Technical Report TR-1053 May, 1981
AFOSR-F49620-80-C-001

Analyzing a Syntactic Family of Complexity Metrics*

Victor R. Basili and David H. Hutchens

Department of Computer Science
University of Maryland
College Park, MD 20742

---

*This work was supported in part by the Air Force Office of Scientific Research Contract AFOSR-F49620-80-C-001 to the University of Maryland. Computer support was provided in part by the Computer Science Center at the University of Maryland.

Abstract

A family of syntactic complexity metrics which contains a number of current metrics is defined. The family is used as a basis for experimental analysis of metrics. Once the family has been implemented, several metrics may be readily formed and computed. This paper uses the family to compare a few simple syntactic metrics to each other. The study also indicates that individual differences have a large impact on the significance of results where many individuals are used. A metric for determining the relative skills of programmers at handling a given level of complexity is also suggested. The study uses the metrics to demonstrate differences between projects on which a methodology was used versus those on which it was not.

1. Introduction

As computer scientists attempt to understand the software process and product, it is natural to try to measure those aspects of software which seem to affect cost. A major problem in computer science is the intellectual control of design which is directly related to the complexity of the product. Many attempts at quantifying the complexity of computer programs have been made [Basili & Reiter 79, Chen, Curtis et al, Dunsmore & Gannon 80, Halstead, McCabe, Sunohara et al]. A good complexity metric could be used as a quality assurance test by software developers and even as a contractual obligation. Current complexity metrics may be roughly divided into two basic groups, (1) static metrics which are measures of the product at one particular point in time, and (2) history metrics which are measures of the product and process taken over time. This paper will deal with static complexity metrics, based upon the physical attributes of a software product. These fall into three basic categories: volume, control organization, and data organization. Each of these categories will be discussed briefly below.

Volume metrics are measures of the size of a product; for example, the number of lines of code, the number of statements, or the number of operators and operands [Halstead]. Of course, the software science volume metric is in this group. Even cyclomatic complexity [McCabe] can be placed in this category since it is the number of decisions plus one. The number of procedures, the average length of procedures, and the number of

variables are examples of volume metrics. The number of input/output formats [Carriere & Thibodeau] and other abstraction metrics are volume metrics as well. Note that these are measures of the logical size, rather than just the physical size, of a program.

Control organization metrics are measures of the comprehensibility of control structures. Thus cyclomatic complexity, when viewed as the number of control paths, is also a control metric. Knots [Woodward et al] and Maximal Intersection Number [Chen] attempt to measure the control complexity by visual properties of program control, either as it is written (in a computer language) or as it would appear in a planer flow graph. Average nesting level has been shown to be a useful control organization metric [Dunsmore]. Essential complexity [McCabe], which will be discussed later, falls in this category as well.

Data organization metrics are measures of data visibility and use as well as the interactions between data within a program. Data binding [Basili & Turner 76; Stevens, Myers & Constantine] is an example of a module interaction metric. A span [Elshoff] is an attempt to measure the proximity of references to each data item. As such it qualifies as a data organization metric. Slicing [Weiser] can also be considered as a data organization metric. A slice is that (not necessarily consecutive) portion of code which is necessary in order to produce some partial output from the program.

2. Definition and Analysis of a Structural Metric Family

All of the above metrics have failed to gain full acceptance as a valid measure of program complexity for at least two reasons. First, there is a lack of experimental evidence to determine what aspects of the system life cycle the metric actually explains. While a metric could correlate well with debugging time, it might still be a poor predictor of the effort required to do maintenance. We need experimental evidence that is focused on the expected uses of the metrics. Second, existing metrics are static (non-parameterized) so they cannot be tuned to the results of exploratory analysis.

A complete development of the structural family of complexity metrics may be found in [Basili & Hutchens]. The structural family includes many metrics from the volume and control organization groups. The data organization group is a subject for future research.

If the family is to include many of the metrics in the literature it must incorporate length, nesting level, control paths, types of control structures, and decomposition simplicity. The family should transcend languages (although specific members may not). The various members may relate to many aspects of software development and maintenance although any one metric may only be useful in a limited way.

Length can be measured by lines of code, with or without comments. However, in a free format language this measure can be

altered by cosmetic revisions of the code, so the number of statements seems to be a more consistent measure. Nesting level might be included explicitly or as a factor to be multiplied with the complexity of the lower levels. Control paths and types of control structures are closely related and are handled in a variety of ways by current metrics, so the family must allow a general mechanism for these concepts. Decomposition simplicity is intended to measure the naturalness with which the intended function is broken into smaller functions.

With these concepts in mind, a recursive definition of a family of control structure complexity metrics (c) could be given by:

$$c(p) = b \sum_{i=1}^{k} c(pi) \quad + \quad f(n,lev,t,s)$$

where p is a program which is decomposed in some fashion into k components p1, p2, ..., pk. The parameter b is used to generate the multiplier for nesting level. The function f, the key to the metric, has four arguments: n, the number of decisions in program p which are not part of a particular subcomponent; lev, the nesting level of component p; t, the type of structure instantiated by p; and s, the structural "niceness" of p.

Some discussion of, and restrictions on, the parameters will clarify their meaning. b is intended to penalize nesting so $b \geq 1$; where, of course, b=1 just removes it from the formula. Since an increase in the number of decisions should not decrease the

complexity, f should be a nondecreasing function of n. At first glance, one might be tempted to place a nondecreasing condition on f with respect to the level, lev. However, there is reason to believe that a concave up function (one which falls at first and latter rises) of lev may be better [Dunsmore]. An example may be found in [Basili & Hutchens].

It should be noted that b is in fact superfluous, for the metric

$$c(p) = b \sum_{i=1}^{k} c(pi) + f(n,lev,t,s)$$

$$= \sum_{i=1}^{k} c(pi) + f'(n,lev,t,s)$$

$$\text{where } f'(n,lev,t,s) = b^{lev} f(n,lev,t,s).$$

In this example, b is reduced to a constant in the function f'. The use of the constant, b, makes the penalties more explicit than does hiding that information in the function. Indeed, many instantiations may use b instead of lev.

The values of t normally range over syntactic entities, such as while, case, and if statements. The parameter s is used to determine if the control structure is "nice." More specifically it normally has two values, "structured construct" and "non-structured" construct.

The control flow of a program may be described by a digraph. A program (equating the program and its digraph) is called a

proper program if it has a single entry and a single exit, and every node of the program lies on some path from the entry to the exit. A proper program is called a prime program if it contains no proper subprograms with two or more nodes. The usual while do od and if then else fi are examples of common prime programs. A prime decomposition is found by continually replacing prime subprograms by function nodes (a node with a single entry and a single exit). A proper program has a unique prime decomposition if successive sequences are treated as a unit [Linger, Mills & Witt].

By letting the parameter s have the two values 1) proper and 2) not proper, the resulting (sub)family is given by:

$$c(p) = b \sum_{i=1}^{k} c(pi) + \begin{cases} f(n,lev,t) & ; \ p \text{ proper} \\ g(n,lev,t) & ; \ p \text{ not proper} \end{cases}$$

This restricted family will be the subject of the rest of this paper. If one assumes that proper programs are less complex than non-proper programs then $f(n,lev,t) \leq g(n,lev,t)$ for all n, lev, and t. The restricted family might reasonably be called a syntactic complexity family since it is based on the syntactic decompositions of the program.

3. Some Members of the Family

The decomposition of p into p1, p2, ..., pk can be based on the syntactic structure of the language. One major benefit of this approach is the ease with which a compiler can be changed

into an automatic metric tool. As a simple example, consider the decomposition of programs into statements (and statements into substatements) where

$$c(p) = \sum_{i=1}^{k} c(pi) + \begin{cases} 1 & ; \text{ p a statement} \\ 0 & ; \text{ otherwise.} \end{cases}$$

Note that this uses the t parameter of the family. The resultant measure is nothing more than a statement count volume metric.

Cyclomatic complexity may be generated by counting the decision constructs in the program plus the number of segments [McCabe]. The measure is just

$$c(p) = \sum_{i=1}^{k} c(pi) + \begin{cases} 1 & ; \text{ p a decision or segment} \\ 0 & ; \text{ otherwise} \end{cases}$$

and eventually each decision will be counted exactly once. Therefore, the member is just the cyclomatic complexity.

The last example of the members of the family follows:

$$(1) \quad c(p) = 1.1 \sum_{i=1}^{k} c(pi) + \begin{cases} 1+\log2(n+1) & ; \text{ p proper} \\ 2*(1+\log2(n+1)) & ; \text{ p not proper} \end{cases}$$

This member exhibits some of the flexibility of the family. The b value of 1.1 penalizes nesting by counting each statement 10% more than it would be at the next outer level. Furthermore, poorly structured code cost twice as much as well structured code. Each statement must contribute at least one to the measure due to the addition of 1 in each of the functions f and g. The

use of the logarithm encourages the use of case statements, the only standard control structure with more than one decision node. Thus, this metric includes consideration of nesting level, length (statement count), structured programming practices, and bonuses for use of an organizing construct (the case statement).

Several other members of the family, including essential complexity and the software science count of total operators and operands, are derived in [Basili & Hutchens].

4. Experimentation

This research focuses on the ability of product metrics to explain the number of program changes made during development as well as the differences in the metrics caused by different development strategies. Program changes were defined by [Dunsmore & Gannon 77] and shown to be closely related to the number of errors made during development. The product metrics used are from the syntactic complexity family. The syntactic complexity family with proper verses not proper statement distinctions has been implemented in the SIMPL-T compiler [Basili & Turner 75]. SIMPL-T is a GOTO-less non-block structured language which allows statement nesting. Loops may be abnormally exited using the EXIT statement and RETURNs are allowed at any point. SIMPL-T is used in many courses at the University of Maryland.

The research reported in this paper uses a database of 19 compilers written by upperclassmen and graduate students at the University of Maryland. The compilers were written under three

different development methodologies: ad hoc individuals (AI), ad hoc teams (AT), and disciplined teams (DT). The ad hoc individuals and ad hoc teams were not given any particular methodologies or techniques to be used in the implementation. They were free to organize the project in any way they desired. The disciplined teams were required to use a list of methodologies and techniques which were taught in their class. These methodologies included chief programmer teams, walkthroughs, and top down design with PDL, among others. Several metrics have already been tested to see if they detect the differences among the groups [Basili & Reiter 79,81].

The results reported here deal with the metric defined in equation (1) (referred to as SC in the rest of this paper) as well as statement counts (ST), call count (CA, the number of calls including procedures and functions whether user defined or predefined in the language) cyclomatic complexity(CV), and the number of decision statements (DS). Appendix 1 contains the coefficients of determination (r square) and the slope of the lines for each of the projects and each of the metrics using simple regression analysis [Neter & Wasserman].

The five metrics considered here are highly correlated as may be seen in the correlation matrix in Table 1. For this reason, multiple regression equations tended to be erratic, with the coefficients changing greatly with the addition of new variables, while producing minimal increases in R square. Of the 19 projects, the addition of a second variable had one of three basic

Correlation Matrix
for the product metrics

	ST	SC	CA	CV	DS
ST	1.000				
SC	.975	1.000			
CA	.845	.770	1.000		
CV	.879	.893	.747	1.000	
DS	.873	.939	.617	.832	1.000

Table 1

results. Four of the projects found no second variable to be of any value at all. Five of the projects found a second variable that was moderately useful. The ten remaining projects added a second variable whose coefficient was negative, while doubling or tripling the coefficient of the original variable, indicating that the two variables were highly related and making the regression model questionable. Therefore, the rest of this paper deals only with simple regression equations.

4.1. The Effects of Individuals

The first attempt at comparing the metrics with the number of program changes was done between projects. That is, the number of program changes for each project was determined and the metrics were computed on each segment (procedure or function) in each project. The complexity of the project was then computed by several methods. These include summing the complexity of each segment as well as summing only the most complex segments (such as the top 10 or 20 percent). All of these attempts at correlat-

ing the number of program changes to the complexity of the project were unsuccessful.

Graph 1 gives an example of the scatterplot of a metric (in this case SC) with the number of program changes, where both values are summed over each project. The relationship between the metric and the number of program changes is almost non-existent.

However, each of the complexities of the individual segments in one project was correlated to the number of program changes made in that particular segment. Only the 6 projects that were developed by ad hoc individuals were used in this part of the study. The coefficient of determination (r square) for SC as a predictor of program changes ranged between .475 and .866 for these 6 projects. The other metrics had slightly lower values but a similar spread (see Appendix 1). Therefore, when an individual is isolated it appears that these metrics do correlate well with the number of program changes.

Graph 2 gives an example plot of a metric (again SC) with program changes where only one ad hoc individual's project is considered. Each point represents a segment (procedure or function) in the final delivered product. It is clear that this approach yields a much closer relationship between the variables of interest than the inter-product comparison of Graph 1.

It is somewhat surprising that a linear fit does better than an exponential fit for almost all cases in terms of both r square

and the distribution of the residuals. Many have argued that segments should be made very small to control their complexity. An exponential fit would imply that the argument is valid, since the sum of the complexities of several very small segments would be much smaller than the complexity of one larger segment with the same amount of code. However, a linear fit implies that there is no advantage to splitting a large segment into many smaller segments unless duplication of code could be reduced.

The 19 projects did demonstrate a linear fit for all five metrics. Only a couple of isolated fits yielded any improvement using exponential fits. The straight lines do intersect fairly close to the origins, so the poor fit of the exponential is not caused by missing the low valued points due to forcing the curve through the origin.

It is highly probable that the linear model appears to fit best because the segments are so small (the average "maximum segment size" for the 19 projects is 66 statements). The exponential tail might show up if there were larger (more complex) segments. It is also possible that programmers naturally limit themselves to smaller segments where they can handle the complexity level.

More interesting, however, is the fact that the slopes of the fitted lines varied from .157 to .729 for SC. Similar variation exists for the slopes of the other metrics. The slope of the line may, in fact, be viewed as a measure of a programmer's

ability to cope with complexity since it is just the number of program changes he makes in developing a program for each unit of complexity. This interpretation is possible because the intercepts of the regression lines are close to zero. It is this variation which accounts for the lack of results using several projects produced by different people.

Experimentation which combines the work of different people is likely to contain a large amount of noise. Any results which can be obtained are of course not invalidated by the noise. However, results may not show statistically even when they exist in the underlying population. Indeed, this phenomena alone may be the cause of many failed experiments.

4.2. Slope Metrics

Using the slope as an indication of a programmer's ability to cope with complexity gives hope of producing an experiment which can quantify a programmer's limitations with respect to the complexity of various applications. The results might be used for management decisions such as assignment of tasks.

The results presented here, however, do not give a total picture of the individual's ability to cope with complexity. One complexity metric is not powerful enough to represent the difficulty of the task.

Since a single complexity metric is unlikely to cover all aspects of complexity, it may be possible for a programmer to

shift the difficulty of development to unmeasured areas. A vector of metrics (and corresponding slopes) might give a better indication of the ability of the programmer to cope with complexity. In fact, such a vector may be useful in determining how to allocate the available programmer resources so that each is working on problems where the complexity is expected to be of the type that he is most capable of handling.

One advantage of a slope metric is its independence of the specification (as long as the specification is not changing). Note that in the case of this experiment, the specification for each of the segments in a given product is different. It might therefore be possible to take measurements from the regular work of the programmers over a long period of time and avoid the need for a special experiment. Thus the programmers will not need to be specifically aware of the experiment so their performance would not be affected by any reactions to the experimental situation.

The benefit of a derived metric like slope might still be realizable even if the fits are nonlinear. For example, if the relationship is exponential, the value of the exponent might provide a measure of the programmers limitations. The use of metrics in the evaluation of programmer's ability to copy with complexity is an area that warrants considerable research attention.

4.3. Comparison of Metrics

The five members of the family have been compared to see how well they predict the number of changes that were made to each segment. The results are summarized in Table 2.

For each project, the coefficient of determination was compared over the five metrics. Friedman's test [Conover] is employed to determine globally (over all five metrics) whether there is reason to believe that any of the metrics performs significantly differently from the others. After concluding that there is a difference in the metrics at the .02 level of significance, a two-tailed sign test [Siegel] was used pairwise to test the null hypothesis that the metrics have equal predictive value. If the level of significance was less than .20, the alternative hypothesis (that there is a difference) with the direction of difference was listed in Table 2. Otherwise, the two metrics are listed as "=", indicating that we may not reject the null

Metric comparisons
(using the sign test)

Friedman yields a .02 level

"SC = ST"		(10/19)
"SC > CV"	at .167 level	(13/19)
"SC > DS"	at .063 level	(14/19)
"SC > CA"	at .019 level	(15/19)
"CV < ST"	at .019 level	(4/19)
"CV = DS"		(7/19)
"CV = CA"		(10/19)
"DS = ST"		(7/19)
"DS = CA"		(11/19)
"CA < ST"	at .063 level	(5/19)

Table 2

hypothesis. The last column contains the ratio of the times that the first listed metric had a better (higher) r square than the second metric, to the total number of data points in the group.

It should be noted that significance levels of .2 are not particularly strong, and in fact anything over .05 is perhaps best read as indicative of a possible trend but inconclusive in the current study.

The results show that ST does better than CV and CA in explaining the number of program changes. Moreover, there is an indication that SC is better than CA, CV, or DS. There is no distinguishable difference between SC and ST or between CA, CV, and DS.

Since the statement count is very easy to calculate and many researchers have found that it does a credible job of measuring the complexity, it must be considered as the metric to beat in studies of this kind. This study has failed to find a metric that is significantly better than statement count.

4.4. Comparison of Methodologies

The five metrics were used to compare the different groups of teams. This part of the study uses the Kruskal-Wallis test and the Mann-Whitney U test [Siegel] to determine if a particular group appears to have a better value than another. The groups were compared with respect to the values of the slope and the coefficient of determination. Note that the slope of the line

has units of changes per unit of complexity. Thus the larger the slope, the more changes made in the face of a given level of complexity and (supposedly) the worse the methodology or programmer which produced it. Statistically, the coefficient of determination is a measure of the amount of variation in the dependent variable (program changes) that may be explained by the variation in the independent variable (the product metric). In this case, we contend that the coefficient of determination is a measure of the consistency of the group with respect to spreading the problems evenly through the code (as measured by the complexity metric). Under the hypothesis that methodology makes a group act more like an individual in terms of the consistency, one would expect that the disciplined teams would have a coefficient of determination which is slightly lower (less consistent) than the ad hoc individuals but larger (more consistent) than the ad hoc teams. The results appear in Tables 3 and 4. The CALLS metric does not appear in these tables because none of the statistics are significant with regard to it. Appendix 2 shows the raw data sorted and displayed to illustrate the contribution of each group.

The Kruskal-Wallis test yields a significance level of between .02 and .05 (depending upon the metric) in favor of there being some difference among the slopes of the groups.

As may be seen in Table 3, the slope of the line is larger for the ad hoc individuals. This means that the disciplined teams do a better job (by requiring fewer program changes) for a

Methodology Comparisons
(using the Mann-Whitney U test)

slopes

SC

Kruskal-Wallis at .05 level
Mann-Whitney AI > AT at .132 level
AI > DT at .014 level
AT = DT

ST

Kruskal-Wallis at .05 level
Mann-Whitney AI > AT at .094 level
AI > DT at .014 level
AT = DT

CV

Kruskal-Wallis at .02 level
Mann-Whitney AI > AT at .180 level
AI > DT at .008 level
AT > DT at .074 level

DS

Kruskal-Wallis at .02 level
Mann-Whitney AI > AT at .026 level
AI > DT at .008 level
AT = DT

Table 3

given amount of complexity than the ad hoc individuals.

Furthermore, the ad hoc teams did better at coping with syntactic complexity than the ad hoc individuals (particularly for the DS metric), indicating that teamwork, even when undisciplined, can show some advantages. The disciplined teams do show a superiority over the ad hoc teams for the CV metric.

For the coefficient of determination, the Kruskal-Wallis test gives a significance level of .03 to .10 in favor of there being some difference among the groups. The ad hoc teams seem to have a lower coefficient of determination than the ad hoc individuals. It is conjectured that this results from the differing

Methodology Comparisons
(using the Mann-Whitney U test)

r square

SC		
	Kruskal-Wallis at .03 level	
	Mann-Whitney AI > AT at .016 level	
	AI > DT at .180 level	
	AT < DT at .052 level	
ST		
	Kruskal-Wallis at .10 level	
	Mann-Whitney AI > AT at .026 level	
	AI = DT	
	AT < DT at .034 level	
CV		
	Kruskal-Wallis at .10 level	
	Mann-Whitney AI > AT at .016 level	
	AI > DT at .128 level	
	AT = DT	
DS		
	Kruskal-Wallis at .10 level	
	Mann-Whitney AI > AT at .042 level	
	AI > DT at .180 level	
	AT < DT at .138 level	

Table 4

abilities of the members of the ad hoc teams causing different parts of the system to be assembled with varying degrees of effectiveness. It is interesting to note that the disciplined teams appear to fall somewhere between the other two groups. This also indicates that a team that works with a set of methodologies tends to be more consistent than a team that does not. In fact, the data indicating that the disciplined teams have a lower coefficient of determination than ad hoc individuals is very weak, lending more justification to the consistency hypothesis.

5. Orthogonality of the metrics

If the complexity of computer programs is to be measured, it is necessary to develop metrics which have a degree of orthogonality, i.e. metrics which measure different aspects of the complexity. As was seen in the correlation matrix of Table 1, the metrics considered so far lack this property. One possible way to gain some orthogonality is to normalize the metrics. For example, if cyclomatic complexity is normalized with respect to length (by computing CV/ST) the resulting metric is a measure of decision density in the code. One might then ask if code with a high decision density requires more program changes than code with a low decision density. For our data, the answer is no. Similar results (or lack thereof) hold for CA and DS normalized by ST.

A mild relationship does seem to exist between SC/ST and program changes. It is conjectured, however, that this is so because the nesting penalties as defined for SC produce a multiple of length for nested statements, so that SC/ST is related to nesting depth which is related to length. The normalized metrics were also tried in multiple regression equations with the all of the original metrics, using incremental regression techniques [Neter & Wasserman]. The normalized metrics proved to yield little additional information in the equations. Hence no truly orthogonal metrics within the study of syntactic metrics which help explain program changes have been uncovered.

We believe that orthogonal metrics may exist outside the realm of syntactic complexity. Metrics that measure other properties of programs and program development, e.g. data metrics [Basili & Turner 76; Dunsmore; Elshoff; Stevens, Myers & Constantine; Weiser], may prove orthogonal to the control structure metrics studied here. We are currently investigating a variety of metric classes.

6. Conclusions

A family of syntactic complexity metrics has been defined which encompasses many of the current metrics. The family has been used in comparing different individuals, metrics, and development methodologies.

It was found that individuals differ widely in the number of program changes required to implement a program of some given complexity. This leads to the possibility of measuring a programmer's ability to cope with complexity. This ability measure concept should be pursued with complexity metrics from other groups of metrics (such as data complexity metrics).

Furthermore, we have some evidence that a team acts more capably than an individual as measured by the slopes of the fitted regression lines. This lends support to the argument that even small projects which one person might be able to do will none the less be done better if more than one person cooperates in the development (at least when they take active steps, such as the use of various methodologies to aid in their communication).

Finally, it was shown that ad hoc teams show far less consistency in their ability to cope with complexity than individuals or disciplined teams, giving further support to the claim that a disciplined team tends to gain the advantages of a team (lower slope) while maintaining the more consistent properties of a single individual.

7. Acknowledgements

The authors would like to thank H.E. Dunsmore and G. Sutton for their efforts in counting the program changes in the projects. We are also indebted to John Gannon for his comments and suggestions.

8. References

[Basili & Hutchens]
V.R. Basili and D.H. Hutchens, "A Study of a Family of Structural Complexity Metrics," Proc. ACM-NBS Nineteenth Annual Technical Symposium: Pathways to System Integrity, Gaithersburg, MD. June 1980, pp. 13-15.

[Basili & Reiter 79]
V.R. Basili and R.W. Reiter, "An Investigation of Human Factors in Software Development," Computer, Dec. 1979, pp. 21-38.

[Basili & Reiter 81]
V.R. Basili and R.W. Reiter, "A Controlled Experiment Quantitatively Comparing Software Development Approaches," IEEE Transactions on Software Engineering, May 1981.

[Basili & Turner 75]
V.R. Basili and A.J. Turner, SIMPL-T: A Structured Programming Language, Paladin House Publishers, Geneva, Ill. 1976.

[Basili & Turner 76]
V.R. Basili and A.J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," IEEE Transactions on Software Engineering, Vol. 1, No. 4, Dec. 1975, pp. 390-396.

[Carriere & Thibodeau]
W.M. Carriere and R. Thibodeau, "Development of A Logistics Software Cost Estimating Technique for Foreign Military Sales," General Research Corporation, Santa Barbara, California, June '79.

[Chen]
E.T. Chen, "Program Complexity and Programmer Productivity," IEEE Transactions on Software Engineering, Vol. 4, No 3, May 1978, pp. 187-193.

[Conover]
W.J. Conover, Practical Nonparametric Statistics, John Wiley & Sons, New York, NY, 1971.

[Curtis et al]
B.Curtis, S.B. Sheppard, P.Milliman, M.A. Borst, and T. Love, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," IEEE Transactions on Software Engineering, March 1979, pp. 96-104.

[Dunsmore]
H.E. Dunsmore, "The Influence of Programming Factors on Program Complexity," Ph.D. Diss., Dept. of Computer Science,

University of Maryland, July '78.

[Dunsmore & Gannon 77]
H.E. Dunsmore and J.D. Gannon, "Experimental Investigations of Programming Complexity," Proc. ACM-NBS Sixteenth Annual Technical Symposium: Systems and Software, Wash., D.C., June 1977, pp. 117-225.

[Dunsmore & Gannon 80]
H.E. Dunsmore & J.D. Gannon, "Analysis of the Effects of Programming Factors on Programming Effort," The Journal of Systems and Software 1, 1980, pp. 265-273.

[Elshoff]
J.L. Elshoff, "An Analysis of some Commercial PL/1 Programs," IEEE Transactions on Software Engineering, Vol. 2, No 2, June 1976, pp. 113-120.

[Halstead]
M. Halstead, Elements of Software Science, Elsevier Computer Science Library, 1977.

[Linger, Mills & Witt]
R.C. Linger, H.D. Mills, B.I.Witt, Structured Programming: Theory and Practice, Addison-Wesley, Reading Mass., 1979.

[McCabe]
T.J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. 2, No. 4, Dec. 1976, pp. 308-320.

[Neter & Wasserman]
J. Neter and W. Wasserman, Applied Linear Statistical Models, Richard D. Irwin, INC., Homewood Ill., 1974.

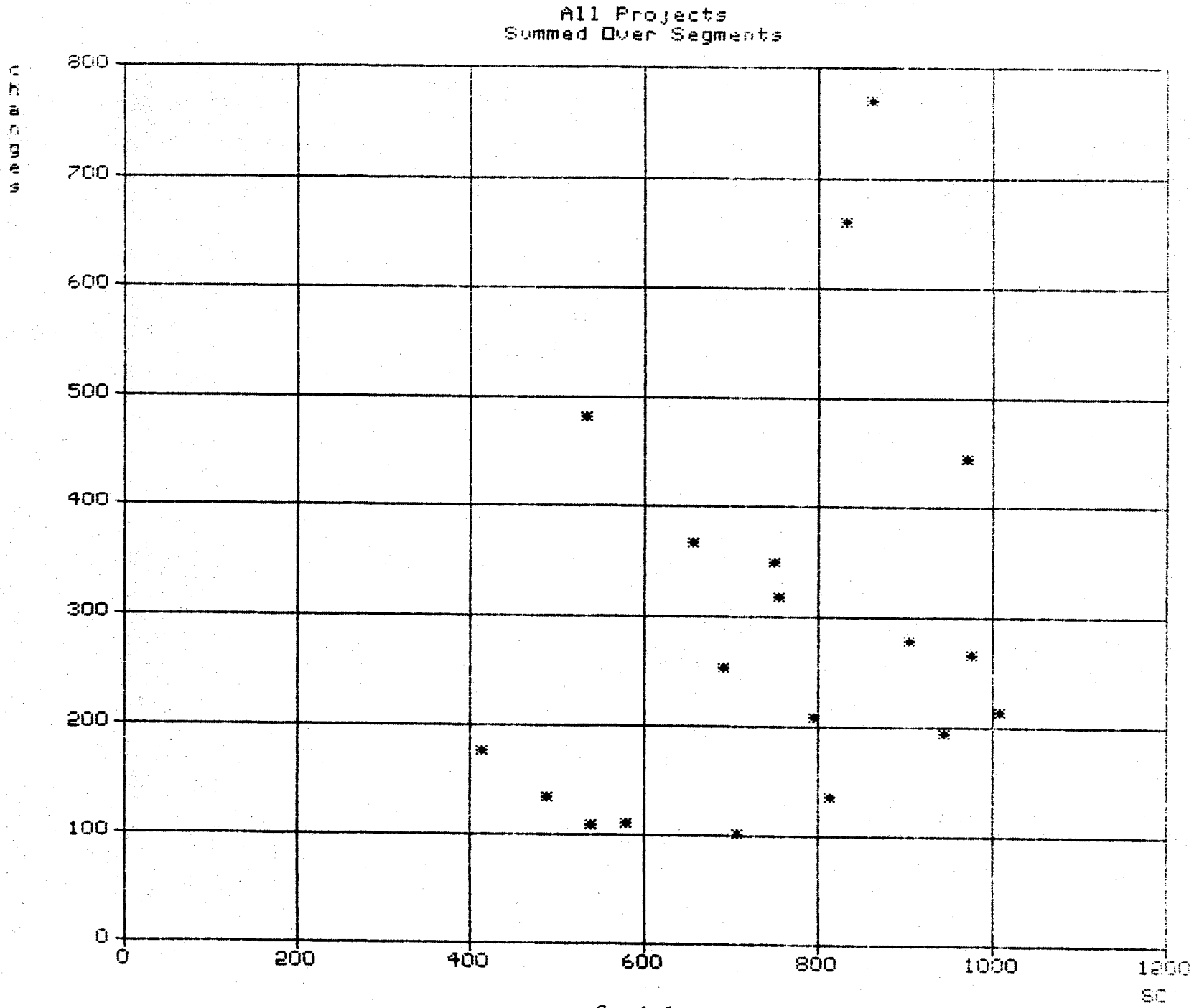
[Siegel]
S. Siegel, Nonparametric Statistics, McGraw-Hill Book Company, New York, NY, 1956.

[Stevens, Myres, & Constantine]
W.P. Stevens, G.J. Myres and L.L. Constantine, "Structural Design," IBM Systems Journal, Vol. 13, No. 2, 1974, pp. 115-139.

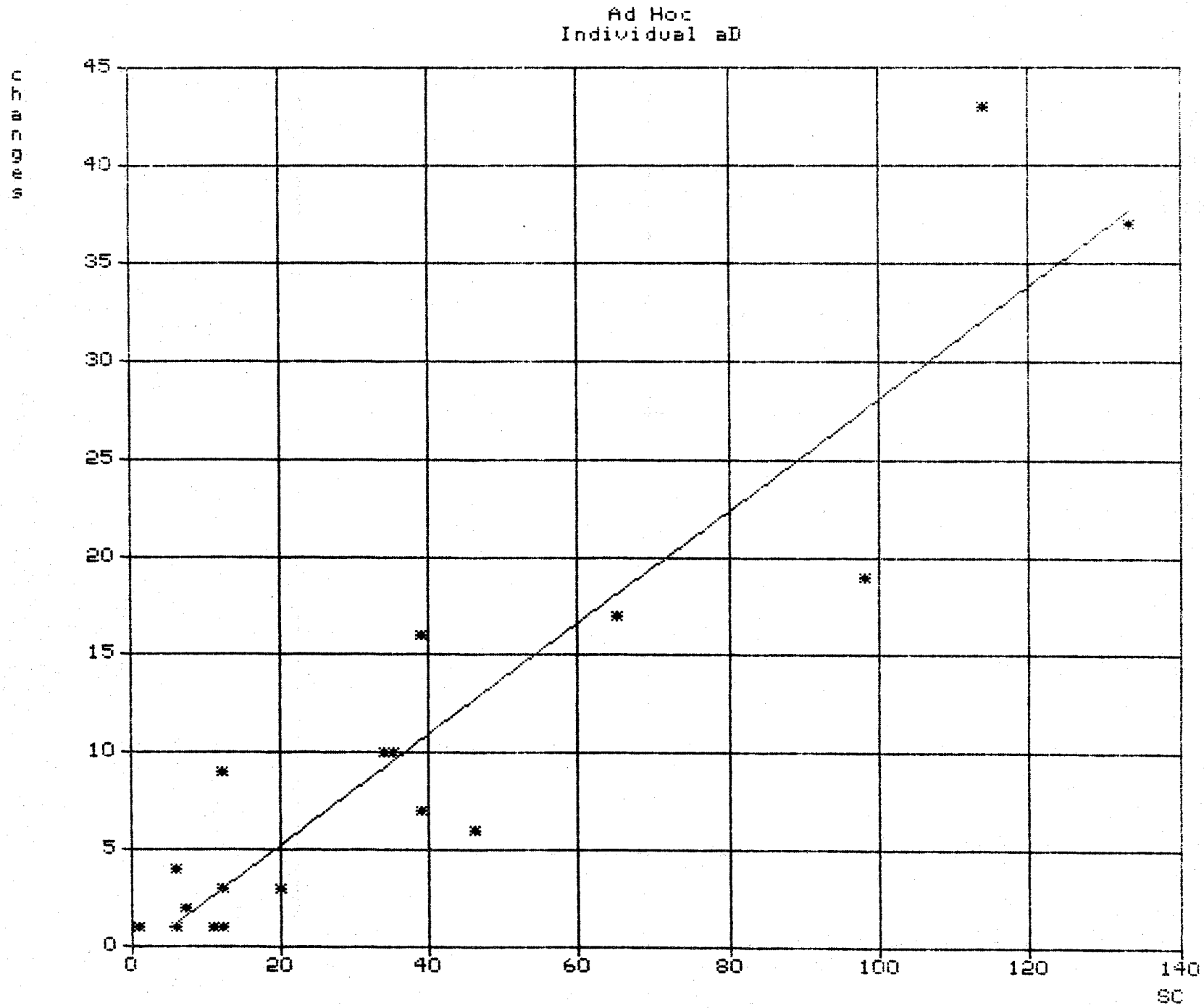
[Sunohara et al]
T. Sunohara, A. Takano, K. Vehara, and T. Ohkawa, "Program Complexity Measure for Software Development Management," Proc. Fifth International Conference on Software Engineering, San Diego, California, March 9-12, 1981, pp. 100-106.

[Weiser]
M.D. Weiser, "Program Slices: Theoretical, Psychological, and Practical Investigations of an Automatic Program Abstraction Method," Ph.D. Diss., Univ. of Michigan, 1979.

[Woodward et al]
M.R. Woodward, M.A. Hennell, and D. Hedly, "A Measure of Control Flow Complexity in Program Text," IEEE Transactions on Software Engineering, Jan. 1979, pp. 45-50.


All Projects
Summed Over Segments
changes
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000
1200
SC

Graph 1.


Ad Hoc
Individual aD
changes
45
40
35
30
25
20
15
10
5
0
0
20
40
60
80
100
120
140
SC

Graph 2.

9. Appendix 1

Slope and Coefficient of Determination Data

slope

	SC	ST	CA	CV	DS
	.729	1.162	1.411	1.776	4.567
	.286	.397	.443	1.013	3.044
AI	.157	.277	.469	.440	1.076
	.576	.809	1.460	2.121	3.114
	.499	.927	3.859	2.788	2.950
	.437	.684	.406	1.318	2.774
	.204	.319	.547	.531	1.060
	.492	.785	1.775	1.244	2.904
AT	.085	.121	.118	.289	.640
	.173	.244	.242	.799	.841
	.456	.743	1.736	1.992	2.840
	.128	.254	.502	.621	.811
	.155	.239	.510	.258	1.035
	.142	.193	.362	.372	1.078
	.161	.278	.390	.583	.887
DT	.102	.143	.181	.281	.932
	.297	.524	.823	.694	1.627
	.189	.320	.542	.499	1.319
	.141	.210	.323	.431	.812

r square

	SC	ST	CA	CV	DS
	.475	.447	.104	.288	.368
	.866	.800	.556	.595	.852
AI	.717	.679	.487	.525	.733
	.521	.469	.454	.396	.372
	.739	.838	.489	.683	.712
	.592	.638	.075	.627	.450
	.490	.504	.289	.257	.376
	.322	.287	.380	.177	.325
AT	.170	.149	.078	.187	.207
	.054	.051	.024	.086	.042
	.585	.551	.533	.589	.515
	.207	.227	.319	.210	.232
	.335	.358	.257	.065	.302
	.351	.309	.312	.163	.382
	.724	.790	.705	.522	.480
DT	.660	.725	.872	.735	.531
	.499	.558	.734	.321	.336
	.682	.625	.398	.494	.672
	.469	.484	.337	.350	.288

10. Appendix 2

Sorted Raw Data
(used by Mann-Whitney U test)

r s q u a r e

CV			DS			SC			ST		
AI	AT	DT	AI	AT	DT	AI	AT	DT	AI	AT	DT
		.065		.042			.054			.051	
	.086			.207			.170			.149	
		.163		.232			.207			.227	
	.177				.288		.322			.287	
	.187				.302			.335			.309
	.210			.325				.351			.358
	.257				.336			.469	.447		
.288			.368			.475			.469		
		.321	.372				.490				.484
		.350		.376				.499		.504	
.396					.382	.521				.551	
		.494	.450				.585				.558
		.522			.480	.592					.625
.525				.515				.660	.638		
	.589				.531			.682	.679		
.595					.672	.717					.725
.627			.712					.724			.790
.683			.733			.739			.800		
		.735	.852			.866			.838		

s l o p e

CV			DS			SC			ST		
AI	AT	DT	AI	AT	DT	AI	AT	DT	AI	AT	DT
		.258		.640			.085			.121	
		.281		.811				.102			.143
	.289				.812		.128				.193
		.372		.841				.141			.210
		.431			.887			.142			.239
.440					.932			.155		.244	
		.499			1.035	.157				.254	
	.531			1.060				.161	.277		
		.583	1.076				.173				.278
	.621				1.078			.189		.319	
		.694			1.319		.204				.320
	.799				1.627	.286			.397		
1.013			2.774					.297			.524
	1.244			2.840		.437			.684		
1.318				2.904			.456			.743	
1.776			2.950				.492			.785	
	1.992		3.044			.499			.809		
2.121			3.114			.576			.927		
2.788			4.567			.729			1.162		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* ANALYZING A SYNTACTIC FAMILY OF COMPLEXITY METRICS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-1053
7. AUTHOR(s) Victor R. Basili and David H. Hutchens		8. CONTRACT OR GRANT NUMBER(s) AFOSR-F4-49620-80-C-001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science University or Maryland College Park, Maryland, 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Math. & Info. Sciences, AFOSR Bolling AFB Washington, D. C. 20332		12. REPORT DATE May 1981 13. NUMBER OF PAGES 29
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

software experiments, software metrics, control structure metrics

structural complexity, syntactic complexity

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A family of syntactic complexity metrics which contains a number of current metrics is defined. The family is used as a basis for experimental analysis of metrics. Once the family has been implemented, several metrics may be readily formed and computed. This paper uses the family to compare a few simple syntactic metrics to each other. The study also indicates that individual differences have a large impact on the significance of results where many individuals are used. A metric for determining the relative skills of programmers at handling a given level of complexity is also suggested. The study uses the metrics to demonstrate differences between projects on which a methodology was used vs. those on which it was not.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*