METRICS IN CODESONAR

The CodeSonar analysis computes and reports a number of metrics.

About Metrics In CodeSonar

Each CodeSonar analysis computes values for a designated set of software metrics. Metric values can then be accessed through the hub GUI or by using the CodeSonar plug-in API. A number of built-in metric classes are provided, along with mechanisms allowing users to define custom metric classes.

The analysis submits the values computed for analysis-, compilation unit-, and file-granularity metrics to the hub, where they are stored with other analysis information. For space reasons, values for procedure-granularity metrics are stored in the project analysis directory (`pfilesname.prj_files/`). Directory-granularity metrics are not currently displayed in the CodeSonar GUI. The consequences for subsequent availability of metric values are described below.

We distinguish between a *metric class* and a *metric value*.

metric class — A metric class contains information about a single software metric:

- A short *tag*.
- A slightly longer, human-readable *description*
- A *granularity*, indicating what kind of component is described by the metric class. There are four standard granularities: procedure, file, compilation unit, and analysis. A fifth granularity - directory - is present but values of this granularity are not currently displayed in the CodeSonar GUI or available through the plug-in API.
- A *computation function*, used to obtain metric values for the class.

In most parts of the CodeSonar GUI, metrics are referred to by their *description* and *granularity* is implicit. For example, the "Total Lines" column in a table of procedures contains values computed for the procedure-granularity metric class with that description.

For clarity in saving metric reports and managing the saved reports, metric class granularity is stated explicitly. The format used in these cases is `Tag (Granularity)`. For example, "TL (Procedure)" refers to the procedure-granularity metric class with tag TL.

metric value — The value computed for a specific metric class as applied to a specific code component.

Built-In Metrics

The following metric classes are built into CodeSonar.

- For C/C++ code, all metrics are available.
- For C# and Java code, only the line count metrics are available (at all granularities): Blank Lines, Code Lines, Comment Lines, Lines with Code, Lines with Comments, Mixed Lines.

Description	Tag	Definition	Computed at Level			
			Analysis	File	Procedure	Directory [*]
Computed by default						
Blank Lines	LB	The number of blank lines, excluding blank lines in comments.	x	x	x	x
Code Lines	LCode Only	The number of lines that contain code only, with no comments.	x	x	x	x
Comment Lines	LCom Only	The number of lines that contain comments only, with no code.	x	x	x	x

Cyclomatic Complexity [*]	vG	The number of linearly independent paths through the control flow graph `G` of a function. Computed as `vG=E-N+2`, where `E` is the number of edges in `G`, and `N` is the number of nodes in `G`.	.	x	x	x
Include file instances	InclF	The number of include-file instances in the analyzed project, excluding system include files.	x	.	.	.
Is Taint Sink *or* Taint Sink Total	TaintSink	Is Taint Sink (TaintSink(Procedure)): 1 if a function contains an operation that must not use a tainted value, 0 otherwise. Taint Sink Total (TaintSink(File)) : sum of TaintSink(p) over all procedures p in the file.	.	x	x	x
Is Taint Source *or* Taint Source Total	TaintSource	Is Taint Source (TaintSource(Procedure)): 1 if tainted data can enter the program through a function, 0 otherwise. Taint Source	.	x	x	x

		Total (TaintSource(File)): sum of TaintSource(p) over all procedures p in the file.				
Lines with Code	LCode	The number of lines that contain code.	x	x	x	x
Lines with Comments	LCom	The number of lines that contain comments.	x	x	x	x
Mixed Lines	LMCC	The number of lines that contain both code and comments.	x	x	x	x
Modified Cyclomatic Complexity [*]	mvG	A variant of Cyclomatic Complexity in which `switch` statements are considered to have the same effect on complexity as `if` statements, regardless of the number of switch cases.	.	x	x	x
Propagates Taint *or* Taint Propagator Total	TaintProp	Propagates Taint (TaintProp(Procedure)): 1 if a tainted value or reference to a tainted value occurs in the procedure, 0 otherwise. Taint Propagator Total (TaintProp(File)): sum of TaintProp(p)	.	x	x	x

		over all procedures p in the file.				
Top-level file instances	TopLF	The number of compilation units in the analyzed project, and thus the number of top-level file instances.	x	.	.	.
Total Lines	TL	The total number of lines.	x	x	x	x
User-defined functions	Modules	The total number of user-defined functions.	x	.	.	.
Not computed by default						
Distinct Operands [**]	n2	Total number of distinct operands for a module	.	x	.	.
Distinct Operators [**]	n1	Total number of distinct operators for a module	.	x	.	.
Essential Complexity [*]	evG	A measure of the amount of unstructured code in a module. Computed as the cyclomatic complexity of the *reduced-CFG* obtained by removing "well-structured" primitive constructs from the module's control flow graph.	.	x	x	x

Halstead Program Volume [**]	V	= N * (log₂ n), where n=n1+n2	.	x	x	x
Halstead Programming Effort [**]	E	= D * V	.	x	x	x
Halstead Programming Time [**]	T	= E/18 seconds	.	x	x	x
Halstead Intelligent Content [**]	I	= (1/D) * V	.	x	x	x
Halstead Program Difficulty [**]	D	= (n1/2)*(N2/n2)	.	x	x	x
Halstead Program Length [**]	N	= N1 + N2	.	x	x	x
Halstead Program Level [**]	L	= (2/n1)*(n2/N2)	.	x	x	x
Integration Complexity [*]	S1	A measure of the number of independent integration tests required for the analyzed project. Computed by taking the sum of the module design complexities (ivG) of all project functions, minus the number of functions, plus 1.	x	.	.	.
Module Design Complexity [*]	ivG	Computed as the cyclomatic complexity of the *reduced-CFG* obtained by removing control structures that do not contain	.	x	x	x

			function calls from the module's control flow graph.				
	Total Operands [**]	N2	Total number of operands for a module	.	x	x	x
	Total Operators [**]	N1	Total number of operators for a module	.	x	x	x

Watson and McCabe Metrics [*]

These metrics are defined in the NIST document Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, by Wallace, Watson, and McCabe.

Halstead Metrics [**]

Halstead metrics are based on definitions of *operators* and *operands*. For this purpose, CodeSonar uses the following definitions, which may differ from the definitions used by other Halstead Measure tools:

Operators	
Arithmetic	`* , / , + , - , % , ++ , --`
Relational and Equality	`== , != , /= , < , <= , > , >=`
Assignment	`= , += , -= , *= , /= , %= , <<= , >>= , &= , ^ , |`
Shift	`<< , >>`
Bitwise	`& , | , ^ , && , ||`
Unary	`- , ! , sizeof , ~ , * , & , +`

Control-points	`if` , `while` , `dowhile` , `for` , `switch-case`
Type specifiers	`void` , `short` , `int` , `long` , `float` , `double` , `signed` , `unsigned` , `struct` , `union` , `enum`
Storage class specifiers	`auto` , `register` , `static` , `extern` , `typedef`
Other	`break` , `continue` , `goto` , `else` , comma , semicolon , type-cast , array-reference , function-call , `return`
Operands	
Identifier	
Literal	where each literal is treated as a distinct operand.
Label	
Name of the module	

User-Defined Metrics

CodeSonar provides two main mechanisms for defining custom metric classes. These mechanisms are available for the C/C++, C#, and Java language modules.

- Use `METRIC_DERIVED_DEF` configuration file rules to define new *derived metric classes*, whose values are computed by mathematically manipulating values of existing metrics.
- Use the CodeSonar Plug-in API to define metric classes with other computation functions.

Availability

Metric availability is based on two factors: which metrics are computed by a particular analysis, and which metrics are stored at any given time.

Availability: Computation

The table of built-in metrics indicates which are computed by default, and which are not.

Use the `METRIC_FILTER` configuration file parameter to

- override these defaults, and
- specify whether or not values should be computed for any user-defined metrics.

Availability: Storage

When an analysis computes metrics at analysis, compilation unit, and file granularities, it submits the metric values to the hub. These values are stored in the hub database with other information about the analysis, and remain there unless the analysis is subsequently deleted. This means that *analysis/compilation-unit/file metric values computed by all analyses on the hub are always available*.

Procedure-granularity metrics values are not submitted to the hub: instead, the analysis stores them in the project analysis directory. Any subsequent analysis in the same directory will delete the stored values and replace them with newly-computed metric values. This means that *procedure-granularity metric values are only available for the most recent analysis of each project*, except in certain cases that are described fully in Procedures: Availability.

Directory-granularity metrics are not currently available in the CodeSonar GUI or through the plug-in API.

Accessing Metric Values

Programmatic access to metric values is available through the plug-in API.

CodeSonar GUI provides access to metric values in several locations, as described in the following table.

Analysis	Provides several mechanisms for accessing metric information: • The Metrics section provides links to report shortcuts and to the Metric Report Creation page. • The Files tab provides columns for all file-granularity metrics computed for the analysis (built-in or user-defined). • The Procedures tab provides columns for all procedure-granularity metrics computed for the analysis (built-in or user-defined).
Home	Columns for all analysis-granularity metrics (built-in or user-defined) are available in the table of projects.
Metric Report	Presents a user-specified subset of the metric values recorded by the hub.
Project	Provides several mechanisms for accessing metric information:

- Columns for all analysis-granularity metrics (built-in or user-defined) are available in the table of analyses.
- The Reports section provides functionality for generating a management report based on the predefined **Metrics Over Time Report** template.

Search

Metrics are available in various search results as follows:

- File Search Results include columns for all file-level metrics.
- Procedure Search Results include columns for all procedure-level metrics.
- Metric Search Results are displayed in a Metric Report.

Source Listing

Values for all file-granularity metrics computed for the file are displayed in the File Details section.