Software Metrics for Predicting Maintainability

Software Metrics Study: Technical Memorandum 2

Marc Frappier
Stan Matwin
Ali Mili

University of Ottawa

January 21, 1994

0 Table of Contents

1. Introduction

The focus of this study is to evaluate software MAINTAINABILITY. Maintainability is defined as the ease with which a system can be maintained. It is an internal attribute of the system which cannot be measured directly. Instead, one can measure maintenance process attributes, such as the time required to make a change, which are influenced by software maintainability. Since we are interested to evaluate maintainability during development and at delivery time, these maintenance process measures are available too late. A solution to this problem is to build models to predict the maintenance process measures from software product measures like cohesion, coupling and complexity.

This document is a first step towards the construction of a maintainability prediction model. It identifies a set of software measures based on correlations published in empirical studies.

The rest of this document is structured as follows. In section 2, we present the methodology for selecting metrics. In section 3, we define a framework for describing metrics. In section 4, we provide a summary list of the metrics selected for review. Section 5 contains a detailed description of the metrics. For each metric, we provide a recommendation and a justification.

2. Methodology for Selecting Metrics

2.1. Terminology

We define a *module* as an independent unit of software that can be called by name. Examples are subroutines and functions in FORTRAN, functions in C, or procedures, functions and packages in ADA. Following [IEEE91], we use the word *fault* for "an accidental condition that causes a functional unit to fail to perform its required function. A fault, if encountered, may cause a *failure*. It is synonymous with bug". A *change* is a modification to a software unit.

2.2. Selection Method

We proceeded in two steps to select software metrics: first selection and detailed evaluation.

a) - First Selection

We selected an initial set of metrics based on their applicability to MDSF and abundance of literature on the metric such as experimental evaluation or theoretical evaluation. In absence of such documentation, we proceed with our own evaluation and suggestion. Naturally, preference will be given to well documented existing metrics. This first selection is limited by the man power and the short time period allocated to this project.

b) - Detailed Evaluation

We evaluated each metric according to the following list of criteria derived from [BBKL78] and [CDS86]:

b.1) - Objectivity and Algorithmic Measurement

"An objective, or algorithmic measurement is one that can be computed precisely according to an algorithm. Its value does not change due to changes in time, place or observer"[CDS86]. Objectivity is critical for establishing the credibility of a metric program. In an experiment on the relationship between software structure and maintainability [GS89], Gibson and Senn observed that system differences were not discernible to programmers. There was no homogeneity of perceptions of relative system complexity among programmers. Subjective metrics are prone to produce inconsistent results. Conte *et al* [CDS86] strongly advocate the use of objective metrics.

We will classify metrics as objective or subjective.

b.2) - Validity

How do we establish if a metric can predict maintainability? A metric may be objective but it may loosely correlate with our intuitive notion of maintainability. To be as objective as possible in the evaluation of a metric, we will look for experimental results published by other researchers establishing a correlation between a metric and *indicators* of maintainability. An indicator is a measure of maintainability based on historical maintenance data. The distinction between an indicator and a metric is when they are computed. A metric is computed on the software products with the intent to predict maintainability indicators. An indicator is an actual value computed after maintenance has started. In MDSF, maintenance indicators could be computed for the first two development increments if the historical maintenance data was sufficiently precise. No comprehensive indicator of maintainability has been proposed yet. Each indicator is a partial view of maintainability. It does not always discriminate between a maintainable and an unmaintainable module. Validation is based on statistical correlation (linear or rank [Sieg55]). The correlation level denotes how well the metric can predict the indicator. The indicators used in the literature are:

IE = internal effort per change and module (i.e. effort spent in a module for a change). This indicator does not take into account the size of a change (e.g. number of lines added, modified or deleted). For instance, a small change on a poorly maintainable module may require the same effort as an extensive change on a highly maintainable module. Programmer's ability and programmer's familiarity with the system are other elements not considered by this indicator. Maintainability should decrease as IE increases.

XE = external effort per change and module (i.e. effort spent in other module for a change). The weaknesses of this indicator are similar to IE. Maintainability should decrease as XE increases.

NB = average number of modules affected by a change, reported by module (i.e. when a module is affected by a change, the number of other modules affected by the change). This indicator does not consider the nature nor the extent of a change. Maintainability should decrease as NB increases.

PP = maintenance productivity by project (difference between final and initial number of non comments source line divided by the number of hours spent for all modules of a maintenance project). This indicator does fully take into account the size of a change (lines added, modified, deleted). Maintainability should decrease as PP increases.

Many experiments use other indicators indirectly related to maintainability. We include them in this review because they sometimes represent the only validation data available for a metric. These indicators are:

FC = number of faults or program changes per module. It is more a reliability indicator than a maintainability indicator. A poorly maintainable module may have less detected faults than a highly maintainable module which was easily tested and verified, and where all faults were detected. The hypothesis is that maintainability decreases as the number of faults in a module increases.

DE = development effort per module (design, coding and testing). The hypothesis is that maintainability decreases as the development effort increases for a module.

Some experiments validate their metric by correlation with a subjective evaluation of maintainability or complexity. Having expressed our reserves about subjectivity, we mention these results if they represent the only experimental data available.

SE = subjective evaluation of complexity or maintainability.

We will classify metrics as XY, where

X : correlation level

H = high correlation, > .70

M = medium correlation, between .50 and .70

L = low correlation, < .50

We restrict to results with a significance less than .05.

Y : one of the maintainability indicators mentioned earlier

Example: HIE = high correlation with internal effort per change and module.

b.3) - Degree of Automation

With this criterion, we classify metrics as:

Auto = the computation can be automated for the product used in MDSF

Expert = the computation is manual and must be performed by an expert

Partially = the computation can be partially automated

b.4) - Availability of Commercial Off-the-shelf Software

With this criterion, we classify metrics as:

Yes = commercial tool available and no manual computation required

No = no tool available

3. Framework for Describing Metrics

To evaluate and select metrics, we need a uniform description framework. We propose the following items:

- General Information: name, short description, author, sources;
- Criterion Measured
- Product Measured
- Formula
- Algorithm
- Interpretation
- Validity
- Benefits
- Cost of buying or developing an automated tool
- Effort Required to Compute the Metric
- Recommendation with justification

4. Summary of Selection

Nineteen metrics were selected for review. Fourteen were recommended for implementation at CSA. Every product of the life cycle is measured by at least one metric. Seven source code metrics can be computed by commercial tools. Some design metrics could be automated if a formal syntax or a case tool was used to write them. Manual metrics must be computed by an expert. Objective metrics can be computed by a single expert, but subjective metrics should be determined by a group of experts. In the latter case, some methodology like the Delphi method [Boeh81] should be used to derive conclusions from a set of evaluations.

The software engineering community has proposed over two hundred software metrics. We certainly cannot claim completeness in this study. With the manpower and time available, we focused on well known metrics applicable in the MDSF environment.

4.1. Summary List of Metrics

Table 1 provides a summary list of the metrics selected for review. Each metric is identified by a mnemonic name and a reference page number where a detailed description is found. The third column, titled *Obj.*, indicates if the metric is objective or not. The next column summarises the validation results published in the literature. The notation for correlation levels is explained on page 5. The mention *No Data* means that no validation results were found. *Is related* refers to experiments revealing a relationship between the metric and some indicator using other techniques than statistical correlation. The next three columns list the products on which the metrics is computed, the degree of automation (see page 6) and the availability of tools. The last column indicates if the metric is recommended for implementation at CSA.

Metric	Page	Obj.	Validity	Products	Compu.	Tool	Rec
UR	13	Yes	No Data	All	Part.	No	Yes
NR	15	Yes	No Data	All	Part.	No	Yes
M-MC	17	Yes	No Data	SDDD & S. Code	Expert	No	Yes
M-MS	19	No	No Data	SDDD & S. Code	Expert	No	Yes
HK-IF	21	Yes	1 HFC, Is Related	SDDD	Expert	No	No
				S. Code	Auto.	No	No
R-IF	23	Yes	1 MIE, 1 HXE, 1 HNB	SDDD	Expert	No	No
				S. Code	Auto.	No	No
KPL-IF	25	Yes	1 LFC, 1 LSE	SDDD	Expert	No	No
				S. Code	Auto.	No	No
IF4	27	Yes	1 HSE	SDDD	Expert	No	No
				S. Code	Auto.	No	No
CA-DC	29	Yes	1 HFC, 1 LDE	SDDD	Expert	No	Yes
				S. Code	Auto.	No	Yes
COCO	31	No	Is Related	SDDD, S. Code	Expert	No	Yes
v(G)	33	Yes	1 LIE, 2 MIE, 1 HIE,1 LNB 1 LFC, 2 MFC, 3 HFC 1 LDE, 1 MDE	S. Code	Auto.	Yes	Yes
Knots	35	Yes	1 HIE	S. Code	Auto.	Yes	Yes
RLC	36	Yes	1 HIE, 1 MPP	S. Code	Auto.	Yes	No

Vcd	37	Yes	Is Related	S. Code	Auto.	Yes	Yes
Vcs	38	Yes	Is Related	S. Code	Auto.	Yes	Yes
Ls	39	Yes	Is Related	S. Code	Auto.	Yes	Yes
LOC	40	Yes	1 LIE, 2 MIE, 1 HIE, 1 LNB, 1 LFC, 3 MFC 1 LDE, 1 MDE, 1 HDE	S. Code	Auto.	Yes	Yes
E	41	Yes	1 LIE, 1 MIE, 2 HIE 1 LFC, 1 MFC, 2 HFC 1 LDE, 1 MDE, 1 HDE	S. Code	Auto.	Yes	Yes
DAR	43	No	No Data	All	Expert	No	Yes
SCC	45	No	No Data	All	Expert	No	Yes

Table 1 - Summary List of Metrics

4.2. List of Metrics by Phase

It is our objective to evaluate maintainability at each phase of the life cycle. Which metric can be computed at each phase depends on what information is available. Table 2 provides a list of metrics per phase and milestone review. Recommended metrics are underlined. A product delivered in one phase is usually modified, or sometimes refined, in subsequent phases. Therefore, the metrics computable on that product should be recomputed at each subsequent phase to reflect the new software status.

Phase	**Milestone Review**	**Metrics**
Requirement Definition	System Requirement Review	None
System Design	System Design Review	UR, NR
Preliminary Design	Preliminary Design Review	UR, NR
Detailed Design	Critical Design Review	UR, NR, M-MC, M-MS, CA-DC, COCO, HK-IF, R-IF, KPL-IF, IF4
Implementation	Test Documentation Review	All
Integration & Test	Test Readiness Review	All
Verification & Acceptance	Acceptance Review	All

Table 2 - List of metrics per phase

4.3. List of Metrics per Product and Criterion

Table 3 illustrates the coverage of the reviewed set of metrics over the criterion identified in technical memorandum 1. Only two criteria are not measured by any metric.

Product	Criterion	Metrics
CEI Specification	Readability	No proper metric found
Requirements Specification	Traceability	NR, UR
	Readability	No proper metric found
Software Preliminary Design Document	Traceability	NR, UR
Software Detailed Design Document	Coupling	M-MC
	Cohesion	M-MS
	Coupling / Cohesion	HK-IF, R-IF, KPL-IF, IF4, CA-DC
	Size	No proper metric found
Source Code	Traceability	NR, UR
	Control Structure	v(g), Knots, RLC
	Independence	No metric proposed
	Readability	Vcd, Vcs, Ls
	Size	LOC, E, RLC
	Doc. Accuracy	DAR
	Consistency	SCC

Table 3 - List of metrics per product and criterion

4.4. Criteria Without Metric

One criterion, readability, has no metric for two products: the CEI Specification and the Requirements Specifications. Some metrics have been proposed to measure readability of english texts: Dale & Chall [DC48], Flesh-Kincaid [KAOC81] and Fog Index [BJ80]. They are

based on word length, number of syllables and phrase length. None of them considers the semantic elements and logical organization of the text. Boehm *et al* [BBKL78] recommend to use the Fog Index. Unfortunately, we have not found any experimental data assessing their suitability for software specification documents. We feel that the reviews performed by experts during the System Requirement Review and the System Design Review are more reliable than any such readability index. It might be useful to provide the reviewers with a methodology for evaluating readability. We were short of time to propose one in this study.

The sole criterion associated with portability, source code independence, was not addressed. The definition of a metric of this type depends on the programming language, the operating system and the computer used. With the appropriate expertise, it should be feasible to derive a set of metrics for each programming language used on MDSF. We were short of time and expertise to devise such metrics. However, since portability is an important factor for MDSF, source code independence should be addressed in the project following the Software Metrics Study.

4.5. Tools

A number of tools were mentioned for each metric. We provide a summary description for each tool. An important aspect of a tool is its capacity to handle file names as they are definable on the VAX. For instance, it is not possible to handle file names with more than eight characters on a PC. To analyze MDSF files on a PC, it is necessary to rename all files to file names not exceeding eight characters, which is a cumbersome operation to perform.

Name: DATRIX
Company: Bell Canada, distributed by Schemacode International
Location: Montreal, Quebec (514) 683-8693
Price: 2500 $ - PC Installation
Product Measured: Source Code
Languages Supported: Fortran, C, Ada, Pascal, Cobol, C++
Metrics Supported: v(G), Vcd, Vcs, Ls, Knots & others

Name: PC-METRIC
Company: Set Laboratories
Location: Mulino, Oregon (503) 829-7123
Price: 400 $ - PC Installation
Product Measured: Source Code
Languages Supported: Fortran, C, Ada, Pascal, Cobol, C++ & others
Metrics Supported: v(G), Halstead's E, LOC

Name: METRIC
Company: Software Research
Location: San Francisco, California (415) 957-1441
Price: 4500 $ - PC Installation
Product Measured: Source Code
Languages Supported: Fortran, C, Ada, C++
Metrics Supported: v(G), Halstead's E

Name: SAP
Company: NASA, distributed by COSMIC, University of Georgia
Location: Athens, Georgia (706) 542-3265
Price: 3125 $ - DEC VAX series
Product Measured: Source Code
Languages Supported: Fortran
Metrics Supported: v(G), Halstead's E, LOC

A combination of Datrix and PC-Metric would cover the greatest number of metrics for a minimal price. Both can export results in a format readable by popular spreadsheet and database programs available on PC.

5. Detailed Description of Metrics

In the sequel, we present a detailed description of the metrics selected for evaluation. For the sake of readability, each metric is described on a separate page.

5.1. Unreferenced Requirements (UR)

Description: The number of original requirements not referenced by a lower document in the documentation hierarchy.

Author: Modified IEEE metrics.

Sources: [IEEE91], [Henn80]

Criteria Measured: Traceability (RS, SDD, Source Code)

Products Measured: CEI Spec, RS, SDD, Source Code

Formula: UR = number of R1 not referenced by a R2, where

a) R1 = an original requirement in CEI Spec

   R2 = an original requirement in the RS

b) R1 = an original requirement in RS

   R2 = an original requirement in the SDD

c) R1 = an original requirement in SDD

   R2 = the source code of a module

Algorithm:

The traceability index is available on a database for a). For b), references are made to the RS within a description of a CSC or a CSU. Using a text editor, it should be simple to search on the keywords introducing a reference. Use a similar approach for c).

In the context of MDSF, some requirements apply to every configuration item. Examples are requirements for maintainability and testability. These requirements are unlikely to be referenced in each configuration item. They should be removed from the count in R1.

Interpretation:

A value greater than zero indicates that some requirements are not implemented.

Validity:

No experimental data available. It has been used successfully on large projects [Henn80].

Benefits:

Unreferenced requirements are detected.

Cost of Buying or Developing:

No tool available. No data available on cost of developing.

Computation Effort:

Computation can be partially automated. No data available on computation effort.

Recommendation: Implement the metric

Traceability is widely recognized [IEEE91], [BO85] as an important property for software maintenance.

5.2. Non Referencing Items (NR)

Description: The number of items not referencing an original requirement.

Author: Modified IEEE metrics.

Sources: [IEEE91], [Henn80]

Criteria Measured: Traceability for RS, SDD and source code.

Products Measured: CEI Spec, RS, SDD, Source Code

Formula: NR = number of R2 not referencing any R1, where

a) R1 = an original requirement in CEI Spec

   R2 = an original requirement in the RS

b) R1 = an original requirement in RS

   R2 = an original requirement in the SDD

c) R1 = an original requirement in SDD

   R2 = the source code of a module

Algorithm:

The traceability index is available on a database for a). For b), references are made to the RS within a description of a CSC or a CSU. Using a text editor, it should be simple to search on the keywords introducing a reference. Use a similar approach for c)

Interpretation:

A value greater than zero indicates that some items have no requirements.

Validity:

No experimental data available. It has been used successfully on large projects [Henn80].

Benefits:

Items without requirements are identified.

Cost of Buying or Developing:

No tool available. No data available on cost of developing.

Computation Effort:

No data available on computation effort.

Recommendation: Implement the metric

Traceability is widely recognized [IEEE91, [BO85] as an important property for software maintenance.

5.3. Module Coupling (M-MC)

Description: A measure of the strength of the relationships between modules.

Author: G.J. Myers

Sources: [Myer75], [BO85]

Criterion Measured: Coupling

Product Measured: SDD, source code

Formula: Myers defines six levels of coupling which are, in order of decreasing strength:

Content: a module makes a direct reference to the contents of another module (e.g. using absolute displacement);

Common: a number of modules share a common data structure (e.g. using FORTRAN COMMON area);

External: a number of modules share a common data item;

Control: a control element is passed to a module;

Stamp: a data structure is passed to a module

Data: none of the above coupling level apply and all arguments passed to a module are data items.

Algorithm:

Myers provides the following checklist to evaluate module coupling (see next page). A blank entry represents an irrelevant condition. If a module satisfies more than one level of coupling, the highest (worse) one is selected.

Direct reference between the modules	Y		N	N	N	N	N
Modules are packaged together		Y	N	N	N	N	N
Some interface data is external or global			Y	Y	N	N	N
Some interface data is control information					Y	N	N
Some interface data is in a data structure			Y	N		Y	N
Content coupling	X	X					

Common coupling			X				
External coupling				X			
Control coupling					X		
Stamp coupling						X	
Data coupling							X

Interpretation:

Maintainability decreases as coupling increases.

Validity:

No correlation data is available. However, this metric has gained wide acceptance in the software engineering community.

Cost of Buying or Developing:

Computation is manual.

Computation Effort:

No data available on computation effort. Each module must be reviewed by an expert. Computation should be integrated in the design review activities.

Recommendation: Implement the metric

Coupling is a critical attribute of design. Other metrics measuring coupling do not distinguish between different levels of coupling.

5.4. Module Strength (M-MS)

Description: A measure of how strongly related are the elements within a module.

Author: G.J. Myers

Sources: [Myer75], [BO85]

Criterion Measured: Cohesion

Product Measured: SDD, source code

Formula: Module strength is a subjective metric. Myers defines seven levels of strength which are, in order of increasing strength:

Coincidental: there is no meaningful relationship among elements of a module;

Logical: instructions are logically similar (e.g. I/O module);

Classical: same as logical, but operations are related in time (e.g. initialization or termination module);

Procedural: all operations are related to a problem procedure;

Communicational: same as procedural, but elements use the same data;

Informational: a module containing more than one functional strength module acting on a common database;

Functional: all operations contribute to carrying out a single function.

Algorithm:

Myers provides the following checklist (see next page) for determining the strength level. A blank entry represents an irrelevant condition. If a module satisfies more than one level of strength, the highest (best) one is selected.

Difficult to describe the module's function	Y	N	N	N	N	N	N	N
Module performs more than one function			Y	Y	Y	Y	Y	N
Only one function performed per invocation			Y	N	N	N	Y	
Each function has an entry point			N				Y	
Module performs related class of functions		N	Y	Y				

Functions are related to problem's procedure				N	Y	Y		
All of the functions use the same data					N	Y	Y	
Coincidental	X	X						
Logical			X					
Classical				X				
Procedural					X			
Communicational						X		
Informational							X	
Functional								X

Interpretation:

Maintainability increases with module strength. The design goal is to have modules with informational or functional strength.

Validity: No correlation data is available. However, this metric is widely accepted in the software engineering community.

Cost of Buying or Developing: Computation is manual.

Computation Effort: No data available on computation effort. Each module must be reviewed by a group of experts to increase objectivity. Computation should be integrated in the design review activities.

Recommendation: Implement the metric

Cohesion is a critical attribute for maintenance. We see it as a complement to the information flow metrics which do not measure cohesion directly.

5.5. Information Flow (HK-IF)

Description: A measure of the control flow and data flow between modules.

Author: S. Henry, D. Kafura

Sources: [HK81], [KH81]

Criterion Measured: coupling, cohesion

Product Measured: SDD, source code

Formula: $IF_m = (\text{fan-in}_m * \text{fan-out}_m)^2 * \text{int-comp}_m$

IF_m = information flow complexity of module *m*

fan-in_m = number of flows into module *m* +
number of global data structures read by module *m*.

fan-out_m = number of flows from module *m* +
number of global data structures updated by module *m*.

intl-comp_m = a measure of the internal complexity of the module. [HK81] used LOC, but suggested that other measures such as v(G) could also be used. At design time, LOC is not available. [Shep92] proposes a subjective metric, *worki*, to estimate module size at design time, based on the traceability between requirements specification and software design. v(G) can be estimated if the module logic is sufficiently detailed in the design.

Algorithm:

A flow of information goes from a *source* module to a *destination* module. Unfortunately, it is difficult to obtain a rigorous definition of a flow from the published literature ([HK81], [KH81]). The definition of a flow in [HK81] is not exactly the same as definition provided in [KH81]. For simplicity, we provide the definition given in [HK81]. There is a flow from module A to module B if:

1) - A calls B;

2) - B calls A and A returns an output value to B

3) - C calls both A and B, and A returns an output value to C which is then passed to B

The inconsistency between [HK81] and [KH81] arises in definitions 1) and 2). A global data structure is a variable accessible to a set of modules without being passed as a parameter. Examples are COMMON variable in FORTRAN or global variables in PASCAL or external variables in C. It is not mentioned in [HK81] or [KH81] if files are also considered as global data structures.

Interpretation:

According to [HK81], as the information flow increases, maintainability should decrease. High information flow would be an indicator of low cohesion and high coupling.

Validity:

[KH81] HFC (.95, measured at the source code level using LOC). Validation was done on the source code of Unix. In [HK81], the authors propose other metrics for cohesion and coupling based on the information flow concept, but they have no experimental validation.

[HS90] correlates information flow measured at the design level and at the source code level. Results show high correlation between design measure and source code measure, even at various levels of design refinement.

[CA88] indicates that the information flow measure does not fit well their experimental data, without providing any specific result in the study.

[KC85] validated HK-IF on data from the Software Engineering Laboratory using outlier analysis. Results show that extreme faulty or effort-intensive modules can be detected using this metric. However, a simple metric like LOC (described further) was performing equally well.

Cost of Buying or Developing:

Computation can be automated for source code. No tool available. No data available on cost of developing.

Computation Effort:

SDD: manual with the current format. Source code: automatic. No data available on manual computation effort.

Recommendation: Do not implement the metric.

The five information flow metrics reviewed in this document present some deficiencies. HK-IF, R-IF and IF4 are defined vaguely or ambiguously. They have been validated only by their developer. They are not very good indicator of cohesion. A module with high cohesion called by many modules may have a high metric value. With some reserves, we recommend CA-DC because of its simplicity and its validation on industrial flight software.

5.6. Integrated Information Flow of Rombach (R-IF)

Description: A measure of intermodule and intramodule complexity based on information flow.

Author: H.D. Rombach

Sources: [Romb84]

Criterion Measured: cohesion, coupling

Product Measured: SDD, source code

Formula: Rombach does not provide a complete formula for his metric. We content ourselves with describing the main primitives.

Intermodule complexity is measured using information flows. The notion of module in [Romb84] is similar to the notion of package in ADA. It is an aggregation of imported and exported functions. There are four types of information flow from a module A to a module B.

1) Explicit Global: If A has write access and B has read access to the same global variable.

2) Implicit Global: If B uses an implicit assumption established in A and not available as data in the code. Examples are buffer size, number of terminals, etc.

3) Local Direct: If A calls or uses B

4) Local Indirect: If either of the two following conditions is satisfied.

a) If B receives data from A due to a call or a use.

b) If C calls or uses A and B in sequence, passing data from A to B.

Intramodule module complexity is measured using the number of interface accesses. An interface access is either an exported function or a call to an imported function.

Algorithm: Not available.

Interpretation:

Maintainability should decrease as intermodule and intramodule complexity increase.

Validity:

The metric was validated on a system called TSS (Time Sharing System) and developed by graduate students at the University of Kaiserslautern. Faults and changes were artificially designed by the experimenter. Maintenance was done by a different group of graduate students. The following correlation levels were obtained: HXE, HNB, MIE.

Cost of Buying or Developing:

Computation could be automated for source code. No tool available. No data available on developing effort.

Computation Effort:

SDD: manual with the current format. Source code: automatic. No data available on manual computation effort.

Recommendation: Do not implement the metric.

The metric is based on the notion of package rather than module. It would be difficult to compute in MDSF. Moreover, we feel the metric requires additional validation on an industrial system with real faults and modifications before recommending its implementation The high correlation found with maintainability indicators are encouraging enough to suggest the usage of at least one information flow metric for MDSF.

5.7. Information Flow by Kitchenham et al (KPL-IF)

Description: A measure of intermodule complexity inspired from Henry & Kafura's information flow metric. Since Kitchenham *et al* experienced some difficulties in understanding the definition of flows provided by Henry & Kafura, they formulated a new set of definitions.

Author: B.A. Kitchenham, L.M. Pickard, S.J. Linkman

Sources: [KPL90]

Criterion Measured: cohesion, coupling

Product Measured: SDD, source code

Formula: $IFC_m = (IFI_m * IFO_m)^2$

IFC_m = information flow complexity for module *m*

IFI_m = number of modules calling module *m* +
number of global data structures read by module *m*

IFO_m = number of modules called by module *m* +
number of output parameters of module *m* +
number of global data structures updated by module *m*

Algorithm: Implicit from formula.

Interpretation:

Maintainability should decrease as information flow complexity increases.

Validity:

The metric was validated using a communications system. Very low correlation were found with faults (.07), changes (.06) and a subjective evaluation of complexity (.12). Using the contingency correlation method, where values are grouped into classes, slightly better but still very low correlations were observed (.31 with faults, .35 with changes and .25 with a subjective evaluation of complexity). IFO alone was performing better than IFC.

Cost of Buying or Developing:

Computation could be automated for source code. No tool available. No data available on developing effort.

Computation Effort:

SDD: manual with the current format. Source code: automatic. No data available on manual computation effort.

Recommendation: Do not implement the metric.

The metric is simple but experimental results show poor performance.

5.8. IF4

Description: A measure of intermodule complexity based on information flow.

Author: M. Shepperd

Sources: [Shep90a], [Shep90b], [SI90], [IS89]

Criterion Measured: cohesion, coupling

Product Measured: SDD, source code

Formula: $IF4_m = (\text{fan-in}_m * \text{fan-out}_m)^2$

$IF4_m$ = information flow complexity of module *m*

fan-in_m = number of *information flows* terminating at module *m*

fan-out_m = number of *information flows* emanating from module *m*

Algorithm:

The definition of information flow is not the same in two publications of the author for reused module [IS89], [Shep90a]. We chose the definition of [Shep90a] because it is more explicit than [IS89]. There is an *information flow* from module A to module B if either of the following three cases occurs:

a) A calls B and passes it an input value

b) B calls A and A returns an output value to B.

c) A updates a global data structure which is read by B

Duplicate flows are not counted. A global data structure can be a global variable or a file.

Interpretation:

Maintainability should decrease as information flow complexity increases.

Validity:

The author has conducted two experiments: one with a game adventure shell developed by second year B. Sc. students, and one industrial real-time aerospace application. In the former, correlation was established with computer connect time. As noted by the author, connect time does not capture other aspect of development time like design, debugging, etc. IF4 showed a correlation of .80 with connect time. In the latter experiment, the author established a correlation with a subjective evaluation of maintainability for 89 modules provided by four experts. A correlation of .70 was obtained.

Cost of Buying or Developing:

Computation could be automated for source code. No tool available. No data available on developing effort.

Computation Effort:

SDD: manual with the current format. Source code: automatic. No data available on manual computation effort.

Recommendation: Do not implement the metric.

Experimental results are encouraging but there is still a lack of validation data with industrial applications.

5.9. Design Complexity of Card & Agresti (CA-DC)

Description: A measure of intermodule and intramodule complexity of a system based on fan-out, number of modules and input/output variables

Author: D. N. Card, W.W. Agresti

Sources: [CA88]

Criterion Measured: cohesion, coupling

Product Measured: SDD, source code

Formula: CA-DC = S + L

S = intermodule complexity (structural complexity)

S = (sum of f_m^2)/n

L = intramodule complexity (local complexity)

L = (sum of v_m /(f_m+1))/n

f_m = fan out = number of calls to other module in module *m*

v_m = number of input/output variables (global variable or argument) in module *m*.

n = number of modules in the system

Algorithm: Unreferenced global variables are not counted in v_m .

Interpretation:

Maintainability should decrease as complexity increases.

Validity:

The only source of empirical validations is from the authors [CA88]. The study involved 7 projects at the Software Engineering Laboratory with an average number of modules of 265 and average number of lines of code of 64 000. The following correlation levels were observed: LDE (-.49), HFC (.83). The behaviour of the measure raises some concerns regarding its theoretical validity. A design favouring the usage of global variables can be less complex than an equivalent one using explicit module argument. This is in contradiction with the widely agreed design principle of coupling [Myer76]. This theoretical limitation is not specific to CA-DC. The same criticism applies to IF4. Also, because fan-in was not affecting their experimental data, it is not included in the metric. This is consistent with [KPL90].

Cost of Buying or Developing:

Computation could be automated for source code. No tool available. No data available on developing effort.

Computation Effort:

SDD: manual with the current format. Source code: automatic. No data available on manual computation effort.

Recommendation: Implement the metric.

The metric is simple. Of the five information flow metrics, CA-DC was validated with the largest industrial experiment, showing good correlation with the number of faults.

5.10. Cocomo Inspired Metric (COCO)

Description: A selection of appropriate adjustment factors of the intermediate Cocomo metric.

Author: B. W. Boehm

Sources: [Boeh81]

Criterion Measured: N/A

Product Measured: SDD, source code

Formula:

In intermediate Cocomo, the maintenance effort is estimated by multiplying a nominal effort by an effort adjustment factor. The nominal effort is estimated o the basis of the size of the software and the yearly ratio of modified software. The effort adjustment effort is estimated in a two step process: first a rating is assigned to each of fifteen cost factors of intermediate assigned to each of fifteen cost factors of intermediate cocomo; then a numerical value is associated to each factor according to its rating. The rationale of this metric is to estimate the effort adjustment factor associated to the maintenance of the given product; we drop the nominal effort from consideration because it is based on two factors that are difficult to estimate at the PDR step: the size of the final product, and the annual change traffic of the product in the operations and maintenance phase.

Algorithm:

The effort adjustment factor is the product of fifteen factors, which are divided into four categories: product attributes, computer attributes, personnel attributes, and project attributes. Given that we are dealing with a design document, only the first category is significant. In this category, we have the following factors: RELY, DATA and CPLX. ratings can be assigned to these factors using table 8-3 of boehm's book (page 119). for RELY, the rating can be converted to a numerical value using table 8-7; for the other two factors, use table 8-2. The product of these three factors is the desired metric. It is in the neighborhood of 1.

Interpretation:

If the value is greater than 1, we predict a high cost of maintenance; if it is smaller, we predict a low cost.

Validity:

Cocomo is one of the most often cited metric for effort estimation. It has been developed after analysis of a database of 60 industrial projects

Cost of Buying or Developing:

No tool available.

Computation Effort:

The computation is manual and the estimation of factors cannot be automated. No data available on cost of computation.

Recommendation: Implement the metric.

The metric is widely used. It gives a rough estimate of the software maintenance cost (low, medium, high).

5.11. Cyclomatic Complexity Number (v(G))

Description: The number of independent basic paths in a program.

Author: T.J. McCabe

Sources: [McCa76], [CDS86], [IEEE91]

Criterion Measured: Control Structure

Product Measured: Source Code

Formula: $v(G) = e - n + 2$ or, equivalently, $v(G) = DE + 1$

e = number of edges in the control graph

n = number of nodes in the control graph

DE = number of predicates

Algorithm:

For the first formula, compute the control graph (or flow chart) of a program. Nodes are program statements. Edges represent the control flow between statements. The control flow is determined by the conditional statements (IF, WHILE, CASE, etc). When a conditional statement is made of a compound predicate (i.e. a condition with some occurrence of the logical operators AND and OR), it is decomposed into simple conditional statements. For instance, the statement IF c1 AND c2 THEN is decomposed into two IF statements: IF c1 THEN IF c2 THEN. c1 and c2 are examples of predicates. Note that some authors have proposed alternative calculations for the number of predicates ([Myer77] and [Hans78]).

For the second formula, there is no need to compute the graph. The algorithm is to count the number of predicates.

Interpretation:

Maintainability should decrease has v(G) increases. McCabe suggested 10 as the ideal maximum value for a module. However, this value seems arbitrary to some authors ([BP84], [Ejio91]). [LV89] noted that the change density was minimum for modules with v(G) between 10 and 15.

Validity:

This metric has been well studied in the literature. The following is a list of correlation levels found:

[Romb84] MIE (.56), LNB (.40)

[BK85] HIE (.81)

[BSP83] LIE to MIE (depending on the analysis: by programmer, project or overall)

LFC to MFC (idem)

LDE to MDE (idem)

[KH81] HFC (.96)

[LV89]MFC to HFC (.68 random sample, .72 for selected system features)

[DL88]MIE (.53 debugging time, .66 construction time)

HFC (.79)

[BP84], surprisingly, shows that the number of faults per module *was decreasing* when v(G) and LOC (see description of LOC) were increasing. This is in contradiction with the interpretation just given. These results are tentatively explained by the fact that larger modules are coded with greater care, and faults are more apparent in smaller module (there may be still be numerous undetected faults in larger modules). [LV89] obtained similar results: the number of faults and changes decreases up to a certain point, and then increases as v(G) increases.

Cost of Buying or Developing:

Datrix - 2500 $

SAP - 3125 $

Metric - 4500 $

PC-Metrics - 400 $

Computation Effort: The computation is automatic.

Recommendation: Implement the metric.

The metric has been extensively validated and correlation with indicators is usually good.

5.12. Knots

Description: The number of crossing lines (unstructured goto statements) in a control flow graph.

Author: M.R. Woodward, M.A. Hennell, D. Hedley

Sources: [WHH79], [CDS86]

Criterion Measured: Control Structure

Product Measured: Source Code

Formula: number of knots

Algorithm:

Define an *ordering* on the lexical elements of a program based on their sequential position from the start of the program. In a language allowing only one statement per line, this ordering could be the line number. A *jump* from position a to position b is represented by the pair (a, b). Jump (a, b) gives rise to a *knot* with respect to jump (p, q) if

$\min(a, b) < \min(p, q) < \max(a, b) < \max(p, q)$

or

$\min(p, q) < \min(a, b) < \max(p, q) < \max(a, b)$

Interpretation:

Maintainability should decrease has the number of knots increases.

Validity:

The following is a list of correlation levels found:

[BK85] HIE (.80)

Cost of Buying or Developing: Datrix - 2500 $

Computation Effort: The computation is automatic.

Recommendation: Implement the metric.

The metric has been validated and correlation with indicators is good.

5.13. Relative Logical Complexity (RLC)

Description: The number of binary decisions divided by the number of statements

Author: T. Gilb

Sources: [Gilb77]

Criterion Measured: Control Structure and Size

Product Measured: Source Code

Formula: v(G) / LOC

Algorithm: See v(G) and LOC

Interpretation:

Maintainability should decrease has relative logical complexity increases.

Validity:

The following is a list of correlation levels found:

[BK85] HIE (.93)

[GK91] MPP(.59)

Cost of Buying or Developing:

SAP - 3125 $, Metric - 4500 $, PC-Metrics - 400 $

Computation Effort: The computation is automatic.

Recommendation: Do not implement the metric.

The metric is dependent on v(G) and LOC which are recommended for implementation. Unmaintainable modules detected by RLC should also be detected by v(G) or LOC, whereas an abnormally high value of v(G) and LOC may result in a normal ratio for RLC. Therefore, v(G) and LOC should be sufficient.

5.14. Comments Volume of Declarations (Vcd)

Description: Total number of characters found in the comments of the declaration section of a module. The declaration section comprises comments before the module heading up to the first executable statement of the module body.

Author: N/A

Sources: [RCC91], [Datrix]

Criterion Measured: readability

Product Measured: source code

Formula: N/A

Algorithm: Blanks, ":" and "." are omitted in the count.

Interpretation:

A module with a low Vcd or high Vcd is considered less readable. The latter can be an indication of a lack of focus. The proposed acceptable range in [RCC91] is 150-2000.

Validity:

[WDS81] shows that comments judiciously located in a module declaration improve comprehensibility. [SBC78] indicates that comments do not harm, but may not help. [Jorg80] found that the number of comments is a factor influencing readability, along with the extensiveness of blanks in the left margin, the number of blanks lines and the average length of variable names.

Cost of Buying or Developing: Datrix 2500 $

Computation Effort: Automatic

Recommendation: Implement the metric.

Experiments show the importance of comments in module understanding.

5.15. Comments Volume of Structures (Vcs)

Description: Total number of characters in the comments found anywhere in the module except in the declaration section. The declaration section comprises comments before the module heading up to the first executable statement of the module body.

Author: N/A

Sources: [RCC91], [Datrix]

Criterion Measured: readability

Product Measured: source code

Formula: N/A

Algorithm: Blanks, ":" and "." are omitted in the count.

Interpretation:

A module with a low Vcs or high Vcs is considered less readable. The latter can be an indication of a lack of focus. The proposed acceptable range in [RCC91] is 150-2000.

Validity:

See Vcd

Cost of Buying or Developing: Datrix 2500 $

Computation Effort: Automatic

Recommendation: Implement the metric.

Experiments show the importance of comments in module understanding.

5.16. Average Length of Variable Names (Ls)

Description: Mean number of characters of all variables used in a module. Unused declared variables are not included.

Author: N/A

Sources: [Jorg80], [Datrix]

Criterion Measured: readability

Product Measured: source code

Formula: N/A

Algorithm: Compute the mean number of characters of all variables used in a module. Unused declared variables are not included. Each variable is counted only once.

Interpretation:

The longer the variable names are, the easier the module is to maintain.

Validity: See Vcd. The acceptable range suggested in RCC91 is 5 to 10.

Cost of Buying or Developing: Datrix - 2500 $

Computation Effort: Automatic

Recommendation: Implement the metric.

Experiment of [Jorg80] shows the importance of variable name length in module understanding.

5.17. Lines of Code (LOC)

Description: The number of lines in the source code excluding blank lines or comment lines.

Author: N/A

Sources: [CDS86]

Criterion Measured: Size

Product Measured: Source Code

Formula: N/A

Algorithm:

Count the number of lines in the source code while excluding blank lines or lines containing comments only.

Interpretation:

Maintainability should decrease as LOC increases. No agreed upon upper bounds for a module. [LV89] noted that the change density was minimum for module with LOC between 100 and 150.

Validity:

[Romb84] HIE (.70), LNB (.45)

[DL88]MIE (.51), HDE (.72), MFC (.65)

[LV89]MFC (.68 random sample, .64 selected system features)

[BSP83] LIE to MIE (depending on the analysis: by programmer, project or overall)

LFC to MFC (idem), LDE to MDE (idem)

Cost of Buying or Developing: PC-Metrics - 400 $, SAP - 3125 $

Computation Effort: Computation is automatic.

Recommendation: Implement the metric.

LOC is a simple measure which everyone can understand. It shows good correlation with indicators.

5.18. Software Science Effort (E)

Description: An estimation of programming effort based on the number of operators and operands. It is a combination of other Software Science metrics.

Author: M.H. Halstead

Sources: [Hals77], [SCD83], [CDS86], [IEEE91]

Criterion Measured: Size

Product Measured: Source Code

Formula:

Effort	$E = VD$
Difficulty	$D = (n1/2)(N2/n2)$
Volume	$V = N\log_2 n$
Observed Length	$N = N1 + N2$
Vocabulary	$n = n1 + n2$

n1 = number of distinct operators (Ex: +, >, :=, IF, WHILE)

n2 = number of distinct operands (i.e. a variable, a label or a constant)

N1 = total occurrences of operators

N2 = total occurrences of operands

Algorithm:

Scan source code to compute n1, n2, N1 and N2. Compute other metrics.

Interpretation:

Maintainability should decrease as the effort increases. No desirable upper bound suggested in the literature.

Validity:

This metric has been well studied in the literature. The following is a list of correlation levels found:

[BK85] HIE (.76)

[DL88]HIE (0.71), HDE (.74), HFC (.82)

[BSP83] LIE to MIE (depending on the analysis: by programmer, project or overall)

LFC to MFC (idem)

LDE to MDE (idem)

Cost of Buying or Developing:

SAP - 3125 $

METRIC - 4500 $

PC-METRIC - 400 $

Computation Effort: The computation is automatic.

Recommendation: Implement the metric.

E is very similar to LOC in that it is based on a count of every statement composing a program. Because it is insensitive to formatting, one could conjecture that it should perform better than LOC. However, experiments are not conclusive, E performing better than LOC in some cases, worse in others. Since it shows good correlation with indicators, we recommend it.

5.19. Documentation Accuracy Ratio (DAR)

Description: A verification of the accuracy of the CEI Spec, RS and SDD with respect to the source code.

Author: M. Frappier

Sources: N/A

Criterion Measured: Documentation accuracy

Product Measured: CEI Spec, RS, SDD, Source Code

Formula: DAR = (sum of d_m)/n

d_m = documentation accuracy level for module *m*

n = number of modules reviewed

Algorithm:

Select a random sample of modules. For each module, compute the documentation accuracy level (d_m) according to the following procedure:

1) set d_m to zero;

2) if the module's function as implemented in the source code corresponds to the requirements defined in the CEI Spec following the traceability index, add .2 to d_m;

3) if the module's function as implemented in the source code corresponds to the requirements defined in the RS following the traceability index, add .2 to d_m;

4) if the module's inputs as implemented in the source code corresponds to the module's inputs specified in the SDD, add .2 to d_m;

5) if the module's algorithm as implemented in the source code corresponds to the module's algorithm specified in the SDD, add .2 to d_m; the SDD should be sufficiently detailed to allow a programmer to write the existing code from it.

6) if the module's outputs as implemented in the source code corresponds to the module's outputs specified in the SDD, add .2 to d_m;

The specification and design documents of MDSF are mostly informal. Therefore, this metric has to be computed by experts. Step 4 and 6 are objective, but step 2, 3 and 5 may be evaluated differently depending on the reviewer's familiarity with the system. Ultimately, to provide an unbiased evaluation, the review should be performed by programmers unfamiliar with the system.

Interpretation:

Maintainability should increase with DAR. DAR's value ranges between 0 and 1.

Validity:

No experimental data available.

Cost of Buying or Developing:

Computation is done manually by a group of experts to increase objectivity.

Computation Effort:

No data available.

Recommendation: Implement the metric.

If a system's documentation is to be of any use, it has to be accurate. Moreover, the functionality of MDSF is extremely specialised. It cannot be expected from ordinary programmers and software engineers to be familiar with the concepts implemented in MDSF.

5.20. Source Code Consistency (SCC)

Description: The extent to which the source code contains uniform notation, terminology and symbology within itself.

Author: Derived from Boehm *et al.*

Sources: [BBKL78]

Criterion Measured: Consistency

Product Measured: Source Code

Formula: SCC = (sum of c_i)/n

c_i =number of inconsistencies for item *i*

n = number of modules reviewed

Algorithm:

Select a set of modules which are related in some manner, for instance the modules are sharing global data structures, are part of the same calling hierarchy, are computing similar functions, or are using similar data. Compute the number of inconsistencies between the modules for each item (c_i). The list of item to check is:

1) identical common area definition: Count the number of common area defintion(COMMON statement in FORTRAN) which differs among the modules.

Example:
```
sub1  COMMON /C1/A,B,C
sub2  COMMON /C1/D,E,F
sub3 COMMON /C1/A(2),TEMP(3)
```

2) consistent type definition: Count the number of variables with a different type definitions.

Example:
```
sub1 real*4   A
sub2 real*8   A
```

3) consistent constant representation: Count the number of constants with different definitions.

Example:
```
sub1 PI = 3.14159
sub2 PI = 3.1416
```

4) same name different entities: Count the number of variables which represent different entities in different modules.

Example:
```
sub1 PI = 3.14159
sub2 PI = point of inertia
```

5) different names same entities: Count the number of variables which represent different entities in different modules.

Example: sub1 VOLUME
sub2 SIZE
sub3 NUMBER

6) similar format of expressions: Count the number of similar expressions differently constructed.

Example: sub1 Y = X**3
sub2 Y = X*X*X

7) consistent comment format: Count the number of modules with different format of comments (mainly the header comment describing the function of the module, its restrictions, inputs, outputs, etc)

Interpretation:

Maintainability should decrease with SCC. SCC's value is greater or equal to zero.

Validity:

The study reported [BBKL78] shows a correlation with a subjective evaluation of maintainability.

Cost of Buying or Developing:

Automated support could be provided for items 1, 2 and 3. The others must be done manually by an expert

Computation Effort:

No data available.

Recommendation: Implement the metric.

6. Acronyms

CA-DC	Design Complexity of Card & Agresti
CEI Spec	Contract End Item Specification
COCO	Cocomo Inspired Metric
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DAR	Documentation Accuracy Ratio
DE	development effort per module
E	Effort metric of Halstead
FC	number of faults or modifications per module
HK-IF	Information Flow Complexity of Henry & Kafura
IE	Internal Effort per module (maintainability indicator)
IF4	Information Flow Complexity of Shepperd
KPL-IF	Information Flow Complexity of Kitchenham *et al*
LOC	Number of non-comment source code lines
Ls	Average Length of Variable Names
M-MC	Module Coupling of Myers
M-MS	Module Strength of Myers
MDSF	Manipulator Development Simulation Facility
NB	average number of modules affected by a change (maintainability indicator)
NR	Non-referencing Item
PP	maintenance productivity by project (maintainability indicator)
R-IF	Integrated Information Flow Complexity of Rombach

RLC	Relative Logical Complexity of Gilb
RS	Requirement Specification
SDD	Software Design Document
SE	subjective evaluation
UR	Unreferenced Requirements
v(G)	Cyclomatic Complexity Number of McCabe
Vcd	Comments Volume of Declaration
Vcs	Comments Volume of Structure
XE	External Effort per module (maintainability indicator)

7. References

[Barr80] Barry, J.G. "Computerized Readability Levels", *IEEE Transactions on Professional Communication*, 23(2), June 1980, pp. 88-90.

[BBKL78] Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., Merrit, M.J. *Characteristics of Software Quality*, TRW series of software technology, North-Holland, 1978.

[BK85] Blaine, J.D., Kemmerer, R.A. "Complexity Measures for Assembly Language Programs", *JSS*, 5, 1985, pp. 229-245.

[BO85] Birrell, N.D., Ould, M.A. *A Practical Handbook for Software Development*, Cambridge, 1985.

[Bohm81] Boehm, B.W. *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[BP84] V. Basili and B.T. Perricone. "Software Errors and Complexity: An Empirical Investigation", *CACM*, 27(1), January 1984.

[BSP83] Basili, V.R., Selby, R.W., Phillips, T-Y. "Metric Analysis and Data Validation Across FORTRAN Projects", *IEEE TSE*, 9(6), Nov. 83, pp. 652-663.

[CA88] Card, D.N., Agresti, W.W. "Measuring Software Design Complexity", *JSS*, 8(3), 1988, pp. 185-197.

[CDS86] Conte, S.D., Dunsmore, H.E., Shen, V.Y., Software Engineering Metrics and Models, Benjamin/Cummings Publishing Company, Menlo Park, 1986 ISBN: 0-8053-2162-4

[Datrix] *DATRIX User's Guide*, Schemacode International, Dollard-des-Ormaux, Quebec, Canada, 1990.

[DC48] Dale, E., Chall, J.S. "A Formula for Predicting Readability", *Educational Research Bulletin*, 27, Feb. 1948, pp. 221-233.

[DL88] Davis, J.S., LeBlanc, R.J. "A Study of the Applicability of Complexity Measures", *IEEE TSE*, 14(9), 1988, pp.1366-1372.

[Ejio91] Ejiogu, L.O., *Software Engineering with Formal Metrics*, QED Technical Publishing Group, 1991.

[Gilb77] Gilb, T. *Software Metrics*, Cambridge, MA., Winthrop Publishers, 1977.

[GK91] Gill, G.K., Kemerer, C.F. "Cyclomatic Complexity Density and Software Maintenance Productivity", *IEEE TSE*, 17(12), Dec. 1991, pp. 1284-1288.

[GS89] Gibson, V.R., Senn, J.A. "System Structure and Software Maintenance Performance", *CACM*, 32(3), March 1989, pp. 347-358.

[Hals77] M.H. Halstead. *Elements of Software Science*. New York, Elsevier North-Holland, 1977.

[Hans78] Hansen, W.J. "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)", *ACM SIGPLAN Notices*, 13(3), Mar 1978, pp. 29-33.

[Hans78] Hansen, W.J. "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)", *ACM SIGPLAN Notices*, 13(3), March 1978, pp. 29-33.

[Henn80] Henninger, K. "Specifying Software Requirements for Complex Systems: New Techniques and their Application", *IEEE TSE*, 6(1), Jan., 1980, pp. 1-14.

[HK81] Henry, S., Kafura, D. "Software structure metrics based on information flow", *IEEE TSE*, 7(5), Sept 81, pp. 510-518

[HS90] Henry, S., Selig, C. "Predicting Source-Code Complexity at the Design Stage", *IEEE Software*, Marc 1990, pp. 36-44.

[IEEE91] IEEE, *Software Engineering Standards Collection*. The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York. IEEE Standards Board, 1991.

[IS89] Ince, D., Shepperd, M. "An Empirical and Theoretical Analysis of an Information Flow-Based System Design Metric", in *Lecture Notes in Computer Science no. 387: Proceedings. 2nd European Software Engineering Conference*, Springer-Verlag, 1989, pp. 86-99.

[Jorg80] Jorgensen, A.H., "A Methodology for Measuring the Readability and Modifiability of Computer Programs", *BIT*, 20, 1980, pp. 394-405.

[KAOC81] Kincaid, J.P., Aagard, J.A., O'Hara, J.W., Cottrell, L.K. "Computer Readability Editing System", *IEEE Trans. on Professional Communication*, 24(1), March 1981, pp. 38-41.

[KC85] Kafura, D. Canning, D. "A Validation of Software Metrics Using Many Metrics and Two Resources", *ICSE-8*, 1985, pp. 378-385.

[KH81] Kafura, D., Henry, S., "Software Quality Metrics Based on Interconnectivity", *JSS*, 2, 1981, pp. 121-131.

[KPL90] Kitchenham, B.A., Pickard, L.M., Linkman, S.G. "An Evaluation of Some Design Metrics", *SEJ*, 5(1), 1990, pp. 50-58.

[LV89] Lind, R.K., Vairavan, K. "An Experimental Investigation of Software Metrics and Their Relationship to Software Development Effort", *IEEE TSE*, 15(5), May 1989, pp. 649-653.

[McCa76] McCabe, T.J., "A Complexity Measure", *IEEE TSE*, 2(4), Dec. 1976, pp. 308-320.

[Myer75] Myers, G.J. *Reliable Software Through Composite Design*, New York, Van Nostrand Reinhold, 1975.

[Myer77] Myers, G.J. "An Extension to the Cyclomatic Measure of Program Complexity", *ACM SIGPLAN Notices*, 12(10), Oct. 1977, pp. 61-64.

[RCC91] Robillard, P.N., Coupal, D., Coallier, F. "Profiling Software Through the Use of Metrics", *Software - Practice and Experience*, 21(5), May 1991, pp. 507-518.

[Romb84] Rombach, H.D. "Design Metrics for Maintenance", *Proc. 9th Annual Software Engineering Workshop*, NASA, Goddard Space Flight Centre, Greenbelt, Maryland, Nov. 1984, pp. 100-121.

[SBC78] Sheppard, S.B., Borst, M.A., Curtis, B. "Predicting Programmers' Ability to Understand and Modify Software", *Symposium Proceedings: Human Factors and Computer Science*, Washington D.C., June 1978, pp. 115-135.

[SCD83] Shen, V.Y., Conte, S.D., Dunsmore, H.E. " Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support, *IEEE TSE*, 9(2), March 1983.

[Shep90a] Shepperd, M. "Early Life-Cycle Metrics and Software Quality Models", *Information and Software Technology*, 32(4), May 1990, pp. 311-316.

[Shep90b] Shepperd, M. "Design Metrics: An Empirical Analysis", *SEJ*, 5(1), 1990, pp. 3-10.

[Shep92] Shepperd, M. "Measurement of Structure and Size of Software Designs", *Information and Software Technology*, 34(11), Nov. 1992, pp. 756-762.

[SI90] Shepperd, M., Ince, D. " Controlling Software Maintainability", in *Proceedings. Second European Conference on Software Quality Assurance*, Oslo, Norway, May 1990.

[Sieg55] Siegel, S. *Nonparametric Statistics for the Behavioral Sciences*, New York, McGraw-Hill, 1955.

[WDS81] Woodfield, S.N., Dunsmore, H.E., Shen, V.Y. "The Effect of Modularization and Comments on Program Comprehension", *ICSE-5*, 1981, pp. 215-223.

[WHH79] Woodward, M.R., Hedlay, D., Hennell, M.A. "A Measure of Control Flow Complexity in Program Text", *IEEE TSE*, 5(1), 1979, pp. 45-50.

