5/7-61

N94-35559

1-19

The Development and Application of Composite Complexity Models and a Relative Complexity Metric in a Software Maintenance Environment

J. M. Hops
Radio Frequency and Microwave Subsystems Section

J. S. Sherif
Software Product Assurance Section
and
California State University, Fullerton

A great deal of effort is now being devoted to the study, analysis, prediction, and minimization of software maintenance expected cost, long before software is delivered to users or customers. It has been estimated that, on the average, the effort spent on software maintenance is as costly as the effort spent on all other software costs. Software design methods should be the starting point to aid in alleviating the problems of software maintenance complexity and high costs. Two aspects of maintenance deserve attention: (1) protocols for locating and rectifying defects, and for ensuring that no new defects are introduced in the development phase of the software process, and (2) protocols for modification, enhancement, and upgrading. This article focuses primarily on the second aspect, the development of protocols to help increase the quality and reduce the costs associated with modifications, enhancements, and upgrades of existing software. This study developed parsimonious models and a relative complexity metric for complexity measurement of software that were used to rank the modules in the system relative to one another. Some success was achieved in using the models and the relative metric to identify maintenance-prone modules.

I. Introduction

A. Project Objectives

The primary objective of this study was to determine whether software metrics could help guide our efforts in the development and maintenance of the real-time embedded systems that we develop for NASA's Deep Space Network (DSN). Generally, the systems that are developed control receivers, transmitters, exciters, and signal paths through the communication hardware. The most common programming language in our systems is PL/M for Intel 8080, 8086, and 80286 microprocessors; and the systems range in size from 20,000 to 100,000 non-commented lines of code (NCLOC). Approximately 65 percent of the fund-

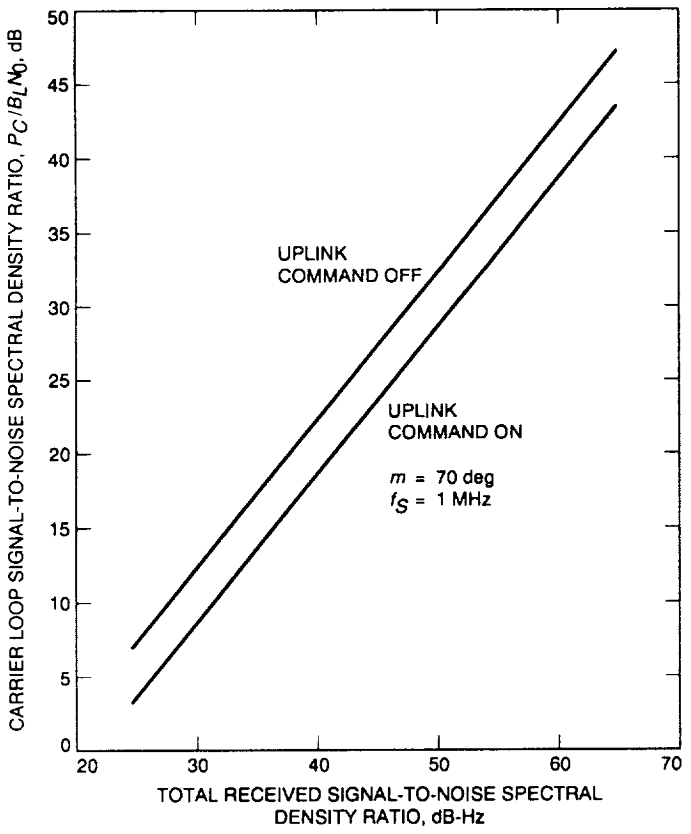

Fig. 13. Carrier loop received signal-to-noise spectral density ratio versus total received signal-to-noise spectral density ratio.

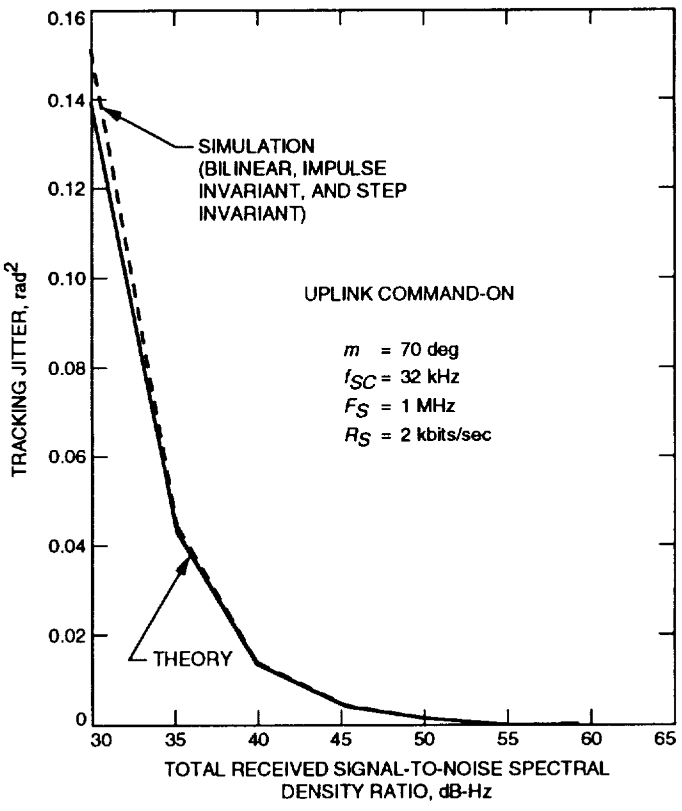

Fig. 14. Comparison of theoretical and simulated tracking phase jitter.

ing received in our environment is dedicated to extending the life span of the previously developed systems; of this, 15 percent is spent on finding and fixing defects, while 85 percent is spent on adding automation features, adding capabilities, and increasing capacity.

Our efforts have been successful in that the life spans of our systems now range from 4 to 8 years and are increasing. As support for new spacecraft becomes necessary, these older systems are being used in new ways, thereby increasing the importance of high-quality, defect-free, and cost-effective enhancements to the software. Protocols and guidance for locating and rectifying defects in the software-sustaining environment were deemed critical, especially with the added complications that the maintainers of the systems are not the original developers and that there is little or no confidence in the software documentation.

Specifically, we were looking for ways to identify which modules should be reengineered and which modules would need extra development and test time in order to maintain. The problems we face in our environment are quite common in the industry. Software maintenance cost is about two to four times the original development cost [3,13,10,21]. Charette [5] emphasizes the fact that 60 to 80 percent of the total software costs are related to maintenance. This will likely remain so for the indefinite future [7,11,24].

Figure 1 shows the initial cost breakdown in developing a new project (unfortunately with maintenance costs hidden), and Fig. 2 shows the costs of software during its life cycle, as discussed by Zelkowitz [34]. Software maintenance is not what people think it is: Software maintenance actually encompasses fixing software errors in addition to software enhancements and adding new functions to existing systems, system conversion, training and supporting users, and improving system performance [31–33]. Error correction, which is often perceived as the substance of maintenance, is only a small part of the software maintenance effort [8,4]. Table 1 shows the distribution of the average time spent on various maintenance tasks for 4 years, as reported by Lientz and Swanson [19]. Note that functional enhancement constitutes the major portion of the time spent on software maintenance. Charette [5] discusses another reason why the cost of software is so high and cites some statistics as reported by the Comptroller General [6] and as shown in Table 2. It is reported that only 2 percent of the software contracted for could work on delivery; 3 percent could work after some rework; 45 percent was delivered, but was never successfully put to use; 20 percent was used, but was either extensively reworked or abandoned; and 30 percent was paid for, but was never delivered.

For the study described in this article, we took the following steps:

(1) Determined what the literature suggests.

(2) Developed a course of action to be tried on one of our operational systems that would be representative of all the others.

(3) Performed the steps and analyzed the results.

The process and results of each of these steps are described below.

B. Suggestions from the Literature and Course of Action

One of the earlier studies encountered pertaining to our objectives was undertaken by Shen, Yu, Thebaut, and Paulsen [27]. They assessed the potential usefulness of product and process metrics in identifying components of the system that were most likely to contain errors. Their goal was to establish an empirical basis for the use of objective criteria in developing strategies for the allocation of testing effort in the software-maintenance environment. It was found that the number of unique operands, as defined by Halstead [14], was the best predictor of problem reports on modules that were reported after the initial delivery. Additionally, simple metrics related to the number of unique operands, such as the cyclomatic complexity (defined by McCabe [20]), also performed well. Shen et al. concluded that these metrics are useful in finding error-prone modules at an early stage [27].

In 1987, Kafura and Reddy [17] published the results of their study on using software complexity metrics during the software maintenance phase of a system. They related seven separate metrics to the experience of maintenance activities on medium-sized systems. Two of the results reported were that the overall complexity of a system grows with time and that the individual complexity scores of the software modules agree well with the expert opinions of the programmers. Their conclusion was that metrics could form the control element in a formal maintenance method.

Harrison and Cook [15,16] discuss the decision, frequently encountered by software maintenance personnel, of whether to make an isolated change in a module or to totally redesign and rewrite the module anew. They developed an objective decision rule to identify modules that should be rewritten rather than modified. This decision rule is whether the total change in the Halstead software science volume metric exceeds a threshold value. This threshold value seems to be subjective since it depends upon the decision maker's risk-taking propensity and experience and since it must be tuned for a particular environment.

Lennselius, Wohlin, and Vrana [18] discuss the possibility of using complexity metrics to identify error-prone, and thus maintenance-prone, modules. They suggest that a module whose complexity lies at least one standard deviation above the acceptable mean of complexity of the project may be considered to be a maintenance-prone module. The authors, however, emphasize that metrics cannot replace the decision-making process of software managers.

Rodriguez and Tsai [23] use discriminant analysis to develop a methodology to evaluate software metrics. They suggest that when classifying units of software as either complex or normal, more attention is usually paid to the complex group to either redesign it or test it more thoroughly. Their methodology is based on the assumption of normal distribution and homogeneity of variances of the two groups. The authors consider 13 metrics depicting Halstead's software science metrics, McCabe complexity metrics, and NCLOC metrics. They conclude that these metrics are correlated.

Stalhane [29] discusses how to estimate the number of defects in a software unit from various software metrics and how to estimate the reliability of the same software. The author also concludes that complexity increases as the size of code increases. Stalhane asserts that misunderstanding the specifications will increase with the specification complexity and that complexity may be transferred to the code and thus lead to maintenance-prone complex code and complex modules.

Munson and Khoshgoftaar [21] employ factor analytic techniques to reduce the dimensionality of the complexity problem space to produce a set of reduced metrics. The reduced complexity metrics are subsequently combined into a single relative complexity measure for the purpose of comparing and classifying programs. In particular, the relative complexity metric can be seen to represent the complexity of a particular software module at a particular level of system release. The authors investigate McCabe complexity metrics, Halstead software science metrics, and NCLOC metrics. The comparison of complexity is again of a relative and subjective nature.

Binder and Poore [2] investigate the possibility of including the number of comments in the code as a variable in determining the quality of the code. They assert that comments only contribute to quality when they are needed and meaningful. The authors suggest a software quality measure called the "LB-ratio," which is defined as the ratio of the number of operators to the sum of the number of operands and the number of comments. The authors agree that their experiments with the LB-ratio need additional work and refinement since including the concept of meaningful comments in the formula seems to be problematic and subjective at best.

The following suggestions were deduced from these sources:

(1) An estimate of errors and reliability can be determined from software product metrics [20,27,29].

(2) Software product metrics could be used to find error-prone modules and could form the control element in a formal software maintenance methodology [15–18].

(3) The software product metrics that may be considered include all of Halstead's software science metrics, McCabe's complexity metric [14,23,27], and NCLOC [21].

(4) Factor analysis can be used to identify those software measures that are highly and significantly related to all other measures. This economy of description will facilitate the analysis of software complexity [21].

(5) Comments in the code contribute to the quality of software [2].

We therefore took the following actions:

(1) Determined the Halstead software science, McCabe complexity, NCLOC, and LB-ratio from sequential releases of a representative software system.

(2) Performed factor analysis on the metrics from the software modules to determine the unique dimensions represented by the metrics.

(3) Proposed a model to calculate a relative metric.

(4) Determined if this metric can identify maintenance-prone modules in the software by using the mean-plus-one standard deviation as the relative metric cut-off value.

II. Method, Analysis, and Results

A. Representative System and Metrics Collection

1. Nature of Software. We analyzed the source program in the very long baseline interferometry (VLBI) receiver controller (VRC) software system by using factor analysis for 16 software measures. The source program is a real-time embedded system in the receiver–exciter subsystem of NASA's DSN. It serves as a communication interface to VLBI subsystems and configures and monitors the status of the narrow-channel bandwidth VLBI receiver assembly. Three releases of the system software were analyzed: OP-B (222 modules), OP-C (224 modules), and a draft version of OP-D (235 modules). These were used as a representative maintenance project in this study. The source code for these three releases was originally written in PL/M but was later converted to C using the PLC86 conversion program (from Micro-Processor Services).

2. Software Metrics and Measures. Software metrics are quantitative measures of certain characteristics of a development project that can be valuable management and engineering tools. Software metrics can be used to achieve various project-specific results, such as predicting source-code complexity at the design phase; monitoring and controlling software reliability and functionality; predicting cost and schedule; and identifying high-risk modules in a software project [28].

The 16 software measures that were used to analyze the VRC software are given in Table 3. The first eight measures belong to the Halstead software science family of software complexity measures. Halstead [14] uses a series of software science equations to measure the complexity of a program based on the lexical counts of symbols used. Generally, the measurements are made for each module, and the total measurements of the modules constitute the measurement of the program. Halstead's metrics become available only after the coding is done, and therefore can be of use only during the testing and maintenance phases. Although Halstead's metrics are useful in determining the complexity of programs, their weaknesses are that they do not measure control flow complexity and have little predictive value.

Measures 9 and 10, i.e., VG_1 and VG_2, belong to McCabe and were adapted from the mathematical concepts of graph theory. McCabe cyclomatic complexity metric VG_1 is a measure of the maximum number of linearly independent circuits in a program control graph. The primary purpose of this metric is to identify software modules that will be difficult to test or maintain, as explained by McCabe [20]. The value of the McCabe metric is available only after the detailed design is done. Although the McCabe metric is very useful for measuring control flow complexity, its weakness is that it is not sensitive to program size; for example, if programs of different sizes are composed exclusively of sequential statements, then they may have the same cyclomatic number.

Measures 11–15 deal with the size of the program or the number of lines. Although many researchers do not find this measure as appealing, Boehm [3] points out that no other metric has a clear advantage over NCLOC as a metric. It is easy to measure, is conceptually familiar to software developers, and is used in most productivity databases and cost estimation models.

Measure 16, the LB-ratio, is defined by Binder and Poore [2] as the ratio of the number of operators to the sum of the number of operands and the number of comments. It appears to capture the idea of distinguishing between meaningful comments in the code and just comments in general. The weakness of this metric is its reliance on defining the number of meaningful comments, which seems to be more subjective than quantitative.

B. Analysis of Data, Models, and Validation

The 16 software measures of the three releases of the VRC code, OP-B, OP-C, and draft OP-D, were analyzed using factor analysis, correlation, analysis of variance, and regression analysis. Table 4 shows the number of modules and the mean value per module for each of the 16 measures. Figures 3–5 show the correlation matrix of the 16 measures for the three releases. The data show a high degree of correlation. Except for the LB-ratio measure, the remaining 15 measures are highly correlated. It can be seen that the Halstead volume metric (V), the McCabe cyclomatic complexity metric (VG_1), and the NCLOC metric are highly and significantly correlated, while the LB-ratio metric is not. These results agree with those of other researchers, such as Ramamurthy and Melton [22], Gill and Kemerer [12], Samadzadeh and Nandakumar [25], Basili and Hutchins [1], Evangelist [9], and Kafura and Reddy [17].

The factor analysis matrix is shown in Table 5. All measures except the LB-ratio are loaded on factor 1, and thus there is no cross-loading. This is a desired result, since cross-loading on many factors makes the interpretation of the result ambiguous. The analysis of variance of the three sets of releases did not show any significant difference at the level of significance of 0.05. This means that, on the average, the values of, say, the McCabe cyclomatic complexity metric (VG_1) of the three releases are not significantly different at alpha of 5 percent. The same is also true for the other 15 measures.

Regression analysis had been used to develop models of relationships of the most interrelated measures. These are the Halstead volume metric (V), the McCabe cyclomatic metric (VG_1), and the non-commented lines of code ($NCLOC$) metric, as discussed next.

1. Factor Analysis Discussion. Three releases of software were analyzed by factor analysis to show the existence of meaningful relationships among known software complexity measures. The analysis shows the number of factors where software complexity measures tend to load high or low, and also the percentage of the variability explained by each factor. This research also shows the matrix of correlation summarizing the relationships among the 16 software complexity measures for each release.

Factor analysis of the three releases of software had shown that the first 15 measures of complexity are closely related to some measure of similarity and are consequently all interrelated. However, the 16th complexity measure (LB-ratio) does not seem to be typical of the other 15 measures, and thus it is unlike the rest of the data set. The 3 releases show 2 factors that concisely state the pattern of relationships within the 16 measures. However, measures 1–15 load most strongly on the first factor with explained variability of 90 to 91 percent, while the second factor displays less interesting patterns with loading of 9 to 10 percent. Factor analysis had also shown that three complexity measures, the McCabe cyclomatic complexity metric (VG_1), the Halstead volume metric (V), and ($NCLOC$), are highly and strongly related. Therefore, in order to achieve an economy of description, these three measures are considered to give a strong similarity and representation of all the 15 measures.

The correlation matrix for each release of the software also shows that the first 15 complexity measures are related, while the LB-ratio measure is not related or interrelated to any of the other 15 measures.

Analysis of variance does not show any significant difference between the three releases at the level of significance of 5 percent. This means that as the software evolves through its releases, the interrelationships between the complexity measures seem to be preserved. However, we should note that without normalization to size, adding on to a program will make a more complex program. This seems to agree with the findings of other researchers, as discussed by Valett and McGarry [30], Harrison and Cook [15], and Schneidewind [26].

Since factor analysis techniques showed that the first 15 software measures are closely related to some measure of similarity, and since 3 of these measures, the McCabe cyclomatic complexity metric (VG_1), the Halstead volume metric (V), and the NCLOC metric, are highly and significantly related, they are considered to give a strong similarity and representation of all 15 measures. This economy of description made it appealing to develop a set of parsimonious models for software complexity measurements using data from the three software releases. The five composite models together with their coefficients of determination (R^2) are shown in Table 6.

Statistical analysis, model back testing, and model testing with independent segments of software are used for validation of the composite models and ascertaining their degree of accuracy. The developed models had shown a high degree of accuracy in predicting software complexity, and thus they can serve as a baseline for other software projects in identifying software modules with high complexity (maintenance prone), so that actions can be taken before their release to users.

2. Back Testing of Models. The five composite complexity models shown in Table 6 were checked with actual data from the three releases, OP-B, OP-C, and OP-D. Table 7 and Fig. 6 show the actual average values of the dependent variables (VG_1) and values predicted by the first three models. Table 8 and Fig. 7 show the actual average values of (V) and values predicted by models 4 and 5. It can be seen that the difference in predicting (VG_1) by the first three composite models ranges from 3.2 to 10.6 percent below the actual average value of (VG_1), as calculated by the McCabe cyclomatic complexity metric. Also, the difference in predicting (V) by models 4 and 5 ranges from 1.2 to 1.3 percent above the actual average value of (V), as calculated by Halstead's volume metric.

3. Testing the Five Composite Models by External Check. The five composite complexity models were tested against four independent segments of software with characteristics as shown in Table 9. A sample calculation of actual average values of (VG_1) and values predicted by model 1 for the four segments of software is shown in Table 10. The summary of the actual grand average values of (VG_1) and (V) and their values, as predicted by models 1, 2, and 3 and models 4 and 5, respectively, for the four segments of software, is shown in Tables 11 and 12 and

Figs. 8 and 9. It can be seen that the difference in predicting (VG_1) by the first three composite models ranges from 17.3 percent below to 0.7 percent above the actual average value of (VG_1). Also, the difference in predicting (V) by models 4 and 5 is 9.7 percent above the actual average value of (V) for the four segments of software.

C. Parsimonious Model and Relative Complexity

Since the five complexity models developed in this study show direct relationships between (VG_1) and (V) and also $(NCLOC)$, we chose the third model,

$$<VG1> = 0.786 + 0.0013(V) + 0.0976(NCLOC)$$

as a representative model for estimating the value of (VG_1), given the measured values of (V) and $(NCLOC)$.

1. Development of the Relative Complexity Metric. We propose to capture the total complexity of a program based on its control flow complexity, the lexical counts of symbols used, and the program size. In essence, a complexity metric that accounts for a program total complexity due to volume and control flow and normalized by the number of lines of code would present a relative complexity metric that is more useful to consider for detecting maintenance-prone programs. The relative complexity metric (RCM) will be derived for each module from the measured value of (V), the estimated value of (VG_1) from model 3, and normalized by the module lines of code. The RCM for a module is

$$(RCM)_i = \left(\frac{<VG_1> + V}{NCLOC}\right)_i$$

2. Analysis of the Three Releases Using the Relative Complexity Metric. The RCM was used to analyze the modules of the three releases, as shown in Table 13. Note that, as reported by Kafura and Reddy [17], the RCM has grown with each release, from a 2799 total in OP-B to a 3470 total in the draft of OP-D.

Using the criterion of the mean relative complexity value plus one standard deviation as a cut-off value for acceptable modules, we can identify those modules that can be considered as outliers, or maintenance-prone modules. Results for the three releases are given in Table 14.

In order to determine whether the modules above the cut-off value were more at risk to be modified for enhancement or fixes than modules below the cut-off value, the transitions between the releases were examined. The results appear in Table 15. Of the 33 modules over the cut-off value of RCM in OP-B, 40 percent were actually modified in order to implement OP-C. Of the 36 modules in OP-C over OP-C's RCM cut-off value, 50 percent were actually modified to implement the draft version of OP-D.

Although the cut-off value seems to evenly divide the modules that were actually modified, the modules over the cut-off value for each release were more likely to be changed than the modules below the cut-off value. The RCM was, therefore, able to identify maintenance-prone modules.

III. Discussion and Conclusion

Given that a metric that measures software complexity should prove to be a useful predictor of software maintenance costs, it is recommended that modules that show a high order of complexity within a release be looked upon as modules with a propensity to become maintenance prone after release and delivery to users. It is imperative that a maintenance-prone module be improved, enhanced, or simplified into two or more modules before final delivery. The composite complexity models and the relative complexity metric developed in this study can be considered as a baseline for comparison with other projects and may serve as a set point for simplifying and reducing complexity of developed software.

Acknowledgments

The authors would like to express their sincere thanks to Dr. Harry Detweiler, Manager, Dr. William J. Hurd, Deputy Manager, and Paul A. Willis, Supervisor, Radio Frequency and Microwave Subsystems Section; Dr. Robert C. Tausworthe, Chief Technologist, Information Systems Division; and Dr. Donald S. Remer, Telecommunications and Data Acquisition Planning, for comments and suggestions that greatly improved this article.

References

[1] V. R. Basili and D. H. Hutchins, "An Empirical Study of a Synthetic Complexity Family," *IEEE Transactions on Software Engineering*, vol. 9, no. 6, pp. 664–672, November 1983.

[2] L. H. Binder and J. H. Poore, "Field Experiments With Local Software Quality Metrics," *Software Practice and Experience*, vol. 20, no. 7, pp. 631–647, July 1990.

[3] B. Boehm, *Software Engineering Economics*, Englewood Cliffs, New Jersey: Prentice Hall, 1981.

[4] B. Boehm and P. Papaccio, "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, October 1988.

[5] R. N. Charette, *Software Engineering Environment*, New York: McGraw–Hill, Inc., 1986.

[6] Comptroller General, *Contracting for Computer Software Development*, General Accounting Office Report, FGMSD-80-4, GAO, 1979.

[7] B. Curtis, S. Sheppard, P. Milliman, M. Borst, and T. Love, "Measuring the Psychological Complexity of Software Maintenance Tasks With the Halstead and McCabe Metrics," *IEEE Transactions on Software Engineering*, vol. 5, pp. 96–104, March 1979.

[8] S. Dekleva, "Software Maintenance: Any News Besides the Name," *The Software Practitioner*, vol. 3, no. 3, pp. 5–8, March 1993.

[9] W. M. Evangelist, "Software Complexity Metric Sensitivity to Program Structure Rules," *Journal of Systems and Software*, vol. 3, no. 3, pp. 231–243, March 1983.

[10] R. E. Fairley, *Software Engineering Concepts*, New York: McGraw–Hill, Inc., 1985.

[11] V. R. Gibson and J. A. Senn, "System Structure and Software Maintenance Performance," *Communications ACM*, vol. 32, no. 3, pp. 347–358, March 1989.

[12] G. K. Gill and C. F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Transactions on Software Engineering*, vol. 17, no. 12, pp. 1284–1288, December 1991.

[13] R. L. Glass, *Software Maintenance Handbook*, Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1981.

[14] M. Halstead, *Elements of Software Science*, New York: Elsevier North Holland, Inc., 1977.

[15] W. Harrison and C. Cook, "A Micro/Macro Measure of Software Complexity," *The Journal of Systems and Software*, vol. 7, no. 2, pp. 213–219, August 1987.

[16] W. Harrison and C. Cook, *Insights on Improving The Maintenance Process Through Software Measurements*, Naval Ocean Systems Center Report TR 90-4, N66001-87-D-0136, 1990.

[17] D. Kafura and G. R. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," *IEEE Transactions on Software Engineering*, vol. 13, no. 13, pp. 335–343, March 1987.

[18] B. Lennselius, C. Wohlin, and C. Vrana, "Software Metrics: Fault Content Estimation and Software Process Control," *Microprocessors and Microsystems*, vol. 11, no. 7, pp. 365–375, September 1987.

[19] B. P. Lientz and E. B. Swanson, *Software Maintenance Management*, Reading, Massachusetts: Addison-Wesley, 1990.

[20] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, December 1976.

[21] J. C. Munson and T. M. Khoshgoftaar, "Application of a Relative Complexity Metric for Software Project Management," *Journal of Systems and Software*, vol. 12, no. 3, pp. 283–291, July 1990.

[22] B. Ramamurthy and A. Melton, "A Synthesis of Software Sciences Measures and the Cyclomatic Number," *IEEE Transactions on Software Engineering*, vol. 14, no. 8, pp. 1116–1121, August 1988.

[23] V. Rodriguez and W. T. Tsai, "Evaluation of Software Metrics Using Discriminant Analysis," *Proceedings of the Eleventh Annual International Computer Software and Applications Conference*, Tokyo, Japan, pp. 245–251, October 1987.

[24] H. D. Rombach, "A Controlled Experiment on the Impact of Software Structure on Maintainability," *IEEE Transactions on Software Engineering*, vol. 13, no. 3, pp. 344–354, March 1987.

[25] M. H. Samadzadeh and K. Nandakumar, "A Study of Software Metrics," *Journal of Systems Software*, vol. 16, no. 3, pp. 229–234, November 1991.

[26] N. F. Schneidewind, "Methodology For Validating Software Metrics," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 410–422, May 1992.

[27] V. Y. Shen, T. Yu, S. M. Thebaut, and L. R. Paulsen, "Identifying Error-Prone Software—An Empirical Study," *IEEE Transactions on Software Engineering*, vol. 11, no. 4, pp. 317-323, April 1985.

[28] Y. S. Sherif, E. Ng, and J. Steinbacher, "Computer Software Development: Quality Attributes, Measurements and Metrics," *Naval Research Logistics*, vol. 35, no. 1, pp. 425–436, January 1988.

[29] T. Stalhane, *A Discussion of Software Metrics as a Means for Software Reliability Evaluation*, Report PB89-210322, U.S. Department of Commerce, National Technical Information Service, 1988.

[30] J. D. Valett and F. E. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *The Journal of Systems and Software*, vol. 9, no. 2, pp. 137–148, February 1989.

[31] I. Vessey and R. Weber, "Some Factors Affecting Program Maintenance: An Empirical Study," *Communications ACM*, vol. 26, no. 2, pp. 128–134, February 1983.

[32] S. Wake and S. Henry, "A Model Based on Software Quality Factors Which Predicts Maintainability," *Proceedings of the Conference on Software Maintenance*, Phoenix, Arizona, pp. 382-387, October 24, 1988.

[33] S. S. Yau and J. S. Collofello, "Some Stability Measures for Software Maintenance," *IEEE Transactions on Software Engineering*, vol. 6, no. 6, pp. 545–552, November 1980.

[34] M. V. Zelkowitz, A. C. Shaw, and J. D. Grannon, *Principles of Software Engineering and Design*, Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1979.

Table 1. Percentage of time spent on various maintenance tasks.

Maintenance tasks	Percentage of time spent			
	1977	1985	1987	1990
Enhancements	59	44	41	43
Corrections	22	15	18	16
Supporting users	NA[a]	21	12	12
Reengineering	NA	NA	10	9
Adaptations	6	8	9	8
Documentation	6	NA	5	6
Tuning	4	NA	3	5
Evaluating requests	NA	8	NA	NA
Other	3	4	2	1

[a] Not applicable.

Table 2. Comptroller General statistics on delivered software.

Quality of software delivered	Percentage of software delivered
Could work on delivery	2
Could work after some rework	3
Never successfully put to use	45
Extensively reworked	20
Useless	30
Total	100

Table 3. Software measures used to analyze the VRC software.

Measure number	Measure	Measure definition
1	n_1	Number of unique operators
2	n_2	Number of unique operands
3	N_1	Number of total operators
4	N_2	Number of total operands
5	N	Length $(N_1 + N_2)$
6	$\hat{N}$	Estimated length $= [n_1(\log_2(n_1)) + n_2(\log_2(n_2))]$
7	V	Volume $= (N)\log_2(n) = (N_1 + N_2)\log_2(n_1 + n_2)$
8	E	Effort $= V/[(2/n_1)(n_2/N_2)]$
9	VG_1	McCabe cyclomatic complexity (number of decisions + 1)
10	VG_2	Extended complexity (decisions + ANDs + ORs +1)
11	LOC	Lines of code (includes blank and comment lines)
12	B/C	Number of blank lines + number of comment lines
13	$<;>$	Number of executable semicolons
14	SP	Average maximum lines between variable references
15	$NCLOC$	Non-commented lines of code $= LOC - B/C$
16	LB-ratio	$[N_1/(N_2 + B/C)]$

Table 4. OP-B, OP-C, and OP-D modules and the mean values of the 16 measures.

Measure number	Measure	OP-B (222 modules) mean	OP-C (224 modules) mean	OP-D (235 modules) mean
1	n_1	12	12	13
2	n_2	12	12	15
3	N_1	70	75	87
4	N_2	42	44	52
5	N	113	119	140
6	$\hat{N}$	103	110	126
7	V	704	721	844
8	E	53,781	58,198	61,715
9	VG_1	4	4	5
10	VG_2	5	4	5
11	LOC	73	78	83
12	B/C	43	46	49
13	$<;>$	12	13	15
14	SP	5	5	6
15	$NCLOC$	30	31	34
16	LB-ratio	1	1	1

Table 5. The factor matrix for the 16 measures of OP-C, OP-B, and OP-D.

Measure number	Measure	OP-B		OP-C		OP-D	
		Factor 1	Factor 2	Factor 1	Factor 2	Factor 1	Factor 2
1	n_1	0.78	−0.17	0.79	−0.12	0.78	−0.17
2	n_2	0.94	−0.02	0.94	−0.02	0.93	−0.03
3	N_1	0.97	0.10	0.98	0.83	0.97	0.08
4	N_2	0.97	0.06	0.97	0.04	0.96	−0.05
5	N	0.98	0.09	0.98	0.07	0.97	0.07
6	$\hat{N}$	0.91	−0.01	0.96	−0.00	0.96	−0.01
7	V	0.96	0.14	0.97	0.09	0.96	0.09
8	E	0.89	0.22	0.90	0.15	0.88	0.15
9	VG_1	0.94	0.09	0.95	0.08	0.93	0.10
10	VG_2	0.77	0.12	0.95	0.07	0.93	0.10
11	LOC	0.94	−0.25	0.96	−0.17	0.95	−0.19
12	B/C	0.61	−0.64	0.72	−0.50	0.70	−0.53
13	$<;>$	0.97	0.03	0.97	0.04	0.97	0.06
14	SP	0.70	−0.05	0.60	−0.01	0.72	0.04
15	$NCLOC$	0.98	0.05	0.98	0.05	0.98	0.05
16	LB-ratio	−0.03	0.83	−0.01	0.92	−0.02	0.90
Percentage of explained variability		90	10	91	9	91	9

Table 6. Five composite complexity models and their coefficients of determination.

Model number	Model	Coefficient of determination, percent
1	$< VG_1 > = 1.48 + 0.005(V)$	$R^2 = 96$
2	$< VG_1 > = 0.510 + 0.136(NCLOC)$	$R^2 = 96$
3	$< VG_1 > = 0.786 + 0.0013(V) + 0.0976(NCLOC)$	$R^2 = 96$
4	$< V > = -206 + 29.5(NCLOC)$	$R^2 = 99$
5	$< V > = -210 + 8.7(VG_1) + 28.3(NCLOC)$	$R^2 = 99$

Table 7. Summary of actual average values of (VG_1) and values predicted by models 1, 2, and 3.

Model	Release	(V) value		Delta,	Error percentage,
		Actual, (A)	Predicted, (P)	(A) − (P)	delta/(A)
1	OP-B	4.45	5.00	−0.55	−12.40
	OP-C	4.53	5.09	−0.56	−12.40
	OP-D	5.30	5.70	−0.40	−7.50
Grand average		4.76	5.26	−0.50	−10.60
2	OP-B	4.45	4.59	−0.14	−3.10
	OP-C	4.53	4.86	−0.33	−7.30
	OP-D	5.30	5.27	−0.03	0.60
Grand average		4.76	4.91	−0.15	−3.10
3	OP-B	4.45	4.62	−0.17	−3.80
	OP-C	4.53	4.84	−0.31	−6.80
	OP-D	5.30	5.30	−0.00	0.00
Grand average		4.76	4.92	−0.16	−3.40

Table 8. Summary of actual average values of (V) and values predicted by models 4 and 5.

Model	Release	(V) value		Delta,	Error percentage,
		Actual, (A)	Predicted, (P)	(A) − (P)	delta/(A)
4	OP-B	704	679	+25	+3.6
	OP-C	722	738	−16	−2.2
	OP-D	845	826	+19	+2.2
Grand average		757	748	+9	+1.2
5	OP-B	704	678	+26	+3.7
	OP-C	722	735	−13	−1.8
	OP-D	845	826	+19	+2.2
Grand average		757	746	−10	+1.3

Table 9. Characteristics of four independent segments of software.

Segment number	Number of modules	Actual average value		
		VG_1	V	$NCLOC$
1	16	16.4	3343	102
2	16	17.9	4016	139
3	50	8.16	1823	64
4	55	11.10	2212	71

Table 10. Sample calculation of actual average values of (VG_1) and values predicted by model 1 for segments 1–4.

Model	Segment	(V) value		Delta, (A) − (P)	Error percentage, delta/(A)
		Actual, (A)	Predicted, (P)		
1	1	16.40	18.19	−1.79	−10.9
	2	17.90	21.56	−3.66	−20.4
	3	8.16	10.59	−2.03	−24.4
	4	11.10	12.54	−1.44	−13.0
Grand average		13.39	15.72	−2.33	−17.3

Table 11. Summary of actual grand average values of (VG_1) and values predicted by models 1, 2, and 3 for segments 1–4.

Model	Segment	(VG_1) grand average value		Delta, (A) − (P)	Error percentage, delta/(A)
		Actual, (A)	Predicted, (P)		
1	1–4	13.39	15.57	−2.33	−17.3
2	1–4	13.39	13.31	+0.08	+0.6
3	1–4	13.39	13.48	−0.09	+0.7

Table 12. Summary of actual grand average values of (V) and values predicted by models 4 and 5 for segments 1–4.

Model	Segment	(VG_1) grand average value		Delta,	Error percentage,
		Actual, (A)	Predicted, (P)	(A) − (P)	delta/(A)
4	1–4	2848	2570	+278	+9.7
5	1–4	2848	2571	+277	+9.7

Table 13. Analysis of three software releases using the relative complexity metric.

Release	Total number of modules	Relative complexity					
		Total	Maximum	Minimum	Median	Mean	Standard deviation
OP-B	222	2799	45	0.4	10.9	12.6	10.0
OP-C	224	2837	45	0.4	10.9	12.7	9.6
OP-D	235	3470	49	0.4	12.2	14.8	11.3

Table 14. Cut-off values of the three software releases.

Release	Total number of modules	(RCM) cut-off value	Number of modules exceeding (RCM) cut-off value	Percentage of modules over (RCM) cut-off value
OP-B	222	22.6	33.0	15.0
OP-C	224	22.3	36.0	16.0
OP-D	235	26.1	35.0	15.0

Table 15. Analysis of transitions between the three software releases.

Transition	Number of modules modified	(RCM) cut-off value	Percentage of modified modules over cut-off value	Percentage of all modules over cut-off value that were actually modified
From OP-B to OP-C	13	22.6	46	40
From OP-C to OP-D	38	22.3	47	50

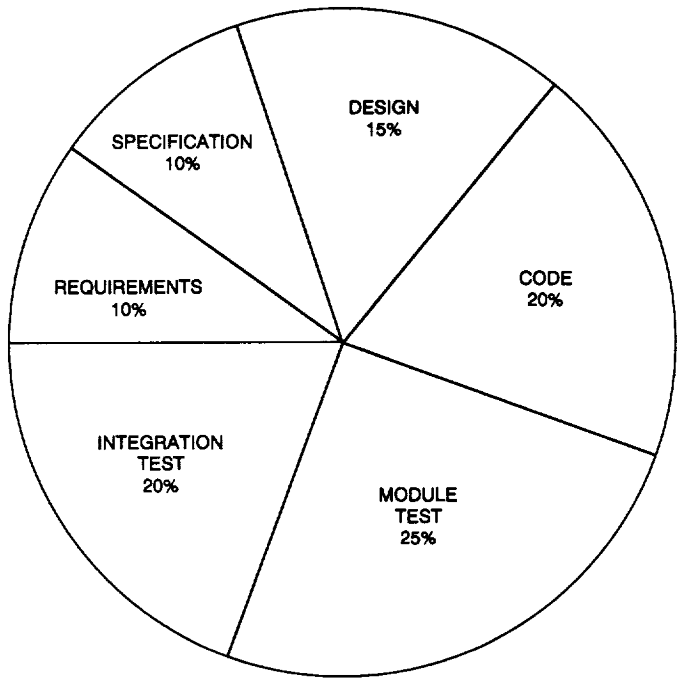

Fig. 1. The initial cost breakdown in developing a new project.

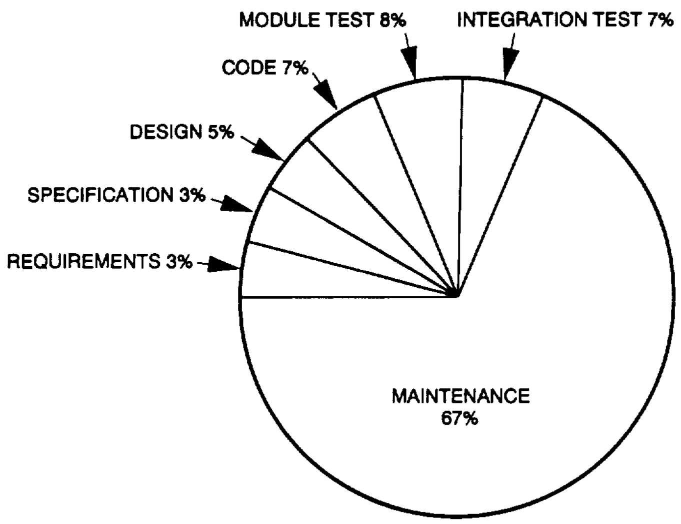

Fig. 2. The cost of software during its life cycle.

	B1	B2	N1	N2	N	NV	V	E	VG1	VG2	LOC	BC	CR	SP	NLOC	LBR
B1	1.00000 0.0	0.77205 0.0001	0.70872 0.0001	0.68055 0.0001	0.70054 0.0001	0.72123 0.0001	0.65058 0.0001	0.54830 0.0001	0.74993 0.0001	0.56794 0.0001	0.75723 0.0001	0.59043 0.0001	0.76131 0.0001	0.71629 0.0001	0.73042 0.0001	-0.03466 0.6075
B2	0.77205 0.0001	1.00000 0.0	0.91155 0.0001	0.93063 0.0001	0.92233 0.0001	0.96182 0.0001	0.91273 0.0001	0.78438 0.0001	0.83249 0.0001	0.73144 0.0001	0.87705 0.0001	0.58032 0.0001	0.93684 0.0001	0.63632 0.0001	0.91476 0.0001	-0.03778 0.5755
N1	0.70872 0.0001	0.91155 0.0001	1.00000 0.0	0.98580 0.0001	0.99801 0.0001	0.90913 0.0001	0.98176 0.0001	0.95128 0.0001	0.93309 0.0001	0.71636 0.0001	0.89288 0.0001	0.51917 0.0001	0.95732 0.0001	0.62622 0.0001	0.97883 0.0001	0.01953 0.7723
N2	0.68055 0.0001	0.93063 0.0001	0.98580 0.0001	1.00000 0.0	0.99437 0.0001	´0.93243 0.0001	0.97363 0.0001	0.90950 0.0001	0.89624 0.0001	0.71234 0.0001	0.91111 0.0001	0.56573 0.0001	0.95695 0.0001	0.61197 0.0001	0.97494 0.0001	-0.00079 0.9906
N	0.70054 0.0001	0.92233 0.0001	0.99801 0.0001	0.99437 0.0001	1.00000 0.0	0.92114 0.0001	0.98277 0.0001	0.93915 0.0001	0.92274 0.0001	0.72116 0.0001	0.90250 0.0001	0.53733 0.0001	0.96022 0.0001	0.62291 0.0001	0.98104 0.0001	0.01181 0.8611
NV	0.72123 0.0001	0.96182 0.0001	0.90913 0.0001	0.93243 0.0001	0.92114 0.0001	1.00000 0.0	0.90741 0.0001	0.79853 0.0001	0.76636 0.0001	0.64927 0.0001	0.85885 0.0001	0.57506 0.0001	0.88360 0.0001	0.47706 0.0001	0.89127 0.0001	-0.02969 0.6600
V	0.65058 0.0001	0.91273 0.0001	0.98176 0.0001	0.97363 0.0001	0.98277 0.0001	0.90741 0.0001	1.00000 0.0	0.95253 0.0001	0.91233 0.0001	0.79787 0.0001	0.86609 0.0001	0.47319 0.0001	0.93203 0.0001	0.58001 0.0001	0.96965 0.0001	0.01466 0.8281
E	0.54830 0.0001	0.78438 0.0001	0.95128 0.0001	0.90950 0.0001	0.93915 0.0001	0.79853 0.0001	0.95253 0.0001	1.00000 0.0	0.89454 0.0001	0.69025 0.0001	0.78596 0.0001	0.37656 0.0001	0.85869 0.0001	0.49117 0.0001	0.91505 0.0001	0.04023 0.5510
VG1	0.74993 0.0001	0.83249 0.0001	0.93309 0.0001	0.89624 0.0001	0.92274 0.0001	0.76636 0.0001	0.91233 0.0001	0.89454 0.0001	1.00000 0.0	0.78359 0.0001	0.86089 0.0001	0.48352 0.0001	0.92553 0.0001	0.76376 0.0001	0.95509 0.0001	0.01807 0.7889
VG2	0.56794 0.0001	0.73144 0.0001	0.71636 0.0001	0.71234 0.0001	0.72116 0.0001	0.64927 0.0001	0.79787 0.0001	0.69025 0.0001	0.78359 0.0001	1.00000 0.0	0.67531 0.0001	0.32942 0.0001	0.70748 0.0001	0.58539 0.0001	0.78232 0.0001	-0.02037 0.7628
LOC	0.75723 0.0001	0.87705 0.0001	0.89288 0.0001	0.91111 0.0001	0.90250 0.0001	0.85885 0.0001	0.86609 0.0001	0.78596 0.0001	0.86089 0.0001	0.67531 0.0001	1.00000 0.0	0.83221 0.0001	0.90148 0.0001	0.65696 0.0001	0.92972 0.0001	-0.15599 0.0201
BC	0.59043 0.0001	0.58032 0.0001	0.51917 0.0001	0.56573 0.0001	0.53733 0.0001	0.57506 0.0001	0.47319 0.0001	0.37656 0.0001	0.48352 0.0001	0.32942 0.0001	0.83221 0.0001	1.00000 0.0	0.56689 0.0001	0.43891 0.0001	0.56954 0.0001	-0.33967 0.0001
CR	0.76131 0.0001	0.93684 0.0001	0.95732 0.0001	0.95695 0.0001	0.96022 0.0001	0.88360 0.0001	0.93203 0.0001	0.85869 0.0001	0.92553 0.0001	0.70748 0.0001	0.90148 0.0001	0.56689 0.0001	1.00000 0.0	0.75125 0.0001	0.95988 0.0001	0.01323 0.8445
SP	0.71629 0.0001	0.63632 0.0001	0.62622 0.0001	0.61197 0.0001	0.62291 0.0001	0.47706 0.0001	0.58001 0.0001	0.49117 0.0001	0.76376 0.0001	0.58539 0.0001	0.65696 0.0001	0.43891 0.0001	0.75125 0.0001	1.00000 0.0	0.68239 0.0001	0.01060 0.8752
NLOC	0.73042 0.0001	0.91476 0.0001	0.97883 0.0001	0.97494 0.0001	0.98104 0.0001	0.89127 0.0001	0.96965 0.0001	0.91505 0.0001	0.95509 0.0001	0.78232 0.0001	0.92972 0.0001	0.56954 0.0001	0.95988 0.0001	0.68239 0.0001	1.00000 0.0	-0.00566 0.9332
LBR	-0.03466 0.6075	-0.03778 0.5755	0.01953 0.7723	-0.00079 0.9906	0.01181 0.8611	-0.02969 0.6600	0.01466 0.8281	0.04023 0.5510	0.01807 0.7889	-0.02037 0.7628	-0.15599 0.0201	-0.33967 0.0001	0.01323 0.8445	0.01060 0.8752	-0.00566 0.9332	1.00000 0.0

Fig. 3. Correlation matrix of 16 measures for OP-B.

	B1	B2	N1	N2	N	M	V	E	VG1	VG2	LOC	BC	CT	SP	NCLOC	LBR
B1	1.00000	0.79503	0.70854	0.68603	0.70381	0.85592	0.66948	0.54823	0.75377	0.75702	0.74387	0.62827	0.77626	0.68827	0.72105	-0.04119
	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.5397
B2	0.79503	1.00000	0.91747	0.92764	0.92585	0.98457	0.90915	0.78541	0.83939	0.83961	0.89238	0.68225	0.95180	0.56033	0.90922	-0.01885
	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.7790
N1	0.70854	0.91747	1.00000	0.98567	0.99822	0.92925	0.99735	0.95202	0.93489	0.93273	0.93188	0.64823	0.95255	0.49795	0.98920	0.02460
	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.7143
N2	0.68603	0.92764	0.98567	1.00000	0.99326	0.91839	0.98180	0.90939	0.89666	0.89658	0.93627	0.67809	0.95307	0.49197	0.97728	0.00933
	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.8896
N	0.70381	0.92585	0.99822	0.99326	1.00000	0.92967	0.99510	0.93890	0.92500	0.92352	0.93758	0.66283	0.95701	0.50224	0.98868	0.01943
	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.7724
M	0.85592	0.98457	0.92925	0.91839	0.92967	1.00000	0.91803	0.81228	0.88911	0.88833	0.89454	0.66532	0.96145	0.61427	0.92292	-0.00182
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.9783
V	0.66948	0.90915	0.99735	0.98180	0.99510	0.91803	1.00000	0.96468	0.92532	0.92202	0.92287	0.63301	0.94184	0.46109	0.98518	0.02808
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.6760
E	0.54823	0.78541	0.95202	0.90939	0.93890	0.81228	0.96468	1.00000	0.89747	0.89059	0.85331	0.54267	0.84510	0.33267	0.93731	0.04231
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.5287
VG1	0.75377	0.83939	0.93489	0.89666	0.92500	0.88911	0.92532	0.89747	1.00000	0.99669	0.89309	0.60974	0.92230	0.65413	0.95515	0.02546
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.7047
VG2	0.75702	0.83961	0.93273	0.89658	0.92352	0.88833	0.92202	0.89059	0.99669	1.00000	0.89196	0.61315	0.91863	0.66041	0.95136	0.02426
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.7180
LOC	0.74387	0.89238	0.93188	0.93627	0.93758	0.89454	0.92287	0.85331	0.89309	0.89196	1.00000	0.87132	0.91200	0.50984	0.95278	-0.12536
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0611
BC	0.62827	0.68225	0.64823	0.67809	0.66283	0.66532	0.63301	0.54267	0.60974	0.61315	0.87132	1.00000	0.65911	0.37603	0.68116	-0.31940
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001
CT	0.77626	0.95180	0.95255	0.95307	0.95701	0.96145	0.94184	0.84510	0.92230	0.91863	0.91200	0.65911	1.00000	0.64830	0.95282	0.01914
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.7757
SP	0.68827	0.56033	0.49795	0.49197	0.50224	0.61427	0.46109	0.33267	0.65413	0.66041	0.50984	0.37603	0.64830	1.00000	0.52798	0.02198
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.7435
NCLOC	0.72105	0.90922	0.98920	0.97728	0.98868	0.92292	0.98518	0.93731	0.95515	0.95136	0.95278	0.68116	0.95282	0.52798	1.00000	0.01062
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.8744
LBR	-0.04119	-0.01885	0.02460	0.00933	0.01943	-0.00182	0.02808	0.04231	0.02546	0.02426	-0.12536	-0.31940	0.01914	0.02198	0.01062	1.00000
	0.5397	0.7790	0.7143	0.8896	0.7724	0.9783	0.6760	0.5287	0.7047	0.7180	0.0611	0.0001	0.7757	0.7435	0.8744	0.0

Fig. 4. Correlation matrix of 16 measures for OP-C.

	B1	B2	N1	N2	N	M	V	E	VG1	VG2	LOC	BC	CT	SP	NCLOC	LBR
B1	1.00000	0.76798	0.69921	0.67628	0.69470	0.83404	0.65911	0.54051	0.70251	0.70917	0.75883	0.65341	0.73748	0.70268	0.72216	-0.06819
	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.2979
B2	0.76798	1.00000	0.91318	0.92727	0.92209	0.98371	0.90476	0.76771	0.81731	0.82739	0.88841	0.67125	0.93009	0.62854	0.90618	-0.02990
	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.6484
N1	0.69921	0.91318	1.00000	0.98554	0.99823	0.93052	0.99630	0.94063	0.89809	0.89640	0.91856	0.62890	0.94779	0.62333	0.97910	0.02353
	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.7197
N2	0.67628	0.92727	0.98554	1.00000	0.99310	0.92374	0.98136	0.89946	0.86324	0.86590	0.92109	0.65207	0.94822	0.61295	0.96792	0.00753
	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.9086
N	0.69470	0.92209	0.99823	0.99310	1.00000	0.93185	0.99426	0.92832	0.88867	0.88864	0.92391	0.64142	0.95138	0.62209	0.97908	0.01824
	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.7809
M	0.83404	0.98371	0.93052	0.92374	0.93185	1.00000	0.92089	0.80819	0.85805	0.86369	0.90027	0.66906	0.93851	0.67833	0.92550	-0.01502
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.8189
V	0.65911	0.90476	0.99630	0.98136	0.99426	0.92089	1.00000	0.95835	0.88574	0.88249	0.90801	0.61560	0.93267	0.58713	0.97180	0.02684
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.6823
E	0.54051	0.76771	0.94063	0.89946	0.92832	0.80819	0.95835	1.00000	0.83132	0.81862	0.82410	0.52488	0.82351	0.48411	0.90390	0.04260
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.5158
VG1	0.70251	0.81731	0.89809	0.86324	0.88867	0.85805	0.88574	0.83132	1.00000	0.99307	0.86436	0.55703	0.92329	0.78499	0.94385	0.02552
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.6971
VG2	0.70917	0.82739	0.89640	0.86590	0.88864	0.86369	0.88249	0.81862	0.99307	1.00000	0.86403	0.56309	0.92001	0.79418	0.93942	0.02379
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0001	0.7168
LOC	0.75883	0.88841	0.91856	0.92109	0.92391	0.90027	0.90801	0.82410	0.86436	0.86403	1.00000	0.86829	0.90087	0.64902	0.94701	-0.13903
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001	0.0332
BC	0.65341	0.67125	0.62890	0.65207	0.64142	0.66906	0.61560	0.52488	0.55703	0.56309	0.86829	1.00000	0.61517	0.43323	0.66295	-0.33398
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.0001	0.0001
CT	0.73748	0.93009	0.94779	0.94822	0.95138	0.93851	0.93267	0.82351	0.92329	0.92001	0.90087	0.61517	1.00000	0.75191	0.96130	0.01875
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.0001	0.7749
SP	0.70268	0.62854	0.62333	0.61295	0.62209	0.67833	0.58713	0.48411	0.78499	0.79418	0.64902	0.43323	0.75191	1.00000	0.69901	0.02089
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.0001	0.7500
NCLOC	0.72216	0.90618	0.97910	0.96792	0.97908	0.92550	0.97180	0.90390	0.94385	0.93942	0.94701	0.66295	0.96130	0.69901	1.00000	0.00643
	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0	0.9218
LBR	-0.06819	-0.02990	0.02353	0.00753	0.01824	-0.01502	0.02684	0.04260	0.02552	0.02379	-0.13903	-0.33398	0.01875	0.02089	0.00643	1.00000
	0.2979	0.6484	0.7197	0.9086	0.7809	0.8189	0.6823	0.5158	0.6971	0.7168	0.0332	0.0001	0.7749	0.7500	0.9218	0.0

Fig. 5. Correlation matrix of 16 measures for OP-D.

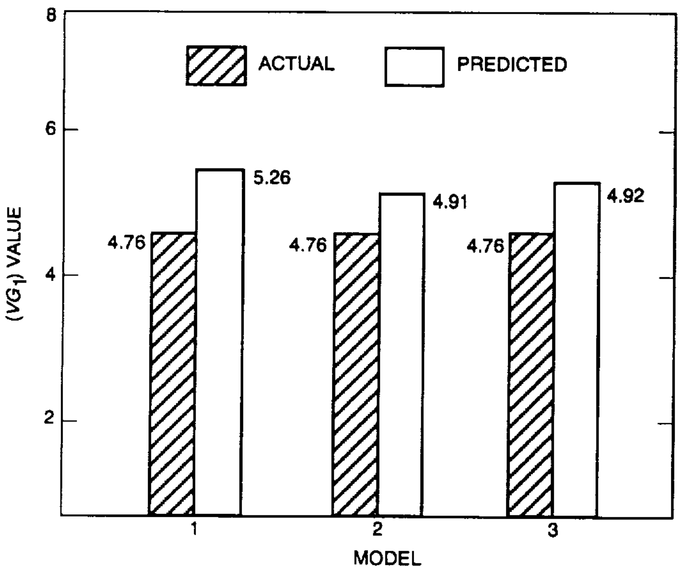

Fig. 6. Actual average values of (VG_1) and values predicted by models 1, 2, and 3.

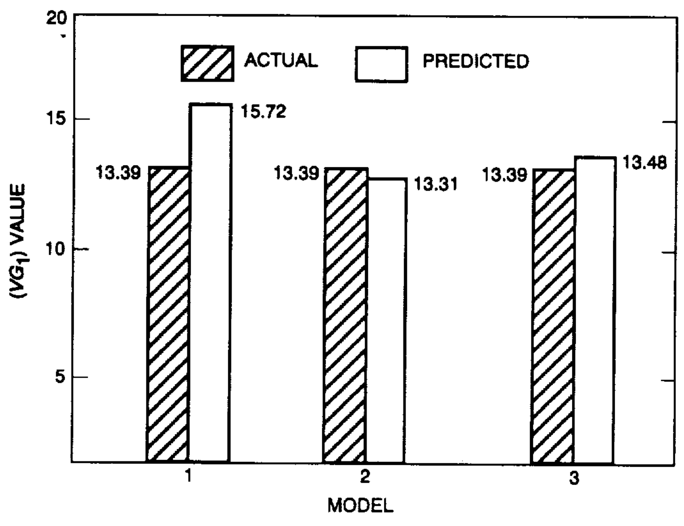

Fig. 8. Actual average values of (VG_1) and values predicted by models 1, 2, and 3 for independent segments of software.

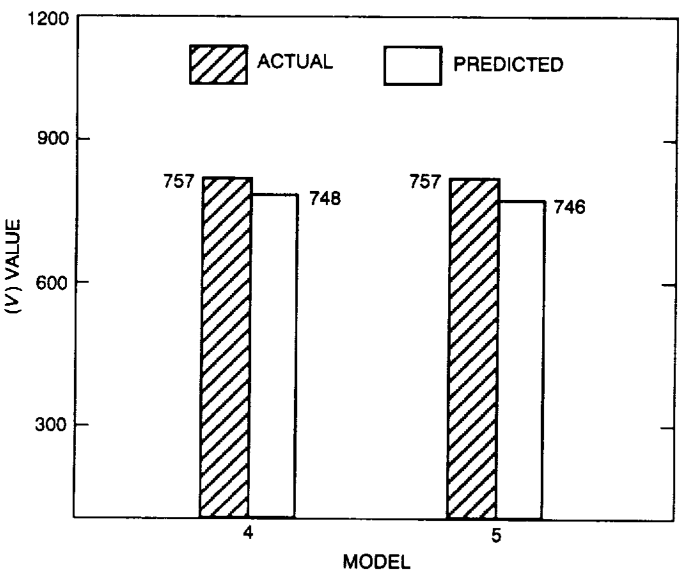

Fig. 7. Actual average values of (V) and values predicted by models 4 and 5.

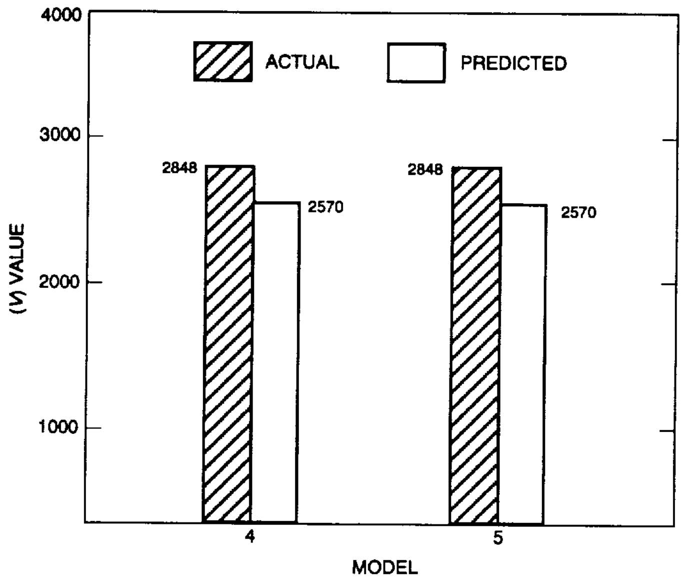

Fig. 9. Actual average values of (V) and values predicted by models 4 and 5 for independent segments of software.