Metrics for Assessing a Software System's Maintainability

Paul Oman and Jack Hagemeister

Software Engineering Test Lab
University of Idaho, Moscow, Idaho 83843
oman@cs.uidaho.edu

Abstract

The factors of software that determine or influence maintainability can be organized into a hierarchical structure of measurable attributes. For each of these attributes we show a metric definition consistent with the published definitions of the software characteristic being measured. The result is a tree structure of maintainability metrics which can be used for purposes of evaluating the relative maintainability of the software system. In this paper we define metrics for measuring the maintainability of a target software system and discuss how those metrics can be combined into a single index of maintainability[1].

I. A Software Maintainability Taxonomy

The characteristics influencing or determining the maintainability of a software system are many and varied. In an earlier study of 35 published works on software maintainability, we coalesced software maintainability definitions into a hierarchical structure containing 92 known maintainability attributes [Oman91b]. The tree structure is refined through successive subtrees until a leaf node, representing an identified maintainability attribute, is defined. Our hierarchy serves as a taxonomic definition for software maintainability that is compatible with the 35 published works upon which it is based. A complete definition and reference list can be found in [Oman91b].

Figure 1 shows the upper level tree structure corresponding to the maintainability taxonomy. As shown in the figure, there are three broad categories of factors which influence and contribute to the maintainability of a software system: (1) The management practices (and related issues) being employed, (2) The operational hardware and software environments involved in working with the target software system, and (3) The target software system (including the source code and supporting documentation) that is being placed under maintenance.

In this paper we focus on the Target Software System, and provide metric descriptions and definitions for the maintainability attributes contained within that subtree. We emphasize that this choice of metrics and their definitions are exemplary and serves our purpose of building an automated maintainability assessment system [Hagemeister92, Oman92b]. Other metrics for many of these attributes have been proposed (see [Zuse91]). We do not claim that this is an exhaustive list, nor do we claim that we have chosen the best possible metric for each maintainability attribute. Rather, we show that for each attribute within the Target Software System subtree, there exists a definable metric which quantifies that characteristic. Figure 2 contains the tree structure defining the maintainability aspects of a Target Software System. In the next section we provide metric definitions for each attribute contained within the Target Software System subtree.

II. Target Software System Metrics

For each of the three major subparts within the Target Software System subtree (Maturity Attributes, Source Code, and Supporting Documentation), we now present a top-level definition and list of metrics that can be used to quantify the maintainability characteristics contained in that subtree.

1. Software maturity attributes

Software maturity attributes are defined as the physical characteristics and measures pertaining to the age and use of the target software system. Metrics for this portion of the maintainability hierarchy were adapted from [Boehm81, Grady87, Musa89, and Pehrson90]. Following are the metric definitions for each of the leaf nodes in the maturity attribute subtree of the maintainability hierarchy:

a. Age = Age of software system, since release, in months.

[1] This study was funded by a grant from the Hewlett-Packard Company. For implementation details contact Mr. Don Coleman, HP Corporate Engineering, 1801 Page Mill Rd. Bldg. 18-D, Palo Alto, CA 94303.

0-8186-2980-0/92 $03.00 © 1992 IEEE

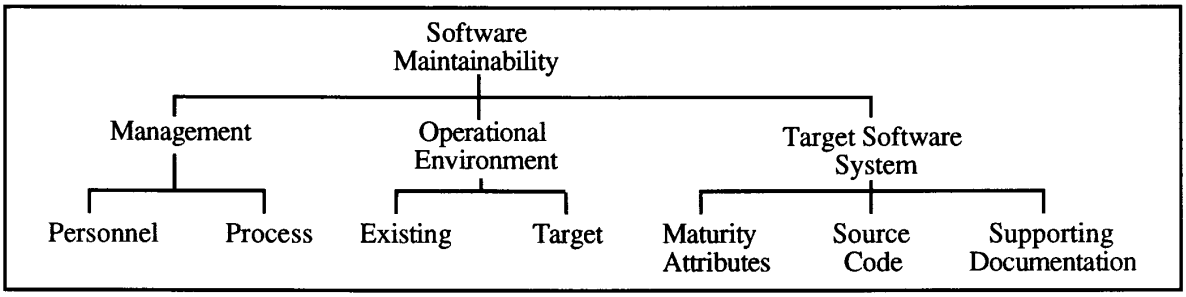

Figure 1. A Software Maintainability Hierarchy

b. Size = KNCSS (Thousands of Non-Commented Source Statements).

c. Stability = 1 - Change Factor (CF) , where the change factor is an exponential function of the percent of lines changed in each of the last *n* quarters.

$$CF = e^{D} * \sum_{i=1}^{n} c_i / i$$

where

$D = C_n - C_1$
C_i = Percentage of lines changed in each of the last *n* quarters.

d. Maintenance Intensity = An exponential function of the Maintenance Activity (MA), where maintenance activity is the percentage of hours expended in maintenance in each of the last *n* quarters.

$$MI = e^{D} * \sum_{i=1}^{n} MA_i / i$$

where

$D = MA_n - MA_1$
$MA_i = h_i / HPQ_i$
h_i = maintenance hours in each of the last *n* quarters.
HPQ_i = maximum work hours per quarter in each of the last *n* quarters.

e. Defect Intensity = An exponential function of the Defect Density (DD), where defect density is the number of defects reported divided by KNCSS for each of the last *n* quarters (or the current KNCSS used as a constant dividend).

$$DI = e^{D} * \sum_{i=1}^{n} DD_i / i$$

where

$D = DD_n - DD_1$
$DD_i = defects_i / KNCSS_i$

f. Reliability = The Musa-Okumoto metric, an exponential function of the Failure Intensity (FI), where failure intensity is the rate of failures (faults) per hour.

$R(t) = e^{-FI * t}$ where

t= time period for which the reliability is estimated.

g. Reuse = The percentage of lines of code of software system that have been reused from other systems (expressed in KNCSS).

h. Subjective Product Appraisals = Evaluations on a 5-point, forced-choice scale (Very Low, Low, Nominal, High, Very High) for each of the following characteristics: Programming language complexity, Application complexity, Development effort expended, Development techniques used, Product requirements volatility, Product dependencies, Complexity of the software "build," Completeness of diagnostics and test coverage, Complexity of test procedures, Installation complexity, Intensity of product use, and Efficiency of the software system.

2. Source code

Characteristics of the source code of the software system are divided into subcategories for (1) Control Structure, (2) Information Structure, and (3) Typography, Naming, and Commenting. Each of these three

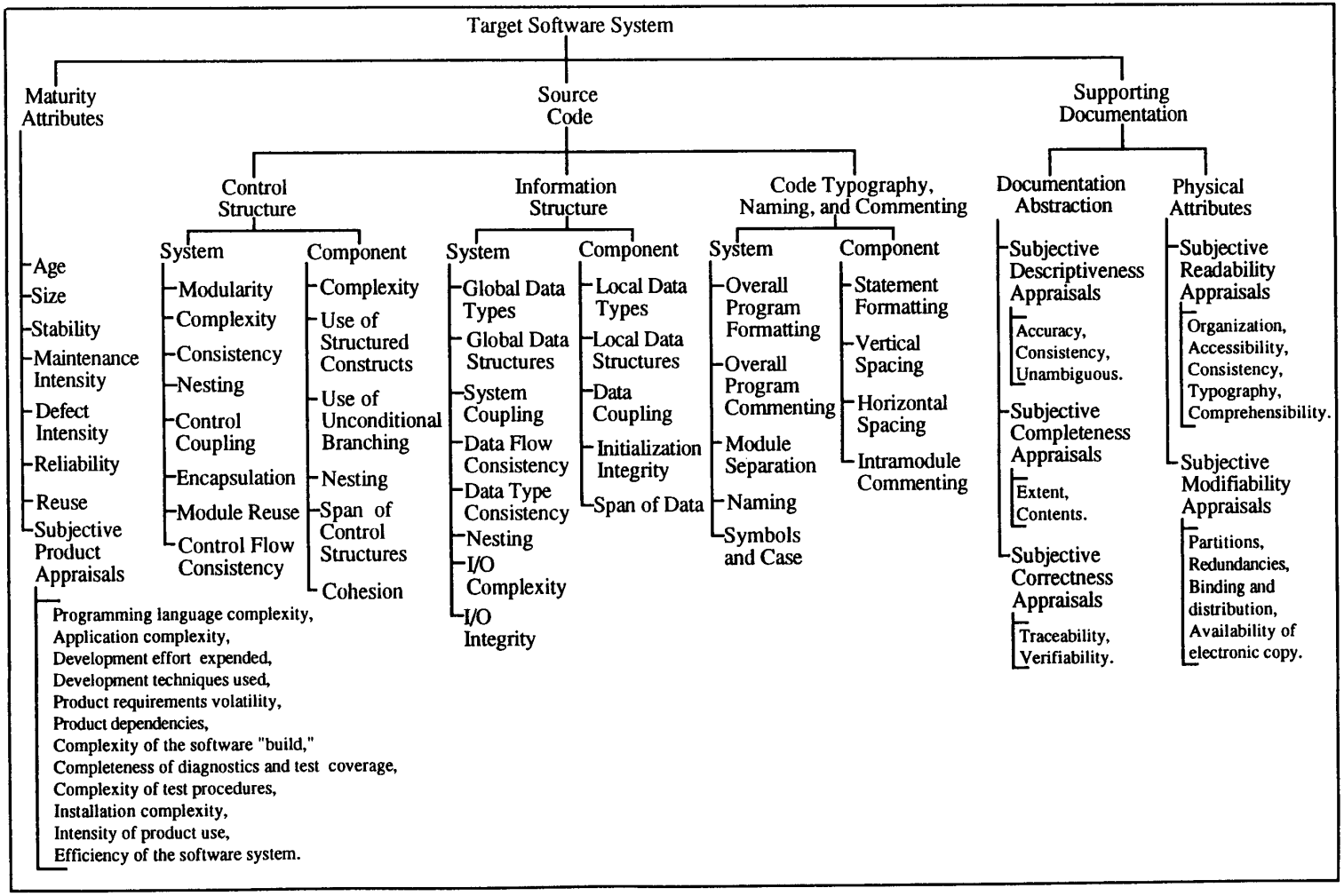

Figure 2. Target Software System Subtree

subtrees is further divided into (i) System and (ii) Component subcategories to distinguish between system-wide attributes and those representative of the modules (i.e., programs, procedures, subroutines or functions) within the system. Metric definitions for the source code subtrees were adapted from [Beizer90, Emerson84, Grady87, Grady92, Halstead77, IEEE88-982.1, IEEE90-610.12, McCabe76, Oman90, and Oman91a].

System control structure characteristics are those pertaining to the intermodular control attributes, the choice and use of control flow constructs, the manner in which the program or system is decomposed into algorithms, and the method in which those algorithms are implemented. Following are the metric definitions for each of the leaf nodes in the system control structure subtree of the maintainability hierarchy:

a. Modularity = A 2-tuple {Number of modules, Average module size in KNCSS}.

b. Complexity = A 2-tuple {V(g), Halstead's E} for the whole system.

c. Consistency = A 2-tuple {Standard deviation of the module size (in KNCSS), Standard deviation of V(g)} calculated over all modules.

d. Nesting = A 3-tuple {Maximum depth of module nesting, Average depth, Percentage of modules nested}

e. Control Coupling = (Fan_out)^2, an estimate of the size of the connectedness of system components, where Fan_out is the number of module calls.

f. Encapsulation = 1 - ((Data_coupling * #modules) / Fan-in), which represents the complement of the ratio of data coupling to module calls (where data coupling is defined in the next section on Information Structures).

g. Module Reuse = 1 - (#modules / Fan_in), the complement of the ratio of the number of modules defined to the number of module calls.

h. Control Flow Consistency = 1 - Percent of code anomalies, where percentage of code anomalies is the number of lines of dead code divided by the size of the system (in KNCSS).

Component control structure characteristics are those pertaining to the intramodular control flow and execution of a module within a program or system, including the choice and manner in which control constructs are implemented. Following are the metric definitions for each of the leaf nodes in the component control structure subtree of the maintainability hierarchy:

a. Complexity = A 2-tuple of the cyclomatic complexity and Halstead's effort {Average V(g), Average E} averaged over all modules.

b. Use of Structured Constructs = Percentage of single-entry-single-exit structures per module, averaged over all modules.

c. Use of Unconditional Branching = Percentage of unconditional branching structures per module, averaged over all modules.

d. Nesting = A 3-tuple {Maximum depth of structure nesting, Average depth, Percentage of structures nested} averaged over all modules.

e. Span of control structures = A 2-tuple {Maximum structure span in NCSS, Average structure span} averaged over all modules.

f. Cohesion = Emerson's cohesion metric k(M) averaged over all modules.

$$k(M) = \frac{\sum_{i}^{v} V(R_i)}{v}$$

where

$V(R_i)$ = V(g) of the set of instructions referencing the ith variable in the set of all variables in module M.
v = the number of variables in module M.

System information structure characteristics are those pertaining to the information storage and flow in a program or system, including global and intermodular data definition and data flow characteristics, and the system input and output characteristics. Following are the metric definitions for each of the leaf nodes in the system information structure subtree of the maintainability hierarchy:

a. Global data types = The number of global data types divided by the total number of defined data types.

b. Global data structures = The number of global data structures divided by the total number of defined data structures.

c. System coupling = (The number of global structures plus the number of passed parameters) divided by the total number of data structures.

d. Data flow consistency = 1 - Percent of data flow anomalies, where percentage of data flow anomalies is the number of data flow anomalies (used before definition, definition without use, redefinition without use) divided by the total number of data structures.

e. Data type consistency = 1 - Percent of data structures which undergo type conversion during assignment operations.

f. Nesting = A 3-tuple {Maximum depth of data structure nesting, Average depth, Percentage of data structures nested}.

g. I/O Complexity = The number of lines of code devoted to I/O divided by NCSS.

h. I/O Integrity = The number of bounded or screened I/O data items divided by the total number of I/O data items.

Component information structure characteristics are those pertaining to the intramodular data storage and manipulation of a module within a program or system, including local data structure and data flow characteristics. Following are the metric definitions for each of the leaf nodes in the component information structure subtree of the maintainability hierarchy:

a. Local data types = The number of local data types divided by the total number of defined data types averaged over all modules.

b. Local data structures = The number of local data structures divided by the total number of defined data structures, averaged over all modules.

c. Data coupling = (The number of global structures and passed parameters within a module) divided by the total number of data structures within the module, averaged over all modules.

d. Initialization integrity = Percentage of well defined variables (i.e., variables initialized prior to use), averaged over all modules.

e. Span of data = Average span of a data structure in NCSS, averaged over all data items across all modules.

System typography, naming and commenting characteristics are those pertaining to the overall layout and commenting of a program or system, including system-wide typographic and commenting conventions and intermodular layout and naming characteristics. By definition, these attributes have no effect on program execution. Following are the metric definitions for each of the leaf nodes in the system-wide typography, naming and commenting subtree of the maintainability hierarchy:

a. Overall program formatting = A 3-tuple {Percentage of blank lines in the whole program, Percentage of modules with blank lines, Percentage of modules with embedded spacing}.

b. Overall program commenting = A 2-tuple {Percentage of comment lines in the whole program, Percentage of modules with header (prologue) comments}.

c. Module separation = Percentage of modules with white-space or comments before/after the first/last line of the module.

d. Naming = The number of identifiers and labels with meaningful names divided by the total number of identifiers and labels, where meaningfulness is determined by the length of the identifier/label.

e. Symbols and case = A 3-tuple {Percentage of identifiers with embedded special symbols (e.g., "-" or "_"), Percentage of identifiers with mixed upper and lower case characters, Percentage of reserved words with mixed upper and lower case characters}.

Component typography, naming, and commenting characteristics are those pertaining to the intramodular typographic layout and commenting of a module within a program or system, including local naming characteristics, but excluding all characteristics affecting execution. Following are the metric definitions for each of the leaf nodes in the system-wide typography, naming and commenting subtree of the maintainability hierarchy:

a. Statement formatting = Percentage of uncrowded statements (no more than one statement per line) per module, averaged over all modules.

b. Vertical spacing = Number of blank lines divided by the total number of lines in the module, averaged over all modules.

c. Horizontal spacing = Number of lines with horizontal spacing (indentation and/or embedded non-singular spacing) divided by the total number of lines in the module, averaged over all modules.

d. Intramodule commenting = Number lines with comments divided by the total number of lines in the module, averaged over all modules.

3. Supporting Documentation

Characteristics of the supporting documentation for the software system are divided into subcategories for (1)

Documentation Abstraction, and (2) Physical Attributes. This serves to distinguish the mapping from the documentation to the code (the abstraction) from the physical attributes of the document set. All measurements for this portion of the maintainability hierarchy are subjective evaluations on a 5-point, forced-choice scale (Very Low, Low, Nominal, High, Very High) for documentation characteristics selected from [AFOTEC89, Boehm81, and IEEE84-830].

Supporting documentation abstraction characteristics are those pertaining to the mapping from the document set to the actual source code and its subsequent execution. Following are the metric definitions for each of the leaf nodes in the supporting documentation abstraction subtree of the maintainability hierarchy:

a. Descriptiveness Appraisals = Subjective evaluations of the document set for each of the following characteristics: Accuracy (relative to the code implementation), Consistency (lack of conflicts and contradictions), and Unambiguous (lack of ambiguous statements).

b. Completeness Appraisals = Subjective evaluations of the document set for each of the following characteristics: Extent of the document set, and Contents of the existing documentation.

c. Correctness Appraisals = Subjective evaluations of the document set for each of the following characteristics: Traceability to and from the code implementation, and Verifiability to the actual specifications.

Supporting documentation physical attributes are those characteristics pertaining to the readability and physical appearance of the supporting document set. Following are the metric definitions for each of the leaf nodes in the supporting documentation physical attributes subtree of the maintainability hierarchy:

a. Readability Appraisals = Subjective evaluations of the document set for each of the following characteristics: Organization, Accessibility via indices and table of contents, Consistency of the writing style, typography, and comprehensibility.

b. Modifiability Appraisals = Evaluations of the document set for each of the following characteristics: Document set partitions, Document set redundancies, Document set binding and distribution, and Availability of electronic copy

III. Assessing Software Maintainability

Given the above hierarchical structure of software maintainability metrics, the next step in assessing the maintainability of a target software system is to combine the metrics into a unified value (or set of values) that can be used as an index of the maintainability of the measured system. To date we have identified several methods of quantifying tree structures. Only the simplest is presented here.

Given the tree structure in Figures 1 and 2, which represent software maintainability, we define the internal nodes of the tree as dimensions or categories that we want to measure and the leaf nodes of the tree as attributes of maintainability *that can be measured.* (For example, the metrics presented in this paper.) Representing the dimensions as circles and the leaf nodes as squares, we derive the tree structure shown in Figure 3.

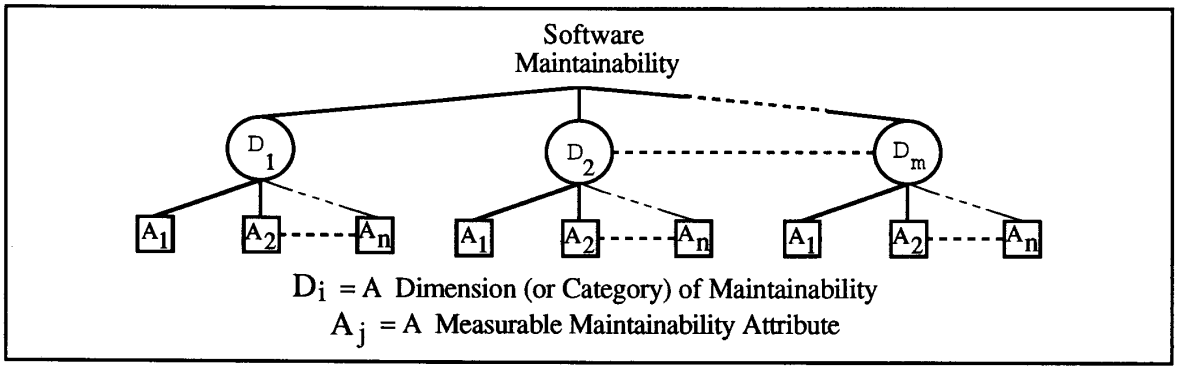

Figure 3. A Software Maintainability Tree

To quantify the maintainability of a tree we can then use the following formula:

$$\prod_{i=1}^{m} W_{D_i} \left(\frac{\sum_{j=1}^{n} W_{A_j} M_{A_j}}{n} \right)_i$$

where

W_{D_i} = Weight of influence of maintainability Dimension D_i

W_{A_j} = Weight of influence of maintainability Attribute A_j

M_{A_j} = Metric or measure of maintainability Attribute A_j

This formula represents the product of the weighted dimensions, where each dimension is measured as the average deviation from a known value of "goodness" for that maintainability attribute. Some explanatory notes are in order:

1. Weights can be applied (if necessary) to both the dimensions and the attribute measures to "adjust" the impact of those characteristics on the maintainability index. Initially, all weights can be set to one. This means that all attributes contribute nominally to the maintainability index.

2. The average deviations from good maintainability attributes must be structured so that values range between 0 and 1. This yields a percentage of "goodness." The product of those percents thereby yields an overall percent (or probability) of maintainability.

3. Not all attributes identified in the tree need to be measured. Only the most significant measures (those with the most predictive value) need to be assessed.

The principles and metrics presented in this paper have successfully been used in a set of tools for predicting and measuring software maintainability:

1. Polynomials for predicting (assessing) maintainability from some of the metrics defined here have been constructed and validated [Oman92a].

2. A prototype source code analyzer which uses the techniques outlined in this paper to assess the maintainability of C or Pascal source code has been implemented [Oman92b].

3. A spreadsheet implementation of the portions of the maintainability taxonomy which are not measurable from the source code has been built and is being tested [Hagemeister92]. It is used in conjunction with the two tools listed above.

There are other ways to quantify tree structures and gauge software quality [Basili83, BDM78, Berns84, Kafura87, Kermerer90, Khoshgoftaar92, Peercy81, Selby89, Yau88]. We are studying those methods to see if and how they would be useful in measuring software maintainability. This paper represents an initial attempt to define maintainability in such a way that it is a measurable characteristic of a software system.

References

[AFOTEC89] *Software Maintainability - Evaluation Guide*, an updated AFOTEC Pamphlet 800-2, vol. 3, Oct. 31, 1989, HQ Air Force Operational Test & Evaluation Center, Kirtland Air Force Base, NM 87117.

[Basili83] V. Basili & D. Hutchens, "An Empirical Study of a Syntactic Complexity Family," *IEEE Transactions on Software Engineering*, Vol. 9(6), Nov. 1983, pp. 664-672.

[BDM78] BDM Corporation, *Analysis of Software Maintainability Evaluation Process*, report #BDM/TAC-78-698-TR, compiled for the Air Force Test and Evaluation Center (Kirtland Air Force Base, NM) by the BDM Corp., 1801 Randolph Road SE, Albuquerque, NM 87106, 1978.

[Beizer90] B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, NY, 1990.

[Berns84] G. Berns, "Assessing Software Maintainability," *Communications of the ACM*, Vol. 27(1), Jan. 1984, pp. 14-23.

[Boehm81] B. Boehm, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[Emerson84] T. Emerson, "A Discriminant Metric for Module Cohesion," *Proceeding of the 7th International Conference on Software Engineering*, IEEE Computer Society Press, New York, NY, 1984, pp. 294-303.

[Grady87] R. Grady & D. Caswell, *Software Metrics: Establishing a Company-wide Policy*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[Grady92] R. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, New York, 1992.

[Hagemeister92] J. Hagemeister & P. Oman, *HP-SMES: Software Maintainability Estimation Spreadsheet*, SETL Report #92-05-ST, June 1992, University of Idaho, Moscow, ID 83843.

[Halstead77] M. Halstead, *Elements of Software Science*, North Holland, Amsterdam, 1977.

[IEEE84-830] IEEE Computer Society, *IEEE Standard Guide for Software Requirements Specifications*, IEEE Std. 830-1984, IEEE, Inc., 345 East 47th Street, New York, NY 10017.

[IEEE88-982.1] IEEE Computer Society, *IEEE Standard Dictionary of Measures to Produce Reliable Software*, IEEE Std. 982.1-1988, IEEE, Inc., 345 East 47th Street, New York, NY 10017.

[IEEE90-610.12] IEEE Computer Society, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990 (supersedes Std. 729-1983), IEEE, Inc., 345 East 47th Street, New York, NY 10017.

[Kafura87] D. Kafura & G. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," *IEEE Transactions on Software Engineering* , Vol. 13(3), Mar. 1987, pp. 335-343.

[Kemerer90] C. Kemerer, *Reliability of Function Points Measurement: A Field Experiment*, Center for Information Systems Research Report, Massachusetts Institute of Technology, Cambridge, MA, 1990.

[Khoshgoftaar92] T. Khoshgoftaar & J. Munson, "An Aggregate Measure of Program Module Complexity," *Papers of the Fourth Annual Oregon Workshop on Software Metrics* (Session V, Track B), Silver Falls, OR, Mar. 1992.

[Musa89] J. Musa, "Tools for Measuring Software Reliability," *IEEE Spectrum*, Vol. 26(2), Feb. 1989, pp. 39-42.

[Oman90] P. Oman & C. Cook, "Typographic Style is More Than Cosmetic," *Communications of the ACM*, Vol. 33(5), May 1990, pp. 506-520.

[Oman91a] P. Oman & C. Cook, "A Programming Style Taxonomy," *Journal of Systems and Software*, Vol. 15(3), July 1991, pp. 287-301.

[Oman91b] P. Oman, J. Hagemeister & D. Ash, *A Definition and Taxonomy for Software Maintainability*, SETL Report #91-08-TR, University of Idaho, Moscow, ID 83843.

[Oman92a] P. Oman & J. Hagemeister, *Construction and Validation of Polynomials for Predicting Software Maintainability*, SETL Report #92-06-TR, July 1992, University of Idaho, Moscow, ID 83843.

[Oman92b] P. Oman, *HP-MAS: A Tool for Software Maintainability Assessment*, SETL Report #92-07-ST, August 1992, University of Idaho, Moscow, ID 83843.

[Peercy81] D. Peercy, "A Software Maintainability Evaluation Methodology," *IEEE Transactions of Software Engineering* , Vol. 7(4), July 1981, pp. 144-152.

[Pehrson90] R. Pehrson & R. Tang, *CPE Centre Product Entry Process*, (unpublished) Hewlett-Packard, Pinewood Information Systems Division, Wokingham, Berkshire, UK, 1990.

[Selby89] R. Selby & A. Porter, "Software Metric Classification Trees Help Guide the Maintenance of Large-Scale Systems," *Proceedings Conference on Software Maintenance - 1989*, (Miami, FL, Oct. 16-19), IEEE Computer Society Press, Washington DC, 1989, pp. 116-123.

[Yau82] S. Yau & J. Collofello, "Design Stability Measures for Software Maintenance," *Proceedings of the 6th Annual International Conference on Software and Applications*, IEEE Computer Society, New York, NY, 1982, pp. 100-108.

[Zuse91] H. Zuse, *Software Complexity Measures and Methods*, Walter de Gruyter & Co., New York, NY, 1991.