A Method for Assessing the Reusability of Object-Oriented Code Using a Validated Set of Automated Measurements

Fatma Dandashi

Mitre Corporation
7515 Coleshire Dr.
McLean, VA 22102-7508
1.703.883.7914
Dandashi@mitre.org

David C. Rine

George Mason University
4400 University Drive
Fairfax, Virginia 22030-4444
1.703.993.1530 ext. 3-1546
drine@cs.gmu.edu

ABSTRACT

A method for judging the reusability of C++ code components and for assessing indirect quality attributes from the direct attributes measured by an automated tool was demonstrated. The method consisted of two phases. The first phase identified and analytically validated a set of measurements for assessing direct quality attributes based on measurement theory. An automated tool was used to compute actual measures for a repository of C++ classes. A taxonomy relating reuse, indirect quality attributes, and measurements identified and validated during the first part of this research was defined. The second phase consisted of identifying and validating a set of measurements for assessing indirect quality attributes. A case study of the feasibility of applying direct measurements to assess the indirect quality attributes was conducted. The comparison and analysis of indirect quality attributes measured by human analysis with direct quality attributes measured by the automated tool provided empirical evidence that the two sets of quality attributes, direct and indirect, do correlate.

Keywords

Direct quality attribute measurements, indirect quality attribute measurements, empirical study.

1. INTRODUCTION

Considerable progress has been made in identifying and developing static, quantitative measurements for software developed using procedural programming languages [10, 12, 18, 19, 20, 29, 31, 32]. Quantitative measurements of software quality for functionally structured software have been extensively studied using these measurements [13, 14, 17, 22, 23, 39, 40, 42, 45]. Some of these measurements have been the subject of numerous critics [15, 16, 36, 37, 41, 49]. Quantitative measurements specific to Object-Oriented (OO) components have also been identified by various researchers [1, 2, 3, 6, 8, 24, 26, 33, 38]. Quantitative measurements for indirect quality attributes, such as adaptability, completeness, maintainability, and understandability have not been thoroughly investigated, or rigorously validated. To date, some studies of limited scope have been conducted to show that relationships exist between the two collections of attributes, direct and indirect [28].

During our research, we identified and analytically validated a set of static, syntactic, directly measurable quality attributes of C++ code components at the class (macro) and method (micro) levels [11]. Our goal was to identify and validate a set of measurements that can be used as effective assessors of indirect quality attributes and predictors of the overall quality and reusability of C++ code components. This objective was achieved through the derivation of a taxonomy and a method that was applied to estimate indirect quality attributes such as understandability and maintainability. This paper describes the measurements identified and analytically validated, the empirical study, the measurement taxonomy, the defined method, and the results obtained.

2. DEFINITIONS

Software product quality is the degree to which software possesses a desired combination of attributes. The term "quality software" is generally understood to embody software product attributes such as adaptability, completeness, maintainability, and understandability [43]. Some of these quality attributes are subjective in nature, difficult to measure, and some are in conflict with each other (e.g., completeness, and understandability). We propose that the degree to which a software product possesses certain quality attributes may be indirectly assessed using a set of directly measurable quality attributes. The following definitions are for terms that were used throughout this paper.

An attribute is a feature or property of an entity (e.g., size of a class or duration of class testing) [15]. Attributes are directly or indirectly measurable:

Direct attribute measurements of a code component are measurements for attributes that can be defined purely in terms of the elements that make up the syntax or behavior of the component [15].

Indirect attribute measurements of a code component are measurements for attributes that cannot be defined directly in terms of the elements that make up the syntax or behavior of the code component. Measurements of an indirect attribute involve the measurement of one or more direct attributes [15].

SAC 2002, Madrid, Spain

3. MICRO LEVEL MEASUREMENTS

Micro level measurements are measurements for attributes that are collected at the method level. The measurements of McCabe's Cyclomatic Number (MCC) [29], which is a function of the number of nodes and edges in a flowgraph of the code; Halstead's Volume [18], which is a function of the number of operators and operands; the number of Physical Source Statements (PSS) (excluding blanks and comments); and the Depth (of nesting statements, where a nesting level of one is assigned to the first statement in a method), were used in this study.

A measure for each method in a class was collected for each of the above measurements using the automated tool PC-Metric [46]. To arrive at a measure for the whole class, the highest collected measure was used as representative of the corresponding class measure. If one method in a class exceeds the desired quality threshold for a measurement, then the class's quality as a whole is affected by the quality measure of this method.

4. MACRO LEVEL MEASUREMENTS

At the macro level, Bieman [2] cites and discusses OO metrics that are similar to ones developed by a widely referenced work by Chidamber and Kemerer [8] which contains a definition of six measurements for C++ code. The Chidamber and Kemerer measurements of Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Response For a Class (RFC), Coupling Between Objects (CBO), and Depth of Inheritance Tree (DIT) Lack of Cohesion in Methods (LCOM), were used in this study. These OO-specific measurements were also collected using PC-Metric [46].

5. ANALYTICAL RESULTS

Several researchers have published properties to validate the correctness and meaningfulness of software complexity measurements [4, 27, 47]. Some of these properties have been criticized and proved to be incomplete, insufficient, and sometimes contradictory [5, 7, 15, 25, 30, 34, 48, 49, 50]. Critiques of the Chidamber and Kemerer measurements have also been published in the literature [9, 21]. The approach of validating the measurements analytically before applying them to software code components ensures that the results obtained are based on sound measurement theory. Zuse [48] provided definitions and properties for scales and an analytical validation for the four micro-level measurements used in our study. Based on the same Zuse properties, we analytically validated the Chidamber and Kemerer macro-level measurements [11]. The analysis validated that WMC, DIT, and NOC measurements are ratio scale measurements, while RFC, CBO, and LCOM are ordinal scale measurements.

Briand et al [4] published a set of properties for the validation of OO measurements. These findings are not inconsistent with our validation except in the case of RFC, CBO, and LCOM (Briand et al state that RFC and CBO are ratio scale measures since they satisfy their properties of size and coupling, and the LCOM measurement could not be classified according to their framework.). The RFC, CBO, and LCOM measurements were classified as ordinal scale measures in our study. The LCOM measurement was not utilized in the empirical study because it was not a good indicator of the quality of software. Therefore, our results are still valid under the Briand at al properties. The analysis conducted during this research effort was aimed solely at providing a theoretical basis for determining the scale of each measurement, thereby identifying the type of operations that are valid on these measurements (e.g. additivity for ratio scale measurements). Briand, et al offered an additional dimension by classifying the measurements according to length, size, complexity, etc. This aspect of their work will be discussed further in future work.

After analytically validating the direct attribute measurements, an empirical study was conducted to gauge indirect quality attributes defined by the National Institute of Standards and Technology (NIST) [35]. The report prepared by NIST researchers identified 10 indirect quality attributes for a software product based on extensive research into the available literature. The attributes are: 1) adaptability, flexibility, expandability, 2) completeness, 3) correctness, 4) efficiency, 5) generality, 6) maintainability, 7) modularity, 8) portability, 9) reliability, and 10) understandability. The focus of the NIST report was on providing measurement information for determining the reusability of software. Our survey was designed to provide us with an assessment of these software quality attributes. The study results are listed in the next section.

6. EMPIRICAL CASE STUDY DEFINITIONS AND VALIDATION

Our objective was to present evidence that will allow us to reject the null hypothesis:

H°: For C++ code components taken from the Object-Oriented Particle-In-Cell Simulations (PICS) problem domain, there is no relationship between direct software quality attribute measures collected using automated tools, and a group of indirect software quality attribute measures collected via a survey instrument.

And to accept the Alternative Hypothesis:

H_1: C++ code components taken from the PICS problem domain, that meet direct software quality attribute measurement thresholds, also possess certain indirect software quality attributes collected via a survey instrument.

6.1 Survey Instrument Construction

To empirically validate the research hypothesis, a survey instrument was designed and a survey was conducted to gather data about the PICS indirect quality attributes. Using a repository of PICS C++ code components, the survey was used to assess the degree to which the code components possess the indirect software quality attributes identified by NIST [35].

To achieve our objective, we identified a taxonomy that relates direct measurements to the set of indirect quality attributes (see Figure 1). The nature of the abstract level connection at Point A is explored further, and the validity of the taxonomy is demonstrated via the case study discussed in subsequent sections of this paper.

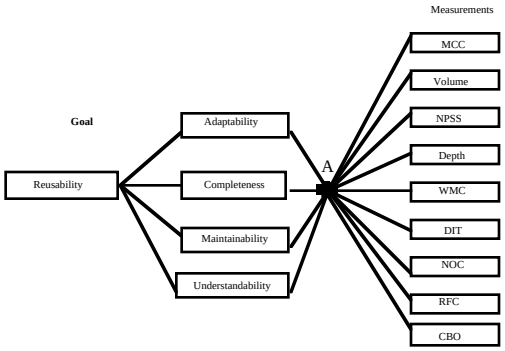

Figure 1: Taxonomy of indirect and direct quality attributes that can be used to assess the reusability of software components

6.2 The Population

In this study, we were trying to obtain objective measures for indirect quality attributes by using experts in the field, whose collective responses comprise these measures. We used repeated applications of the survey instrument to collect responses. The responses were used as measures of the quality of the code components.

A first application of the survey instrument, where the respondents were the authors of the PICS code, was conducted to avoid analysis problems and pitfalls that might lead us to conduct an invalid study and draw invalid conclusions. Some lessons learned from carrying out the first application of the survey instrument were:

- Eliminating the quality attribute *Reliability* from the survey. This was done because reliability is an attribute that can only be estimated by actually running the software several times with a variety of test data and then inspecting the defects uncovered or the number of times that the code terminates normally with the expected output.
- The number of statements per each indirect attribute was reduced because the survey was deemed too long.

The subjects chosen to participate in the second application consisted of two groups. The first was made up of C++ moderated newsgroup readers. A call for survey participation was posted in the newsreader for the C++ newsgroup. Interested individuals responded by indicating their interest in completing the survey. A copy of the survey was then e-mailed to them individually, and most completed and e-mailed back the survey. The second group was made up of students in a graduate level Software Engineering class of mostly software professionals that was being offered at a local university. More control variables were introduced to group these respondents, such as the level of C++ experience, GUI and/or scientific computation programming experience levels. During this second phase, we concentrated on a validation of our survey instrument by relating the group of statements that were designed to gauge each indirect quality attribute. We accepted as valid for measurement all the closely related responses that were gathered during this survey application. This resulted in a third phase study of only those indirect software quality attributes that were validated by the second phase. These were adaptability, completeness, maintainability, and understandability. We also concluded from the second survey application of the survey instrument that our C++ GUI code components were not useful study variables since this code was compiler and platform dependent. We narrowed our case study to the PICS problem domain. The third phase of the case study is detailed in the next section. The complete analysis of the responses appears in another work by these same authors [11].

7. CASE STUDY PROCEDURE

The procedure that was followed to conduct the empirical study consisted of:

1. Collect the indirect attribute measurements (via the survey instrument) for a PICS class from a respondent with experience in C++ and the problem domain, and use the responses with the most frequency for each set of statements as a measure of the indirect attribute.
2. Repeat this step using several experienced C++ programmers with similar experience in programming scientific computation applications. Each programmer is asked to evaluate a different class, thus obtaining indirect attribute measures for several classes.
3. Collect correlation coefficients for the values obtained via the survey instrument and the values obtained for that class via the automated tools.
4. Use the results obtained to predict a measure for reusability.

Our objective was to present evidence that will allow us to reject the null hypothesis, and to accept our Alternative Hypothesis. Twenty-five (25) surveys were handed out, sixteen (16) were completed and returned. Each indirect attribute was gauged via the survey by a collection of 4-6 statements. The most frequent response given by a respondent from the collection of responses applicable to a particular attribute was chosen as the overall class-attribute value. The data collected for this last phase of the study appears in another work by these same authors [11].

8. EMPIRICAL CASE STUDY RESULTS

Figure 2 below shows the summarized data (one value per indirect attribute) for every PICS class that was included in the last phase of the case study. Higher values indicate lower abilities (e.g., the responder who completed a survey for the class DADIXY believes that the class exhibits very low maintainability potential, shown below as a 5 rating). In general, the line graphs are consistent with each other, in that every attribute exhibits the same trend as the other attributes for the same class. That is, when one attribute is high for one class, the other attributes for the same class are also high. This data supports our choice to use these four indirect attributes.

Figure 3 consists of line graphs that plot the collected direct attribute measures for each class used in this case study. Higher values are indicative of lower quality. Figure 3 shows values for the four non OO-specific direct measurements: McCabe's Cyclomatic Complexity, Halstead's Volume, in addition to a count of the number of physical source statements and the depth of nested statements. The four measurements have traditionally been classified as measuring different dimensions of code component quality [4, 17].

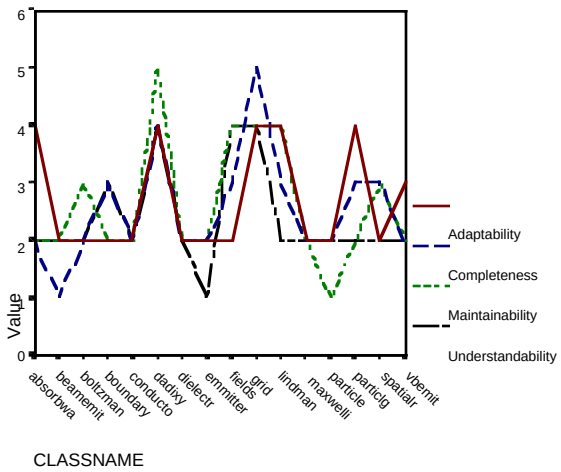

Figure 2: Plot of class indirect quality attribute measures collected via the survey instrument

However, we observe that for the PICS classes chosen for this case study, the four measures go up or down in tandem. We cannot conclude that they measure the same attributes. However, we can corroborate the intuitive premise that if the size of a software code component (as measured by PSS, for example) is large, then the other three collected measures follow suit. Regardless of the measurement classification (size, complexity, etc.), a large number of physical source statements usually also results in a larger number of nodes and edges (representing if, do, and while statements), and a larger Volume. It does not necessarily follow that the nesting depth of a large component would also be large (that is, we may have a code component that consists of a large number of simple statements). However, based on the measures collected here for MCC, V, PSS, and Depth, we observe that a high source line count is usually accompanied by high measures for MCC and V, nested depth of statements.

We also observe that the OO-specific measurements do not exhibit the same kind of measure agreement (see Figure 4). This may be a reflection of the kinds of attributes that are being measured via these two groups of measurements. We conclude that the OO-specific measurements measure unrelated attributes, whereas volume, PSS, MCC and Depth, are closely related size measures of the static, syntactical aspects of the software artifact.

The indirect attribute values collected via the survey instrument for the PICS classes also exhibit similar trends (Figure 2). This is also significant in that we are now able to draw conclusions about the appropriateness of these indirect attributes chosen for the last phase of our study, and their usefulness in predicting the overall reuse potential of a software artifact.

9. INTERPRETATION OF THE RESULTS

Using the statistical package SPSS [44], we computed correlation coefficients between every direct quality attribute measures collected for every class included in our case study, and the measures collected for the indirect quality attributes obtained via the survey instrument.

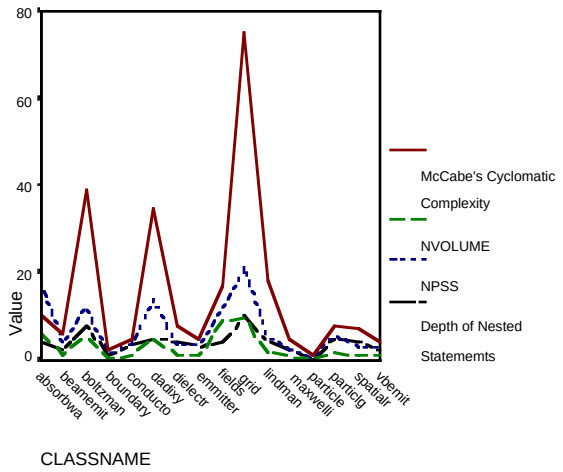

Figure 3: Plot of the direct quality attribute measures collected via PC-Metric

The correlation coefficients indicate that indirect quality attribute measurements (adaptability, completeness, maintainability, and understandability) are proportional to some of the direct quality attribute measures (namely MCC, Volume, PSS, Depth of Nested Statements, and WMC). We observe that as the measures for the direct attributes increase, so do the adaptability measures (lower values are "good" values).

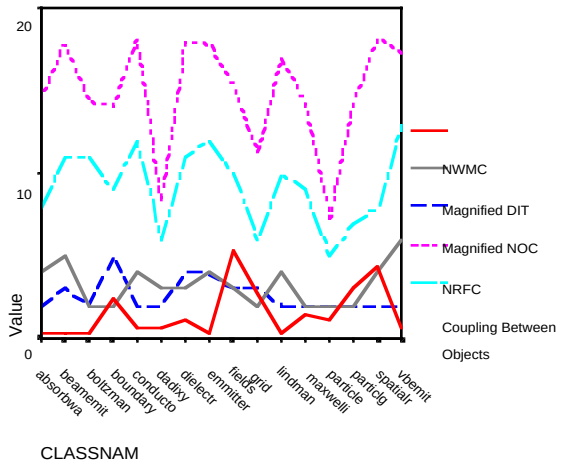

Figure 4: Plot of the direct quality attribute measures collected via PC-Metric

There is no strong relationship between the DIT and NOC measures and indirect quality attributes. We can observe that as DIT and NOC increase, (higher levels of inheritance), the adaptability, completeness, maintainability, and understandability (as perceived by a respondent), of a PICS class decrease. This is in general agreement with other researchers' evidence [8] supporting the statement that increasingly high levels of inheritance, while theoretically increasing initial development-reuse levels, make the classes:

1. increasingly difficult to adapt (and subsequently difficult to reuse), as these high inheritance levels imply an increasing degree of specialization in terms of the objects they represent.

2. more dependent on other inherited classes (more difficult to reuse than stand-alone or self-contained components).

3. less maintainable as stand-alone software artifacts in that their proper maintenance (perfective, adaptive or corrective) may require the modification of increasing numbers of inherited classes.

4. less understandable as stand-alone software artifacts, in that their understandability may require the analysis of increasing numbers of inherited classes.

In summary, the data collected for the indirect quality attribute adaptability, completeness, maintainability, and understandability supports our research hypothesis that direct attribute measures may be used to deduce the adaptability, completeness, maintainability, and understandability of a PICS class.

The correlation coefficients for adaptability, completeness, maintainability, and understandability versus RFC and CBO respectively are interpreted next. There does not seem to be a very strong relation between RFC and CBO measures and indirect quality attribute measures, as perceived by our respondents. We observe a weak trend (i.e. one that is not consistent for every class chosen for this study) that shows that indirect quality attribute measures increase when the RFC and CBO measures decrease. A class exhibits less indirect quality attributes as the coupling between the class in question and other classes decreases. Upon further examination of the kinds of classes that were chosen for the study, we observed that relations between RFC, CBO, and the four indirect quality attributes (adaptability, completeness, maintainability, and understandability) reflect the following class characteristics. As a class grows and becomes more self-contained, relying on its own methods to execute all functionality, coupling between such a class and other classes decreases, thus resulting in lower RFC and CBO measures. At the same time, the respondents found such a class to be, in general, less adaptable, less complete, less maintainable, and less understandable. This is in agreement with an intuitive analysis of OO software components: classes that contain large numbers of methods, while less reliant on methods in other classes, are generally perceived to be of lesser quality than smaller, simpler classes. In general, OO classes with a large number of methods and attributes are more difficult to adapt, maintain, and understand, and are perceived to be less complete (i.e. they implement functionality that does not belong in the class).

In the next section, a relation and a method for gauging the reusability of C++ code components will be discussed.

10. QUALITY ATTRIBUTE ESTIMATION METHOD

Based on the results observed during the case study, two rules are formulated:

1. The measures collected for Complexity, Volume, PSS, Depth, and WMC are proportional to the levels of adaptability, completeness, maintainability, and understandability of the class.

2. The measures collected for NOC, DIT, RFC, and CBO, are inversely proportional to the levels of adaptability, completeness, maintainability, and understandability of the class.

These rules were used in the following quality attribute estimation method. Given a C++ class we can,

1. Collect the direct attribute measures of Complexity, Volume, PSS, and Depth via an automated tool. For each class we use, we will get separate Complexity, Volume, PSS, and Depth measures for every method in that class.

2. Collect the direct attribute measures of WMC, NOC, DIT, RFC, and CBO via an automated tool.

3. Use the highest measure collected for each of the methods as the class measure. For example, if a class consists of 3 methods, and the Complexity measures collected are equal to 2, 4, and 3 respectively for each of the methods, then the overall class Complexity measure is 4.

4. Based on the direct quality attribute measures obtained, assign a measure representing the indirect quality attributes, to the class in question. For this measure, use one of the estimator values: Very High, High, Average, Low, Very Low. An estimator value for the indirect quality attributes: adaptability, completeness, maintainability, and understandability should conform to the following relations. Each is directly proportional to the direct quality attribute measurements MCC, V, PSS, Depth, and WMC; and inversely proportional to the measurements DIT, NOC, CBO, and RFC. Since our case study has shown that these four indirect attributes are related to each other, an assigned measure, based on the direct attribute measures collected, represents the class's potential for adaptability, completeness, maintainability, and understandability. This assigned value may also be used as an estimator of the potential for the reusability of the class.

11. SUMMARY AND CONCLUSION

During a previous study [11], we proposed that software engineers and practitioners may use indirect quality attribute measurements as assessors of reusability. The validity of the measurements was demonstrated via an empirical study conducted on C++ code components from the PICS problem domain. This paper reported on that work which included:

1. An identification of a set of measurements for OO code components that measure static, syntactic attributes of the code.

2. An analytical validation of the scale types, and their permissible transformations for measurements for OO code components. The objective was to identify and classify, according to a group of properties, the scale type for each of these measurements. The validation of the scale types provided a proven theoretical foundation for the analysis of the collected measures. In addition, some of the weaknesses in the published literature were exposed.

3. The collection of simple non-parametric correlation coefficients to explore relations between the various quality attribute measures.

4. Identification of a set of relations that map directly measurable software quality attributes to another set of quality attributes that can only be measured indirectly.

5. An empirical study, and a validated method for the estimation of the reusability of C++ code components.

Based on this study, we concluded that reuse of a C++ code component, may be estimated from four indirect quality attributes: adaptability, completeness, maintainability, and understandability. A quantitative value for reuse potential for such a code component, may be gauged by automatically collecting measures for the direct attributes of: McCabe's Complexity, Halstead's Volume, The number of Physical Source Statements (PSS), The Depth of nested statements, Weighted Methods per Class (WMC), Depth of Inheritance tree, Number of Children (immediate descendants only), Response for a class (RFC), Coupling Between Objects (CBO).

Future research includes plans to re-apply the survey instrument to new OO code components to further verify the applicability of this reuse estimation method to new problem domains and to other OO languages. The goal is to verify that the method validated during this research may be applied for the reliable certification of OO software code components, and to facilitate their reuse in a product-line manufacturing process.

12. REFERENCES

[1] Bieman, J. 1991, "Deriving Measures of Software Reuse in Object-Oriented Systems," TR CS-91-112, Colorado State University.

[2] Bieman, J. 1995a, "Metric Development for Object-Oriented Software," *Software Measurement*, Austin Melton, ed., International Thomson Computer Press, London, UK, 75-92.

[3] Bieman, J. and J. Xia Zhao 1995b, "Reuse Through Inheritance: A Quantitative Study of C++ Software," *Proc. ACM Symposium on Software Reusability (SSR '95)*, Seattle, WA, 47-52.

[4] Briand, L. C., S. Morasca, and V. R. Basili 1996, "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, 22(1), 68-85.

[5] Briand, L. C., S. Morasca, and V. R. Basili 1997, "Response to: Comments on Property-Based Software Engineering Measurement: Refining the Additivity Properties," *IEEE Transactions on Software Engineering*, 23(8), 196-198.

[6] Briand, L. C., J.W. Daley & J. K. Wust 1999, "A unified Framework for coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, 25(1), 91-121.

[7] Cherniavsky, J. C., and C. H. Smith 1991, On Weyuker's Axioms for Software Complexity measures, *IEEE Transactions on Software Engineering*, 17(6), 636-638.

[8] Chidamber, S. and C. Kemerer 1994, A Metrics Suite for Object-oriented Design, *IEEE Transactions on Software Engineering*, 20(6), 476-493.

[9] Churcher, N. I., and M. J. Shepperd 1995, Comments on A Metrics Suite for Object-oriented design, Correspondence in *IEEE Transactions on Software Engineering*, 21(3), pp. 263-265.

[10] Conte, S.D., H.E. Dunsmore, & V. Y. Shen 1986, *Software Engineering Metrics and Models*, Benjamin/Cummings, New York.

[11] Dandashi, F. 1998, *A Method for Assessing the Reusability of Object-oriented Code Using a Validated Set of Automated Measurements*, Ph.D. Dissertation, SITE, George Mason University, Fairfax, VA.

[12] Davis, J.S. and R.J. LeBlanc 1988, A Study of the Applicability of Complexity Measures, *IEEE Transactions on Software Engineering*, 14(9), 1366-1372.

[13] Dhama, H. 1995, Quantitative Models of Cohesion and Coupling in Software, *J. Systems Software*, Vol. 29, Elsevier Science Inc., NY, NY, 65-74.

[14] Evangelist, W.M. 1983, Software Complexity Metric Sensitivity to Program Structuring Rules, *Journal of Systems and Software*, 3, 231-243.

[15] Fenton, N. 1994a, "Software Measurement: A Necessary Scientific Basis," *IEEE Transactions on Software Engineering*, 20(3), 199-206.

[16] Fenton, N., S.L. Pfleeger, & R.L. Glass 1994b, Science and Substance: A Challenge to Software Engineers, *IEEE Software*, 11(4), 86-95.

[17] Fonash, P. 1993, *Metrics for Reusable Code Components*, Ph.D. Dissertation, SITE, George Mason University, Fairfax, Virginia.

[18] Halstead, M. 1977, *Elements of Software Science*, Elsevier North Holland, New York, NY.

[19] Hansen, W.J. 1978, Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count), *ACM SIGPLAN Notices*, 13(3), 29-33.

[20] Henry, S. and D. Kafura 1981, Software Structure Metrics Based on Information Flow, *IEEE Transactions on Software Engineering*, 7(5), 510-518.

[21] Hitz, M., and B. Montazeri 1996, Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective, Correspondence in *IEEE Transactions on Software Engineering*, 22(4), 267-271.

[22] Jensen, H.A. and K. Vairavan 1985, An Experimental Study of Software Metrics for Real-Time Software, *IEEE Transactions on Software Engineering*, 11(2), 231-234.

[23] Kafura, D. and G.R. Reddy 1987, The Use of Software Complexity Metrics in Software Maintenance, *IEEE Transactions on Software Engineering*, 13(3), 335-343.

[24] Karunanithi, S. and J. M. Bieman 1992, Candidate Reuse Metrics for Object-oriented and Ada Software, *TR CS-92-142*, Colorado State University.

[25] Kitchenham, B. A., S.L. Pfleeger, and N. Fenton 1997, Reply to: Comments on *Toward a Framework for Software Measurement Validation*, IEEE *Transactions on Software Engineering*, 23(8), 189-189.

[26] Lake, A., & C. Cook 1992, A Software Complexity Metric for C++, *TR 92-60-03*, Computer Science Dept., Oregon State University, Corvallis, OR.

[27] Lakshmanan, K. B., S. Jayaprakash, & P. K. Sinha 1991, Properties of Control-Flow Complexity Measures, *IEEE Transactions on Software Engineering*, 17(12), 1289-1295.

[28] Li, Wei & S. Henry 1993, Object-Oriented Metrics that Predict Maintainability, *J. Systems Software*, Elsevier Science Publishing Co, 23, 111-122.

[29] McCabe, T. J. 1976, A Complexity Measure, *IEEE Transactions on Software Engineering*, 2(4), 308-320.

[30] Morasca, S., L.C. Briand, V. R. Vasili, E.J. Weyuker, & M. V. Zelkowitz 1997, Comments on Toward a Framework for Software Measurement Validation, IEEE *Transactions on Software Engineering*, 23(8),187-188.

[31] Myers, G.J. 1977, An Extension to the Cyclomatic Measure of Program Complexity, *ACM SIGPLAN Notices*, 12(10), 61-64.

[32] Offutt, A. J., M. J. Harrold, & P. Kolte 1993, A Software Metric System for Module Coupling, *J. Systems Software*, Vol. 20, Elsevier Science Publishing Co. Inc., New York, NY, 295-308.

[33] Ott, L. M., J. M. Bieman, B.-K. Kang, & B. Mehra 1995, Developing Measures of Class Cohesion for Object-Oriented Software, in *Proc. Annual Oregon Workshop on Software Metrics* (AOWSM '95), 1995.

[34] Poels, G. and G. Dedene 1997, Comments on Property-Based Software Engineering Measurement: Refining the Additivity Properties, *IEEE Transactions on Software Engineering*, 23(3), 190-195.

[35] Salamon W.J. and D.R. Wallace 1994, Quality Characteristics and Metrics for Reusable Software (preliminary Report), US DoC for US DoD Ballistic Missile Defense Organization, NISTIR 5459.

[36] Schneidewind, N.F. 1992, Methodology for Validating Software metrics, *IEEE Transactions on Software Engineering*, 18(5), 410-422.

[37] Schneidewind, N.F. 1993, Report on the IEEE Standard for a Software Quality Metrics Methodology, *ACM Software Engineering Notes*, 18(3), A95-A98.

[38] Sheetz, S. D., D. P. Tegarden, & D. E. Monarchi 1991, Measuring Object-Oriented System Complexity, *Proc. 1st Workshop on information Technologies and Systems*.

[39] Shen, V.Y., T-J Yu, S.M. Thebaut, & L.R. Paulsen 1985, Identifying Error-Prone Software-An Empirical Study, *IEEE Transactions on Software Engineering*, 11(4), 317-323.

[40] Shepperd, M. 1988, A Critique of Cyclomatic Complexity as a Software Metric, *Software Engineering Journal*, 3(2), 30-36.

[41] Shepperd, M. & D.C. Ince 1994, A Critique of Three Metrics, *J. Systems Software*, Volume 26, Elsevier Science Inc., NY, NY, 197-210.

[42] Shooman, M. L. 1983, *Software Engineering: Design Reliability and Management*, McGraw Hill Inc., NY, NY.

[43] Sommerville, I. 1996, *Software Engineering*, Sixth Edition, Addison-Wesley Publishing Company, Reading, Massachusetts.

[44] SPSS Inc. 1997, *SPSS 7.5 for Windows*, Chicago, IL.

[45] Stark, G., R.C. Durst, & C.W. Vowell 1994, Using Metrics in Management Decision Making, *IEEE Computer*, Vol. 27, No. 9, 42-48.

[46] Versaw, L. 1989, PC-METRIC - A Measuring Tool For Software, *The C Users Journal*, 8(1).

[47] Weyuker, E.J. 1988, Evaluating Software Complexity Measures, *IEEE Transactions on Software Engineering*, 14(9), 1357-1365.

[48] Zuse, Horst 1990, *Software Complexity: Measures and Method*s, Walter de Gruyter publishers, Berlin, Germany.

[49] Zuse, Horst 1993, Support of Experimentation by Measurement Theory, H. Rombach, V. Basili, and R. Selby, Editors, *Experimental Software Engineering Issues*, (Lecture Notes in Computer Science, Volume 706), Springer-Verlag, New York, NY, 137-140.

[50] Zuse, Horst 1997, Reply to Property-Based Software Engineering Measurement, *IEEE Transactions on Software Engineering*, 23(8), 533-533.