8 ... 36014

CALCULATION AND USE OF AN ENVIRONMENT'S CHARACTERISTIC SOFTWARE METRIC SET

Victor R. Basili [1] and Richard W. Selby, Jr. [2]

[1] Department of Computer Science, University of Maryland, College Park, MD 20742, USA
[2] Department of Information and Computer Science, University of California, Irvine, CA 92717, USA; was with the Department of Computer Science, University of Maryland, College Park, MD 20742, USA

ABSTRACT

Since both cost/quality goals and production environments differ, this study presents an approach for customizing a characteristic set of software metrics to an environment. The approach is applied in the Software Engineering Laboratory (SEL), a NASA Goddard production environment, to 49 candidate process and product metrics of 652 modules from six (51,000 – 112,000 line) projects. For this particular environment, the method yielded the characteristic metric set {source lines, fault correction effort per executable statement, design effort, code effort, number of I/O parameters, number of versions}. The uses examined for a characteristic metric set include forecasting the effort for development, modification, and fault correction of modules based on historical data.

1. Introduction

Several metrics have been proposed to predict product cost/quality and to capture distinct project aspects [8, 12, 18, 19, 21]. The effectiveness of the metrics in capturing what is intended, however, has depended on the particular environment examined [1, 4, 9, 10, 13, 17, 27, 28, 29]. A particular software metric that has been useful to characterize, evaluate, or predict aspects of software development in one environment may have limited usefulness elsewhere. The differing cost/quality goals among environments and the diversity in methodology, software type, etc. contribute to the inconsistent performance of metrics. Thus, it seems inappropriate to attempt to select a set of software metrics that have universal effectiveness across all software environments. The selection of a set of metrics appropriate for a particular environment must consider its individual features; that is, a metric set must be customized to a specific environment.

Section 2 describes the idea of characteristic software metric sets. Section 3 presents an approach for customizing a characteristic set of cost and quality metrics to an environment. The application of the approach in a software production environment is discussed in Section 4. Section 5 investigates the use of a characteristic metric set as a management tool. Section 6 presents the conclusions from this work.

2. Characteristic Software Metric Sets

The successful management of software projects requires a diverse range of capabilities, including monitoring and controlling the evolving software system and forecasting the outcome of the development. Techniques that assist in these management functions may lead to more successful projects, and possibly higher product requirement conformance and operational reliability. The idea of a characteristic software metric set supports several aspects of software management.

A characteristic software metric set is a concise collection of metrics that capture distinct factors in a software development/maintenance environment. A characteristic metric set can be thought of as a vector of metrics that represents different areas of importance in an environment. Since both cost/quality goals and production environments differ, the particular factors that are captured by the metrics in the set will differ across environments. The calculation of a characteristic metric set should be based on the particular cost and quality goals in an environment, and reflect the inherent differences of an environment from others.

A characteristic metric set may be used to 1) characterize an environment, 2) compare an environment with others, 3) monitor current project status, or 4) forecast project outcome relative to past projects, when metrics in the set are available early in development. Once the distinct factors in an environment's set are determined, the set then characterizes what aspects are important in the environment. Comparing the characteristic set of factors in one environment with the sets of other environments provides a format to distinguish and contrast among them. Within an individual environment, the actual values of the metrics in the set characterize a particular project or project subsystem. The change in the metric values during a project can be used to monitor project status and its change over time. The characteristic set in conjunction with historical data can be used to forecast the outcome of the current project relative to past project outcome.

The goals for this study are threefold. I.) Develop an approach for customizing a set of metrics to particular cost/quality goals in a specific environment. II.) Apply the approach to calculate the characteristic set for the NASA/SEL environment. III.) Examine the usability of

the approach as a management tool for predicting outcome of system parts.

3. Approach for Set Calculation

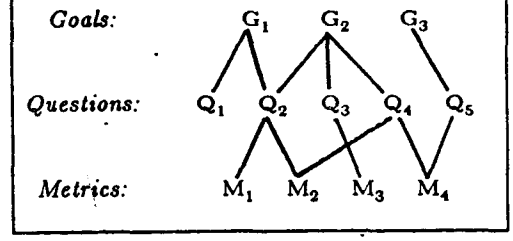
A proposed approach for calculating a characteristic metric set consists of three steps: 1) formulate the goals and questions that represent cost/quality factors in an environment; 2) list all metrics that capture information relating to the goals and questions; and 3) condense metrics into a set capturing distinct factors. This approach satisfies the two key aspects of customizing a characteristic metric set to an environment: sensitivity to the cost/quality goals of importance in the environment, and capturing the features that give the environment its identity.

The first step is to generate a cost/quality goal and question framework for the environment on which to base the generation of all potential metrics (see Figure 1). After the goals and questions have been specified for an environment, all possible metrics are listed that represent relevant information. These first two steps are an application of the goal-question-metric paradigm [6, 7]. Since a software environment is in some sense defined by the projects it develops, applying the metrics listed to those projects reflects an environment's identifying features. The third step is to condense the collection of candidate metrics into a characteristic set. Factor analysis may be applied to accomplish this step [22, 24]. This data reduction task actually groups the metrics listed according to how they relate to the distinct factors in an environment. Appropriate metrics that relate to each of the factors can then be selected based on some criteria, such as ease of calculation or phase availability. In very heterogeneous environments, cluster analysis [16, 24] may first be used to identify demographically similar projects or subsystems, followed by factor analysis within the groups. Section 4.3, "NASA/SEL Set Calculation," describes the application of these steps in a software production environment.

3.1. An Alternate Approach

An alternate approach to determining a small set of characteristic metrics was examined in [15]. In this approach, twenty candidate complexity metrics were calculated on 585 PL/I procedures. The name of each procedure was put into a large "complexity pot" once for each time the procedure appeared in the top decile of a candidate complexity metric. Since there were twenty candidate metrics, the name of a given procedure could then appear up to twenty times in the pot. The procedures identified by a single metric were then compared with those in the total pot. For each appearance of a procedure name in the total pot, a candidate metric was awarded one point if that name was in the metric's top decile. The candidate complexity metric that scored the highest would be selected for the characteristic set. All occurrences of procedure names were then removed from the pot that appeared in the top decile of the first metric selected. The scores for the metrics were then recalculated based on the remaining procedures and another metric would then be selected, continuing until no procedures remained in the pot.

Figure 1. Goal/question/metric paradigm.



This first approach for calculating a metric set is simple and straightforward. However, there are some drawbacks resulting from the simplicity, including the technique used to select metrics for the characteristic set and a fundamental assumption in the calculation. Including a large number of highly dependent program metrics in the collection examined (e.g., the software "quantity" group of executable statements, length, volume, vocabulary, ...) increased disproportionately the number of appearances of routines commonly selected by that group in the pot of "complex" programs. It is therefore no surprise that the metric that selected the greatest percentage of the appearances in the pot is one member of the "quantity" group (length). In each of the twenty program metrics examined, the top decile of programs was chosen as the most complex according to that metric. This decision relied on the implicit assumption that software complexity is a monotonically increasing function of each of the metrics, which is possibly troublesome.

Our paper presents an approach for calculating a characteristic set that advances the above approach by 1) selecting candidate metrics based on an environment's cost/quality goals, and 2) abstracting relationships (e.g., correlations) among (in)dependent metrics into a set of environmental factors. The use of values of characteristic metrics to identify modules with particular attributes, such as those of high "complexity" as was done in [15], is discussed in Section 5.

4. Application in the NASA/SEL Environment

This section describes the NASA/SEL environment, the data collection, and the resulting characteristic metric set.

4.1. NASA/SEL Environment

The Software Engineering Laboratory (SEL) [2, 3, 11, 25] is a joint venture between the University of Maryland, NASA/Goddard Space Flight Center, and Computer Sciences Corporation. The purpose of the SEL has been to provide an experimental database for examining relationships among the factors that affect the software development process and the delivered product. The software comprising the database is ground support software for satellites. The six systems analyzed in this study consisted of 51,000 to 112,000 lines of FORTRAN source code, and took between 6900 and 22,300 person-hours to develop over a period of 9 to 21 months. There

are from 200 to 600 modules (e.g., subroutines) in each system and the staff size ranges from 8 to 23 people per project, including the support personnel. Anywhere from 10 to 61 percent of the source code is reused or modified from previous projects.

4.2. Data Collection

The data discussed in this study are extracted from several sources. Among the data analyzed are the effort to design, code, and test the various modules of the systems as well as the changes and faults that occurred during their development. Effort data were obtained from a collection form that is filled out weekly by all programmers on the project. They report the time they spent on each module in the system partitioned into the 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 module is defined as any named object in the system; that is, a module is either a main procedure, block data, subroutine or function. The faults and changes are reported on another data collection form that is completed by a programmer each time a change is made to the system. A static code analysis program called SAP [14] automatically computed several of the static metrics examined in this analysis.

4.3. NASA/SEL Set Calculation

In the application of the approach in the SEL environment, there were two major reasons to use just six recent projects. First, changes and improvements in development technologies and personnel tend to be reflected in the projects developed (as they are intended to be). Therefore, the consideration of projects not recently completed would not be representative of the current environment. Second, several development environments do not have a long history of data collection. Discussing an approach that required a large project database would have little utility for them.

Three goal areas were defined for the SEL environment. The first goal area was to analyze the system development effort. An example question under this goal is "What are the attributes of modules that result in high development effort?". The second goal area was to analyze the system modifications. An example question here is "What are the attributes of modules that will be difficult to change?". Analyzing the system faults was the third goal area. An example question would be "What are the attributes of modules that will be fault-prone?". The generated list of metrics based on these three goal areas appears in Table 1; a total of 49 metrics was examined. The metrics are grouped according to the general areas of size/complexity [21], effort, faults/changes, and software science [19]. The set notation in the table signifies the ratio of one metric over another, e.g., amount of code effort was considered alone and divided by the amount of testing effort, overhead effort, and total effort. In addition to being examined alone, several effort and faults/changes metrics were divided by size/complexity metrics.

From the six projects, this analysis focuses on 652 newly developed modules with complete data for the metrics listed in Table 1. The use of principal factor analysis (with orthogonal varimax rotation) [22, 24] isolated a set of six distinct factors, {size, modification and fault correction effort density, development effort, code and test effort, η_2^*, #versions}, which are listed in descending order of overall importance and cumulatively explained 79% of the variance. The η_2^* metric is the number of I/O parameters in a module. Some appropriate metrics that related well to each of the factors in the set were a) size – source lines, executable statements, and N (the total number of operators and operands); b) modification and fault correction effort density – fault correction effort / executable statement; c) development effort – design effort, total effort / executable statement, and design effort / subroutine call; d) code and test effort – code effort, code effort / subroutine call, and test effort / subroutine call; e) η_2^* – η_2^*; and f) #versions – number of module versions. Thus, a feasible characteristic metric set for the SEL environment is {source lines, fault correction effort per executable statement, design effort, code effort, number of I/O parameters, number of versions}.

Table 1. List of measures examined in the SEL environment.

Size/Complexity Area
source lines (SRC)
executable statements (XQT)
comments
comments/SRC
Cyclomatic_complexity
calls
{Cyclomatic_complexity} over {XQT}
Effort Area
total_effort
design_effort
code_effort
testing_effort
{design_effort} over {code_effort}
{code_effort} over {testing_effort, overhead_effort, total_effort}
{design_effort, code_effort, testing_effort} over {calls}
{design_effort, code_effort} over $\{\eta_2^*\}$
{total_effort} over {SRC, SRC-comments, XQT, calls}
Faults/Changes Area
version
total_changes
weighted_changes
total_faults
weighted_faults
{total_faults, weighted_faults} over {SRC, XQT}
Software Science

η_1	η_2	η_2^*	N1	N2/η_2
N	N^	V	V*	L^
1/L^	E^	E^^	E*	B^
λ				

Table 2. Fraction of past SEL modules in the upper quartile of the dependent variables.

2a.) Module Development Effort

Characteristic Set Metric M_i	Quartile of Metric M_i			
	Upper	Second	Third	Lower
code effort	.71	.18	.04	.04
design effort	.56	.18	.13	.13
source lines	.51	.26	.14	.09
η_2^*	.48	.24	.17	.11
version	.44	.37	.13	.06
fault correction effort / XQT	.41	.28	.15	.16

2b.) Module Modification Effort

Characteristic Set Metric M_i	Quartile of Metric M_i			
	Upper	Second	Third	Lower
fault correction effort / XQT	.65	.18	.08	.09
version	.52	.33	.11	.04
code effort	.50	.27	.17	.06
source lines	.50	.28	.13	.09
η_2^*	.45	.24	.23	.08
design effort	.41	.25	.18	.17

2c.) Module Fault Correction Effort

Characteristic Set Metric M_i	Quartile of Metric M_i			
	Upper	Second	Third	Lower
fault correction effort / XQT	.81	.19	.00	.00
version	.50	.35	.12	.03
code effort	.48	.29	.15	.08
source lines	.42	.33	.14	.11
η_2^*	.42	.28	.19	.11
design effort	.36	.25	.20	.19

5. Use as a Management Tool

Although a characteristic set has the several uses outlined in Section 2, this study focuses on the use of metrics in the set to forecast the outcome of modules in projects. Several studies have pointed to the unsatisfactory use of metrics as direct predictors of software cost and quality [5, 20, 26]. This inadequacy motivates the use of software metrics from a new perspective – the examination of how well the metrics in the characteristic set can identify system parts (or whole systems) resulting in high or low cost/quality. System parts with interesting cost or quality attributes include those with high/low development effort, high/low modification effort, or high/low fault correction effort.

An approach for using metrics to identify system parts having interesting attributes is as follows. First, select some interesting cost or quality aspect of a system part, such as the total development effort for a module. Then, choose a set of modules that would be useful to identify, such as those modules that might eventually be in a project's upper quartile of total development effort. Next, from past projects determine how often metric value ranges (e.g., quartiles) contained modules that ended up in the upper quartile of development effort. Finally, characterize and identify modules in a current project that are likely, based on past metric data, to end up in the upper quartile of total development effort. The calculation of a characteristic metric set and the use of corresponding metric data from past projects is intended to help identify interesting modules in a system.

5.1. Metric Data from Past Projects

The data displayed in Table 2 were calculated from six SEL projects, and are interpreted as follows. The table is divided into three sections, corresponding to the three SEL goal areas discussed above. There is a table section for each dependent variable: total module development effort, total effort for module modification, and total effort for fault correction in a module. The characteristic set of six metrics that represent the different environmental factors is listed in each section of the table. Consider the section on total module development effort. The table displays the fraction of modules contained in the upper quartile of total development effort, based on their final quartile rankings for the

Table 3. Fraction of past SEL modules in the lower quartile of the dependent variables.

3a.) Module Development Effort

Characteristic Set Metric M_i	Quartile of Metric M_i			
	Upper	Second	Third	Lower
code effort	.00	.00	.23	.77
source lines	.10	.12	.24	.54
version	.06	.14	.30	.50
η_2^*	.09	.21	.25	.45
design effort	.02	.23	.37	.38
fault correction effort / XQT	.12	.25	.32	.31

3b.) Module Modification Effort

Characteristic Set Metric M_i	Quartile of Metric M_i			
	Upper	Second	Third	Lower
version	.09	.15	.28	.48
fault correction effort / XQT	.01	.13	.43	.43
η_2^*	.14	.19	.25	.42
source lines	.11	.18	.30	.41
code effort	.11	.18	.34	.37
design effort	.18	.28	.27	.27

3c.) Module Fault Correction Effort

Characteristic Set Metric M_i	Quartile of Metric M_i			
	Upper	Second	Third	Lower
fault correction effort / XQT	.00	.00	.50	.50
version	.18	.18	.27	.37
source lines	.21	.19	.29	.31
code effort	.18	.24	.27	.31
η_2^*	.20	.24	.25	.31
design effort	.18	.25	.29	.28

characteristic metrics. For example, 74% of the modules in the upper quartile of code effort were also in the upper quartile of total module development effort. Only 9% of the modules in the lower quartile of source lines were in the upper quartile of total development effort. The interpretation is the same for the other dependent variables of module modification effort and module fault correction effort. Table 3 is analogous to Table 2, except it displays the fraction of modules contained in the *lower* (instead of the upper) quartile of the respective dependent variable. For example, 50% of the modules in the lower quartile of number of versions were also in the lower quartile of total module development effort.

5.2. Data Interpretation

The information in these tables could be used to forecast the outcome of modules in a system. At the end of the design phase, the η_2^* metric and the amount of effort spent in design are known. The modules in the upper quartile of design effort should be identified by a project manager because 56% of these modules ended up in the upper quartile of total development effort. That is, in this environment the modules in the upper quartile of design effort were more than twice (=.56/.25) as likely than by chance to be the most expensive to develop overall; these modules were approximately 28 (=.56/.02) times more likely to be in the upper quartile of total development effort than to be in the lower quartile of total development effort. Modules in the upper quartile of the η_2^* metric were almost twice as likely than by chance to require the most effort to develop, modify, and correct. Other observations include 1) it is easiest to identify those modules that will have high development effort; 2) it is most difficult to identify those modules that will require little fault correction effort; and 3) the metrics of design effort and η_2^* are reasonably similar in forecasting ability, except that η_2^* seems superior in identifying modules that will require little modification effort.

The two tables help characterize the SEL development environment. The total development effort for a module tends to be indicated by the module's coding effort – modules in the extreme quartiles of coding effort are three times more likely than by chance to be in the corresponding extreme quartiles of total development effort. Since the programmers in the SEL are quite experienced in the application area and with appropriate design approaches, the dominance of coding effort seems reasonable. In other environments, the amount of design effort might better indicate the total development effort required. Other observations include 1) high density of fault correction effort (fault correction effort per executable statement) indicates high total modification effort and high total fault correction effort; and 2) an extreme (high or low) number of program versions reflects a corresponding amount of modification effort and correction effort.

Ideally, the metrics in the characteristic set would all be available early in development and have strong relationships with the dependent variables of interest. Some metrics, such as fault correction effort per executable statement, have limited usefulness as a predictor because of not being available until late in project development. An assumption is needed in order to use metric data from past projects to forecast the outcome of modules from a current project. The assumption is that the relationship between a module's current metric quartile and its eventual outcome (i.e., development, modification, and correction effort) is the same as the relationship between the final metric quartiles of past projects' modules and their outcome. This assumption is reasonable when using data from recent projects that are similar to the current project, and when predicting from metrics whose final quartiles are reasonably certain early in development (e.g., the number of I/O parameters in a module tends to remain relatively constant once specified in the design phase, and therefore, the metric's value does not tend to change quartiles). Note that the examples and metric data presented are from a particular environment, project data from other environments may differ.

Using a characteristic metric set with corresponding data from past projects enables the monitoring of a small set of customized metrics to forecast current project outcome. A characteristic set is usable as a management tool as soon as the metrics in the set are available.

6. Conclusions

A characteristic software metric set is intended to help support the effective management of software development and maintenance. The approach examined for building a characteristic metric set is adaptable to different cost/quality goals and to different environments. The calculation and use of the set could be coupled to an automated project monitor and database. The major results of this study are 1) an approach has been described for customizing a characteristic software metric set to an environment; 2) the application of the approach to the SEL production environment yielded the characteristic software metric set {source lines, fault correction effort per executable statement, design effort, code effort, number of I/O parameters, number of versions}; and 3) the use of a characteristic metric set with corresponding historical data can assist in project management by forecasting the outcome of system parts.

Further investigation in this area includes incorporating the characteristic metric set data (from Section 5) into a knowledge-based system. A statistical pattern classification scheme [23] is under consideration, although such an approach applies Bayes' Theorem and would assume independence among the metrics in the characteristic set. In this environment independence between, for example, design effort and number of I/O parameters is reasonable, while independence between source lines and code effort is questionable. A knowledge-based system that could use information from several metrics simultaneously would characterize system parts more effectively and forecast their outcomes more precisely. This work is intended to advance the understanding of the use of various metrics to characterize and predict aspects of software cost and quality.

7. Acknowledgement

Research supported in part by the Air Force Office of Scientific Research Contract AFOSR-F49620-80-C-001 and the National Aeronautics and Space Administration Grant NSG-5123 to the University of Maryland. Computer support provided in part by the Computer Science Center at the University of Maryland.

8. References

[1] J. W. Bailey and V. R. Basili, A Meta-Model for Software Development Resource Expenditures, *Proc. Fifth Int. Conf. Software Engr.*, San Diego, CA, pp. 107-116, 1981.

[2] V. R. Basili, M. V. Zelkowitz, F. E. McGarry, R. W. Reiter, Jr., W. F. Truszkowski, and D. L. Weiss, The Software Engineering Laboratory, Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD, Rep. SEL-77-001, May 1977.

[3] V. R. Basili and M. V. Zelkowitz, Analyzing Medium-Scale Software Developments, *Proc. Third Int. Conf. Software Engr.*, Atlanta, GA, pp. 116-123, May 1978.

[4] Victor R. Basili, *Tutorial on Models and Metrics for Software Management and Engineering*, IEEE Computer Society, New York, 1980.

[5] V. R. Basili, R. W. Selby, Jr., and T. Y. Phillips, Metric Analysis and Data Validation Across FORTRAN Projects, *IEEE Trans. Software Engr.* **SE-9**, 6, pp. 652-663, Nov. 1983.

[6] V. R. Basili and R. W. Selby, Jr., Data Collection and Analysis in Software Research and Management, *Proceedings of the American Statistical Association and Biometric Society Joint Statistical Meetings*, Philadelphia, PA, August 13-16, 1984.

[7] V. R. Basili and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data*, *Trans. Software Engr.* **SE-10**, 6, pp. 728-738, Nov. 1984.

[8] C. A. Behrens, Measuring the Productivity of Computer Systems Development Activities with Function Points, *IEEE Trans. Software Engr.* **SE-9**, 6, pp. 648-651, Nov. 1983.

[9] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[10] W. D. Brooks, Software Technology Payoff: Some Statistical Evidence, *J. Systems and Software* 2, pp. 3-9, 1981.

[11] D. N. Card, F. E. McGarry, J. Page, S. Eslinger, and V. R. Basili, The Software Engineering Laboratory, Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD Rep. SEL-81-104, Feb. 1982.

[12] E. T. Chen, Program Complexity and Programmer Productivity, *IEEE Trans. Software Engr.*, pp. 187-194, May 1978.

[13] B. Curtis, S. B. Sheppard, and P. M. Milliman, Third Time Charm: Stronger Replication of the Ability of Software Complexity Metrics to Predict Programmer Performance, *Proc. Fourth Int. Conf. Software Engr.*, pp. 356-360, Sept. 1979.

[14] W. J Decker and W. A. Taylor, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1), Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD, Rep. SEL-78-102, May 1982.

[15] J. L. Elshoff, Characteristic Program Complexity Metrics, *Proc. Seventh Int. Conf. Software Engr.*, Orlando, FL, pp. 288-293, 1984.

[16] B. S. Everitt, *Cluster Analysis, 2nd ed.*, Heineman Educational Books Ltd., London, 1980.

[17] A. R. Feuer and E. B. Fowlkes, Some Results from an Empirical Study of Computer Software, *Proc. Fourth Int. Conf. Software Engr.*, pp. 351-355, 1979.

[18] J. E. Gaffney and G. L. Heller, Macro Variable Software Models for Application to Improved Software Development Management, *Proc. Workshop on Quantitative Software Models for Reliability, Complexity and Cost*, IEEE Comput. Society, 1980.

[19] M. H. Halstead, *Elements of Software Science*, North Holland, New York, 1977.

[20] P. G. Hamer and G. D. Frewin, M. H. Halstead's Software Science -- A Critical Examination, *Proc. Sixth Int. Conf. Software Engr.*, Tokyo, Japan, pp. 197-206, Sept 13-16, 1982.

[21] T. J. McCabe, A Complexity Measure, *IEEE Trans. Software Engr.* **SE-2**, 4, pp. 308-320, Dec. 1976.

[22] S. A. Mulaik, *The Foundations of Factor Analysis*, McGraw-Hill, New York, 1972.

[23] J. A. Reggia, Knowledge-Based Decision Support Systems: Development through KMS, Ph.D. Dissertation, Dept. Com. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1121, Oct. 1981.

[24] Statistical Analysis System (SAS) User's Guide, SAS Institute Inc., Box 8000, Cary, NC, 27511, 1982.

[25] Annotated Bibliography of Software Engineering Laboratory (SEL) Literature, Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD Rep. SEL-82-006, Nov. 1982.

[26] V. Y. Shen, S. D. Conte, and H. E. Dunsmore, Software Science Revisited: A Critical Evaluation of the Theory and Its Empirical Support, *Trans. Software Engr.* **SE-9**, 2, pp. 155-165, March 1983.

[27] J. Vosburgh, B. Curtis, R. Wolverton, B. Albert, H. Malec, S. Hoben, and Y. Liu, Productivity Factors and Programming Environments, *Proc. Seventh Int. Conf. Software Engr.*, Orlando, FL, pp. 143-152, 1984.

[28] C. E. Walston and C. P. Felix, A Method of Programming Measurement and Estimation, *IBM Systems J.* **16**, 1, pp. 54-73, 1977.

[29] J. C. Zolnowski and D. B. Simmons, Taking the Measure of Program Complexity, *Proc. National Computer Conference*, pp. 329-336, 1981.