HIS Source Code Metrics

Stand: 01.04.2008
Version: 1.3.1
Autor: Kuder, Helmar

Mitarbeit:

Audi	Albrecht Korn, Erwin Haunschild
BMW Group	Dr. Bernhard Kalusche
DaimlerChrysler	Helmar Kuder, Martin Huber, Andreas Krüger, Dr. Eric Sax (MBtech GmbH)
Porsche	Dr. Rüdiger Dorn, Jesper Hansson
Volkswagen	Herbert Tschinkel, Jörg Kluge

Inhaltverzeichnis

History

Version	Date	Change
1.0	31.05.2005	First Official Version
1.1	30.11.2005	Range Reference to the Document, "HIS requirements for Software Test" New Glossary.
1.2	14.02.2006	Implementation of Tables 3-1 and 3-2, spelling corrections, limit value for ap-cg-cycle, 'Binding' Metrics without limits
1.3	11.05.2007	Update comments for metrics CALLING, stability index S_i, NOMV, NOMVPR and ap_cg_cycle.
1.3.1	01.04.2008	Adding titel for ap_cg_cycle: “number of recursions“

1 Background

1.1 General

HIS (**H**ersteller **i**nitiative **S**oftware - Manufacturer's software initiative) five working groups from the Automotive manufacturer's Audi, BMW Group, DaimlerChrysler, Porsche and Volkswagen whose goal is the production of agreed standards within the areas of Standard software modules for networks, Development of process maturity, Software test, Software tools and Programming of ECU's.

Software Metrics are the basis for efficient project and quality management. With software Metrics statements can be made about the quality of the software product and the software development process.

In this document HIS specifies a fundamental set of Metrics to be used in the evaluation of software.

1.2 Reference Documents

Table 1-1 Reference Documents

Document	Title
HIS Subset MISRA C 1.0.2	General Subset of the MISRA C Guidelines Dated: 22.03.2004; Version: 1.0.2
ISO/IEC 9899:1999	Programming languages - C Stand: 1999
Definition „Cyclomatic Complexity“	Complexity Metric (Arthur H. Watson, Thomas J. McCabe; Computer Systems Laboratory; National Institute of Standards and Technology; Gaithersburg, MD 20899-0001; United States of Americe) Date: August 1996 http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/chapter2.htm

1.3 Glossary

Metric

A measure (Metric) is the objective allocation of a value to an entity, in order to characterise a specific feature. (Source: German-language group of users for software Metrics and expenditure estimates http://www.dasma.

Statement

An action that can be implemented on an individual basis.
(Source [ISO/IEC 9899:1999]; Kapitel 6.8: „A statement specifies an action to be performed.“)

2 Outline

2.1 Metric Groups

The fixed Metrics are raised on the basis of a "compilable unit"[1]. A Metric Statement should be raised for each function.

2.2 Variation of Metrics

The software supplier is responsible for the documentation of the measured values in accordance with the software Metrics defined in this document.

A fundamental distinction exists between Metrics which only need to be documented (table 3-2 Metrics without boundary limits) and in Metrics (table 3-1 Metrics with boundary limits), with which a violation of the boundary limit will lead to further action.

All metrics have to be reported, all violations of the agreed boundary limits at the function level have to be justified.

Correction of violations of the boundary limits of the metrics is user-specific (e.g. Functional Software, Operating System) and OEM-Specific. Each OEM could specify, for example, outside of the definitions in Table 3-l, which we'll call the "Green range", wider ranges within which he accepts the software subject to rework, the "Yellow range " or completely rejects it, the "Red range 'I.

[1] Different Analysis tools define "compilable unit" to be "total software " even if the external interfaces refer to the opposite (partial extent), i.e. Limit values can differ dependent on whether "total software "or "partial extent " is being referenced.

3 Metrics – Measurement

3.1 Metrics with limits

Table 3-1 Metrics with limits

Metric	*Description*	*Comment*	*Range*
Comment Density “COMF“	Relationship of the number of comments (outside of and within fubctions) to the number of statements	Comprehensibility, clarity in the code. With violation documentation to decide on the acceptance or rejection. According to its definition the value cannot be > 1 For example Logiscope: COMF = (BCOM + BCOB) / STMT For example QA-C: COMF = (STM28 / STM22)	> 0,2
Number of paths “PATH“	Number of non cyclic remark paths (i.e. minimum number of necessary cases of test)	Measurement for the reduction of PATH:Allocation into several functions, Display in sub-functions. For example Logiscope: ct_path, PATH For example QA-C: STPTH	1 - 80
Number of Go to Statements “GOTO“	Number of Go to Statements	This drastically increases the number of paths and thus reduces testability For example Logiscope: ct_goto, GOTO For example QA-C: STGTO	0
Cyclomatic Complexity “v(G)“	In accordance with the Cyclomatic Number [Definition „Cyclomatic Complexity“]	Measurement for the decrease in v(G): Allocation across several functions with paging into and out of sub-functions. For example Logiscope: ct_vg, VG, ct_cyclo For example CodeSurfer: vG For example QA-C: STCYC	1 - 10
Number of Calling Functions “CALLING“	By how many subfunctions is this function called?	The range 1 to 5 only makes sense when the entire system is analysed. With subsystems and libraries 0 to 5 is permissible. For the complete system there is the exception of main(), since this will only be called by the Start-up-Code. For example Logiscope: dc_calling, NBCALLING For example QA-C: STM29	0 - 5
Number of called functions „CALLS“	How many different functions does this function call? Calling the same subfunction counts only once.	Examine width of nesting For example Logiscope: dc_calls, DRCT_CALLS For example QA-C: STCAL	0 - 7

Metric	Description	Comment	Range
Number of Function Parameters „PARAM“	How complex is the function interface?	Complexity of the function, need for stack Structures and Arrays hide the complexity in the same way. For example Logiscope: ic_param For example QA-C: STPAR	0 - 5
Number of Instructions per function „STMT“	How complex is the function?	Empty functions fall through For example Logiscope: lc_stat, STMT For example QA-C: STST3	1 - 50
Number of call Levels „LEVEL“	Depth of nesting of a function.	Maximum nesting levels within a function + 1 For example Logiscope: LEVL For example QA-C: STMIF	0 - 4
Number of return points „RETURN“	Number of return points within a function	Complexify of a function, Maintainability of a function, When does a function return. 0 = a function without a specific return statement For example Logiscope: RETU For example QA-C: STM19	0 - 1
The stability index „S_i“	The stability index supplies a measure of the number of the changes (changes, deletions, additions) between two versions of a piece of software. Stability Index Si = (STMT - (S_{change} + S_{new} + S_{del}))/STMT	To be collected at a function level or at the total software level. The value is per definition <= 1. The lower this value of this metric the more changes the new software version contains. If no changes had been made overall then the value of this metric would be 1. As might be expected the stability index for two early successive versions of a piece of software will be rather low and should always then continue to rise for the later versions. For example Logiscope: not supported For example QA-C: not supported	0 - 1

Metric	Description	Comment	Range
Language scope „VOCF“	The language scope is an indicator of the cost of maintaining/changing functions. VOCF = (N1 + N2) / (nl + n2), where nl = Number of different operators N1 = Sum of all Operators n2 = Number of different Operands N2 = Sum of all Operands	Higher value = similar or duplicated code portions; Calls inlout of sub-functions need to be considered For example Logiscope: VOCF For example QA-C: Ermittelbar aus STOPN (n2), STOPT (n1), STM21 (N1), STM22 (N2)	1 - 4
„NOMV“	Total number of the violations of the Rules of the [HIS Subset MISRA C 1.0.2]. NOMV = Number of MISRA HIS Subset violations	Compliance of HIS rules within the Software production process For example Logiscope: nicht vorhanden For example QA-C: can be determined over Compliance Report.	0
„NOMVPR“	Number of violations of each rule of the [HIS Subset MISRA C 1.0.2], Classified according to the rules. NOMVRP = Number of MISRA violations per rule	Compliance of HIS rules within the software production process. Should also contain a rule to identify ,,Dead Code” For example Logiscope: nicht vorhanden For example QA-C: can be determined over Compliance Report.	0
Number of recursions „ap_cg_cycle“	Call graph recursions	Recursions over one or more functions. For example Logiscope: ap_cg_cycle , GA_CYCLE For example QA-C: STNRA	0

3.2 Metrics without limits

The Metrics specified in the following table represent pure measured values which must still be documented in every case.

Table 3-2 Metrics without limits

Metric	*Description*	*Comment*	*Range*
„S_{change}“	STMT(changed) The number of statements in a piece of software which have changed between the previous and the current version of the software.	Evaluation of the extent of change since the last official software release. This value is needed for the calculation of the stability index of Si. For example Logiscope: not supported For example QA-C: not supported	–
„S_{del}“	STMT(deleted) The number of statements in a piece of software which have been deleted between the previous and the current version of the software.	Evaluation of the extent of change since the last official software release. This value is needed for the calculation of the stability index of Si. For example Logiscope: not supported For example QA-C: not supported	–
„S_{new}“	STMT(new) The number of statements in a piece of software which have added between the previous and the current version of the software.	Evaluation of the extent of change since the last official software release. This value is needed for the calculation of the stability index of Si. For example Logiscope: not supported For example QA-C: not supported	–