NISTIR 8289

Quantities and Units for Software Product Measurements

David Flater

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8289

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 8289

Quantities and Units for Software Product Measurements

David Flater
Software and Systems Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8289

March 2020

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Interagency or Internal Report 8289
Natl. Inst. Stand. Technol. Interag. Intern. Rep. 8289, 82 pages (March 2020)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8289

Abstract

International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 80000, the International System of Quantities, collects and organizes the most important physical quantities into a coherent system. In a similar fashion, this report collects and organizes the most important quantities used in software metrics, focusing on software as a product rather than its development process.

Key words

Measurement; metrics; quantities; software; units.

Table of Contents

0 Measurement concepts

0.1 Normative references

The following referenced documents are taken to be canonical for the established system of metrology:

- "The SI Brochure:" International Bureau of Weights and Measures (Bureau International des Poids et Mesures, BIPM). The International System of Units (Le Système international d'unités, SI), 9th edition, 2019. http://www.bipm.org/en/publications/si-brochure/

- "The VIM:" Joint Committee for Guides in Metrology (JCGM). International vocabulary of metrology (Vocabulaire international de métrologie, VIM)—Basic and general concepts and associated terms, 3rd edition. JCGM 200:2012. http://www.bipm.org/en/publications/guides/vim.html

- "The GUM:" Joint Committee for Guides in Metrology. Evaluation of measurement data—Guide to the expression of uncertainty in measurement (GUM). JCGM 100:2008. http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf

- "The International System of Quantities (ISQ):" International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC) 80000, Quantities and units.

0.2 Basic terms

The following terms are defined by the 3rd edition of the VIM [VIM]:

> **quantity:** property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference.

[The "reference" is typically an expression in terms of SI units.]

> **quantity value:** number and reference together expressing magnitude of a quantity. Example 1: Length of a given rod: 5.34 m or 534 cm.
>
> **measured quantity value:** quantity value representing a measurement result.
>
> **measurement result:** set of quantity values being attributed to a measurand together with any other available relevant information.

[The "set of quantity values" is intended to accommodate uncertainty, given that a single true quantity value generally cannot be determined.]

In common software jargon (and thus, within this document), the term "metric" is used in a broad manner to include not just measurable quantities in the strict sense but calculated values and assigned classifications of any sort that are treated as measurement results. Similarly, "measure" used as a noun can mean measurement (as in the idiomatic "unit of measure"), metric, or unit (as in "measure of X"), depending on context.

0.3 Extended units

A quantity in the SI can be stated as a mathematical expression—the product of a numerical value and a unit of measurement. The magnitude of a quantity can be expressed in terms of the seven SI traditional base quantities length (m), mass (kg), time (s), electric current (A), thermodynamic temperature (K), amount of substance (mol), and luminous intensity (cd), either individually or in combinations. These quantities correspond to physical dimensions as used in dimensional analysis.

However, many kinds of quantities have no extent in any of the seven standard dimensions. For example, a *counted quantity* is a number of some distinguishable kind of thing, such as 32 bits. Unfortunately for computer science, "amount of data" is not an SI dimension, and bits and bytes are not SI units. Ratios of two quantities of the same kind, such as mass fractions (kg/kg), are a similar major category of quantities. The SI Brochure regards both of these categories of quantities as dimensionless.

In the SI, the unit of measurement for dimensionless quantities is the special unit one. Depending on context, it may be regarded as the derived unit that algebraically results from setting the exponents on all seven of the SI traditional base quantities to zero, or it may instead be regarded as a further base unit that is common to all measurement systems [VIM, SI]. A suggestion that it would be clearer to refer to *dimension number*, with Z as its symbol and 1 as its coherent unit of measurement [Krystek], has become popular.

To avoid user surprise at the canonical SI treatment of amounts of data and other dimensionless quantities, software libraries and packages that implement quantities and units functions often apply workarounds such as adding an explicit base unit for 1, adding many non-SI dimensions, and allowing users to introduce arbitrary irreducible units (effective extra dimensions). Different software has applied different workarounds, creating subtle problems for transfer of scientific data.

In this document, we follow a model that extends the interpretation of dimensionless quantities by subtyping the special unit one with "extended units." For a complete discussion of this model, related work, and alternative approaches, please see Ref. [Flater].

0.4 "Amount of data" as a dimension

For most counted quantities, there is only one obvious unit to use (the counted entity or event), and the question of dimension is obviated by the type system for dimensionless quantities that was mentioned above. However, the coexistence of multiple "natural units" of data (bits, bytes, and occasionally words) means that often it is less misleading to cite *dimension data* than to specify any such unit.

In physical metrology, the National Institute of Standards and Technology (NIST) has proposed that angle be included as an SI dimension with the radian as its coherent unit of measure and the cycle as a non-coherent unit that is equal to 2π radians. In parallel fashion, we sometimes find it convenient to think of data as an added dimension with the bit as its coherent unit of measure and the byte as a non-coherent unit that is equal to 8 bits. The fact that amounts of data are counted quantities that cannot be subdivided indefinitely is not always an important factor; the same is true of amounts of substance in the SI.

Stating that a quantity has the dimension of data, rather than the units of bits or bytes, makes it clear that the choice of unit is not an essential part of the definition of the quantity. One can use other counting units, such as data structures of a particular type that can be reduced to a count of bits, without losing traceability.

Dimensions that proved useful in describing software quantities are provided in Sec. 3. They are: time, data, information, and work. Other "effective" dimensions corresponding to the many kinds of nonphysical quantities that are described in this document can easily be posited; however, in most cases, there is nothing to be gained by doing so. The units and/or scale of the result provide complete information.

0.5 Traceability

[SI, Sec. 2.3.3] states that counts are traceable to the SI via the special unit one and "appropriate, validated measurement procedures." However, in general, counting involves characterizing what is being counted (say, lines of code), and this characterization involves a standard (definition of line of code) that is not part of the SI. Therefore, the task of defining most extended units falls on the downstream users of the SI.

Traceability is complicated further when some kind of count is used as a surrogate measure of another kind of quantity. For example, the durations of some software processes may be expressed in Central Processing Unit (CPU) cycles rather than in seconds. A given number of CPU cycles translates to a variable number of seconds because the CPU frequency varies. Analogously, a program may transfer a fixed number of data entities of a given type, but if these entities vary in size, then the number of bits transferred will vary.

Practitioners may find it expedient to use the non-traceable units because the resulting expression apparently is more precise. However, sacrificing traceability means that the actual duration of the process (in seconds) or the actual amount of data transferred (in bits) has not been quantified. A process that took more CPU cycles to run may actually have taken *less* time. This may or may not be an important consideration, depending on the use of the measured quantity values, but it is anathema to theories of measurement that seek to relate all quantities to real, independent, objective properties.

0.6 Scales

Scale theory is a small part of the broad discipline of measurement theory. It was popularized in 1946 by Stanley S. Stevens [Stevens] and subsequently extended, formalized, reinterpreted, and criticized by many others.

The following scales suffice for the purposes of this document. N.B., These traditional scale definitions are incompatible with the formal definitions used in [Zuse].

- A dichotomic scale has only two values, typically named yes/no or true/false, which have no particular ordering (thus "scale" is a misnomer).
- Nominal is the "scale" of measurements that assign identifiers that have no particular ordering. A "nominal set," in which the identifiers are not mutually exclusive, is reducible to a set of independent dichotomic measures.
- Ordinal is the scale of measurements that assign numbers in a fashion that preserves relative ordering but nothing more. There is no unit of measurement for results on an ordinal scale because the magnitudes have no meaning beyond relative ordering.
- Interval is the scale of measurements that have a meaningful unit but not a meaningful zero point. For example, temperature can be measured on either the Celsius or Fahrenheit scales, but their zero points are different and do not correspond to a physical minimum.
- Ratio is the scale on which physical measurements of length, mass, time, etc. are made. Since they have a meaningful zero point, quantities on a ratio scale can be transformed by a simple multiplicative scaling factor without losing information; e.g., 5 m is the same quantity as 500 cm.

In this document, a derived number for which no unit or representational structure is obvious is deemed ordinal, even though the metric may not even preserve a relative ordering by the measurand. When a unit is derivable but not consistent with how the metric is used, the scale may also be deemed ordinal.

1 Guide to the system

1.1 Scope, goals, and non-goals

The word "software" covers a lot of territory, either directly or indirectly. A notional software attribute can be different things depending on which artifact or process serves as the object of measurement. Our objects of measurement include the following:

- Architecture, design
- Requirements
- Specification
- Algorithm
- Implementation (source code or script)

- Executable (binary or bytecode)
- Execution (of binary, bytecode, or script)

The purpose of this document is to provide a systematic reference for metrics that are in use. Non-goals of this document include:

- Providing a tutorial on computer science, software measurement, or metrology;
- Creating another self-consistent vocabulary for computer science, software measurement, or metrology, to compete with [ISO-Vocab], [VIM], and similar standards;
- Creating a complete catalog of every software metric that was ever used;
- Explaining the use or usefulness of individual metrics;
- Evaluating the meaningfulness, validity, or formal properties of individual metrics;
- Explaining or mitigating the deficiencies of individual metrics beyond what is necessary to integrate them into the system without confusing the reader;
- "Picking winners" among competing candidates for metrics of a given type.

We prioritize practical usability over rigid consistency and eschew organizing principles that would require commonly associated metrics to be separated from one another in the document.

1.2 Criteria for inclusion

The determination of which metrics to include was made according to the following criteria:

1. Species: the ostensible metric must actually be a metric and not merely a framework, model, methodology, method, architecture, paradigm, foundation, or theory.

2. Scope: the metric must apply to a software artifact, i.e., one of the objects listed in Sec. 1.1, and it must measure some objective property. Metrics that resemble customer satisfaction surveys are excluded. Edge case: the security metrics in Ch. 10.

3. Clarity: the metric and its result must not depend on models or concepts that are too abstruse to summarize reasonably in this document.

4. Completeness: the metric must not depend on made-up numbers (e.g., arbitrary weighting factors, typically denoted by w_i), unspecified threshold values, or quantities for which no reliable measurement method is known (e.g., number of incorrect requirements, total number of possible use cases). Edge case: the combinatorial coverage metrics in Sec. 9.2 depend on the selection of a finite set of valid values for variables.

5. Notability: there must exist references to the metric by sources other than the originating author or organization. Large, complicated metrics that are difficult to incorporate need correspondingly stronger evidence of notability. Edge case: some obscure metrics have been referenced in surveys that are more notable and available than the original sources. Such references beget more references in later surveys

even if the metrics never had any practical application, as each survey aims for completeness.

6. Availability: the definition of the metric must be published and obtainable with a web search or an interlibrary loan, or it must appear in an international standard. Edge case: [DO-178C] is an important reference but is not widely published.
7. Not a hardware reliability metric: generic systems reliability metrics like mean time to failure (MTTF) that sometimes are applied to software are adequately covered by other standards. A sample of such measures is provided in Institute of Electrical and Electronics Engineers (IEEE) 982.1, IEEE Standard Dictionary of Measures of the Software Aspects of Dependability (2005).
8. Not a checklist or laundry list: the metric must not depend on a list of disparate rules or factors that would be unreasonable to quote in its entirety in this document.
9. Not overspecialized: to avoid an infinite proliferation of variations on a theme, a metric that is an obvious specialization or derivative of something already included here, need not be included here. For example, given that number of operations is included, we do not necessarily need entries for the number of *illegal* operations for every possible definition of illegal, the *mean* number of operations per every possible denominator, or the *proportion* of operations that are legal, unless the derivative forms are especially notable in themselves.

The metrological validity and soundness of metrics was not evaluated. The inclusion of a metric is not a recommendation or endorsement, and the exclusion of a metric is not a criticism or condemnation.

1.3 Guide to tables

Ch. 2 and later chapters detail the software system of quantities in a series of tables. The columns of those tables are explained below.

- The Names column gives the intelligible short description or alternative descriptions of the quantity.
- The Symbols column attempts to provide either the canonical symbol assigned by a primary reference or the symbols that are most commonly used in practice to identify a quantity. This document makes no attempt to catalog every symbol ever used for a given quantity; the proliferation is too great and is not to be encouraged.
- The Definitions column defines the quantity and includes any necessary discussion. In especially complex cases, a summary is given and readers must consult the cited references for the complete definition. References to input quantities that are defined elsewhere in the document may be indicated by bold font. Superscript numbers refer to the copyright notes listed in Ch. 12.
- The References column contains one or more of the following:

- Supertypes: the References column of the tables in Ch. 2 identifies the immediate supertypes of dimensionless units within a type system as described in [Flater].
- Unit: an expression in terms of the counting units of Ch. 2 and SI units as applicable. For ratios of two quantities of the same kind, such as compression ratio, the units are shown in unsimplified form (bit/bit) for clarity.
- Dimension: when specific units are over-constraining, an expression in terms of the dimensions of Ch. 3 and SI dimensions may be given instead.
- Scale: if a unit or dimension was specified, a ratio scale is implied; otherwise, an interval, ordinal, nominal, or dichotomic scale is specified.
- Range: further explanation of the meaning of the output values, when needed.
- Reference: the source of the definition or the document to refer to for details of the measurement method. In the absence of quotation marks, the definition may be a paraphrase, summary, reduction, or rewrite of some portion of the cited source. The source may provide multiple definitions of which only one was selected. When no reference is given, the definition is a best effort to fill gaps in the canon.

Tables at the beginning of a section that are introduced as "model" or "defined inputs" provide definitions that the quantities in a subsequent table depend on. These inputs are not necessarily quantities, but may be sets or other abstractions that have neither units nor scale. The information provided is a subset of what was described above.

1.4 History and future

This document was started in 2017 by Sumaiyah Sarwat for a Summer Undergraduate Research Fellowship (SURF) with David Flater as research advisor. It was finished by David Flater and submitted to the International Electrotechnical Commission (IEC) in 2018 as initial basis for new work item proposal PNW 25-631 to create 80000-18, Quantities and units—Part 18: Software product. It received the necessary ⅔ majority of votes for approval, but only 3 of the approving members nominated experts to participate in development. A minimum of 4 experts from approving members was required, so the new work item proposal ultimately was rejected. Subsequently, the document was made a NIST publication with copyright clearance assistance by Karen Reczek and technical reviews by Paul E. Black and David B. Newell.

In current practice, many of the elementary quantities of software measurement are multiply-defined and/or ill-defined. Improving and standardizing the definitions of these base quantities is within the traditional scope of international standards work. As software metrology becomes a mature discipline, the document should become less descriptive (following the practice) and more prescriptive (normative). This transition requires consensus, and the proper venue for such a consensus to emerge is international standards.

2 Countable entities and events

In the following tables, **entity** and **event** serve as the most general counting units at the top of the type system (disregarding unit one), following the pattern that was initiated in [Mohr].

Metrology practice maintains a clear distinction between symbols that denote quantities, such as *l* denoting length, and symbols that denote units, such as m denoting the meter (the unit of length). To express that a length is 5 meters, one would write *l* = 5 m but not m = 5. In contrast, the usage of symbols in software practice often is equivocal over whether a symbol refers to a counted quantity (e.g., LOC = 5) or a type of entity that is used as a counting unit (5 LOC). In such cases, the counted quantity can be described simply as "Number of [countable entity]," and a definition of the countable entity would be sufficient to define the quantity. Indeed, many software metrics are nothing more than these counts, embellished by description. For example, the number of LOC, the number of classes, or the number of terms could all be described, in some fashion, as the "size" (more accurately, *a* size) of a software artifact. However, certain symbols, such as Halstead's N_1 and η_1, qualify the method of counting so that merely cataloging the countable entity types is insufficient.

Many higher-level, derived software metrics define input quantities in-line with expressions like "...where *N* is the number of (...)." In a more mature system of quantities, commonly-used counts might have standard symbols, and n and N might not be so overloaded.

2.1 Elementary entities

Names	Symbols	Definitions	References
Bit	b	Quantum unit of data. The definition of 'b' as the symbol for bit was made in [IEEE-100] (now withdrawn), and thence indirectly by [IEEE-1541] which makes normative reference to it, but [IEC] does not use it. In practice, it appears widely as bandwidths are quoted in units of "Mbps" (meaning Mb/s) or "Gbps" (meaning Gb/s). The 'b' symbol is also used for the barn, a non-SI unit of area.	Entity [IEEE-100]
Qubit Quantum bit		Bit, in the context of quantum computing.	Bit [Schumacher]

Names	Symbols	Definitions	References
Byte	B	= 8 bit In many contexts, the byte is practically used as the quantum unit of data because individual bits cannot be directly addressed. The derived unit byte, symbol B, has *de facto* been standardized as 8 bits, but may vary in historical uses. [IEC] suggests the less ambiguous unit octet, symbol o, but this is seldom seen in practice. The 'B' symbol is also used for the bel, a non-SI unit of logarithmic ratio quantities.	[IEC]
Word		"The normal unit in which information may be stored, transmitted, or operated on within a given computer." [10] Like the byte, the word is defined as a number of bits, but the number is architecture-dependent.	[IEEE-100]
Character	char	"A sequence of one or more bytes representing a single graphic symbol." [10] The number of bytes per symbol may vary, as it does in the popular encoding UTF-8 (8-bit Unicode Transformation Format).	Entity [IEEE-100] [Fenton, p. 346]
Pixel	px	Etymologically derived from *picture element*, pixels are the elementary constituents of raster graphics images.	Entity
Instruction, Operation	op	Unit of machine code / assembly language. Depending on the implementation, instructions may be indivisible or they may translate into blocks of microcode.	Entity
Microinstruction, Micro-operation	μop	Quantum unit of microcode. (**Caution:** while the name and symbol imply that 1 op = 10^6 μop, instructions and microinstructions are not actually comparable. The use of the micro- prefix here, established in software jargon, is misleading and incompatible with the SI.)	Entity

2.2 Source-level entities

Many of the entities given in this section have problematic definitions. Some have standard definitions that conflict with their meanings within widely-used software metrics, and some have no standard definitions at all. As was mentioned in Sec. 1.4, gaining consensus on an adequate set of definitions for these base quantities is a work item for the standardization of software metrology.

Names	Symbols	Definitions	References
Line of code, Source line of code, Physical line of code	LOC, SLOC	Segment of source code that is delimited by the applicable end-of-line string (usually a single line feed character or a carriage return-line feed combination) or by the beginning or end of the file. See [Park] for a framework of many different yet equally plausible definitions of this quantity. [Misa] defines SLOC as LOC minus blank lines and comment lines (= NCLOC, see below).	Entity [Park]
Logical line of code	LLOC	Ambiguous. Possible meanings include: 1. NCLOC 2. LOC after concatenating continuation lines 3. NCLOC after concatenating continuation lines 4. **Statement**	Entity (not necessarily LOC)
Comment line of code	CLOC	LOC containing nothing but comments and whitespace, or that is completely empty (blank).	LOC [Fenton, pp. 340–341]
Noncommented line of code, Effective line of code	NCLOC	LOC that is not a CLOC. LOC = NCLOC + CLOC	LOC [Fenton, pp. 340–341]
Line of comments	CM, CMT	LOC that contains a comment (*and possibly other content*); "a physical line on which there is a comment." This usage seems rare but is referenced in Sec. 7.12.	LOC [Welker, p. 130]
Function, Procedure, Subroutine, Submodule		"Portion of a computer program that is named and that performs a specific action." [6] This concrete source code entity should not be confused with the abstract unit of functionality **function** appearing in Sec. 2.6.	Entity [ISO-Vocab]
Module		"Program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading." "Collection of both data and the routines that act on it." [6]	Entity [ISO-Vocab]
Class		"Static programming entity in an object-oriented program that contains a combination of functionality and data." [6] Sometimes compared to a **module** unit of procedural programming (for larger interpretations of **module**).	Entity [ISO-Vocab]

Names	**Symbols**	**Definitions**	**References**
Package, Subsystem, Cluster		"Separately compilable software component consisting of related data types, data objects, and subprograms." [6] In object-oriented contexts, a package consists of related **classes**. "A subsystem is a collection of **classes** that support a set of end-user functions."	Entity [ISO-Vocab, Lorenz]
Data type		"Set of values and operations on those values." [6]	Entity [ISO-Vocab]
Abstract data type	ADT	"**Data type** for which only the properties of the data and the operations to be performed on the data are specified, without concern for how the data will be represented or how the operations will be implemented." [6] Sometimes synonymous with **class**.	Data type [ISO-Vocab]
Block		Group of contiguous **statements** that are treated as a unit. [6]	Entity [ISO-Vocab]
Method, Operation		Source code unit from object-oriented programming; a **function** that is scoped by and contained within a **class**. "Operation" is the term used in [UML], but in this document it is too easily confused with its other definitions, so herein we prefer "method."	Function
Statement		"In a programming language, a meaningful expression that defines data, specifies program actions, or directs the assembler or compiler." [6] (The term "expression" is used in a more general sense in the preceding quote than the way it is defined below.) A unit corresponding to a single "command."	Entity [ISO-Vocab]
Expression		"Sequence of constants, variables, and functions connected by operators to indicate a desired computation." [10]	Entity [IEEE-100]
Data object, Data element, Data item		**Entity** that occupies storage or consumes bandwidth.	Entity
Variable		"Quantity or **data item** whose value can change." [6]	Data object [ISO-Vocab]
Parameter, Argument, In-parameter		An input to a function or procedure. In many contexts it is necessary to distinguish the declared "formal parameter" from the "actual parameter" provided at invocation.	Operand, Data object
Result, Return value, Out-parameter		An output of a function or procedure.	Operand, Data object

Names	**Symbols**	**Definitions**	**References**
Operand		"Variable, constant, or function upon which an operation is to be performed." [6] A **parameter** or **result**.*	Term [ISO-Vocab] [Halstead]
Call, Invocation, Operation, Message send		Source code **entity** that causes a call/invocation **event** (see profiling units). An **operator** together with its **operands**.	Entity
Exception		Source code **entity** that represents or describes an **exception** (event).	Entity
Operator		"Mathematical or logical symbol that represents an action to be performed in an operation." [6] 1. Narrowly: the predefined programming language functions that include mathematical, logical, grouping, indexing, dereferencing, address-of, scoping, character string, etc. functions. 2. More broadly: any function or procedure that has one or more operands. 3. [Halstead] also considers control constructs to be operators [Halstead p. 7]. See *mises en pratique*.*	Term [ISO-Vocab] [Halstead]
Token		**Operator** or **operand**.* (This generalization is the unit of several of Halstead's metrics, but is not given a name in [Halstead].)	Entity [Halstead]
Site		Source code **entity** that indicates where a **weakness** may exist or does exist, i.e., a code location with characteristics relevant to bug classes. "A location in code where a weakness might occur."	Entity [IR8113]
Branch, Decision-to-decision path	DD-path	One of the outbound paths from a conditional statement; e.g., an if-then-else conditional has two branches, one of which is executed if the expression is true and the other of which is executed if the expression is false.	Entity [ISO-Vocab]
Condition		"A Boolean **expression** containing no Boolean operators except for the unary operator (NOT)." [11]	Expression [DO-178C]
Decision		"A Boolean **expression** composed of conditions and zero or more Boolean operators. If a condition appears more than once in a decision, each occurrence is a distinct condition." [11]	Expression [DO-178C]
Entry point, Entry		"Point in a software module at which execution of the module can begin." [6]	Statement [ISO-Vocab]
Exit point, Exit		"Point in a software module at which execution of the module can terminate." [6]	Statement [ISO-Vocab]
Data flow, Information flow		Transfer of data from one **module** to another. (Note that this refers to the implementation of such transfer in software, not the event of it occurring at run time.)	Entity [Henry]

* Halstead's model that "*an algorithm consists of operators and operands, and of nothing else*" [Halstead p. 8] is difficult to reconcile with complex programming languages that implement various declarations, control constructs, pragmas, etc. "The counting rules for the basic metrics are ill-defined, arbitrary and not applicable to languages with structured and abstract data types" [Hamer]. Halstead's counting units consequently get defined through unofficial "*mises en pratique*"—third-party documents that specify how to count them (or how they are counted in fact by some tool, for better or worse) for specific programming languages.

2.3 Graph entities

> "The material which follows comes mostly from a larger area of mathematics known as the theory of graphs. Unfortunately, there is as yet no standard terminology in this field, and so the author has followed the usual practice of contemporary books on graph theory, namely to use words that are similar but not identical to the terms used in any *other* books on graph theory." [Knuth, Sec. 2.3.4]

Software metrology depends on graph theory due to the common use of control flow graphs, data flow graphs, dependency graphs, and other graphs in software measurement.

It is apparently the case that graph-theoretic terms such as walk, trail, circuit, path, chain, and cycle have not been standardized and are used differently in different sources. See Mathematics Stack Exchange, "What is difference between cycle, path and circuit in Graph Theory" and similar discussions. The impacted definitions below are tagged with the following attributes, which act as constraints:

N = reuse of nodes is prohibited (except special case for initial-final node in a cycle)
E = reuse of edges is prohibited
D = direction of arcs matters
C = closed; end of sequence is required to be the same as the beginning.

Names	Symbols	Definitions	References
Edge, Arc	e [McCabe], m [Berge]	A line or arrow in a graph.	Entity [Berge]
Node, Vertex	n [Berge, McCabe]	A point in a graph.	Entity [Berge]
Predicate node	π [McCabe], d [Fenton]	Flowgraph **node** with out-degree greater than 1. Using the symbol π for the count of predicate nodes conflicts with its canonical interpretation as a mathematical constant.	Node [McCabe]
Connected component	p [Berge, McCabe]	A class of the equivalence relation [x=y, or x≠y and there exists a **chain** in G connecting x and y].	Entity [Berge]

Names	Symbols	Definitions	References
Chain		"A sequence $\mu = (u_1, u_2, ..., u_q)$ of **arcs** of G such that each **arc** in the sequence has one endpoint in common with its predecessor in the sequence and its other endpoint in common with its successor in the sequence".	Entity [Berge]
Elementary chain		"**Chain** that does not encounter the same vertex twice." N	Chain [Berge]
Simple chain		"**Chain** that does not use the same arc twice." E	Chain [Berge]
Cycle		**Simple chain** whose endpoints are the same **vertex**. EC	Simple chain [Berge]
Elementary cycle		**Cycle** in which "no vertex is encountered more than once (except, of course, the initial vertex which is also the terminal vertex)." NEC The exception for the initial-final vertex in this representation of cycles means that elementary cycle is not a subtype of elementary chain.	Cycle [Berge]
Path		**Chain** "in which the terminal endpoint of arc u_i is the initial endpoint of arc u_{i+1} for all i<q." D	Chain [Berge]
Circuit		**Cycle** "such that for all i<q the terminal endpoint of u_i is the initial endpoint of u_{i+1}." EDC	Cycle [Berge]
Knot		Place where two arrows are forced to cross each other in some prescribed graph layout.	Entity [Woodward]

2.4 Dependency and definition/use entities

Uses, interactions, and dependencies have to do with relationships among source code entities. As such, they may exist only in a model of the software, or they may be ascribed to one of the involved entities (e.g., the dependency of A on B may be ascribed to A as the point of use). There are several alternative models and vocabularies.

Names	Symbols	Definitions	References
Definition	def	Variable occurrence "in which a value is stored in a memory location."	Entity [Frankl]
Use	use	Variable occurrence "in which a value is fetched from a memory location."	Entity [Frankl]
Computation use	c-use	**Use** that "directly affects the computation being performed or outputs the result of some earlier definition."	Use [Frankl]
Predicate use	p-use	**Use** that "directly affects the flow of control through the subprogram, and thereby may indirectly affect the computations performed."	Use [Frankl]

Names	Symbols	Definitions	References
Data declaration-data declaration interaction	DD-interaction	"A data declaration *A* DD-interacts with another data declaration *B* if a change in *A*'s declaration or use may cause the need for a change in *B*'s declaration or use."	Entity [Briand93]
Data declaration-subprogram interaction, Data declaration-method interaction	DS-interaction, DM-interaction	"A data declaration DS-interacts with a subprogram if it DD-interacts with at least one of its data declarations." "There is a *DM-interaction* between data declaration *a* and method *m*, if *a* DD-interacts with at least one data declaration of *m*. Data declarations of methods include their parameters, return type and local variables."	CI [Briand93, Briand98]
Cohesive interaction	CI	"The set of cohesive interactions in a module is the union of the sets of DS-interactions and DD-interactions, with the exception of those DD-interactions between a data declaration and a subprogram formal parameter."	Entity [Briand93]
Interaction		Reference to a class as the type of an attribute, parameter, or result, or invocation of one of its methods.	Entity [Briand97]

2.5 Class diagram entities

Names	Symbols	Definitions	References
Relationship		**Edge** in a class diagram (e.g., association, generalization, aggregation).	Edge [UML]
Attribute		"Identifiable association between an object and a value." [6]	Data object [ISO-Vocab]

2.6 Units of functionality

Although some shared definitions have been placed in a more general context to avoid duplication, the overlaps between the various standards for functional size measurement (FSM, c.f. Sec. 7.9) have not been fully sorted out.

Names	Symbols	Definitions	References
Requirement, Compliance point		"Condition or capability that must be met or possessed by" the software "to satisfy an agreement, standard, specification, or other formally imposed documents." [6]	Entity [ISO-Vocab]
Use case, Scenario script		"Sequence of tasks that a system can perform, interacting with users of the system and providing a measurable result of value for the user." [6]	Entity [ISO-Vocab]

Names	Symbols	Definitions	References
Feature		"Distinguishing characteristic of a system item." [6]	Entity [ISO-Vocab]
Function		"Defined objective or characteristic action of a system or component." [6] This abstract unit of functionality should not be confused with the concrete source code entity **function** appearing in Sec. 2.2.	Entity [ISO-Vocab]
Function point	FP	Ostensible unit corresponding to a difference of 1 in the result of one of several functional size measurement methods. However, most of these methods yield ordinal values. See Sec. 7.9.	
IFPUG function point	FP	"Unit of measure for functional size" [3] ISO nomenclature: FP (IFPUG-IS)	FP [ISO-IFPUG]
FiSMA function point	Ffp	FP as resulting from the measurement method defined in [ISO-FiSMA].	FP [ISO-FiSMA]
COSMIC function point	CFP	"The size of one data movement" FP as resulting from the measurement method defined in [COSMIC, ISO-COSMIC]. ISO nomenclature: CFP (ISO/IEC 19761:2011)	FP [COSMIC, ISO-COSMIC]
MkII function point	MkII FP	FP as resulting from the measurement method defined in [ISO-MkII]. ISO nomenclature: MkII FP (ISO/IEC 20968:2002)	FP [ISO-MkII]
NESMA function point	FP	FP as resulting from the measurement method defined in [ISO-NESMA]. ISO nomenclature: FP (ISO/IEC 24570:2018)	FP [ISO-NESMA]
Automated function point	AFP	FP as resulting from the measurement method defined in [AFP].	FP [AFP]
Base functional component	BFC	"Elementary unit of Functional User Requirements defined by and used by an FSM Method for measurement purposes" [1]	Entity [ISO-FSM]
Data function		"Functionality provided to the user to meet internal or external data storage requirements" [3]	BFC [ISO-IFPUG]
Transactional function		"Elementary process that provides functionality to the user to process data" [3]	BFC [ISO-IFPUG]
External input	EI	"Elementary process that processes data or control information sent from outside the boundary" [3]	Transactional function [ISO-IFPUG]
External output	EO	"Elementary process that sends data or control information outside the boundary and includes additional processing logic beyond that of an External Inquiry" [3]	Transactional function [ISO-IFPUG]
External inquiry	EQ	"Elementary process that sends data or control information outside the boundary" [3]	Transactional function [ISO-IFPUG]

Names	Symbols	Definitions	References
Internal logical file	ILF	"User recognizable group of logically related data or control information maintained within the boundary of the application being measured" [3]	Data function [ISO-IFPUG]
External interface file	EIF	"User recognizable group of logically related data or control information, which is referenced by the application being measured, but which is maintained within the boundary of another application" [3]	Data function [ISO-IFPUG]
Data element type	DET	"Unique, user recognizable, non-repeated attribute" [3] "Unique, user-recognizable, non-repeated field in a BFC" [7]	Entity [ISO-IFPUG, ISO-FiSMA]
Record element type	RET	"User recognizable sub-group of data element types within a data function" [3]	Entity [ISO-IFPUG]
File type referenced	FTR	"Data function read and/or maintained by a transactional function" [3]	Entity [ISO-IFPUG]
Data movement		"Base Functional Component which moves a single data group" [2]	BFC [ISO-COSMIC]
Entry		"Data movement that moves a data group from a functional user across the boundary into the functional process where it is required" [2]	Data movement [ISO-COSMIC]
Exit		"Data movement that moves a data group from a functional process across the boundary to the functional user that requires it" [2]	Data movement [ISO-COSMIC]
Read		"Data movement that moves a data group from persistent storage [to] within reach of the functional process which requires it" [2]	Data movement [ISO-COSMIC]
Write		"Data movement that moves a data group lying inside the functional process to persistent storage" [2]	Data movement [ISO-COSMIC]
Logical transaction		"Smallest complete unit of information processing that is meaningful to the end user in the business" [4]	BFC [ISO-MkII]
Input data element type	Ni	DET that is an input to a logical transaction.	DET [ISO-MkII]
Data entity type	Ne	"Fundamental thing of relevance to the user, about which information is kept." [4]	Entity [ISO-MkII]
Output data element type	No	DET that is an output from a logical transaction.	DET [ISO-MkII]
Interactive end-user navigation and query service	q	"Interactive end-user navigation and query services specify all parts of the interactive user interface where there is no maintenance of persistent data stored in the system." [7] Seven subtypes are defined.	BFC [ISO-FiSMA]

Names	Symbols	Definitions	References
Interactive end-user input service	i	"Interactive end-user input services specify all parts of the interactive user interface where there is maintenance of data store(s) of the software." [7] Three subtypes are defined.	BFC [ISO-FiSMA]
Non-interactive end-user output service	o	"Non-interactive end-user output services specify all parts of the user interface which are non-interactive and do not maintain data store(s) of the software." [7] Four subtypes are defined.	BFC [ISO-FiSMA]
Interface service to other applications	t	"Interface services to other applications specify all automatic data transfers that move data from the measured piece of software to another application or any device." [7] Three subtypes are defined.	BFC [ISO-FiSMA]
Interface service from other applications	f	"Interface services from other applications specify all automatic data transfers that receive data groups that are provided and sent by another application or any device." [7] Three subtypes are defined.	BFC [ISO-FiSMA]
Data storage service	d	"Data storage services specify a group or collection of related and self-contained data in the real world, about which the user requires the software to provide one or more data stores." [7] Two subtypes are defined.	BFC [ISO-FiSMA]
Algorithmic and manipulation service	a	"Algorithmic and manipulation services are user-defined, independent data manipulation functions." [7] Six subtypes are defined.	BFC [ISO-FiSMA]
Reading reference		"Data storage entity or record, or interface record from another software or system containing data retrieved in a BFC" [7]	Entity [ISO-FiSMA]
Writing reference		"Data storage entity or other record, or interface record to another software or system to which data is written in a BFC" [7]	Entity [ISO-FiSMA]
Operation		"Arithmetic or logical **operation** performed in an algorithmic and manipulation BFC" [7]	Entity [ISO-FiSMA]

2.7 Units of failure, interruption, and termination

Names	Symbols	Definitions	References
Fault		"Incorrect step, process, or data definition in a computer program." [6]	Entity [ISO-Vocab]
Failure		"**Event** in which a system or system component does not perform a required function within specified limits." [5]	Event [ISO-Vocab-2010]

Names	Symbols	Definitions	References
Error		An incorrect result; a "difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition." [6]	Entity [ISO-Vocab]
Bug, Defect		Less formal terms that may mean either **fault** or **error** depending on context and viewpoint. The relationship between **errors** observed by users and **faults** identified by developers is many-to-many.	Entity
Bug report, Problem report		**Event** of a user or tester asserting the existence of a **fault** in the software.	Event
Weakness		**Fault** that is security-relevant.	Fault
Vulnerability		**Weakness** that is exploitable.	Weakness [SP800-30r1]
Panic		Self-initiated emergency stop of an operating system kernel.	Crash
Crash		Emergency stop of a running process.	Failure
Hang		**Failure** of a process to make progress.	Failure
Lockup		**Hang** accompanied by abnormal unresponsiveness to signals or control inputs.	Hang
Abort		Controlled interruption and shutdown of a running process that has not reached its "normal" termination point.	Exit
Exit		Controlled termination of a running process.	Event
Timeout		**Event** of a **latency** exceeding a threshold value.	Event
Exception		"Abnormal" **event** that necessitates the "normal" flow of control of a running process to be interrupted. (See also the source code entity.)	Event [ISO-Vocab]
Interrupt		"Suspension of a process to handle an event external to the process." [6]	Event [ISO-Vocab]
Signal		Specific kind of **interrupt** on Unix-like operating systems.	Interrupt

2.8 Profiling units

Names	Symbols	Definitions	References
Call, Invocation, Message		**Event** of pushing the current function onto the stack and transferring control to another function. Not to be confused with the source code **entity** that causes it to occur. The term "message" is used for Smalltalk and Objective C.	Event
Sample		**Event** of a profiling interrupt being fired. In non-intrusive profiling, the execution of a program is "sampled" either at periodic intervals or at aperiodic times when arbitrary, defined conditions are met.	Event

Names	Symbols	Definitions	References
Cycle		A single "tick" of the internal clock source of a processing unit such as a Central Processing Unit (CPU) or Graphics Processing Unit (GPU).	Event
Operation		Unspecified unit of processing used to parameterize algorithmic performance and throughput metrics. Not to be confused with operation as machine code instruction (elementary entity) or source code entity.	Event
Iteration		Single execution of the **block** inside a looping control construct.	Event
Run		Single execution of a program, from start to exit or crash.	Event
Transaction		Application-specific unit of processing, usually intended to execute atomically.	Event
Resource		"Any physical or virtual component of limited availability within a computer system available for a given purpose and managed by the runtime platform." [6] E.g., CPU cores, GPU cores, file descriptors, memory.	Entity [ISO-Vocab]

2.9 Testing units

Names	Symbols	Definitions	References
Variable-value configuration		For a set of *t* variables, a variable-value configuration is a set of *t* valid values, one for each of the variables. For test coverage purposes, a finite and practically testable set of values is selected and deemed "valid" for each variable. In design-of-experiments vocabulary, the variables are factors, the values are levels, and the variable-value configurations are treatments.	Entity [Kuhn]
Combination		*k* distinct elements chosen from a set of cardinality $\geq k$ (the standard definition from combinatorics).	Entity [Kuhn]
Linear code sequence and jump, Jump-to-jump path	LCSAJ, JJ-path	An LCSAJ triple "consists of a linear sequence of code... plus a jump to a particular location." For Fortran programs, the start point of a LCSAJ "is either the first line of the program or any line which has a jump to it from elsewhere in the program, other than the preceeding line," and the end point "is either the end of the program or a line which contains a jump to other than the succeeding line."	Entity [Hennell]

3 Dimensions

See Sec. 0.4 for background. The legitimacy of data and information as dimensions, and of cycles and samples as units of time, need not be defended here. The purpose of this chapter is simply to identify the ranges of alternative units that could be used when one of the following dimensions is referenced in subsequent sections.

Names	Symbols	Definitions	References
Time	**T**	All conventional units of time longer than the second have multiple, competing definitions. E.g., the existence of leap seconds confounds the definition of a minute as 60 s, an hour as 3600 s, etc.; summer time (daylight savings time) causes certain days to contain more or fewer than 24 hours; and conflicting definitions of the year (calendar years and astronomical years) are most notorious.	second [SI], minute, hour, day..., cycle, sample
Data			bit, byte, data object
Information			shannon (a.k.a. "bit" of information), hartley, nat [IEC]
Work		The essential product of a processing unit (Central Processing Unit (CPU), Graphics Processing Unit (GPU), or suchlike).	transaction, iteration, operation (event), instruction (entity)

4 Basic quantities

4.1 Physical quantities

It should be noted that the relative quantities given below are seldom identified as such in practice; e.g., both self time and relative self time are just called "self time."

Names	Symbols	Definitions	References
Execution time, Run time, Wall time, Wall clock time, Elapsed time, Real time, Real world time		Time that a process takes to run, or that a process or system has been monitored.	Dimension: time
Resource time		Time that a specified class of resource was used, possibly by a specified process, thread, or group thereof. Resource time may exceed elapsed time if more than one resource (e.g., multiple CPU cores) was used.	Dimension: time

Names	Symbols	Definitions	References
Relative resource time, Resource utilization		= **Resource time / real time** This value may exceed unity if more than one resource (e.g., multiple CPU cores) was used. C.f. resource utilization in the next section (same name, different quantity).	Dimension: time/time
Self time		Time that a specified function was executing (running), i.e., that the instruction being executed by the CPU was actually part of that function.	Dimension: time
Relative self time		= **Self time / CPU time**	Dimension: time/time
Total time		Time that a specified function either was executing (**self time**) or was on the stack (while a called subfunction or event handler was being executed).	Dimension: time
Relative total time		= **Total time / CPU time**	Dimension: time/time
Latency, Delay, Lag, Response time		Time between the final **event** that enables or causes something to occur and the **event** of it actually occurring.	Dimension: time
Energy		Amount of energy (generally electric) used to run a process.	Unit: J (joule) [SI] or the non-SI kWh (kilowatt hour).
Power		Rate of energy use for running a process.	Unit: W (watt) = J/s [SI]

4.2 Resources, processing, and transmission

Names	Symbols	Definitions	References
Size		[Of a **data object**]	Dimension: data
Storage capacity, Storage size	M	"Amount of data that can be contained in a storage device, expressed as a number of specified data elements" [8] "Storage" here generalizes all kinds of memory, hard disks (HDD), solid state devices (SSD), etc., and the "device" may be either physical or logical. ‘M’ is also used for mass fraction and as the prefix for 10^6.	Dimension: data [IEC]

Names	Symbols	Definitions	References
Resource utilization		Proportion of the resources of a specified class that were used. C.f. resource utilization = relative resource time in the previous section (same name, different quantity).	Unit: resource/resource
Storage utilization		Proportion of storage capacity used; i.e., **resource utilization** where the resource is storage.	Dimension: data/data
Transfer rate	r, ν	"Quotient of the number of specified data elements transferred in a time interval by the duration of this interval" [8]	Dimension: data/time [IEC]
Bandwidth		Maximum available **transfer rate**; transfer capacity.	Dimension: data/time
Throughput		Work performed in a given period of time. [6]	Dimension: work/time [ISO-Vocab]
Information content	I(x)	$= \log_2 \frac{1}{p(x)}$ Sh where p(x) is the probability of **event** x. [8]	Dimension: information [IEC]
Compression ratio (storage)		$\frac{\text{Uncompressed size}}{\text{Compressed size}}$	Dimension: data/data
Compression ratio (transmission)		$\frac{\text{Uncompressed transfer rate}}{\text{Compressed transfer rate}}$	Dimension: (data/time)/(data/time)

4.3 Graph metrics

Many software metrics are derived using a graph model of the software, such as a control flow graph, data flow graph, call graph, dependency graph, or attack graph. The following metrics apply to graphs in general (directed, undirected, or both).

Names	Symbols	Definitions	References
Cyclomatic number, First Betti number, Circuit rank, Nullity	V(G), v(G)	The number of independent **elementary cycles** in a graph, derived as $e - n + p$ Where: e is number of **edges** n is number of **nodes** p is number of **connected components**	Unit: elementary cycle Cyclomatic number is defined for undirected graphs. The same number is indicative of several different things. See Wikipedia, "circuit rank," for alternative uses. [Berge] See also, cyclomatic complexity in Sec. 6.4.
Depth (of graph)		"Length of the longest path from the root node to a leaf node."	Unit: arc [Fenton, p. 405]
Width		"Maximum number of nodes at any one level."	Unit: node [Fenton, p. 405]
Edge-to-node ratio			Unit: edge/node [Fenton, p. 405]

[Fenton, p. 405] further notes that graph size may be measured by the counts of nodes and edges (Sec. 2.3).

In addition, the following metrics apply to a given node within a graph.

Names	Symbols	Definitions	References
Depth (of node)		Length of the longest path from the root node to a given node.	Unit: arc [Zuse]
Number of ancestors / ascendents		Number of nodes that are reachable from a given node by following edges toward the root.	Unit: node [Zuse]
Number of descendants / successors		Number of nodes that can reach a given node by following edges toward the root.	Unit: node [Zuse]
Proportion of ancestors / ascendents		Number of ancestors divided by the number of nodes.	Unit: node/node
Proportion of descendants / successors		Number of descendants divided by the number of nodes.	Unit: node/node

5 Compatibility metrics

Compatibility metrics indicate the CPU architectures, operating systems, and user environments with which software is compatible. Note that "fat binaries" may support multiple, mutually incompatible architectures and operating systems.

The terms in all capital letters in the table below are a mixture of acronyms, former acronyms that evolved into proper names, and acronym-like names that were invented for branding purposes. ARM, CP/M, DOS, and MIPS are proper names in current usage; their historical expansions are irrelevant. The instruction set extensions are commonly expanded as follows:

ADX	Multi-precision add-carry instruction extensions
AVX	Advanced Vector Extensions
DSP	Digital Signal Processing
MMX	Multimedia Extensions
MPX	Memory Protection Extensions
SGX	Software Guard Extensions
SIMD	Single Instruction, Multiple Data
SSE	Streaming SIMD (Single Instruction, Multiple Data) Extensions
TSX	Transactional Synchronization Extensions
TXT	Trusted Execution Technology
VFP	Vector Floating Point
VT-d	Virtualization Technology for directed input/output
VT-x	Virtualization Technology extensions

Names	Symbols	Definitions	References
Architecture, Instruction set architecture	ISA	E.g., x86, ARM, or MIPS. By convention, x86-64 is often listed separately from x86, but technically it is x86 with 64-bit extensions.	Nominal set
Word size		Historically, "the number of bits in a **word**." In current practice, it identifies classes of platforms with an implied ordering of relative capability: 4-bit = mostly embedded controllers 8-bit = vintage computer 16-bit = DOS, CP/M 32-bit = legacy and low-end, ≤ 4 GiB RAM 64-bit = mainstream, ≥ 4 GiB RAM	Unit: bit (for historical usage) or ordinal (as a platform classifier)
Microarchitecture		Name of the oldest and/or simplest processor microarchitecture that is capable of running the software. Within a sequence of backward-compatible microarchitecture iterations, the values form an ordinal scale. E.g., Prescott < Core < Nehalem < Sandy Bridge < Haswell < Skylake ARMv1 < ARMv2 < ARMv2a < ARMv3 etc.	Nominal set or ordinal

Names	Symbols	Definitions	References
Instruction set extensions		Names of "additional" or "supplementary" sets of instructions that are needed to run the software. E.g., x86-64, MMX, 3DNow, SSE, SSE2, ..., SSE4.2, ADX, AVX, AVX2, AVX-512, MPX, TXT, TSX, SGX, VT-x, VT-d AArch64, Thumb, Thumb-2, DSP, SIMD, VFPv1, VFPv2, ..., VFPv5-D16-M, Neon	Nominal set
Operating system		Versions of operating systems that are able to run the software. Within a sequence of backward-compatible iterations, the values form an ordinal scale.	Nominal set or ordinal
Privileges, Roles		Security properties that must be granted for the software to run. E.g., root or administrator, or access to file system, location, contacts database, camera, or microphone.	Nominal set

6 Algorithm metrics

6.1 Performance

Computational and space complexity are typically quoted for best, average, and worst cases, using big O notation.

Names	Symbols	Definitions	References
Computational complexity, Time complexity		Number of **operations** required, expressed as a function of the number of **data objects** in the input.	Unit: operation
Memory complexity, Space complexity		Amount of storage required, expressed as a function of the number of **data objects** in the input.	Dimension: data
Computational efficiency, Time efficiency		Ratio of the theoretical minimum number of **operations** required divided by the **computational complexity** of the algorithm.	Unit: operation/operation
Memory efficiency, Space efficiency		Ratio of the theoretical minimum amount of storage required divided by the **space complexity** of the algorithm.	Dimension: data/data

6.2 Hash function metrics

Attack resistance refers to the amount of work (e.g., number of hash function evaluations) that is expected to be required for an attack to succeed. Depending on context, the "expectation" may be an upper bound (i.e., what is required to complete an exhaustive search), a statistical average, or an order-of-magnitude estimate; the amounts required in a particular case may be more or less, and there can be tradeoffs.

Names	Symbols	Definitions	References
[Max] message size		Maximum allowed size of the input to a hash function. If length size (see below) = l bits, then message size = $2^l - 1$ bits.	Dimension: data
Length size		Size of the scalar data item used to record the length of the message.	Dimension: data
Output size, Message digest size, Hash value size		Size of the output of a hash function.	Dimension: data
Internal state size		Size of the intermediate hash result.	Dimension: data
Block size		Size of the data segments into which the input is separated for processing.	Dimension: data
Number of rounds		Number of iterations of the work within the hash algorithm.	Unit: iteration
Security bits, Security strength		$\log_2$ of the value of one of the following four resistance quantities.	Dimension: work Logarithmic scale
Collision resistance		Expected amount of work required to find two inputs that produce the same hash value.	Dimension: work
Chosen prefix collision resistance		Expected amount of work required to find two inputs that produce the same hash value when the beginning of each input has been predetermined.	Dimension: work
Preimage resistance		Expected amount of work required to find an input that has a specific hash value.	Dimension: work
Second-preimage resistance		Expected amount of work required to find a second input that has the same hash value as a specified input.	Dimension: work

6.3 Block cipher metrics

In cryptography jargon, the "time/memory/data" triple refers to the amount of work (e.g., number of cipher evaluations), the amount of storage (memory), and the amount of input data (e.g., number of known plaintext-ciphertext pairs) respectively that are expected to be required for an attack to succeed. Depending on context, the "expectation" may be an upper bound (i.e., what is required to complete an exhaustive search), a statistical average, or an order-of-magnitude estimate; the amounts required in a particular case may be more or less, and there can be tradeoffs. Work may be divided between a "preprocessing" phase and a "realtime" phase.

Names	Symbols	Definitions	References
Block size, Block length	*Nb*	Amount of data that comprises each of the input block, output block, state (intermediate cipher result), and round key.	Dimension: data [FIPS 197]
Key size, Key length	*Nk*	Size of the cipher key that is used by the key expansion routine to generate a set of round keys.	Dimension: data [FIPS 197]
Word size, Word length		Size of the data objects in each column of the state array.	Dimension: data [FIPS 197]
Number of rounds	*Nr*	Number of iterations of the work within the cipher algorithm (including the final iteration, which is a special case).	Unit: iterations [FIPS 197]
Key recovery resistance		Expected amount of work, storage, and/or input data required to determine the cipher key.	Dimensions: work, data, data
Plaintext recovery resistance		Expected amount of work, storage, and/or input data required to determine the plaintext.	Dimensions: work, data, data
Distinguishing resistance		Expected amount of work, storage, and/or input data required to distinguish encrypted data from random data.	Dimensions: work, data, data

6.4 Cyclomatic complexity

Cyclomatic complexity is derived from the control flow graph of a program. It is based on the more generic cyclomatic number from graph theory (see Sec. 4.3).

As specified in [McCabe, Section V], compound predicates such as "if C1 and C2" and case statements should be reduced to simple conditionals, "if C1 then if C2 then," before counting. Failure to note this apparently gave rise to so-called *extended* cyclomatic complexity, which in fact merely corrects for a faulty reading of the original quantity.

The only primary source that is cited for extended cyclomatic complexity is [MyersCC]. This reference is problematic for two reasons. First, it asserts that there is ambiguity about how compound predicates should be counted, without mentioning the specification in [McCabe, Section V]. Second, what it proposes is not a scalar metric, as [Welker] and [Oy] assume it to be, but an interval that covers both the greedy and conservative methods of counting.

Names	Symbols	Definitions	References
Cyclomatic complexity	V(G), v(G), CC	The number of linearly independent **paths** in a control flow graph, derived as $e - n + 2p$ Where: e is number of **edges** n is number of **nodes** p is number of **connected components** Alternate derivation: $d + 1$ Where d is the number of **predicate nodes**.	Unit: path The scale is ratio as long as V(G) is treated only as a count of linearly independent paths. As a measure of complexity, the scale is ordinal. [McCabe] See also, cyclomatic number in Sec. 4.3.
"Extended" cyclomatic complexity	V(g'), v(g'), CC2, VG2	= V(G)	See section comments above.

6.5 Woodward, Hennell, and Hedley complexity

This measure of control flow complexity is most accurately derived from source code or pseudocode that has been annotated with arrows corresponding to jumps. If it is derived from a control flow graph instead, only upper and lower bounds can be calculated.

Names	Symbols	Definitions	References
Woodward, Hennell, and Hedley complexity		Number of **knots** in the graph that results from drawing arrowed lines on one side of the source code indicating where a jump occurs from one line of code to another.	Unit: knot [Woodward]

7 General design and implementation metrics

7.1 Generic quantities

Names	Symbols	Definitions	References
Fanin, Fan-in		(1) The number of **calls** (source code entity, not event) to a given **module**. (2) "Number of local flows into procedure *A* plus the number of data structures from which procedure *A* retrieves information."	(1) Unit: call (2) Unit: data flow [Henry]

Names	Symbols	Definitions	References
Fanout, Fan-out		(1) The number of **calls** (source code entity, not event) from a given **module**. (2) "Number local flows from procedure *A* plus the number of data structures which procedure *A* updates."	(1) Unit: call (2) Unit: data flow [Henry]
Number of entries	e_i	"Number of **entry points** for the *i*th **module**." [9]	Unit: entry point [IEEE-982.1-1988]
Number of exits	x_i	"Number of **exit points** for the *i*th **module**." [9]	Unit: exit point [IEEE-982.1-1988]
Number of entries and exits	m_i	$= e_i + x_i$ [9]	Unit: statement [IEEE-982.1-1988]
Defect density		$\frac{\text{Number of known defects}}{\text{Product size}}$	Unit: defect/entity, where entity may be e.g. LOC, class, module, function point, etc. [Fenton, p. 450]
Depth of nesting		Number of loop statements, conditional statements, and scoping blocks within which a statement is enclosed.	Unit: statement [Conte, p. 75]
Span, Reference span		"Number of **statements** between two textual references to the same identifier."	Unit: statement [Elshoff]
Locality of data	LD	Proportion of **variables** accessed by a class or module that are local to that class or module, "excluding all trivial read/write methods for instance variables." Generalized from object-oriented definition. [Hitz] defines "local" as "non-public instance variables of class C, inherited protected instance variables of its superclasses, and static variables defined locally" in the methods.	Unit: variable/variable [Hitz]

7.2 Belady and Evangelisti clustering complexity metric

The first two equations have been rearranged for clarity.

Names	Symbols	Definitions	References
Total complexity	C	$= NE_0 + \sum_{j=1}^{K} n_j e_j$ where K = number of clusters (packages) N = number of nodes (modules) n_j = number of nodes in *j*th cluster e_j = number of intracluster edges (relationships) in *j*th cluster E_0 = number of intercluster edges	Unit: module · relationship [Belady]
Normalized complexity	$\bar{C}$	$= \frac{C}{NE} = \frac{E_0}{E} + \sum_{j=1}^{K} \left(\frac{n_j}{N}\right)\left(\frac{e_j}{E}\right)$ (* Corrected apparent typo in [Belady] where e_j was normalized by N instead of E.)	Unit: (module · relationship)/ (module · relationship) [Belady]
Approximate complexity	$\tilde{C}$	Substituting n_j = N/K in $\bar{C}$ produces $\tilde{C} = \frac{1}{K}\frac{E_i}{E} + \frac{E_0}{E}$ where E_i = total number (over all clusters) of intracluster edges	Unit: (module · relationship)/ (module · relationship) [Belady]

7.3 Henry and Kafura information flow complexity metric

Names	Symbols	Definitions	References
Information flow complexity	IFC	(Of a **procedure**) = $(\text{fan-in} \cdot \text{fan-out})^2$ "The complexity of a **module** is defined to be the sum of the complexities of the procedures within the module."	Unit: $(\text{data flow})^4$ or ordinal [Henry]
Weighted information flow complexity	Weighted IFC	(Of a **procedure**) = LOC · $(\text{fan-in} \cdot \text{fan-out})^2$ "This measure includes imbedded comments but does not include comments preceding the procedure statement."	Ordinal [Henry]

7.4 Cruickshank and Gaffney coupling metric

Names	Symbols	Definitions	References
Coupling		$= \frac{\sum_{i=1}^{n} Z_i}{n}$ where $Z_i = \frac{\sum_{j=1}^{m} M_j}{m}$ M_j = sum of the number of input and output items shared between components *i* and *j* Z_i = average number of input and output items shared over *m* components with component *i* *n* = number of components in the software product	Unit: data item [IR5459, p. 19] Less formally, in [Cruickshank]

7.5 Structured Design scales of coupling and cohesion

The ordinal scale of cohesion that was called *binding* in [StevensWP] evolved, grew, and was forked into two different scales for the same measurand. Versions of both resulting scales are provided in consecutive rows below.

Names	Symbols	Definitions	References
Binding, Module cohesion		From most cohesive (best) to least cohesive (worst), with "magic," "artificial values" from [Yourdon, p. 136]: • Functional = 10: "every element of processing is an integral part of, and is essential to, the performance of a single function" • Sequential = 9: "the output data (or results) from one processing element serve as input data for the next processing element" • Communicational = 7: "all of the elements operate upon the same input data set and/or produce the same output data" • Procedural = 5: elements of a module are "elements of a common procedural unit" • Temporal = 3: "all occurrences of all elements of processing in a collection occur within the same limited period of time during the execution of the system" (such as a start-up module) • Logical = 1: elements of a module fall into the same logical class of similar or related functions • Coincidental = 0: "little or no constructive relationship among the elements of a module"	Ordinal [Yourdon]

Names	Symbols	Definitions	References
Module strength, Module cohesion		From most cohesive (best) to least cohesive (worst): • Functional: "performs a single specific function," and/or Informational: "1. It contains multiple entry points. 2. Each entry point performs a single specific function. 3. All of the functions are related by a concept, data structure, or resource that is hidden within the module. 4. There are no control-flow connections among the logic for each function." • Communicational: "performs multiple sequential functions, where the sequential relationship among all of the functions is implied by the problem or application statement, and where there is a data relationship among all of the functions" • Procedural: "performs multiple sequential functions, where the sequential relationship among all of the functions is implied by the problem or application statement" • Classical: "performs multiple sequential functions where there is a weak, but nonzero, relationship among all of the functions" • Logical: "performs a set of related functions, one of which is explicitly selected by the calling module" • Coincidental: a module whose function cannot be defined or that performs multiple, completely unrelated functions	Ordinal [MyersSD]
Module coupling		From least coupling (best) to most coupling (worst): • No direct coupling: none of the below • Data: the modules directly communicate and use only "homogenous data items"* to do so • Stamp: "reference the same nonglobal data structure" • Control: "one module explicitly controls the logic of the other" • External: reference a "homogenous global data item"* • Common: reference a global data structure like a Fortran blank common block • Content: one directly references the internals of the other or the normal linkage conventions are bypassed * Typical modern variables that are consistently, unambiguously named and typed when referenced in different modules would be considered "homogenous."	Ordinal [MyersSD]

7.6 Embley and Woodfield scales of coupling and cohesion

The metrics of [Embley] follow in the footsteps of the previous section, but are applied to abstract data types (ADTs) rather than modules.

Names	Symbols	Definitions	References
(ADT) Cohesion, Strength		From most cohesive (best) to least cohesive (worst): • Model: "1. logically exports one and only one domain D, 2. logically exports only operations that apply to D and should not be delegated to other ADTs, and 3. does not contain a concealed ADT" • Concealed: "does not have non-delegation, multifaceted, or separable strength and it logically contains a hidden ADT" • Non-delegation: "does not have multifaceted or separable strength and it includes an operator that should logically be delegated to a more-primitive ADT" • Multifaceted: "does not have separable strength and it logically exports two or more domains" • Separable: "if any one of the following conditions holds: 1. There exists a logically-exported operator p of A such that p does not utilize any logically-exported domain of A. 2. A has two or more logically-exported domains, at least one of which is not utilized by any operator of A. 3. A has two or more logically-exported domains D1, D2, ..., Dn and the operators of A can be partitioned into n blocks such that the operators in block i, $1 \leq i \leq n$, utilize Di and only Di"	Ordinal [Embley]
(ADT) Coupling		From least coupling (best) to most coupling (worst): • Export: "1. no function of A1 accesses the implementation of A2 and 2. no function of A1 makes any assumption about the implementation of A2" • Surreptitious: "(A1,A2) does not have visible coupling, but A1 uses knowledge about the implementation of A2" • Visible: "A1 accesses the implementation of A2"	Ordinal [Embley]

7.7 Briand, Morasca, and Basili metrics

Model:

Symbols	Definitions	References
CI(c)	For a module or class *c*, the set of all **CI**s.	[Briand98]
Max(c)	The set of all possible or potential **CI**s (combinatorically).	[Briand98]

Symbols	Definitions	References
K(c)	The set of **CI**s that are known to exist.	[Briand98]
U(c)	The set of **CI**s whose existence or non-existence is unknown.	[Briand98]
Global(m)	"The set of all the external data declarations imported by a module *m*."	[Briand93]
Local(m)	"The set of all the locally defined data declarations in module *m*."	[Briand93]
Scope(m)	"The set of all data declarations declared outside the module for which the internal data declarations of module *m* are visible."	[Briand93]
DD-interactions(m,n)	Number of **DD-interactions** between *m* and *n*.	[Briand93]

Metrics:

Names	Symbols	Definitions	References
Ratio of cohesive interactions	RCI	$RCI(c) = \frac{\lvert CI(c)\rvert}{\lvert Max(c)\rvert}$	Unit: CI/CI [Briand98]
Neutral ratio of cohesive interactions	NRCI	$NRCI(c) = \frac{\lvert K(c)\rvert}{\lvert Max(c)\rvert - \lvert U(c)\rvert}$	Unit: CI/CI [Briand98]
Pessimistic ratio of cohesive interactions	PRCI	$PRCI(c) = \frac{\lvert K(c)\rvert}{\lvert Max(c)\rvert}$	Unit: CI/CI [Briand98]
Optimistic ratio of cohesive interactions	ORCI	$ORCI(c) = \frac{\lvert K(c)\rvert + \lvert U(c)\rvert}{\lvert Max(c)\rvert}$	Unit: CI/CI [Briand98]
Import coupling	IC	IC(m) = DD-interactions(Global(m), Local(m)) (Including both direct and transitive interactions.) For generic modules: "The import coupling of a generic module is the cardinality of the union of the sets of DD-interactions between the data declarations in the software system and those of each of its instances." [Briand94, p. 20]	Unit: DD-interaction [Briand93, Briand94]

Names	Symbols	Definitions	References
Actual export coupling	EC-Actual	EC-Actual(m) = DD-interactions(Local(m), Scope(m)) (Including both direct and transitive interactions.) For generic modules: "When calculating export coupling, we take into account the DD-interactions between the data declarations of each of its instances and those of the software system. Consistent with the definition of DD-interaction, generic formal parameters DD-interact with their particular generic actual parameters (i.e. type, object) when the generic module is instantiated, since a change in the former may imply a change in the latter." [Briand94, p. 20]	Unit: DD-interaction [Briand93, Briand94]
Potential export coupling	EC-Potential	EC-Potential(m) = |Local(m)| · |Scope(m)|	Unit: DD-interaction [Briand93]
Relative dependency	RD	RD(m) = IC(m) / (DD-interactions(Local(m), Local(m)) + IC(m))	Unit: DD-interaction/DD-interaction [Briand93]
Coupling type	CT	CT(m) = IC(m)/(EC-Actual(m) + IC(m))	Unit: DD-interaction/DD-interaction Can be reduced to a dichotomic scale as follows: < 0.5 server ≥ 0.5 client [Briand93]
Visibility control	VC	"The visibility control of a set of modules SM (VC(SM)) is measured by means of the Spearman's rank correlation coefficient between the actual Export Coupling and the potential Export Coupling." $VC(SM) = 1 - \frac{\sum_{m \in SM}(D(m))^2}{\|SM\|(\|SM\|^2 - 1)/6}$ where D(m) = Rank(EC-Actual(m)) − Rank(EC-Potential(m))	Unit: 1 [Briand93]

7.8 Halstead system

Defined input quantities:

Names	Symbols	Definitions	References
Stroud number	S	18 Halstead consistently sets S = 18 "elementary discriminations" per second. This value came from [Halstead], not [Stroud].	Unit: operation/s [Halstead]
Unique operator count	η_1	"Number of unique or distinct **operators**" appearing in an implementation. Secondary sources may replace the Greek η with n [Abran p. 146] or even μ [Fenton p. 345].	Unit: operator [Halstead pp. 2, 6]
Total operators	N_1	"Total usage of all of the **operators**" appearing in an implementation.	Unit: operator [Halstead pp. 2, 6]
Unique operand count	η_2	"Number of unique or distinct **operands**" appearing in an implementation. Secondary sources may replace the Greek η with n [Abran p. 146] or even μ [Fenton p. 345].	Unit: operand [Halstead pp. 2, 6]
Total operands	N_2	"Total usage of all of the **operands**" appearing in an implementation.	Unit: operand [Halstead pp. 2, 6]
Potential operand count	η_2^*	"Number of conceptually unique **operands**"	Unit: operand [Halstead pp. 20, 28]

The following is not an exhaustive list of Halstead's measures, but includes those that are used as input quantities by other metrics in this document.

Chapter 8 of [Halstead] reinterpreted several quantities to refer to mental operations instead of bits. To avoid confusion, these should have been defined as new quantities that were just numerically equal to the previous ones when expressed in incompatible units.

Names	Symbols	Definitions	References
Vocabulary size	η	$\eta_1 + \eta_2$	Unit: token [Halstead p. 2]
Potential vocabulary	η^*	$2 + \eta_2^*$	Unit: token [Halstead p. 2]
Program length	N	$N_1 + N_2$	Unit: token [Halstead p. 2]
Program volume	V	$N \log_2 \eta$	Unit: bit [Halstead p. 2] “Mental comparisons” [Halstead pp. 46–47]

Names	Symbols	Definitions	References
Potential volume	V*	$\eta^* \log_2 \eta^*$ Halstead uses "potential" in the sense of hypothetical ideal or optimal value; e.g., "the most succinct form in which an algorithm could ever be expressed" in any programming language that one might construct.	Unit: bit [Halstead p. 2]
Effort	E	$V/L = V^2/V^* = D \cdot V$	Unit: bit [Halstead p. 2] “Elementary mental discriminations” [Halstead p. 47]
[Estimated] implementation time	T, $\hat{T}$	$\frac{E}{S}$ Where *S* is the Stroud number defined above.	Unit: s (Assuming 1 "elementary discrimination" = 1 bit) [Halstead pp. 2, 48, 52]
Program level	L	V*/V [Halstead p. 47] reinterprets L as a ratio of “mental comparisons” to “elementary mental discriminations.” This is difficult to reconcile.	Unit: bit/bit [Halstead p. 2]
Difficulty	D	1/L	Unit: bit/bit [Halstead p. 2]
Approximated program level	$\hat{L}$	$\frac{2}{\eta_1}\frac{\eta_2}{N_2}$	Unit: $(\text{token/token})^2$ [Halstead pp. 2, 27]

7.9 Functional size (of a software application)

Functional size is generically defined in [ISO-FSM] as "size of the software derived by quantifying the Functional User Requirements." There are multiple standards for how this is determined. The details of these quantities have been elided; please refer to the relevant standards.

Names	Symbols	Definitions	References
Application function point count	AFP	Sum of the functional sizes of all **BFC**s (see Sec. 2.6). The functional size of a BFC is a table-driven function of the numbers of **RET**s or **FTR**s (for data and transactional functions respectively) and **DET**s, and of the function type.	FP (ordinal) [ISO-IFPUG] [ISO-NESMA]
Automated function point size	AFPs	Variant of ibid. in which the input quantities are determined automatically from source code and other artifacts.	AFP (ordinal) [AFP]

Names	Symbols	Definitions	References
COSMIC functional size	FS	Sum of the functional sizes of all **BFC**s. The functional size (FS) of a BFC is the number of **data movements** (entries, exits, reads, and writes).	CFP [ISO-COSMIC]
FiSMA functional size	S	Sum of the functional sizes of all **BFC**s. The functional size of a BFC is a function of the numbers of **DET**s, **reading references**, **writing references**, and **operations**, and of the BFC type.	Ffp (ordinal) [ISO-FiSMA]
MkII functional size	FS	"The weighted sum over all Logical Transactions, of the Input Data Element Types (Ni), the Data Entity Types Referenced (Ne), and the Output Data Element Types (No)." = 0.58 Ni + 1.66 Ne + 0.26 No [4]	MkII FP (ordinal) [ISO-MkII]

7.10 Card and Glass complexity metrics

Defined input quantities:

Names	Symbols	Definitions	References
	n	The number of **modules** in the system.	Unit: module [Card, Ch. 5]
	f(*i*)	The **fanout** of module *i*. "The fanout count defined here does not include calls to system or standard utility routines, but does include calls to modules reused from other application programs."	Unit: call [Card, Ch. 5]
	v(*i*)	The number of input/output (I/O) **variables** in module *i*. I/O variables means "distinct arguments in a calling sequence (an array counts as one variable) as well as referenced COMMON variables."	Unit: variable [Card, Ch. 5]

Metrics:

Names	Symbols	Definitions	References
System complexity (overall)	Ct	Ct = St + Dt	Ordinal [Card, Eqn. 5-1]
Structural (intermodule) complexity	St	Inferred from Eqn. 5-2 and 5-3 that $St = \sum f^2(i)$	Unit: call^2 [Card, Ch. 5]
Data (intramodule) complexity	Dt	Inferred from Eqn. 5-2 and 5-4 that $Dt = \sum \left\{ \frac{v(i)}{[f(i)+1]} \right\}$	Unit: variable/call [Card, Ch. 5]
Relative system complexity	C	$C = \frac{Ct}{n} = \frac{St}{n} + \frac{Dt}{n}$	Ordinal [Card, Eqn. 5-2]

Names	Symbols	Definitions	References
[Relative] structural (intermodule) complexity	S	$S = \frac{\sum f^2(i)}{n}$	Unit: call^2/module [Card, Eqn. 5-3]
Data complexity [of a module]	D(i)	$D(i) = \frac{V(i)}{f(i)+1}$	Unit: variable/call [Card, Ch. 5] unnumbered equation
[Relative] data (intramodule) complexity	D	$\frac{\sum \left\{ \frac{v(i)}{[f(i)+1]} \right\}}{n}$	Unit: (variable/call) /module [Card, Eqn. 5-4]

7.11 Program Complexity Analysis Methodology (PCAM) metrics

The following tables are based primarily on [McClure1]. The definitions in [McClure2] are mostly equivalent, just using different symbols. However, there are a few substantive differences:

- [McClure2] contains a more elaborate definition of module complexity (M) that addresses abort routines. The input quantity Z_m is used only by the [McClure2] definition.
- Both references assert that the complexity of each module should be minimized and that the complexity among modules in a well-structured program should be evenly distributed. However, an example in which these criteria are tested using the mean module complexity and the maximum deviation from that mean appears only in [McClure2, p. 119]. [McClure1], in contrast, proceeds to define partitioning scheme complexity (PS) as the sum of module complexities.

Defined inputs:

Names	Symbols	Definitions	References
	P	= {p_1, ..., p_n} is the set of modules defined in the design of a well-structured program	[McClure1]
	γ	The root module	[McClure1]
	f	f : P → X is the invoking function such that $X \subset P$	[McClure1]
Program control hierarchical system	PCHS	The triple (P, γ, f)	[McClure1]
	n	Number of unique modules in the PCHS	[McClure1]
	s	Total number of control variables	[McClure1]
Invocation control variable set		Set of control variables upon whose values a particular invocation of a module depends	[McClure1]
Branch control variable set		Set of control variables upon whose values a branch may be made to an abort routine	[McClure2]
	F_{p_i}	Set of direct ancestors of module p_i	[McClure1]

Names	Symbols	Definitions	References
	G_{p_i}	Set of direct descendants of module p_i	[McClure1]
	H_{p_i}	Set of ancestors of module p_i	[McClure1]
	L_{p_i}	Set of descendants of module p_i	[McClure1]
	A_v	Set of modules in which the value of control variable v is accessed, $= M_v \cup R_v$	[McClure1]
	E_v	Set of modules whose invocation is dependent upon the value of control variable v	[McClure1]
	M_v	Set of modules in which the value of control variable v is modified	[McClure1]
	R_v	Set of modules in which the value of control variable v is strictly referenced (i.e., referenced but not modified)	[McClure1]
	T_v	$= \left\{ p_j \mid p_j \in A_v \land \exists p_k \in F_{p_j} \ni p_k \notin A_v \right\}$ (The symbol ∋ after an existential denotes "such that.")	[McClure1]
	U_v	$= \left\{ p_j \mid p_j \in R_v \land \exists p_k \in L_{p_j} \ni p_k \in M_v \right\}$	[McClure1]
	W_v	$= \{ p_i \mid p_i \in A_v \land \exists p_j \in M_v \ni p_j \notin H_{p_j}$ and p_j is listed above p_i in the PCHS structure$\}$	[McClure1]
Owner module	α_v	"Local root of the smallest PCHS subhierarchy which contains all members of the set A_v"	[McClure1]
Degree of ownership	D(v)	= 1 if the value of control variable v is modified exclusively in α_v or never modified 2 if it is modified in α_v and in at least one descendant of α_v 3 if it is strictly referenced in α_v and modified in at least one descendant of α_v 4 if it is not accessed in α_v and is modified in at least one descendant of α_v	[McClure1]

Quantities:

Names	Symbols	Definitions	References
Interaction complexity	I(v)	$= q_v + u_v + w_v + t_v$ where $q_v = \lvert M_v \cap E_v \rvert$ $u_v = \lvert U_v \rvert$ $w_v = \lvert W_v \rvert$ $t_v = \lvert T_v \rvert$	Unit: module [McClure1]
Control variable complexity	C(v)	$C(v) = D(v) \cdot I(v) / n$	Ordinal Range: $0 \leq C(v) < 8$ [McClure1]

Names	Symbols	Definitions	References
	X(p)	If x=0, X(p) = 0; else, $$X(p) = \sum_{j=1}^{x} \frac{b_j \cdot \sum_{i=1}^{e} C(v_{ji})}{x}$$ where x = number of invocation control variable sets used in the invocation of module p e = number of control variables in the j^{th} invocation control variable set where $1 \le j \le x$ v_{ji} = the i^{th} control variable in the j^{th} invocation control variable set where $1 \le i \le e$ b_j = 1 if the j^{th} invocation is within a selection structure or a set of nested selection structures, or 2 if the j^{th} invocation is within a repetition structure	Ordinal [McClure1]
	Y(p)	If y=0, Y(p) = 0; else, $$Y(p) = \sum_{j=1}^{y} \frac{b_j \cdot \sum_{i=1}^{k} C(v_{ji})}{y}$$ where y = number of invocation control variable sets referenced by module p to invoke its direct descendants k = number of control variables in the j^{th} invocation control variable set where $1 \le j \le y$ v_{ji} = the i^{th} control variable in the j^{th} invocation control variable set for $1 \le i \le k$ b_j = 1 if the j^{th} invocation is within a selection structure or a set of nested selection structures, or 2 if the j^{th} invocation is within a repetition structure	Ordinal [McClure1]
	Z_m	If z=0, Z_m = 0; else, $$Z_m = \frac{\sum_{j=1}^{z} \sum_{i=1}^{a} C(v_{ji})}{z}$$ where z = number of branch control variable sets used to branch from module m to abort routines a = number of control variables in the j^{th} branch control variable set where $1 \le j \le z$ v_{ji} = the i^{th} control variable in the j^{th} branch control variable set where $1 \le i \le a$	Ordinal [McClure2]

Names	Symbols	Definitions	References
Module complexity	M(p)	$M(p) = f_p \cdot X(p) + g_p \cdot Y(p)$ where $f_p = \lvert F_p \rvert$ and $g_p = \lvert G_p \rvert$.	Ordinal Range: $0 \le M(p) < 16$ s n [McClure1] (See defined inputs above for s and n)
Module complexity	M(m)	$M(m) = T_1 + T_2 + T_3$ where T_1 and T_2 are equivalent to the two terms of M(p) above and $T_3 = B_m \times Z_m$ where B_m is the number of abort routines to which module m may branch.	Ordinal Range: $0 \le M(m) <$ 16 s n [McClure2]
Partitioning scheme complexity	PS	$= \sum_{i=1}^{n} M(p_i)$	Ordinal [McClure1]

7.12 Maintainability index

The maintainability indices referenced by Software Engineering Institute (SEI) and Microsoft are derived from what has been called the Coleman-Oman model [Liso]. A survey of publications authored by Don Coleman, Paul Oman, and their associates between 1993 and 1997 (see list in Coleman-Oman maintainability model subsection following the Bibliography) revealed 13 different candidate definitions for this metric.

Names	Symbols	Definitions	References
Maintainability index (SEI version)	MI	$171 - 5.2 \cdot \ln(\text{aveV}) - 0.23 \cdot \text{aveV}(g') - 16.2 \cdot \ln(\text{aveLOC}) + 50 \cdot \sin\left(\sqrt{2.4 \cdot \text{perCM}}\right)$ where aveV = average Halstead V per module aveV(g') = average extended cyclomatic complexity per module aveLOC = average LOC per module; and, **optionally**, perCM = average "percent"* of lines of comments per module (* actually the proportion aveCMT/aveLOC; see footnote 2 in [Oman]) In [Coleman], the input quantities were averaged by submodule, defined as "function or procedure." In the same context, [Welker] wrote: "This paper	Ordinal [SEI] [Welker, Eqn. 5]

<table>
<tr><th>Names</th><th>Symbols</th><th>Definitions</th><th>References</th></tr>
<tr><td></td><td></td><td>uses the term 'subroutine' or 'module' for any named lexical component of a program, which given a specific programming language might be a function, a procedure, a subroutine, a section, a module, etc."</td><td></td></tr>
<tr><td>Maintainability index (Microsoft version)</td><td></td><td>$$\max\left(0, (171 - 5.2 \cdot \ln(\mathrm{V}) - 0.23 \cdot \mathrm{V(G)} - 16.2 \cdot \ln(\mathrm{LOC})) \cdot \frac{100}{171}\right)$$
The metric is calculated for each type or method instead of using averages as inputs.</td><td>Ordinal
Range: 0 to 100
The scale is further reduced to a 3-level ordinal scale as follows:<table><tr><td>20 to 100</td><td>green/high/ good</td></tr><tr><td>10 to 19</td><td>yellow/ moderate</td></tr><tr><td>0 to 9</td><td>red/low/bad</td></tr></table>[Microsoft]</td></tr>
</table>

7.13 Maturity index

<table>
<tr><th>Names</th><th>Symbols</th><th>Definitions</th><th>References</th></tr>
<tr><td>Software maturity index</td><td>SMI</td><td>[The available definition is equivocal over whether the units to be counted are functions or modules.]
$$= \frac{M_T - (F_a + F_c + F_{del})}{M_T}$$
where
M_T = number of software functions (modules) in the current delivery
F_c = number of software functions (modules) in the current delivery that include internal changes from a previous delivery
F_a = number of software functions (modules) in the current delivery that are additions to the previous delivery
F_{del} = number of software functions (modules) in the previous delivery that are deleted in the current delivery [9]</td><td>Unit: function/function or module/module
Range: $-\infty$ to 1
(Negative values due to large F_{del} may have been unintentional)
[IEEE-982.1-1988]
The origin of this metric is unknown; no citation was given.</td></tr>
</table>

Names	Symbols	Definitions	References
Estimated software maturity index	SMI	If F_a and F_{del} are unavailable, SMI may be estimated as $= \frac{M_T - F_c}{M_T}$ 9	Unit: function/function or module/module Range: 0 to 1 [IEEE-982.1-1988] The origin of this metric is unknown; no citation was given.

8 Object-oriented design and implementation metrics

8.1 Eder, Kappel, and Schrefl scales of coupling and cohesion

[Eder] adapted the earlier, general, ordinal scales of coupling and cohesion to object-oriented software, resulting in six ordinal-scaled metrics.

Names	Symbols	Definitions	References
Interaction coupling		(Based on module coupling, Sec. 7.5) From most coupling (worst) to least coupling (best): • Content: "One method directly accesses parts of the internal structure, i.e., the implementation of another method." • Common: "Methods communicate via an unstructured, global, shared data space." • External: Methods communicate via a structured, global, shared data space. • Control: Methods "communicate exclusively via parameter passing... but one method controls the internal logic of the other method." • Stamp: "Whole data structures are passed as parameters although only parts of the data structure would suffice." • Data: Methods "communicate only by parameters and these parameters are relevant as a whole." • No direct coupling: "Two methods do not (directly) depend on each other."	Ordinal [Eder]

Names	Symbols	Definitions	References
Component coupling		From most coupling (worst) to least coupling (best): • Hidden: "C′ shows up neither in the specification nor in the implementation of C, although an object of C′ is used in the implementation of a method of C." • Scattered: "C′ is used as domain in the definition of some local variable or instance variable in the implementation of C yet C′ is not included in the specification of C." • Specified: "C′ is included in the specification of C whenever it is a component of C." • Nil: "No direct component coupling."	Ordinal [Eder]
Inheritance coupling		From most coupling (worst) to least coupling (best): • Modification: "Inherited information is changed arbitrarily or is even deleted." Includes signature and implementation modification. • Refinement: "Inherited information is only changed due to predefined rules." Includes signature and implementation refinement. • Extension: "The subclass only adds methods and instance variables but neither modifies nor refines any of the inherited ones." • Nil: "No inheritance relationship."	Ordinal [Eder]

Names	Symbols	Definitions	References
Method cohesion		(Based on module cohesion, Sec. 7.5.) From lowest (worst) to highest (best): • Coincidental: "The elements of a method have nothing in common besides being within the same method." • Logical: "The elements with similar functionality, such as input/output handling and error handling, are collected in one method." • Temporal: "The elements of a method have logical cohesion and are performed at the same time." • Procedural: "The elements of a method are connected by some control flow." • Communicational: "The elements of a method are connected by some control flow and operate on the same set of data." • Sequential: "The elements of a method have communicational cohesion and are connected by a sequential control flow." • Functional: "The elements of a method have sequential cohesion, and all elements contribute to a single task of the problem domain."	Ordinal [Eder]
Class cohesion		(Based on ADT cohesion, Sec. 7.6.) From lowest (worst) to highest (best): • Separable: The class represents multiple unrelated abstractions that could easily be partitioned. • Multifaceted: The class represents multiple unrelated abstractions, but "at least one method references instance variables or invokes methods of the different semantic concepts, such that the cohesion of the corresponding class cannot be rated separable." • Non-delegated: "One method uses instance variables which describe only a component of the respective class." • Concealed: "There exists some useful data abstraction concealed in the data abstraction represented by the class." • Model: "The class represents a single, semantically meaningful concept without containing methods which should be delegated to other classes and without containing concealed classes."	Ordinal [Eder]

Names	Symbols	Definitions	References
Inheritance cohesion		"Inheritance cohesion in strong if this hierarchy is a generalization hierarchy in the sense of conceptual modeling, and it is weak if the inheritance hierarchy is merely used for code sharing among otherwise unrelated classes." Uses the same ordinal scale as class cohesion.	Ordinal [Eder]

8.2 Martin's package metrics

Names	Symbols	Definitions	References
Relational cohesion	H	$= \frac{R+1}{N}$ where R is "the number of class relationships that are internal to the package (i.e., that do not connect to classes outside the package)" and N is "the number of classes within the package."	Unit: relationship/class [Martin, p. 282]
Afferent couplings	C_a	"The number of **classes** outside this **package** that depend on **classes** within this package."	Unit: class [Martin, pp. 262, 282]
Efferent couplings	C_e	"The number of **classes** inside this **package** that depend on **classes** outside this **package**."	Unit: class [Martin, pp. 262, 282]
Instability	I	$I = \frac{C_e}{C_a + C_e}$ (I.e., the proportion of class dependencies that are efferent.)	Unit: class/class Range: 0 (maximally stable) to 1 (maximally unstable) [Martin, pp. 262, 282]
Abstractness	A	$A = \frac{N_a}{N_c}$ Where N_c is the number of **classes** in the **package** and N_a is the number of abstract **classes** in the **package**. (I.e., the proportion of classes that are abstract.)	Unit: class/class [Martin, pp. 265, 282]
Distance [from the main sequence]	D	$D = \frac{\lvert A + I - 1\rvert}{\sqrt{2}}$	Unit: class/class Range: 0 (best) to $1/\sqrt{2}$ (worst) [Martin, pp. 266, 282]
Normalized distance	D'	$D' = \lvert A + I - 1\rvert$	Unit: class/class Range: 0 (best) to 1 (worst) [Martin, pp. 267, 282]

[Fenton] proposed a redefinition of relational cohesion (which he calls RC(P) instead of H):

Names	Symbols	Definitions	References
Relational cohesion	RC'(P)	$= \frac{R(P)+1}{NP(P)}$ where R(P) is "the number of relations between classes and interfaces in a package" and NP(P) is "the number of possible relations between classes and interfaces in the package." On p. 420, NP(P) is defined to be $N(P) \cdot (N(P) - 1)$; the absence of a factor of 2 in the denominator suggests that the relations are considered one-directional.	Unit: relationship/relationship [Fenton, pp. 420–421]

8.3 Chidamber and Kemerer class metrics

A suite of six object-oriented class metrics (metrics applied to a given class) is defined in [Chidamber]. These metrics are defined differently in earlier publications by the same authors.

The weighted methods per class (WMC) metric is excluded because its definition is incomplete (see Section 1.2). The authors wrote, "Complexity is deliberately not defined more specifically here in order to allow for the most general application of this metric" [Chidamber, footnote 13].

Names	Symbols	Definitions	References
Depth of inheritance tree	DIT	The length of the longest path from a given class to the root of the inheritance tree (a.k.a. depth of node, Sec. 4.3).	Unit: arc (graph entity) [Chidamber]
Number of children	NOC	"Number of immediate **subclasses** subordinated to a class in the class hierarchy."	Unit: class [Chidamber]
Coupling between object classes	CBO	"CBO for a class is a count of the number of other **classes** to which it is coupled." "Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class."	Unit: class [Chidamber]
Response for a class	RFC	The cardinality of the response set RS, "the set of methods that can potentially be executed in response to a message received by an object of that class," only up to the first level of nesting of method calls. $RS = \{M\} \bigcup_{\text{all } i} \{R_i\}$ where $\{R_i\}$ = set of methods called by method *i* and {M} = set of all methods in the class.	Unit: method [Chidamber]

Names	Symbols	Definitions	References
Lack of cohesion in methods	LCOM	For a class with n methods $M_1...M_n$, let $\{I_i\}$ = set of instance variables used by method M_i. $P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset\}$ $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset\}$ $LCOM = \lvert P\rvert - \lvert Q\rvert$, if $\lvert P\rvert > \lvert Q\rvert$ $= 0$ otherwise "The LCOM value provides a measure of the relative disparate nature of methods in the class. ... Lack of cohesion implies classes should probably be split into two or more subclasses."	Unit: method [Chidamber]

Revised definitions of LCOM proposed by other authors also are in use:

Names	Symbols	Definitions	References
Hitz & Montazeri LCOM	LCOM(X)	Let X denote a class, I_X the set of its instance variables, and M_X the set of its methods. Consider a simple, undirected graph $G_X(V,E)$ with $V=M_X$ and $E = \left\{\langle m,n\rangle \in V \times V \,\middle\vert\, \begin{pmatrix}\exists i \in I_X\text{: (m accesses i)} \\ \wedge \text{ (n accesses i)}\end{pmatrix} \vee \text{(m calls n)} \vee \text{(n calls m)}\right\}$. LCOM(X) is then defined as the number of **connected components** of G_X ($1 \leq$ LCOM(X) $\leq \lvert M_X\rvert$).	Unit: connected component (graph entity) [Hitz]
First version of LCOM*	LCOM*	For a set of methods $\{M_i\}$ (i = 1..m) accessing a set of attributes $\{A_j\}$ (j = 1..a), let $\alpha(M_i)$ be the number of attributes accessed by M_i and let $\mu(A_j)$ be the number of methods that access A_j. Then LCOM* = $\dfrac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right)\left(\frac{1}{m}\sum_{i=1}^{m}\alpha(M_i)\right) - am}{1 - am}$	Unit: (attribute · method) / (attribute · method) Range: 0 (full cohesion) to 1 (no cohesion) [Henderson-Sellers]
Second version of LCOM*	LCOM*	Using the same definitions as ibid., LCOM* = $\dfrac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1 - m}$	Unit: method/method Range: 0 (full cohesion) to 1 (no cohesion) [Henderson-Sellers]

8.4 Bieman and Kang cohesion metrics

[Bieman] defines general and local versions of two cohesion metrics, Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC), which depend on an abstract model and several input quantities. The items in the abstract model are sets and multisets:

Names	Symbols	Definitions	References
Abstracted method	AM(M)	"A method is represented as a set of instance **variables** directly or indirectly used by the method." For the purposes of defining the metrics to follow, it suffices that AM(M) is the model's proxy for a **method**, and that constructors and destructors are not included in the model.	[Bieman]
	V(C)	"V(C) is a set of all visible **methods** in class C and the ancestor classes of C."	[Bieman]
	LV(C)	"LV(C) are the visible **methods** defined within class C."	[Bieman]
Abstracted class	AC(C)	"A collection of AM's where each AM corresponds to a visible **method** in the **class**." $AC(C) = [\![AM(M) \mid M \in V(C)]\!]$ The double-bracket notation denotes a multi-set that may contain duplicate elements, necessary because the AM representations of different methods can be identical.	[Bieman]
Local abstracted class	LAC(C)	"A collection of AM's where each AM corresponds to a visible **method** defined only within the **class**:" $LAC(C) = [\![AM(M) \mid M \in LV(C)]\!]$	[Bieman]

Quantities are derived from a graph in which nodes represent **methods** and edges represent *connections* as defined below:

Names	Symbols	Definitions	References
	NP(C)	"The total number of pairs of abstracted methods in AC(C). NP is the maximum possible number of direct or indirect connections in a class. If there are N methods in a class C," $NP(C) = N \cdot (N-1)/2$.	Unit: edge (graph entity) [Bieman]
	NDC(C)	"The number of direct connections in AC(C)" "If there exists one or more common instance variables between two method abstractions then the two corresponding methods are *directly connected*."	Unit: edge (graph entity) [Bieman]
	NIC(C)	"The number of indirect connections in AC(C)" "Two methods that are connected through other directly connected methods are *indirectly connected*. The indirect connection relation is the transitive closure of direct connection relation."	Unit: edge (graph entity) [Bieman]
Tight class cohesion	TCC(C)	Described as "the relative number of directly connected methods," but it is computed as a proportion of possible connections, not of methods: $TCC(C) = NDC(C)/NP(C)$	Unit: edge/edge [Bieman]
Loose class cohesion	LCC(C)	Described as "the relative number of directly or indirectly connected methods," but it is computed as a proportion of possible connections, not of methods: $LCC(C) = (NDC(C) + NIC(C))/NP(C)$	Unit: edge/edge [Bieman]

Names	Symbols	Definitions	References
Local class cohesion		"Local class cohesion measures are defined by using the local abstracted class (LAC) rather than the abstracted class (AC)." Thus there would be "local" versions of both LCC and TCC, substituting LAC(C) for AC(C) in the input quantities.	Unit: edge/edge [Bieman]

8.5 Li and Henry coupling metrics

Names	Symbols	Definitions	References
Message-passing coupling	MPC	"Number of send statements defined in a class." [Briand99] interprets [Li] as excluding invocations of the class' own methods and send statements in inherited methods.	Unit: call [Li, Briand99]
Data abstraction coupling	DAC	"Number of ADTs [**abstract data types**] defined in a class." [Briand99] provides two interpretations.	Unit: ADT [Li, Briand99]

8.6 Briand, Devanbu, and Melo coupling metrics

[Briand97] defines a suite of 18 object-oriented coupling metrics using a combinatoric approach. All the metrics "correspond to particular counts of **interactions** and are of the generic form:"

$$Metric(c_i) = \sum_{c_j \in Relationship(c_i)} Interactions(c_i, c_j)$$

The metrics are identified by 5 or 6-letter acronyms constructed from three parts:

1. Relationship type
   - IF: inverse friend, Friends^{-1}(c), "the set of classes that have c as a friend"
   - F: friend, Friends(c), "the set of classes that are the friends of c"
   - D: descendant, Descendants(c), "the set of classes that are the descendants of c"
   - A: ancestor, Ancestors(c), "the set of classes that are the ancestors of c. Ancestors(c) refers to the base classes of c, and their base classes, and so on (closure)."
   - O: others, Others(c), the set of other classes that have no inheritance or friendship relationship with c
2. Interaction type (note, "when we discuss attributes and methods of a class C, we only mean newly defined or overriding methods and attributes of C, not ones inherited")
   - CA: Class-Attribute interaction, the type of an attribute of one class refers to another class

- CM: Class-Method interaction, the signature of a method of one class refers to another class
- MM: Method-Method interaction, a method of one class invokes a method of another class

3. Locus of impact
   - IC: import coupling, class c is the using class
   - EC: export coupling, class c is the used class

Import coupling uses only IF, A, and O relationships, and export coupling uses only F, D, and O relationships, so the resulting number of metrics is 3 × 3 × 2 = 18.

8.7 Lee et al. coupling and cohesion metrics

Model:

Names	Symbols	Definitions	References
Message tuple	mt	mt = (fn, na) where fn is the **function** name and na is the "argument number" (number of **arguments**).	[Lee, Def. 3.1]
Function name	fn(mt)	Function name element of an mt	[Lee, Def. 3.1]
Argument number	na(mt)	Argument number element of an mt	[Lee, Def. 3.1]
Message-count tuple	mct	mct = (mt, nc) where nc ≥ 0 is the number of mts	[Lee, Def. 3.2]
Message-count tuple set	M(f)	Set of mcts for f, where f is a basic program **entity**.	[Lee, Def. 3.2]
External flow set	$M_E^C(f)$	For member function (method) f of class C, the set of f's mcts whose mts go to functions defined in other classes.	[Lee, Def. 3.3]
Internal flow set	$M_I^C(f)$	For member function (method) f of class C, the set of f's mcts whose mts go to functions defined in C.	[Lee, Def. 3.3]
Inheritance flow set	$M_{IH}^C(f)$	Subset of $M_E^C(f)$ where the target functions are defined in a superclass of C.	[Lee, Def. 3.4]
Non-inheritance flow set	$M_{NIH}^C(f)$	Subset of $M_E^C(f)$ where the target functions are defined outside the scope of C but not in a superclass of C.	[Lee, Def. 3.4]

Quantities:

Names	Symbols	Definitions	References
I-based coupling contribution (of f to C)	$ICP^C(f)$	$ICP^C(f) = \sum_{i=1}^{ne(f)} [(1 + na(mt_i)) * nc(mct_i)]$ where $mct_i \in M_E^C(f),\ ne(f) = \lvert M_E^C(f)\rvert$, and $nc(mct_i)$ is the number of mct_i's appearances in f.	Unit: argument · method [Lee, Def. 3.5]
I-based cohesion contribution (of f to C)	$ICH^C(f)$	$ICH^C(f) = \sum_{j=1}^{ni(f)} [(1 + na(mt_j)) * nc(mct_j)]$ where $mct_j \in M_I^C(f),\ ni(f) = \lvert M_I^C(f)\rvert$, and $nc(mct_j)$ is the number of mct_j's appearances in f.	Unit: argument · method [Lee, Def. 3.5]
I-based inheritance coupling contribution	$IH\text{-}ICP^C(f)$	Like $ICP^C(f)$, but substituting $M_{IH}^C(f)$ for $M_E^C(f)$.	Unit: argument · method [Lee, Def. 3.5]
I-based non-inheritance coupling contribution	$NIH\text{-}ICP^C(f)$	Like $ICP^C(f)$, but substituting $M_{NIH}^C(f)$ for $M_E^C(f)$.	Unit: argument · method [Lee, Def. 3.5]
I-based coupling (of class C)	ICP(C)	$ICP(C) = \sum_{k=1}^{n} ICP^C(f_k)$	Unit: argument · method [Lee, Def. 3.6]
I-based inheritance coupling (of class C)	IH-ICP(C)	Like ICP(C), but substituting $IH\text{-}ICP^C(f_k)$ for $ICP^C(f_k)$.	Unit: argument · method [Lee, Def. 3.6]
I-based non-inheritance coupling (of class C)	NIH-ICP(C)	Like ICP(C), but substituting $NIH\text{-}ICP^C(f_k)$ for $ICP^C(f_k)$.	Unit: argument · method [Lee, Def. 3.6]
I-based cohesion (of class C)	ICH(C)	$ICH(C) = \sum_{k=1}^{n} ICH^C(f_k)$	Unit: argument · method [Lee, Def. 3.6]

[Lee, Def. 3.7] proceeds to define analogous ICP(H) and ICH(H) for a class hierarchy H using definitions that are similar to the above except that they are scoped to the class hierarchy instead of a single class.

8.8 MOOD2 metrics

[Abreu] contains formal definitions of the following, expressed in Object Constraint Language (OCL). MOOD2 is a superset of the older MOOD (Metrics for Object-Oriented Design) suite of metrics, with the exception that the coupling factor was redefined. The old definition is included below for completeness.

Names	Symbols	Definitions	References
Attribute inheritance factor	AIF	"Quotient between the number of inherited attributes in all classes of the specification and the number of available attributes (locally defined plus inherited) for all classes of the current specification."	Unit: attribute/attribute Range: 0 to 1 [Abreu]
Operations inheritance factor, Methods inheritance factor	OIF, MIF	"Quotient between the number of inherited operations in all classes of the specification and the number of available operations (locally defined plus inherited) for all classes of the current specification."	Unit: method/method Range: 0 to 1 [Abreu]
Internal inheritance factor	IIF	"Quotient between the number of inheritance links where both the base and derived classes belong to the current specification and the total number of inheritance links originating in the current specification."	Unit: relationship/relationship Range: 0 to 1 [Abreu]
Attribute hiding factor	AHF	"Quotient between the sum of the invisibilities of all attributes defined in all classes in the current specification and the total number of attributes defined in the specification." "The invisibility of an attribute is the percentage [proportion] of the total classes in the specification from which this attribute is not visible"	Unit: attribute/attribute Range: 0 to 1 [Abreu]
Operations hiding factor, Methods hiding factor	OHF, MHF	"Quotient between the sum of the invisibilities of all operations defined in all classes in the current specification and the total number of operations defined in the specification." "The invisibility of an operation is the percentage [proportion] of the total classes in the specification from which this operation is not visible"	Unit: method/method Range: 0 to 1 [Abreu]

Names	Symbols	Definitions	References
Attributes hiding effectiveness factor	AHEF	"Quotient between the cumulative number of the specification classes that *do* access the specification attributes and the cumulative number of the specification classes that *can* access the specification attributes."	Unit: class/class Range: 0 to 1 [Abreu]
Operations hiding effectiveness factor	OHEF	"Quotient between the cumulative number of the specification classes that *do* access the specification operations and the cumulative number of the specification classes that *can* access the specification operations."	Unit: class/class Range: 0 to 1 [Abreu]
Behavioral polymorphism factor, Polymorphism factor	BPF, POF	"Quotient between the *actual* number of possible different polymorphic situations and the *maximum* number of possible distinct polymorphic situations (due to inheritance)"	Unit: method/method Range: 0 to 1 [Abreu]
Parametric polymorphism factor	PPF	"Percentage [proportion] of the specification classes that are parameterized"	Unit: class/class Range: 0 to 1 [Abreu]
Class coupling factor	CCF	Square root of the "Quotient between the actual number of coupled class-pairs within the specification and the maximum possible number of class-pair couplings in the specification. This coupling is the one not imputable to inheritance."	Transformed ratio Range: 0 to 1 [Abreu]
Coupling factor	COF	$= CCF^2$ (As defined in the original MOOD set)	Unit: relationship/relationship Range: 0 to 1 [Abreu]
Internal coupling factor	ICF	"Quotient between the number of coupling links where both the client and supplier classes belong to the current specification and the total number of coupling links originating in the current specification."	Unit: relationship/relationship Range: 0 to 1 [Abreu]
External inheritance factor	EIF(S)	"Quotient between the number of external inheritance links to specification “s” and the total number of inheritance links originating in the current specification.	Unit: relationship/relationship Range: 0 to 1 [Abreu]

Names	Symbols	Definitions	References
External coupling factor	ECF(S)	"Quotient between the number of external coupling links to specification “s” and the total number of coupling links originating in the current specification."	Unit: relationship/relationship Range: 0 to 1 [Abreu]
Potential reuse factor	PRF(S)	"Percentage [proportion] of the available operations in the current specification that were imported from the “s” specification."	Unit: method/method Range: 0 to 1 [Abreu]
Actual reuse factor	ARF(S)	"Percentage [proportion] of the available operations in the current specification that corresponds to effectively used operations imported from the “s” specification"	Unit: method/method Range: 0 to 1 [Abreu]
Reuse efficiency factor	REF(S)	"Percentage [proportion] of the imported operations (from the “s” specification) that are effectively used"	Unit: method/method Range: 0 to 1 [Abreu]

8.9 Lorenz and Kidd metrics

Many of the metrics defined in [Lorenz] are simple counts or obvious derivatives (e.g., average number of support classes per key class = NSC/NKC), but their rich set of symbols is used in other works. Those without symbols or with already-defined symbols (LOC) have been omitted.

Project metrics, application size:

Names	Symbols	Definitions	References
Number of scenario scripts	NSS	Number of **use cases**.	Unit: use case [Lorenz]
Number of key classes	NKC	Number of **classes** "that are deemed to be of central importance to the business."	Unit: class [Lorenz]
Number of support classes	NSC	Number of **classes** that are "not central to the business domain."	Unit: class [Lorenz]
Number of subsystems	NOS	Number of **packages**.	Unit: package [Lorenz]

Design metrics, method size:

Names	Symbols	Definitions	References
Number of message sends	NOM	"Number of messages sent in the method"	Unit: call [Lorenz]

Design metrics, method internals:

Names	Symbols	Definitions	References
Method complexity	MCX	Sum of the following, weighted as shown: Application Programming Interface (API) calls: 5.0 Assignments: 0.5 Binary expressions (Smalltalk) or arithmetic operators (C++): 2.0 Keyword messages (Smalltalk) or messages with parameters (C++): 3.0 Nested expressions: 0.5 Parameters: 0.3 Primitive calls: 7.0 Temporary variables: 0.5 Unary expressions (Smalltalk) or messages without parameters (C++): 1.0	Ordinal [Lorenz]
Strings of message sends	SMS	Number of **expressions** where messages/calls are strung together like self.account().balance().print().	Unit: expression [Lorenz]

Design metrics, class size:

Names	Symbols	Definitions	References
Number of public instance methods	PIM	Number of instance **methods** with public visibility.	Unit: method [Lorenz]
Number of instance methods	NIM	Number of instance **methods** (public, protected, and private).	Unit: method [Lorenz]
Number of instance variables	NIV	Number of instance **variables** (public, protected, and private).	Unit: variable [Lorenz]
Number of class methods	NCM	Number of class **methods**.	Unit: method [Lorenz]
Number of class variables	NCV	Number of class **variables**.	Unit: variable [Lorenz]

Design metrics, class inheritance:

Names	Symbols	Definitions	References
Hierarchy nesting level	HNL	= DIT (Sec. 8.3)	Unit: arc [Lorenz]
Multiple inheritance	MUI	Use of multiple inheritance (yes or no).	Dichotomic [Lorenz]

Design metrics, method inheritance:

Names	Symbols	Definitions	References
Number of methods overridden	NMO	Number of **methods** overridden by a subclass.	Unit: method [Lorenz]

Names	Symbols	Definitions	References
Number of methods inherited	NMI	Number of **methods** inherited by a subclass.	Unit: method [Lorenz]
Number of methods added	NMA	Number of **methods** added by a subclass.	Unit: method [Lorenz]
Specialization index	SIX	For each class, = HNL · NMO / methods	Ordinal [Lorenz]

Design metrics, class internals:

Names	Symbols	Definitions	References
Global usage	GUS	Number of references to global **variables**.	Unit: operand [Lorenz]
Instance variable usage	IVU	Number of references to instance **variables**.	Unit: operand [Lorenz]
Parameters per method	PPM	= parameters / methods	Unit: parameter/method [Lorenz]
Friend functions	FFU	Use of friend functions (yes or no).	Dichotomic [Lorenz]
Function-oriented code	FOC	Proportion of functions that are outside of classes. As with MUI and FFU, [Lorenz] sets a threshold of zero, suggesting a dichotomic metric, but the section heading is "percentage of function-oriented code."	Unit: function/function [Lorenz]
Comment lines per method	CLM	= CMT / methods	Unit: CMT/method [Lorenz]
Percentages of commented methods	PCM	Proportion of **methods** that have *any* comments in them.	Unit: method/method [Lorenz]
Problem reports per class	PRC	= **bug reports** / **classes**	Unit: bug report/class [Lorenz]

The following metrics with symbols either were not clearly defined or are out of scope:

Names	Symbols	Category
Class cohesion	CCO	Class internals
Class coupling	CCP	Class externals
Class reuse	CRE	Class externals
Number of classes thrown away	NCT	Class externals
Number of collaborations	NCO	Class externals
Intersubsystem relationships	ISR	Subsystem coupling
Interclass relationships	ICR	Subsystem coupling
Person-days per class	PDC	Staffing size

Names	Symbols	Category
Classes per developer	CPD	Staffing size
Number of major iterations	NMI	Scheduling
Number of contracts completed	NCC	Scheduling

9 Testability (test coverage) metrics

The following chapter lists test coverage metrics. Given any test coverage metric, a corresponding testability metric that is applicable to the software product in isolation can be produced based on the amount of testing that the metric would require to indicate full coverage. For example, given the path coverage metric, one can inversely relate testability to the number of linearly independent paths.

9.1 Rapps, Frankl, and Weyuker data flow coverage metrics

Definitions quoted below from [Frankl] are more formalized versions of ones that appeared earlier in [Rapps].

Model:

Symbols	Definitions	References
V	The set of **variables**.	[Frankl]
N	The set of **nodes**, which correspond to the **blocks** of the subprogram.	[Frankl]
E	The set of **edges**, which indicate possible flow of control between **blocks**.	[Frankl]
def(i)	{x∈V | x has a global **definition** in block i}	[Frankl]
c-use(i)	{x∈V | x has a global **c-use** in block i}	[Frankl]
p-use(i,j)	{x∈V | x has a **p-use** in edge (i,j)}	[Frankl]
dcu(x,i)	{j∈N | x∈c-use(j) and there is a **def**-clear **path** wrt x from i to j}	[Frankl]
dpu(x,i)	{(j,k)∈E | x∈p-use(j,k) and there is a **def**-clear **path** wrt x from i to (j,k)}	[Frankl]

Metrics:

Names	Symbols	Definitions	References
All-paths		Coverage of every **path** in a def/use graph.	Dichotomic [Frankl]

Names	Symbols	Definitions	References
All-du-paths		All du-paths from i to j with respect to x for each j∈dcu(x,i) and all du-paths from i to (j,k) with respect to x for each (j,k)∈dpu(x,i). A path (n_1, ..., n_j, n_k) is a du-path with respect to a variable x if n_1 has a global definition of x and either 1) n_k has a global c-use of x and (n_1, ..., n_j, n_k) is a def-clear simple path with respect to x, or 2) (n_j, n_k) has a p-use of x and (n_1, ..., n_j, n_k) is a def-clear loop-free path with respect to x. A *simple path* is one in which all nodes, except possibly the first and last, are distinct. A *loop-free path* is one in which all nodes are distinct.	Dichotomic [Frankl]
All-uses		For each node i and each x∈def(i), coverage of all (i,j,x) s.t. j∈dcu(x,i) and all (i,(j,k),x) s.t. (j,k)∈dpu(x,i).	Dichotomic [Frankl]
All-c-uses / some-p-uses		For each node i and each x∈def(i), coverage of all (i,j,x) s.t. j∈dcu(x,i). In addition, if dcu(x,i) is empty, then some (i,(j,k),x) s.t. (j,k)∈dpu(x,i).	Dichotomic [Frankl]
All-p-uses / some-c-uses		For each node i and each x∈def(i), coverage of all (i,(j,k),x) s.t. (j,k)∈dpu(x,i). In addition, if dpu(x,i) is empty, then some (i,j,x) s.t. j∈dcu(x,i).	Dichotomic [Frankl]
All-defs		For each node i and each x∈def(i), coverage of some (i,j,x) s.t. j∈dcu(x,i) or some (i,(j,k),x) s.t. (j,k)∈dpu(x,i).	Dichotomic [Frankl]
All-c-uses		For each node i and each x∈def(i), coverage of all (i,j,x) s.t. j∈dcu(x,i).	Dichotomic [Frankl]
All-p-uses		For each node i and each x∈def(i), coverage of all (i,(j,k),x) s.t. (j,k)∈dpu(x,i).	Dichotomic [Frankl]
All-edges		Coverage of every **edge** in a def/use graph.	Dichotomic [Frankl]
All-nodes		Coverage of every **node** in a def/use graph.	Dichotomic [Frankl]

9.2 Other coverage metrics

Names	Symbols	Definitions	References
e coverage		Generic: for any specification or source code entity type *e* (e.g., requirement, statement, LOC, module, function, class, branch), the proportion of *e* that are covered.	Unit: *e*/*e*

Names	Symbols	Definitions	References
Path coverage		"Test a basis set of **paths** through the control flow graph of each module." The linearly independent **paths** form a basis set. McCabe does not refer to this as path coverage, but rather "the structured testing criterion."	Dichotomic [SP 500-235]
Modified condition/decision coverage	MC/DC	"Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome." [11]	Dichotomic [DO-178C]
Total variable-value configuration coverage, Total *t*-way coverage		For a given combination of *t* variables, the proportion of **variable-value configurations** that are covered by at least one test case in a test set. This metric depends on the sets of values that are deemed "valid." Results based on different sets of valid values are not mutually comparable.	Unit: configuration/ configuration [Kuhn]
Simple *t*-way combination coverage		For a given test set for *n* variables, the proportion of *t*-way combinations of *n* variables for which all valid **variable-value configurations** are fully covered. This metric depends on the sets of values that are deemed "valid." Results based on different sets of valid values are not mutually comparable.	Unit: combination/ combination [Kuhn]
Linear code sequence and jump coverage, Jump-to-jump path coverage, Test effectiveness ratio 3	TER3	The proportion of **LCSAJ** triples that are covered.	Unit: LCSAJ/LCSAJ [Hennell]

10 Security metrics

The metrics covered in this section are Likert-type scales as used in psychometrics. While arguments have been made that these can be interval scales in Stevens' taxonomy instead of merely ordinal, the greater concern is that they may indicate a subjective assessment of a system rather than a measurement of an objective property of a software artifact.

Hash function and cipher metrics are in Sec. 6.2 and 6.3.

10.1 Common Weakness Scoring System

The Common Weakness Scoring System (CWSS) defines a number of ordinal quantities where lower values are better. For detailed descriptions of the levels, please refer to [CWSS].

Names	Symbols	Definitions	References
Technical impact	TI	"Potential result that can be produced by the weakness, assuming that the weakness can be successfully reached and exploited." Critical (C) / Not applicable (NA) = 1.0 High (H) = 0.9 Medium (M) / Default (D) = 0.6 Unknown (UK) = 0.5 Low (L) = 0.3 None (N) = 0.0	Ordinal [CWSS]
Acquired privilege	AP	"Type of privileges that are obtained by an attacker who can successfully exploit the weakness." Administrator (A) / Not applicable (NA) = 1.0 Partially-privileged user (P) = 0.9 Regular user (RU) / Default (D) = 0.7 Limited / guest (L) = 0.6 Unknown (UK) = 0.5 None (N) = 0.1	Ordinal [CWSS]
Acquired privilege layer	AL	"Operational layer to which the attacker gains privileges by successfully exploiting the weakness." Application (A) / Enterprise infrastructure (E) / Not applicable (NA) = 1.0 System (S) / Default (D) = 0.9 Network (N) = 0.7 Unknown (UK) = 0.5	Ordinal [CWSS]
Internal control effectiveness	IC	"Ability of the control to render the weakness unable to be exploited by an attacker." None (N) / Not applicable (NA) = 1.0 Limited (L) = 0.9 Moderate (M) = 0.7 Default (D) = 0.6 Indirect (I) / Unknown (UK) = 0.5 Best-available (B) = 0.3 Complete (C) = 0.0	Ordinal [CWSS]

Names	Symbols	Definitions	References
Finding confidence	FC	"Confidence that the reported issue is a weakness that can be utilized by an attacker." Proven true (T) / Not applicable (NA) = 1.0 Proven locally true (LT) / Default (D) = 0.8 Unknown (UK) = 0.5 Proven false (F) = 0.0	Ordinal [CWSS]
Required privilege	RP	"Type of privileges that an attacker must already have in order to reach the code/functionality that contains the weakness." None (N) / Not applicable (NA) = 1.0 Limited / guest (L) = 0.9 Regular user (RU) / Default (D) = 0.7 Partially-privileged user (P) = 0.6 Unknown (UK) = 0.5 Administrator (A) = 0.1	Ordinal [CWSS]
Required privilege layer	RL	"Operational layer to which the attacker must have privileges in order to attempt to attack the weakness." Application (A) / Enterprise infrastructure (E) / Not applicable (NA) = 1.0 System (S) / Default (D) = 0.9 Network (N) = 0.7 Unknown (UK) = 0.5	Ordinal [CWSS]
Access vector	AV	"Channel through which an attacker must communicate to reach the code or functionality that contains the weakness." Internet (I) / Not applicable (NA) = 1.0 Intranet (R) / Private network (V) = 0.8 Default (D) = 0.75 Adjacent network (A) = 0.7 Local (L) / Unknown (U) = 0.5 Physical (P) = 0.2	Ordinal [CWSS]
Authentication strength	AS	"Strength of the authentication routine that protects the code/functionality that contains the weakness." None (N) / Not applicable (NA) = 1.0 Weak (W) = 0.9 Default (D) = 0.85 Moderate (M) = 0.8 Strong (S) = 0.7 Unknown (UK) = 0.5	Ordinal [CWSS]

Names	Symbols	Definitions	References
Level of interaction	IN	"Actions that are required by the human victim(s) to enable a successful attack to take place." Automated (A) / Not applicable (NA) = 1.0 Typical/limited (T) = 0.9 Moderate (M) = 0.8 Default (D) = 0.55 Unknown (UK) = 0.5 Opportunistic (O) = 0.3 High (H) = 0.1 No interaction (NI) = 0.0	Ordinal [CWSS]
Deployment scope	SC	"Whether the weakness is present in all deployable instances of the software, or if it is limited to a subset of platforms and/or configurations." All (A) / Not applicable (NA) = 1.0 Moderate (M) = 0.9 Default (D) = 0.7 Rare (R) / Unknown (UK) = 0.5 Potentially reachable (P) = 0.1	Ordinal [CWSS]
Business impact	BI	"Potential impact to the business or mission if the weakness can be successfully exploited." Critical (C) / Not applicable (NA) = 1.0 High (H) = 0.9 Medium (M) / Default (D) = 0.6 Unknown (UK) = 0.5 Low (L) = 0.3 None (N) = 0.0	Ordinal [CWSS]
Likelihood of discovery	DI	"Likelihood that an attacker can discover the weakness." High (H) / Not applicable (NA) = 1.0 Medium (M) / Default (D) = 0.6 Unknown (UK) = 0.5 Low (L) = 0.2	Ordinal [CWSS]
Likelihood of exploit	EX	"Likelihood that, if the weakness is discovered, an attacker with the required privileges/authentication/access would be able to successfully exploit it." High (H) / Not applicable (NA) = 1.0 Medium (M) / Default (D) = 0.6 Unknown (UK) = 0.5 Low (L) = 0.2 None (N) = 0.0	Ordinal [CWSS]

Names	Symbols	Definitions	References
External control effectiveness	EC	"Capability of controls or mitigations outside of the software that may render the weakness more difficult for an attacker to reach and/or trigger." None (N) / Not applicable (NA) = 1.0 Limited (L) = 0.9 Moderate (M) = 0.7 Default (D) = 0.6 Indirect (I) / Unknown (UK) = 0.5 Best-available (B) = 0.3 Complete (C) = 0.1	Ordinal [CWSS]
Prevalence	P	"How frequently this type of weakness appears in software." Widespread (W) / Not applicable (NA) = 1.0 High (H) = 0.9 Common (C) = 0.8 Default (D) = 0.85 Limited (L) = 0.7 Unknown (UK) = 0.5	Ordinal [CWSS]
Base finding subscore		$= [(10 \cdot TI + 5 \cdot (AP+AL) + 5 \cdot FC) \cdot f(TI) \cdot IC] \cdot 4.0$ $f(TI) = 0$ if $TI = 0$; otherwise $f(TI) = 1$	Ordinal Range: 0 to 100 [CWSS]
Attack surface subscore		$[20 \cdot (RP+RL+AV) + 20 \cdot SC + 15 \cdot IN + 5 \cdot AS] / 100.0$	Ordinal Range: 0 to 1 [CWSS]
Environmental subscore		$[(10 \cdot BI + 3 \cdot DI + 4 \cdot EX + 3 \cdot P) \cdot f(BI) \cdot EC] / 20.0$ $f(BI) = 0$ if $BI = 0$; otherwise $f(BI) = 1$	Ordinal Range: 0 to 1 [CWSS]
CWSS 1.0 score		= BaseFindingSubscore · AttackSurfaceSubscore · EnvironmentalSubscore	Ordinal Range: 0 to 100 [CWSS]

10.2 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS)[12] defines a number of ordinal quantities where lower values are better. For detailed descriptions of the levels, please refer to [CVSS].

[CVSS] also defines modified base metrics and an environmental score to characterize the impact of "modifications that exist within the analyst's environment." These have been omitted for brevity.

Names	Symbols	Definitions	References
Attack vector	AV	"Context by which vulnerability exploitation is possible." Network (N) = 0.85 Adjacent (A) = 0.62 Local (L) = 0.55 Physical (P) = 0.2	Ordinal [CVSS]
Attack complexity	AC	"Conditions beyond the attacker's control that must exist in order to exploit the vulnerability." Low (L) = 0.77 High (H) = 0.44	Ordinal [CVSS]
Privileges required	PR	"Level of privileges an attacker must possess *before* successfully exploiting the vulnerability." None (N) = 0.85 Low (L) = 0.62 (if scope = U) or 0.68 (if scope = C) High (H) = 0.27 (if scope = U) or 0.50 (if scope = C)	Ordinal [CVSS]
User interaction	UI	"Requirement for a user, other than the attacker, to participate in the successful compromise of the vulnerable component." None (N) = 0.85 Required (R) = 0.62	Ordinal [CVSS]
Scope	S	"Ability for a vulnerability in one software component to impact resources beyond its means, or privileges." Unchanged (U) Changed (C) See PR for numerical effect.	Ordinal [CVSS]
Confidentiality impact	C	"Impact to the confidentiality of the information resources managed by a software component due to a successfully exploited vulnerability." High (H) = 0.56 Low (L) = 0.22 None (N) = 0	Ordinal [CVSS]
Integrity impact	I	"Impact to integrity of a successfully exploited vulnerability." High (H) = 0.56 Low (L) = 0.22 None (N) = 0	Ordinal [CVSS]

Names	Symbols	Definitions	References
Availability impact	A	"Impact to the availability of the impacted component resulting from a successfully exploited vulnerability." High (H) = 0.56 Low (L) = 0.22 None (N) = 0	Ordinal [CVSS]
Exploit code maturity	E	"Likelihood of the vulnerability being attacked" High (H) / Not defined (X) = 1 Functional (F) = 0.97 Proof-of-concept (P) = 0.94 Unproven (U) = 0.91	Ordinal [CVSS]
Remediation level	RL	Unavailable (U) / Not defined (X) = 1 Workaround (W) = 0.97 Temporary fix (F) = 0.96 Official fix (O) = 0.95	Ordinal [CVSS]
Report confidence	RC	"Degree of confidence in the existence of the vulnerability and the credibility of the known technical details." Confirmed (C) / Not defined (X) = 1 Reasonable (R) = 0.96 Unknown (U) = 0.92	Ordinal [CVSS]
Confidentiality requirement	CR	Importance of confidentiality to the organization. High (H) = 1.5 Medium (M) / Not defined (X) = 1 Low (L) = 0.5	Ordinal [CVSS]
Integrity requirement	IR	Importance of integrity to the organization. High (H) = 1.5 Medium (M) / Not defined (X) = 1 Low (L) = 0.5	Ordinal [CVSS]
Availability requirement	AR	Importance of availability to the organization. High (H) = 1.5 Medium (M) / Not defined (X) = 1 Low (L) = 0.5	Ordinal [CVSS]
Impact sub score	ISC	If scope = U: $6.42 \cdot ISC_{BASE}$ If scope = C: $7.52 \cdot (ISC_{BASE} - 0.029) - 3.25 \cdot (ISC_{BASE} - 0.02)^{15}$ where $ISC_{BASE} = 1-[(1-C)\cdot(1-I)\cdot(1-A)]$	Ordinal [CVSS]
Exploitability sub score	ESC*	$8.22 \cdot AV \cdot AC \cdot PR \cdot UI$ * [CVSS] does not assign a symbol. ESC was introduced here to parallel ISC.	Ordinal [CVSS]

<table>
<tr><th>Names</th><th>Symbols</th><th>Definitions</th><th>References</th></tr>
<tr><td>Base score</td><td></td><td>= 0 if ISC ≤ 0. Otherwise,
If scope = U: round up(min(ISC+ESC,10))
If scope = C: round up(min(1.08·(ISC+ESC),10))
where "round up" is defined as the smallest number, specified to one decimal place, that is equal to or higher than its input. For example, round up (4.02) is 4.1; and round up (4.00) is 4.0.</td><td>Ordinal
Range: 0 to 10
The scale is further reduced to a 5-level ordinal scale as follows:<table><tr><td>0.0</td><td>None</td></tr><tr><td>0.1 to 3.9</td><td>Low</td></tr><tr><td>4.0 to 6.9</td><td>Medium</td></tr><tr><td>7.0 to 8.9</td><td>High</td></tr><tr><td>9.0 to 10.0</td><td>Critical</td></tr></table>[CVSS]</td></tr>
<tr><td>Temporal score</td><td></td><td>Round up(Base score · E · RL · RC)
See definition of "round up" under base score.</td><td>Ordinal
Range: 0 to 10
Reduction to 5-level scale same as base score.
[CVSS]</td></tr>
</table>

10.3 Vulnerability severity (SP 800-30)

Names	Symbols	Definitions	References
Vulnerability severity		Very high = 96 to 100 (/100) or 10 (/10) "The vulnerability is exposed and exploitable, and its exploitation could result in severe impacts. Relevant security control or other remediation is not implemented and not planned; or no security measure can be identified to remediate the vulnerability." High = 80 to 95 (/100) or 8 (/10) "The vulnerability is of high concern, based on the exposure of the vulnerability and ease of exploitation and/or on the severity of impacts that could result from its exploitation. Relevant security control or other remediation is planned but not implemented; compensating controls are in place and at least minimally effective." Moderate = 21 to 79 (/100) or 5 (/10) "The vulnerability is of moderate concern, based on the exposure of the vulnerability and ease of exploitation and/or on the severity of impacts that could result from its exploitation. Relevant security control or other	Ordinal [SP800-30r1, Table F-2]

Names	Symbols	Definitions	References
		remediation is partially implemented and somewhat effective." Low = 5 to 20 (/100) or 2 (/10) "The vulnerability is of minor concern, but effectiveness of remediation could be improved. Relevant security control or other remediation is fully implemented and somewhat effective." Very low = 0 to 4 (/100) or 0 (/10) "The vulnerability is not of concern. Relevant security control or other remediation is fully implemented, assessed, and effective."	

11 Bibliography

[Abran] Abran, A. *Software Metrics and Software Metrology*. Hoboken: John Wiley & Sons, Inc., 2010.

[Abreu] Fernando Brito e Abreu. "Using OCL to formalize object oriented metrics definitions." Technical Report ES007/2001, version 1.0, June 2001. https://pdfs.semanticscholar.org/53fb/564942459946ef08ad2b40625a07b0b9187b.pdf.

[AFP] Automated Function Points (AFP), version 1.0. Document formal/2014-01-03, Object Management Group, January 2014. https://www.omg.org/cgi-bin/doc?formal/2014-01-03.

[Belady] L. A. Belady and C. J. Evangelisti. "System Partitioning and Its Measure." *Journal of Systems and Software*, 2, pp. 23–29, 1981.

[Berge] Claude Berge. *Graphs*. North-Holland, second revised edition, 1985.

[Bieman] James M. Bieman and Byung-Kyoo Kang. "Cohesion and Reuse in an Object-Oriented System." In Proceedings of the 1995 Symposium on Software Reusability (SSR'95), pp. 259–262. https://doi.org/10.1145/223427.211856.

[Briand93] Lionel C. Briand, Sandro Morasca, and Victor R. Basili. "Measuring and Assessing Maintainability at the End of High Level Design." In Proceedings of the Conference on Software Maintenance (CSM), pp. 88–97, September 1993. https://doi.org/10.1109/ICSM.1993.366952.

[Briand94] Lionel Briand, Sandro Morasca, and Victor R. Basili. "Defining and validating high-level design metrics." University of Maryland technical report CS-TR 3301, 1994. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.4744&rep=rep1&type=ps

[Briand97] Lionel Briand, Prem Devanbu, and Walcelio Melo. "An Investigation into Coupling Measures for C++." In Proceedings of the 19th International Conference on Software Engineering (ICSE), pp. 412–421, May 1997. https://doi.org/10.1145/253228.253367.

[Briand98] Lionel C. Briand, John. W. Daly, and Jürgen Wüst. "A Unified Framework for Cohesion Measurement in Object-Oriented Systems." *Empirical Software Engineering*, 3, pp. 65–117, 1998.

[Briand99] Lionel C. Briand, John. W. Daly, and Jürgen Wüst. "A Unified Framework for Coupling Measurement in Object-Oriented Systems." *IEEE Transactions on Software Engineering*, 25(1), pp. 91–121, Jan/Feb 1999.
[Card] David N. Card and Robert L. Glass. *Measuring Software Design Quality*. Prentice-Hall, 1990.
[Chidamber] Shyam R. Chidamber and Chris F. Kemerer. A metric suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6), pp. 476-493, June 1994. https://doi.org/10.1109/32.295895.
[Coleman] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, August 1994. https://doi.org/10.1109/2.303623.
[Conte] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software Engineering Metrics and Models*. Benjamin/Cummings, 1986.
[COSMIC] Common Software Measurement International Consortium. The COSMIC Functional Size Measurement Method Version 4.0.1 Measurement Manual, April 2015. https://cosmic-sizing.org/publications/measurement-manual-401/.
[Cruickshank] R. D. Cruickshank and J. E. Gaffney, Jr. "Measuring the Development Process: Software Design Coupling and Strength Metrics." In Proceedings from the Fifth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, November 1980. https://ntrs.nasa.gov/search.jsp?R=19820016124.
[CVSS] Common Vulnerability Scoring System v3.0: Specification Document (version 1.8). https://www.first.org/cvss/v3.0/specification-document.
[CWSS] Steve Christey Coley, ed. Common Weakness Scoring System (CWSS) version 1.0.1. MITRE, 2014-09-05. https://cwe.mitre.org/cwss/cwss_v1.0.1.html.
[DO-178C] "Software Considerations in Airborne Systems and Equipment Certification," RTCA DO-178C, December 13, 2011.
[Eder] Johann Eder, Gerti Kappel, and Michael Schrefl. Coupling and cohesion in object-oriented systems. Technical report, University of Klagenfurt, 1994. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.5819.
[Elshoff] James L. Elshoff. "An Analysis of Some Commercial PL/I Programs." *IEEE Transactions on Software Engineering*, SE-2(2), June 1976, pp. 113–120.
[Embley] D. W. Embley and S. N. Woodfield, "Cohesion and Coupling for Abstract Data Types," in 6th International Phoenix Conference on Computers and Communications (IPCCC), IEEE Computer Society Press, Arizona, 1987, pp. 229–234.
[Fenton] Norman Fenton and James Bieman. Software Metrics: A rigorous and practical approach. CRC Press, 2015.
[Flater] David Flater, "Redressing grievances with the treatment of dimensionless quantities in SI," Measurement v. 109, October 2017, pp. 105–110. https://doi.org/10.1016/j.measurement.2017.05.043.
[Frankl] Phyllis G. Frankl and Elaine J. Weyuker. "An Applicable Family of Data Flow Testing Criteria." *IEEE Transactions on Software Engineering*, 14(10), pp. 1483–1498, October 1988. https://doi.org/10.1109/32.6194.
[Halstead] Maurice H. Halstead. Elements of Software Science. Elsevier, 1977.
[Hamer] Peter G. Hamer and Gillian D. Frewin. M. H. Halstead's Software Science—A Critical Examination. In Proceedings of the 6th International Conference on Software Engineering (ICSE'82), September 1982, pp. 197–206.

[Henderson-Sellers] B. Henderson-Sellers, L. L. Constantine, and I. M. Graham. "Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design," *Object-Oriented Systems*, 3(3), 1996, pp. 143–158.
[Hennell] M. A. Hennell, M. R. Woodward, and D. Hedley. "On program analysis." *Information Processing Letters*, 5(5), November 1976, pp. 136–140.
[Henry] Sallie Henry and Dennis Kafura. "Software Structure Metrics Based on Information Flow." *IEEE Transactions on Software Engineering*, SE-7(5), pp. 510–518, September 1981. https://doi.org/10.1109/TSE.1981.231113.
[Hitz] Martin Hitz and Behzad Montazeri. "Measuring Coupling and Cohesion In Object-Oriented Systems." In Proceedings of the International Symposium on Applied Corporate Computing (ISAAC'95), October 1995. http://www.isys.uni-klu.ac.at/PDF/1995-0043-MHBM.pdf.
[IEC] IEC. Quantities and units—Part 13: Information science and technology, 1.0 edition. IEC 80000-13:2008, http://www.iec.ch/.
[IEEE-100] IEEE 100-2000. The Authoritative Dictionary of IEEE Standards Terms, 7th edition. https://doi.org/10.1109/IEEESTD.2000.322233. Standard withdrawn, DOI broken; use http://ieeexplore.ieee.org/servlet/opac?punumber=4116785.
[IEEE-982.1-1988] IEEE 982.1-1988. IEEE Standard Dictionary of Measures to Produce Reliable Software. https://doi.org/10.1109/IEEESTD.1989.86055. (Superseded)
[IEEE-1541] IEEE 1541-2002. IEEE Standard for Prefixes for Binary Multiples. https://doi.org/10. 1109/IEEESTD.2009.5254933. DOI broken; use http://ieeexplore.ieee.org/servlet/opac?punumber=5254929.
[IR5459] W. J. Salamon and D. R. Wallace. NIST IR 5459. Quality Characteristics and Metrics for Reusable Software (Preliminary Report). May 1994.
[IR8113] Paul E. Black and Athos Ribeiro. NIST IR 8113. SATE V Ockham Sound Analysis Criteria. March 2016 (updated May 2019). https://doi.org/10.6028/NIST.IR.8113.
[ISO-Vocab-2010] ISO/IEC/IEEE 24765-2010. Systems and software engineering—Vocabulary. https://doi.org/10.1109/IEEESTD.2010.5733835. (Superseded)
[ISO-Vocab] ISO/IEC/IEEE 24765:2017. Systems and software engineering—Vocabulary. https://doi.org/10.1109/IEEESTD.2017.8016712.
[ISO-COSMIC] ISO/IEC 19761:2011. Software engineering—COSMIC: a functional size measurement method.
[ISO-FiSMA] ISO/IEC 29881:2010. Information technology—Systems and software engineering—FiSMA 1.1 functional size measurement method.
[ISO-FSM] ISO/IEC 14143-1:2007. Information technology— Software measurement—Functional size measurement, Part 1: Definition of concepts. Second edition, 2007-02-15.
[ISO-IFPUG] ISO/IEC 20926:2009. Software and systems engineering—Software measurement—IFPUG functional size measurement method 2009.
[ISO-MkII] ISO/IEC 20968:2002. Software engineering—Mk II Function Point Analysis—Counting Practices Manual.
[ISO-NESMA] ISO/IEC 24570:2018. Software engineering—NESMA functional size measurement method—Definitions and counting guidelines for the application of function point analysis. 2nd ed., 2018-02.
[Knuth] Donald E. Knuth. The Art of Computer Programming, Volume 1: Fundamental Algorithms. 2nd ed. Addison-Wesley, 1973.

[Krystek] M. P. Krystek. The term 'dimension' in the international system of units. *Metrologia* 52(2), pp. 297–300, 2015. https://doi.org/10.1088/0026-1394/52/2/297.
[Kuhn] D. Richard Kuhn, Itzel Dominguez Mendoza, Raghu N. Kacker, and Yu Lei. Combinatorial coverage measurement concepts and applications. In Second International Workshop on Combinatorial Testing, Sixth International Conference on Software Testing, Verification and Validation, pp. 352–361, March 2013. https://doi.org/10.1109/ICSTW.2013.77.
[Lee] Yen-Sung Lee, Bin-Shiang Liang, Shu-Fen Wu, and Feng-Jian Wang. "Measuring the coupling and cohesion of an object-oriented program based on information flow." Proceedings of the International Conference on Software Quality, Maribor, Slovenia, pp. 81-90, 1995.
[Li] Wei Li and Sallie Henry. "Object-oriented metrics that predict maintainability." *Journal of Systems and Software*, 23(2), pp. 111–122, Nov. 1993. https://doi.org/10.1016/0164-1212(93)90077-B.
[Liso] Aldo Liso. Software maintainability metrics model: An improvement in the Coleman-Oman model. *CrossTalk (Journal of Defense Software Engineering)*, pp. 15–17, August 2001. http://static1.1.sqspcdn.com/static/f/702523/9456986/1290003183740/200108-Liso.pdf.
[Lorenz] Mark Lorenz and Jeff Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice Hall, 1994.
[Martin] Robert Cecil Martin. Agile software development: principles, patterns, and practices. Prentice Hall, 2003.
[McCabe] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, December 1976. https://doi.org/10.1109/TSE.1976.233837.
[McClure1] Carma L. McClure. "A model for program complexity analysis." Proceedings of the 3rd International Conference on Software Engineering (ICSE), May 1978, pp. 149–157. https://dl.acm.org/citation.cfm?id=803205.
[McClure2] Carma L. McClure. *Reducing COBOL Complexity through Structured Programming*. Van Nostrand Reinhold, 1978.
[Microsoft] Maintainability Index Range and Meaning. In Microsoft Code Analysis Team Blog, 2007-11-20. https://blogs.msdn.microsoft.com/codeanalysis/2007/11/20/maintainability-index-range-and-meaning/.
[Misa] Subhas Chandra Misra. Modeling design/coding factors that drive maintainability of software systems. *Software Quality Journal*, 13(3):297–320, September 2005. https://doi.org/10.1007/s11219-005-1754-7.
[Mohr] Peter J. Mohr and William D. Phillips. "Dimensionless units in the SI." *Metrologia*, 52(1):40–47, 2015. https://doi.org/10.1088/0026-1394/52/1/40.
[MyersCC] Glenford J. Myers. "An extension to the cyclomatic measure of program complexity." *ACM SIGPLAN Notices*, 12(10):61–64, October 1977. https://doi.org/10.1145/954627.954633.
[MyersSD] Glenford J. Myers. Composite/Structured Design. Van Nostrand Reinhold, 1978.
[Oman] Paul Oman and Jack Hagemeister. Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software*, 24(3):251–266, March 1994. https://doi.org/10.1016/0164-1212(94)90067-1.

[Oy] Aivosto Oy. Complexity metrics, Project Analyzer v10.2 help, 2017. http://www.aivosto.com/project/help/pm-complexity.html.
[Park] Robert E. Park. Software size measurement: A framework for counting source statements. CMU/SEI-92-TR-020, Software Engineering Institute, September 1992. https://www.sei.cmu.edu/reports/92tr020.pdf.
[Rapps] Sandra Rapps and Elaine J. Weyuker. "Selecting Software Test Data Using Data Flow Information." *IEEE Transactions on Software Engineering*, SE-11(4), pp. 367–375, April 1985. https://doi.org/10.1109/TSE.1985.232226.
[Schumacher] Benjamin Schumacher. "Quantum coding." *Physical Review A*, 51(4), April 1995. https://doi.org/10.1103/PhysRevA.51.2738.
[SEI] Software Engineering Institute. Maintainability index technique for measuring program maintainability, in *Software Technology Roadmap*, May 2008. https://goo.gl/k2SG1N.
[SI] BIPM. The International System of Units (SI), 9th edition, 2019. https://www.bipm.org/en/publications/si-brochure/.
[SP500-235] Arthur H. Watson and Thomas J. McCabe. NIST Special Publication 500-235. Structured testing: A testing methodology using the cyclomatic complexity metric. September 1996.
[SP800-30r1] NIST Special Publication 800-30 Revision 1. Guide for Conducting Risk Assessments. September 2012.
[StevensSS] Stanley S. Stevens. On the theory of scales of measurement. Science, 103(2684):677–680, June 1946. https://doi.org/10.1126/science.103.2684.677.
[StevensWP] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured Design," *IBM Systems Journal*, 13(2):115–139, 1974. https://doi.org/10.1147/sj.132.0115.
[Stroud] John M. Stroud. The Fine Structure of Psychological Time. *Annals of New York Academy of Sciences*, 1966, pp. 623–631.
[UML] Unified Modeling Language version 2.5. OMG document formal/2015-03-01, March 2015. https://www.omg.org/spec/UML/2.5/.
[VIM] Joint Committee for Guides in Metrology. International vocabulary of metrology—Basic and general concepts and associated terms (VIM), 3rd edition. JCGM 200:2012, https://www.bipm.org/en/publications/guides/vim.html.
[Welker] Kurt D. Welker, Paul W. Oman, and Gerald G. Atkinson. Development and application of an automated source code maintainability index. *Journal of Software Maintenance: Research and Practice*, 9(3):127–159, May 1997. https://goo.gl/pXF5Yd.
[Woodward] Martin R. Woodward, Michael A. Hennell, and David Hedley. "A Measure of Control Flow Complexity in Program Text." *IEEE Transactions on Software Engineering*, SE-5(1), January 1979, pp. 45–50.
[Yourdon] Edward Yourdon and Larry L. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Prentice-Hall, 1979.
[Zuse] Horst Zuse. A Framework of Software Measurement. Walter de Gruyter, 1998.

Coleman-Oman maintainability model references

Fang Zhuo, Bruce Lowther, Paul Oman, and Jack Hagemeister. Constructing and testing software maintainability assessment models. In Proceedings, 1st International Software Metrics Symposium, pages 61–70. IEEE, May 1993. https://doi.org/10.1109/METRIC.1993.263800.

[Coleman] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, August 1994. https://doi.org/10.1109/2.303623.

[Oman] Paul Oman and Jack Hagemeister. Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software*, 24(3):251–266, March 1994. https://doi.org/10.1016/0164-1212(94)90067-1.

Don Coleman, Bruce Lowther, and Paul Oman. The application of software maintainability models in industrial software systems. Journal of Systems and Software, 29(1):3–16, April 1995. https://doi.org/10.1016/0164-1212(94)00125-7.

Troy Pearse and Paul Oman. Maintainability measurements on industrial source code maintenance activities. In Proceedings, International Conference on Software Maintenance, pages 295–303. IEEE, October 1995. https://doi.org/10.1109/ICSM.1995.526551.

Kurt D. Welker and Dr. Paul W. Oman. Software maintainability metrics models in practice. CrossTalk (Journal of Defense Software Engineering), pages 19–23, 32, November/December 1995.

[Welker] Kurt D. Welker, Paul W. Oman, and Gerald G. Atkinson. Development and application of an automated source code maintainability index. *Journal of Software Maintenance: Research and Practice*, 9(3):127–159, May 1997. https://goo.gl/pXF5Yd.

12 Copyright notes

[11] Copyright RTCA, Inc. Used with permission. The complete document may be purchased from RTCA, Inc., 1150 18th Street NW Suite 910, Washington, DC 20036; (202) 833-9339; www.rtca.org.

[12] CVSS is owned and managed by FIRST.Org, Inc. (FIRST), a US-based non-profit organization. While FIRST owns all right and interest in CVSS, it licenses it to the public freely for use, subject to the conditions below. Membership in FIRST is not required to use or implement CVSS. FIRST does, however, require that any individual or entity using CVSS give proper attribution, where applicable, that CVSS is owned by FIRST and used by permission. Further, FIRST requires as a condition of use that any individual or entity which publishes scores conforms to the guidelines described in the original specification and provides both the score and the scoring vector so others can understand how the score was derived.