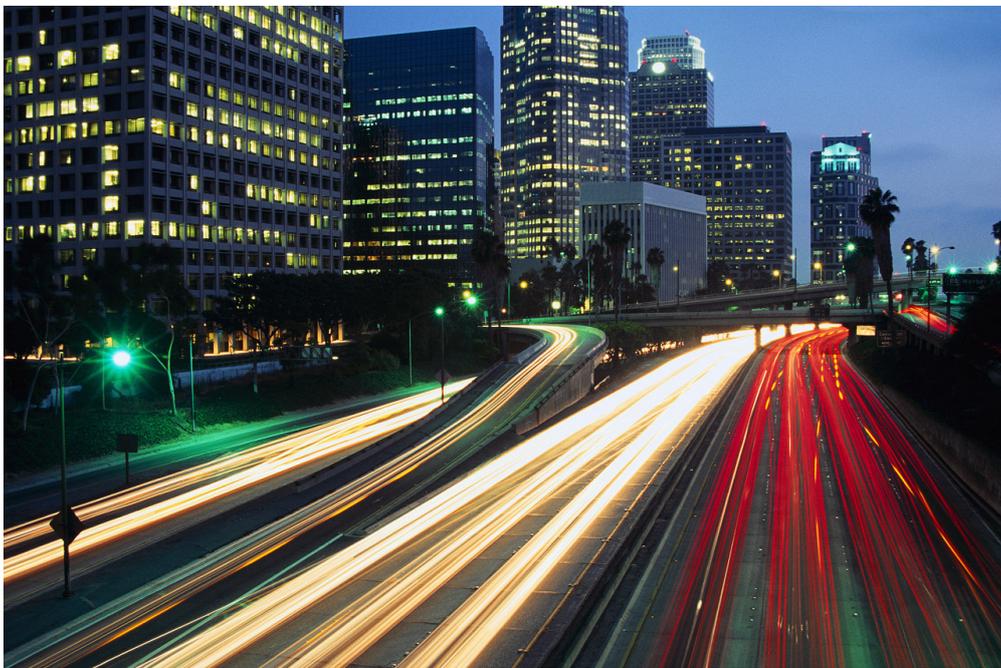




Metric Cards for Automotive Software Projects



Technical Report

Version 1.0 – October 2012

How to reference this document:

MASP, *Metric Cards for Automotive Software Projects, version 1.0*, TR-2012-01, Metrics for Automotive Software Projects (MASP) working group, Automotive SPIN Italy, Technical Report, October 2012

For more information about other Process Improvement, Software Measurement & Quality issues, please visit:

< <http://www.automotive-spin.it/> > or contact the Authors by email at luigi.buglione@computer.org

Copyright © 2012 Automotive SPIN Italia. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the consensus of the Author.

First Printing: October 2012

Table of Contents

1	Document Information.....	4
1.1	Executive Summary	4
1.2	History	4
1.3	Working Group (WG) participants	4
1.4	Acronyms.....	4
1.5	References	5
2	Introduction	6
2.1	Background and Rationale.....	6
2.2	Metric Cards: Fields and Layout.....	6
2.3	What to Measure?	7
2.4	Are selected measures ‘balanced enough’?	8
3	Metrics cards	9
3.1	CBO – (Avg of) Coupling Between Objects.....	9
3.2	CC1 – McCabe Cyclomatic Complexity	11
3.3	CDRE – Company Defect Removal Efficiency	12
3.4	CRE – Change Request Effort.....	13
3.5	CTA – Class Type Attributes	14
3.6	ENC – Engineering Non-Conformance	15
3.7	EXC – External Calls.....	16
3.8	IFC – Information Flow Complexity	17
3.9	OCC – Memory Occupation.....	18
3.10	PSM – Product Software Modification	19
3.11	RDR – Rule Deviation Rate.....	20
3.12	REI – Reliability Index.....	22
3.13	RES – Requirement Stability	23
3.14	SRD – Software Robustness Distribution	24
3.15	SFIN – Structural Fan-In.....	26
3.16	SFOUT – Structural Fan-Out.....	27
3.17	WPU – Work Product Usage	28

1 Document Information

1.1 Executive Summary

The purpose of this document is to propose a balanced set of measures [3] to be used in a measurement plan conforming to the ISO/IEC 15504 Process Reference Model (PRM) [1]. Such measures are defined and described using a template derived from the Measurement Information Model (MIM) proposed in Appendix A of ISO/IEC 15939 standard [2]. This document is the first outcome of the **MASP** (Metrics in Automotive Software Projects) working group of Automotive SPIN Italy (www.automotive-spin.it).

1.2 History

Revision	Date	Changes since last revision
1.00	October 2012	• First issue

1.3 Working Group (WG) participants

Name/Surname	Affiliation
Concetta ARGIRI	TXT Group
Roberto BAGNARA	Univ. Parma/BUGSENG Srl
Marina BORGHI	---
Luigi BUGLIONE	Engineering.IT SpA
Domenico DI LEO	Univ. Federico II
Lorenzo FALAI	Resiltech Srl
Mario FUSANI	ISTI/CNR
Giuseppe LAMI	ISTI/CNR
Leonardo RICCI	Magneti Marelli
Francesco ROSSI	Resiltech Srl
Isabella RUOCCO	Magneti Marelli

1.4 Acronyms

Acronym	Description
A-SPIN	Automotive SPIN Italia (www.automotive-spin.it)
BMP	Balancing Multiple Perspectives
BSC	Balanced Scorecard
ENG	Engineering process group (ISO/IEC 15504)
GQM	Goal-Question-Metric
IEC	International Electrotechnical Commission (www.iec.ch)
IS	International Standard
ISO	International Organization for Standardization (www.iso.org)
LOC	Line of Code
MAN	Management process group (ISO/IEC 15504)
MASP	Metrics in Automotive Software Projects
MIM	Measurement Information Model (ISO/IEC 15939:2007, App.A)
PAM	Process Assessment Model
PRM	Process Reference Model
SLC	Software Life Cycle
SPICE	Software Process Improvement Capability dEtermination (ISO/IEC 15504)
SUP	Support process group (ISO/IEC 15504)
WG	Working Group

1.5 References

Ref	Title
[1]	ISO/IEC, <i>IS 15504-2:2003 – Information Technology – Process Assessment – Part 2: Performing an assessment</i> , International Organization for Standardization, October 2003, URL: www.iso.org
[2]	ISO/IEC, <i>IS 15939:2007 – Systems and Software Engineering – Measurement process</i> , International Organization for Standardization, February 2007, URL: www.iso.org
[3]	Buglione L., <i>Top metrics for SPICE-compliant projects</i> , Automotive-SPIN Italia, 5° Automotive SPIN workshop, Milan (Italy), June 4 2009, URL: www.automotive-spin.it
[4]	Buglione L. & Abran A., <i>Multidimensional Project Management Tracking & Control - Related Measurement Issues</i> , Proceedings of SMEF 2005, Software Measurement European Forum, 16-18 March 2005, Rome (Italy), pp. 205-214, URL: www.dpo.it/smef2005/filez/proceedings.pdf
[5]	Automotive SIG., <i>Automotive SPICE® Process Reference Model (PRM)</i> , v4.5, May 10 2010, URL: www.automotivespice.com
[6]	Buglione L., <i>Top10 Metrics: Metric Cards</i> , version 1.0, WP-2011-01, White Paper, April 1 2011, URL: www.semq.eu
[7]	Buglione L., Ebert C., <i>Estimation</i> , Encyclopaedia of Software Engineering, Taylor & Francis Publisher, June 2012, ISBN: 978-1-4200-5977-9

2 Introduction

2.1 Background and Rationale

During the past two years, several discussions and presentations at Automotive SPIN Italia focused on the need of measuring software projects in order to improve the monitoring and controlling processes. It was thus decided to establish a working group whose objective is to provide advice and recommendations about the deployment of software measurement in the automotive field. This technical report, which is the first outcome of the working group, contains several ‘metric cards’, each one defining one software metric. Some of these metrics are well known; others are less known and/or contain original material developed by the working group. In all cases, the presentation based on metric cards allows for a clear and uniform presentation of all metrics. The objective is to help the people in charge of collecting, interpreting and basing actual decisions upon the measures to apply the same definition for the same concept. This reduces the likeliness of ending up with historical data that are not comparable or that need several (possibly questionable) assumptions in order to derive ‘numbers’.

This report should be interpreted as a ‘living document’: the metric cards it contains are meant to be regularly revised and new metric cards will be added as the need arises.

The objective of this work is twofold: on the one hand, it aims to foster a culture of measurement in the involved organizations, following the well-known motto by Tom Demarco that ‘*you cannot manage what you cannot measure*’ but ‘*you cannot measure what you cannot define*’. On the other hand, it endeavours to promote awareness of the importance to measure the right amount and selection of phenomena: basing a project on a single measure or basing it on too many measures are both dangerous/wasteful mistakes. The last aspect is stressed in the BMP technique [4] and briefly summarized in Section 2.2 of [6] (*How much to measure?*).

2.2 Metric Cards: Fields and Layout

A set of ‘metric cards’ is proposed in Section 3, with the following structure and fields:

- **Measure title/code:** title and an optional code for the measure
- **A-SPICE PRM process:** associated Automotive SPICE (A-SPICE) PRM process
- **Purpose:** a short sentence summarizing the informative goal of the measure
- **Entity:** measurable entity for the measure : {organization | project | resource | process | product}
- **Attribute:** the related attribute for the measured entity
- **SLC phase where applied:** the SLC phase where the measure can be applied, according to the adopted type and taxonomy
- **Unit of measure:** the countable unit for such measure
- **Measurement scale:** {absolute | interval | ordinal | absolute | nominal }
- **Counting Rule:** a brief sentence summarizing what and how must be counted
- **Formula and Legend:** the mathematical expression for the previous field
- **Responsible for Gathering Data:** the people assigned to gather the data required for computing the measure
- **Gathering Frequency:** the suggested frequency for gathering the measure
- **Gathering Methodology:** the suggested methodology/technique for gathering the measure

- **Counting examples:** one or more short calculation examples for showing how the data should be used for computing the measure
- **Comments/Notes:** optional additional comments and notes about that measure
- **Possible Associated Questions:** a list of possible associated questions in a sort of reverse-QQM analysis.

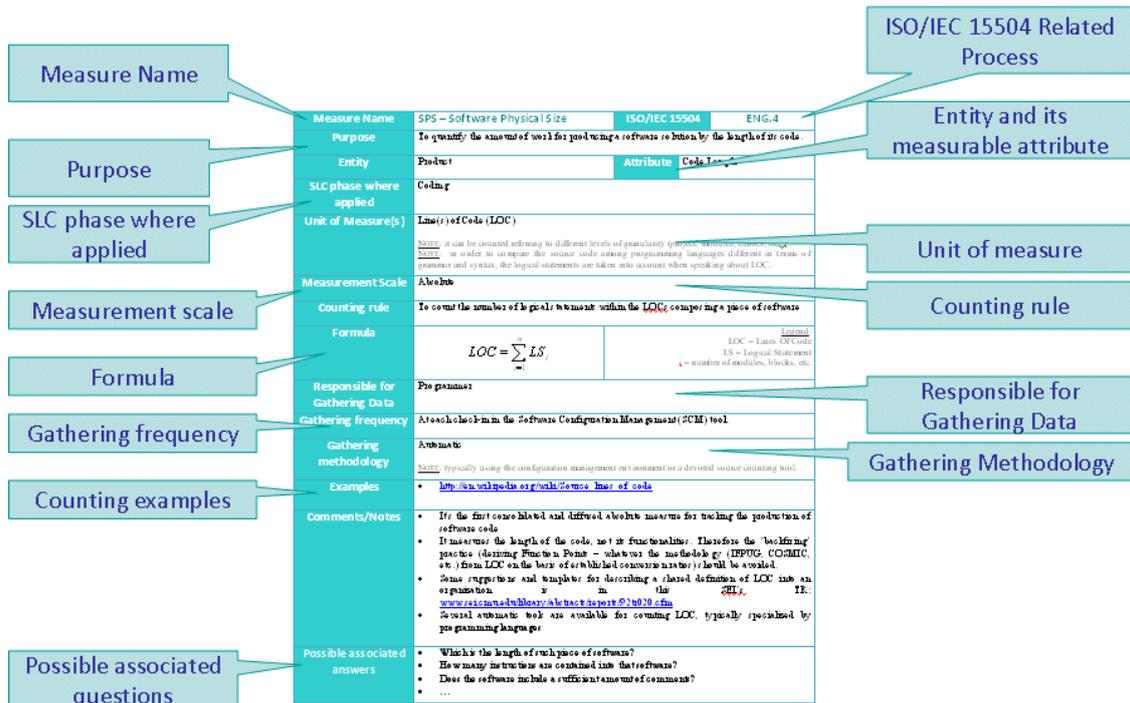


Fig. 1 – Metric Card: structure and fields

2.3 What to Measure?

Here in the following table the list of measures classifies the measures in Section 3, classified according to the EAM taxonomy [7]. Its purpose is to help in establishing whether a proper level of distribution among measurable entities and their attributes has been reached (or not) in a BMP analysis [6].

Entity (E)	Attribute (A)	Measure (M)	A-SPICE
Process	Testability	CDRE – Company Defect Removal Efficiency	ENG.8, ENG.10
Process	Process Performance	ENC – Engineering Non-Conformance	SUP.1
Process	Reliability	REI – Reliability Index	MAN.3, MAN.5
Process	Req. Elicitation capability	RES – Requirement Stability	ENG.1, ENG.4
Product	Code Stability	CBO – (Avg of) Coupling Between Objects	SUP.1
Product	Maintainability	CC1 – McCabe Cyclomatic Complexity	ENG.5, ENG.6
Product	Testability	CTA – Class Type Attributes	SUP.1
Product	Code Stability	EXC – External Calls	SUP.1
Product	Maintainability	IFC – Information Flow Complexity	ENG.6
Product	Maintainability	PSM – Product Software Modification	ENG.6, MAN.3

Entity (E)	Attribute (A)	Measure (M)	A-SPICE
Product	Reliability	RDR – Rule Deviation Rate	ENG.5, ENG.6
Product	Reliability	SDR – Software Robustness Distribution	ENG.5, ENG.6
Product	Maintainability	SFIN – Structural Fan-In	ENG.5, ENG.6
Product	Maintainability	SFOUT – Structural Fan-Out	ENG.5, ENG.6
Project	Changeability	CRE – Change Request Effort	MAN.3, SUP.10
Project	Effectiveness	WPU – Work Product Usage	PA 2.2, 3.2, 4.2, 5.2
Resource	Maintainability	OCC – Memory Occupation	ENG.6

2.4 Are selected measures 'balanced enough'?

The measures presented in this technical report represent only a compilation of experiences by the authors, but cannot of course be exhaustive in terms both of information completeness and coverage. For such purpose, next figure propose the BMP matrix [4], helping to making a quick check for properly balancing the list of measures an organization intends to put in action into its own measurement plan(s) by two criteria yet elicited in each measure card and proposed by the EAM analysis [7]: the measured entity and the attribute that such measure expresses.

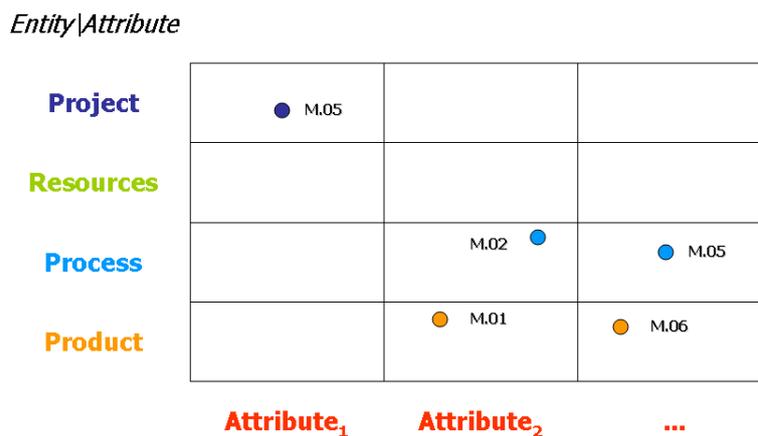


Fig. 2 – BMP matrix: generic structure

In the above example, the following elements could arise from the BMP analysis:

- No resource measures (is it correct?)
- Concentration of measures (4 out of 5) about process and product measures (is it ok?)
- Only a single measure at the project level (→ is it about time and/or cost?)
- Are we missing some relevant attributes from our GQM-based analysis in terms of mandatory/recommended informative goals to be answered?
- ...

remembering that [4]:

- the real issue is **not** to reduce the cost of the measurement process,
- **but** optimising it against the informative value provided by the number of measures/indicators balancing them by each perspective of analysis.

3 Metrics cards

3.1 CBO – (Avg of) Coupling Between Objects

Measure Name	CBO – Avg of Coupling Between Objects	ISO/IEC 15504	SUP. 1
Purpose	To minimize the presence of systematic faults.		
Entity	Product	Attribute	Code Stability
SLC phase where applied	Software Validation		
Unit of Measure(s)	Relationships between objects		
Measurement Scale	Ratio		
Counting rule	To count the number of logical statements within LOCs composing a piece of software		
Formula	$AVG_CBO = \frac{\sum_{classes} (cl_func_calle + cl_data_class)}{ap_clas}$		Legend: <u>cl_func_calle</u> = Sum of external calls <u>cl_data_class</u> = Sum of class-type attribute <u>ap_clas</u> = No. of classes in the application
Responsible for Gathering Data	Software Designers		
Gathering frequency	At each check-in done by the Software Configuration Management (SCM) tool.		
Gathering methodology	Automatic <ul style="list-style-type: none"> E.g. IBM Rational Logiscope QualityChecker 		
Examples	<p>AVG is an average value depending on the number of rows that make up the classes involved. Supposing to have established those threshold AVG_CBO values: min=0; max=10, here in the following an example:</p> <pre>File Check.cpp Check::Check (void) {} void Check::Trim(){ Persona p; p.Nome(); p.Cognome(); } void Check::Trim2(){ Persona p; p.Nome(); p.Cognome(); } File Pesona.cpp std::string Persona::Nome(){ return "Nome"; } std::string Persona::Cognome(){ return "Cognome"; }</pre> <p>In this case the value of <i>cl_fun_calle</i> e <i>cl_data_class</i> are respectively 2 and 2. The sum is 4 and the average is 2.</p>		
Comments/Notes	<ul style="list-style-type: none"> AVG_CBO can be used as part of static check to be run on the code used for safety critical applications. And is often used to assess the stability and maintainability of the application. 		

	<ul style="list-style-type: none"> • Original publication: Chidamber S.R., Kemerer C.F., A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, Vol.20, No.6, June 1994, pp. 476-493; URL: http://goo.gl/bg5w2 • AVG_CBO values out of thresholds show that a large number of classes in the application have a high coupling rate. A high coupling rate shows that the application includes a large number of interconnections, and therefore any change becomes extremely delicate. • The lower the CBO value, the higher the overall code quality for the observed project.
Possible associated questions	<ul style="list-style-type: none"> • Is the project easily maintainable and functionally stable?

3.2 CC1 – McCabe Cyclomatic Complexity

Measure Name	CC1 – McCabe Cyclomatic Complexity	ISO/IEC 15504	ENG.5, ENG.6
Purpose	To increase the maintainability of the source code		
Entity	Product	Attribute	Maintainability
SLC phase where applied	Coding		
Unit of Measure(s)	Edges; Nodes		
Measurement Scale	Interval		
Counting rule	Count of the number of linearly independent paths through the source code		
Formula	$CCI = E - N + 2P$	Legend: E = the number of edges of the graph N = the number of nodes of the graph P = the number of connected components	
Responsible for Gathering Data	Developer		
Gathering frequency	At the end of each build/release		
Gathering methodology	Automatic <ul style="list-style-type: none"> E.g.: Scitools (http://www.scitools.com/) 		
Examples	<ul style="list-style-type: none"> A counting example is presented in this NIST reports: http://goo.gl/OV8QQ 		
Comments/Notes	<ul style="list-style-type: none"> It measures the complexity of source code. A high cyclomatic complexity value represents the symptom for code difficult to debug and to maintain. It also gives a measure of the number of test cases needed to guarantee complete code coverage It can be applied to different levels of granularity (program, subroutine, method). <u>Original publication</u>: McCabe T., A Complexity Measure, IEEE Transaction on Software Engineering, Vol. SE-2, No.4, December 1976 (URL: http://goo.gl/tzqcF) 		
Possible associated questions	<ul style="list-style-type: none"> How many test cases are needed? Is the code too complex to be debugged and maintained? 		

3.3 CDRE – Company Defect Removal Efficiency

Measure Name	CDRE – Company Defect Removal Efficiency	ISO/IEC 15504	ENG.8, ENG.10
Purpose	To measure the ability of an organization to intercept and remove the largest number of possible Faults		
Entity	Process	Attribute	Testability
SLC phase where applied	Testing; Post-Delivery		
Unit of Measure(s)	Fault		
Measurement Scale	Ratio		
Counting rule	Compute the ratio between the Nr Fault found and the Global Number of Faults		
Formula	$CDRE = \frac{INTFT}{(INTFT + CUSFT)}$ <p style="text-align: right;"><u>Legend:</u> INTFT = Nr unique Fault found internally CUSFT= Nr unique Faults found by customer</p>		
Responsible for Gathering Data	Test Manager		
Gathering frequency	After each test and delivery phase		
Gathering methodology	Automatic or manual		
Examples	Having INTFT = 100 and CUSFT = 1 : CDRE= 99% <ul style="list-style-type: none"> It means that the Organization is able to discover the 99% of defects before the Unit (SW) Delivery. 		
Comments/Notes	<ul style="list-style-type: none"> This may be considered a new metric. A similar metric (DRE: Defect Removal Efficiency) is used to verify the ability of an organization to discover the maximum number of defects using faults injection. 		
Possible associated questions	<ul style="list-style-type: none"> Is the 99% discovered acceptable according to Safety requirements? Are we allocating the right resources for removing defects? Are they properly skilled? Do we track and properly classify defects within the organization? 		

3.4 CRE – Change Request Effort

Measure Name	CRE – Change Request Effort	ISO/IEC 15504	MAN.3, SUP.10
Purpose	To measure the impact/cost of changes from the customer in a project		
Entity	Project	Attribute	Changeability
SLC phase where applied	All the SLC Phases		
Unit of Measure(s)	Time – Note: it could be expressed in man/days or man/hours		
Measurement Scale	Absolute		
Counting rule	Sum the extra effort due to change requests by the customer per each SLC phase		
Formula	$CRE = \sum_{i=1}^N \sum_{j=1}^M CRTBE_i$	Legend: CRE = Change Request Effort CRTBE = Change Request Treatment Budgeted Effort N=Tot. No.of CRs in the project M = Tot. No. of SLC phases	
Responsible for Gathering Data	Project Manager		
Gathering frequency	At each iteration closure		
Gathering methodology	Semi-Manual <ul style="list-style-type: none"> Note: effort data gathering and CRM/Tracking tools could be used for making faster the calculation 		
Examples	<ul style="list-style-type: none"> --- 		
Comments/Notes	<ul style="list-style-type: none"> In Automotive software projects usually the number of change requests from the customer is typically high. The effort due to change requests is difficult to be estimated at the beginning. The availability of the ACE value (possibly associated with the specific customer and characteristics of the project) allows a better estimation of the project effort. The number of change requests from the customer may depend on customer market needs or poor requirements elicitation at the beginning of the project. In Software Engineering this phenomenon is also referred as 'scope creep' and should be possibly minimized. Monitoring in a regular way such measure could help in identify eventual root causes to be faced in advance during the project lifecycle and take the proper corrective/improvement action (e.g. seasonality, missing requirement elicitation capability, etc.) 		
Possible associated questions	<ul style="list-style-type: none"> Is my project effort estimation realistic considering the specific customer and the characteristics of the project? Should the requirements elicitation phase be improved for this specific project? 		

3.5 CTA – Class Type Attributes

Measure Name	CTA – Class Type Attributes	ISO/IEC 15504	SUP.1
Purpose	To minimize the presence of systematic faults.		
Entity	Product	Attribute	Testability
SLC phase where applied	Software Validation		
Unit of Measure(s)	Class-type attribute		
Measurement Scale	Absolute		
Counting rule	Sum the class-type attributes for each class.		
Formula	$cl_data_class = \sum_{methods} c_type_att$	<u>Legend:</u> C_type_att : class-type attributes for the class	
Responsible for Gathering Data	Software Developer		
Gathering frequency	At each check-in from the Software Configuration Management (SCM) tool.		
Gathering methodology	Automatic <u>Note:</u> e.g IBM Rational Logiscope QualityChecker		
Examples	Suppose to analyze the following code chunk: <pre>class Sample { private string name; }</pre> In this case CTA = 1.		
Comments/Notes	<ul style="list-style-type: none"> This metric is typically used to calculate the Average Value of CBO (coupling between object) and assess the stability of the application, as part of static check to be run on the code used for safety critical applications. The higher the CTA value, the better the code quality. It is not possible to predefine the thresholds for CTA because its value depends on the number of rows in the class. 		
Possible associated questions	<ul style="list-style-type: none"> Is the project easily stable? Was the number of data class(es) within the established threshold yet from the Design phase? 		

3.6 ENC – Engineering Non-Conformance

Measure Name	ENC – Engineering Non-Conformance	ISO/IEC 15504	SUP.1
Purpose	To quantify the number of Non-Conformities (NCs) from Engineering processes		
Entity	Process	Attribute	Process Performance
SLC phase where applied	All SLC phases.		
Unit of Measure(s)	Non-Conformity		
Measurement Scale	Absolute		
Counting rule	Sum the NCs collected related to product, technical documentation and management processes		
Formula	$ENC = NC_P + NC_{TD} + NC_{MP}$		Legend: NC _P = Product NC NC _{TD} = Technical documentation NC NC _{MP} = Management Process NC
Responsible for Gathering Data	<ul style="list-style-type: none"> QA Manager 		
Gathering frequency	<ul style="list-style-type: none"> Monthly 		
Gathering methodology	<ul style="list-style-type: none"> Manual 		
Examples	<ul style="list-style-type: none"> --- 		
Comments/Notes	<p>Design phase outputs can generate three types of Non-Conformities (NCs) to be eventually graded by severity (see e.g. ISO 19011:2012 standard on audits):</p> <p>A. <u>Product NCs</u> Related to products (outputs) not compliant with customer requirements.</p> <p>Origin During the design process: <i>errors in the design verification/design reviews</i> <i>reports of customers within/outside the organization</i></p> <p>After material and/or equipment purchasing: <i>not positive results of controls during product manufacturing and testing</i></p> <p>During construction and commissioning: <i>presence of interference or unexpected anomalies in the site</i> <i>interface errors</i> <i>construction as planned not possible</i> <i>negative outcomes of all or part of the commissioning</i> <i>reports of failures/faults during the pre-operation</i></p> <p>After customer delivery: <i>reports of failures/faults during the operation</i></p> <p>B. <u>Technical documentation NCs</u> Related to works performed without any valid/complete technical-construction documentation that could have an impact on the management of the project and/or of the design of the software solution.</p> <p>C. <u>Management process NCs</u> Related to any failure from the application of the Quality Management System (QMS)</p>		
Possible associated questions	<ul style="list-style-type: none"> What is an Engineering Non-Conformance? In which life cycle phase has an NC been identified? In which life cycle phase has the NC been originated? 		

3.7 EXC – External Calls

Measure Name	EXC – External Calls	ISO/IEC 15504	SUP.1
Purpose	To minimize the presence of systematic faults		
Entity	Product	Attribute	Code Stability
SLC phase where applied	Software Validation		
Unit of Measure(s)	External call		
Measurement Scale	Absolute		
Counting rule	Compute the sum of total number of calls from the class methods to non-member functions or member functions of other classes		
Formula	$EXC = \sum_{methods} dc_callpe$	<u>Legend:</u> dc_call : Number of External Direct Calls	
Responsible for Gathering Data	Software Designers		
Gathering frequency	At each check-in from the Software Configuration Management (SCM) tool.		
Gathering methodology	Automatic <u>Note:</u> e.g IBM Rational Logiscope QualityChecker		
Examples	<p>Suppose to analyze the following code chunk:</p> <pre> #include <iostream.h> //Declaration of the function to be made as friend for the C++ Tutorial sample int AddToFriend(int x); class CPP_Tutorial { int private_data; friend int AddToFriend(int x); public: CPP_Tutorial() { private_data = 5; } int call() { AddToFriend(this.private_data); }; int AddToFriend(int x) { CPP_Tutorial var1; return var1.private_data + x; } </pre> <p>AddToFriend is a non-member method and it is called by <i>call()</i> method. So EXC calculated for the CPP_Tutorial class is equal to 1.</p>		
Comments/Notes	<ul style="list-style-type: none"> • This metric is often used to assess the stability of the application. • This metric is typically used to calculate the Average Value of CBO (coupling between object) and can be used as part of static check to be run on the code used for safety critical applications. • The higher the EXC value, the better the code quality. • It is not possible to predefine the thresholds for EXC because its value depends on the number of rows in the class. 		
Possible associated questions	<ul style="list-style-type: none"> • Is the project and functionally stable? 		

3.8 IFC – Information Flow Complexity

Measure Name	IFC – Information Flow Complexity	ISO/IEC 15504	ENG.6
Purpose	To determine the degree of maintainability of a module/function/file		
Entity	Product	Attribute	Maintainability
SLC phase where applied	Coding		
Unit of Measure(s)	Complexity		
Measurement Scale	Absolute		
Counting rule	The IFC is given by doubling the multiplication of the FanIn and FanOut. FanIn is the total number of functions (or methods) that call the function A. FanOut is the total number of functions (or methods) called by the function A.		
Formula	$IFC = Fanin_A * Fanout_A * 2$	<u>Legend:</u> Fan-in = The fan-in of procedure A Fan-.out = The fan-out of procedure.	
Responsible for Gathering Data	Programmer		
Gathering frequency	At each check-in in the Software Configuration Management (SCM) tool		
Gathering methodology	Automatic. <u>NOTE</u> : typically using the SCM environment or a devoted source counting tool		
Examples	<ul style="list-style-type: none"> • Function B and C invoke function A • Function A call function Z, X, Y, K • Therefore, FanIn = 2, FanOut = 4 → IFC = 16 		
Comments/Notes	<ul style="list-style-type: none"> • It can be applied at different levels of granularity (individual functions, modules, methods, classes of a program) • A high value for fan-in means that X is tightly coupled to other modules; as a consequence a change of X will have extensive propagation effects. • A high value for fan-out suggests that the overall complexity of X may be high because X has to interact with several modules. • Original publication: Henry, S.; Kafura, D., <i>Software Structure Metrics Based on Information Flow</i>, IEEE Transactions on Software Engineering Volume SE-7, Issue 5, Sept. 1981, pp. 510 – 518. 		
Possible associated questions	<ul style="list-style-type: none"> • How much reusable are the software modules? • What impact has a change of a module over the rest of software architecture? • In which extent does a software defects in the module X can propagate over the software architecture? 		

3.9 OCC – Memory Occupation

Measure Name	OCC – Memory Occupation	ISO/IEC 15504	ENG.6																		
Purpose	To track project progress by ROM and RAM occupation in order to document resources consumption and monitor consumption targets.																				
Entity	Resource	Attribute	Maintainability																		
SLC phase where applied	Implementation																				
Unit of Measure(s)	Kbyte																				
Measurement Scale	Ratio																				
Counting rule	To calculate the ratio between the memory occupied and the overall memory available (ROM, RAM)																				
Formula	$OCC = \frac{\sum_0^n var}{SIZE}$	Legend: OCC= Percentage of occupation var= variable-length code SIZE=Overall memory available																			
Responsible for Gathering Data	<ul style="list-style-type: none"> Software developer 																				
Gathering frequency	<ul style="list-style-type: none"> At each software release 																				
Gathering methodology	<ul style="list-style-type: none"> Semi-automatic <u>Note:</u> using internal script to sum the variable occupation 																				
Examples	<ul style="list-style-type: none"> Suppose to allocate 2 RAM sections: RAM_NEAR; RAM_FAR to manage computations which run at different task rates; the compiler will measure the occupation of this two sections; the sum is the numerator; the physical RAM dimension is the denominator <table border="1" data-bbox="528 1144 1347 1267"> <thead> <tr> <th>MEM_CLASS</th> <th>START_ADDR</th> <th>END_ADDR</th> <th>SIZE_KB</th> <th>OCC_KB</th> <th>OCC_PERC</th> </tr> </thead> <tbody> <tr> <td>FLASH</td> <td>0x00000000</td> <td>0x0017FFFF</td> <td>1536</td> <td>951,19</td> <td>61,93%</td> </tr> <tr> <td>RAM</td> <td>0x40000000</td> <td>0x400177FF</td> <td>94</td> <td>40,26</td> <td>42,83%</td> </tr> </tbody> </table>			MEM_CLASS	START_ADDR	END_ADDR	SIZE_KB	OCC_KB	OCC_PERC	FLASH	0x00000000	0x0017FFFF	1536	951,19	61,93%	RAM	0x40000000	0x400177FF	94	40,26	42,83%
MEM_CLASS	START_ADDR	END_ADDR	SIZE_KB	OCC_KB	OCC_PERC																
FLASH	0x00000000	0x0017FFFF	1536	951,19	61,93%																
RAM	0x40000000	0x400177FF	94	40,26	42,83%																
Comments / Notes	<ul style="list-style-type: none"> The memory occupations are monitored in order to have an indication of targets compliance and an indication of available resources for implementing new functionalities The measurement is generally available by the compiler or can be calculated with a simple sum of memory sections http://en.wikibooks.org/wiki/Embedded_Systems/Memory 																				
Possible associated questions	<ul style="list-style-type: none"> How do ROM and RAM occupation increase in comparison to consumption targets? How many resources are available for further developments? Is it possible to reduce HW costs resizing memory devices? 																				

3.10 PSM – Product Software Modification

Measure Name	PSM – Product Software Modification	ISO/IEC 15504	ENG.6, MAN.3						
Purpose	To measure the amount of modification applied to a software product								
Entity	Product	Attribute	Maintainability						
SLC phase where applied	Coding, Maintenance								
Unit of Measure(s)	It can be applied by several levels of granularity to different units (e.g. individual functions, modules, methods, classes of a program)								
Measurement Scale	Ratio								
Counting rule	Compute the ratio between the total number of actually modified/added lines of the software over the lines of code already existing Note: the total number of already existing LOC shall be measured before applying modification.								
Formula	$PSM = \frac{MLC}{LOC}$	<u>Legend:</u> PSM= Product Software Modification MLC = number of modified/added logical statements LOC= number of existing logical statements							
Responsible for Gathering Data	Programmer								
Gathering frequency	At the end of a release of a milestone or a build.								
Gathering methodology	Automatic								
Examples	Suppose to have the following values:								
	<table border="1"> <thead> <tr> <th>MLC</th> <th>LOC</th> <th>PSM</th> </tr> </thead> <tbody> <tr> <td>5</td> <td>15</td> <td>0.3</td> </tr> </tbody> </table>			MLC	LOC	PSM	5	15	0.3
MLC	LOC	PSM							
5	15	0.3							
Comments/Notes	<ul style="list-style-type: none"> PSM can be applied at several levels of granularity, choosing different units of measure (e.g. modules, functions, methods, software components). Values close to or greater than one means the target software have been profoundly changed. 								
Possible associated questions	<ul style="list-style-type: none"> Which software component, module, function requires more maintenance? What is the effort for maintaining a software component/module/function? 								

3.11 RDR – Rule Deviation Rate

Measure Name	RDR – Rule Deviation Rate	ISO/IEC 15504	ENG.5, ENG.6
Purpose	To improve the management of costs and risks associated to the construction or acquisition of software from third parties		
Entity	Product	Attribute	Reliability
SLC phase where applied	Coding, Testing		
Unit of Measure(s)	Logical source lines of code (LSLOC); Deviations		
Measurement Scale	Ratio		
Counting rule	Divide the number of weighted rule deviations by the number of thousands of LSLOC of the program/module being measured.		
Formula	$RDR = \frac{WRD}{KLSLOC}$		Legend: RDR = Rule Deviation Rate WRD = Weighted Rule Deviations KLSLOC = 1000 Logical Source Lines of Code
Responsible for Gathering Data	<ul style="list-style-type: none"> Programmer (to assist development or evaluation) Tester (to help focusing the testing effort) 		
Gathering frequency	<ul style="list-style-type: none"> Weekly during main development When needed, afterwards 		
Gathering methodology	Automatic Note: it is possible to use a coding rule checking platform that is flexible enough (such as ECLAIR, http://bugseng.com), or semiautomatic, using any coding rule checker followed by suitable post-processing.		
Examples	<ul style="list-style-type: none"> An existing code base is subject to new coding rules: RDR provides a measure of how far the code base is from compliance. A coding standard is being decided for a new project in some organization: measuring the RDR for similar projects developed in the same organization helps the decision process. A software project defines a set of required coding rules and a set of advisory coding rules: managers monitor the RDR of the advisory coding rules. Third-party software needs to be evaluated in terms of maintainability, changeability, reliability, testability: this can be done by measuring RDR with respect to a suitable set of coding rules. The need arises whereby a piece of software has to be evaluated in terms of how pointer-intensive it is. RDR is computed with respect to coding rules that limit the use of pointers, e.g., by constraining the use of pointer-arithmetic. Similarly, one wants to quickly assess how difficult it could be to port a piece of software to a different architecture. This can be achieved by measuring RDR with respect to a set of coding rules that constrain the use of architecture-dependent features. For example, let us consider the task of quickly evaluating the cost of porting a C/C++ application from a 32-bit to a 64-bit architecture. The main issues arising in such migration processes are well studied as are their likeliness to be problematic and the corresponding remediation costs. Here is a short selection, sorted by increasing gravity: (1) use of “magic constants” such as 4, 32, and 0x7FFFFFFF; (2) mixing size types and other integral types; (3) use of pointer arithmetic with non-portable integer types. These issues can be encoded as coding rules and a (flexible) coding rule verifier can automatically detect all of their occurrences (as coding rule violations), compute the WRD with user-supplied weights, and finally compute RDR. Continuing with the example, suppose issues (1), (2) and (3) are given the weights 0.9, 0.8 and 0.2, respectively. Code that has been written with 32-64-bit portability in mind should have an RDR close to 0; values of RDR of 5 strongly suggest a careful consideration of the potential costs of the migration task. 		
Comments/Notes	<ul style="list-style-type: none"> A <i>coding rule</i>, or <i>coding guideline</i>, is a prescriptive text that constrains the use of a programming language, e.g., by limiting the use of features that are error-prone or frequently miscompiled or that, in general, impede maintainability, changeability, 		

	<p>reliability and testability of software programs. A set of coding rules is called a <i>coding standard</i>. For example, <i>MISRA C</i> is a well-known coding standard for the C programming language developed by MISRA (Motor Industry Software Reliability Association -- www.misra.org.uk/). Here are two examples of (advisory) MISRA C rules: <i>Sections of code should not be “commented out”</i> (rule 2.4); <i>The declaration of objects should contain no more than 2 levels of pointer indirection</i> (rule 17.5).</p> <ul style="list-style-type: none"> • Coding rule deviations can be weighted with respect to: <ul style="list-style-type: none"> ▪ severity of the risks involved in the deviation; ▪ likelihood of the deviation to actually cause problems; ▪ remediation costs of such problems; and ▪ when applicable, a measure of the deviation (for example, for MISRA C rule 2.4, a block of commented out code can be weighted by its size; for rule 17.5 the extra levels of pointer indirection may contribute to the weight of the deviation). • A Logical Source Line of Code (LSLOC) is a software metric that approximates the number of executable “statements” independently from code layout. Even though the actual definition is more complex than that (and it can be defined very precisely based on the language grammar), for C-like programming languages the LLOC measure roughly corresponds to the number of statement-terminating semicolons (see, e.g., www.sei.cmu.edu/reports/92tr020.pdf). • A set of coding rules is selected. Optionally, a weight is assigned to each rule violation. An automatic tool gathers both the violations and the number of logical source lines of code in which they occur. • RDR can be applied to different levels of granularity (e.g., per-program or per-module). • RDR data for a set of coding rules can be shown on a per-rule basis, in order to study the various contributions to the RDR aggregate value.
Possible associated questions	<ul style="list-style-type: none"> • How far are we from compliance with respect to the coding rules we have chosen to conform to? • How far are we from compliance with respect to coding rules that we are not compelled to follow, even though we recognize their contribution to increased maintainability, changeability and so forth? • How do these third-party modules compare in terms of compliance with respect to the coding rules we care about?

3.12 REI – Reliability Index

Measure Name	REI – Reliability Index	ISO/IEC 15504	MAN.3, MAN.5												
Purpose	To measure the probability that no failure occurs during a given time interval														
Entity	Process	Attribute	Reliability												
SLC phase where applied	Testing, Maintenance														
Unit of Measure(s)	It can be applied to several levels of granularity by different units (e.g. individual functions, modules, methods, classes of a program)														
Measurement Scale	Ratio														
Counting rule	Compute the probability that no failure occurs during the interval $(0,t)$.														
Formula	$REI_i = 1 - \frac{Nf_{(0,t)}}{N}$	Legend: REI= reliability index estimated over the interval $(0,t)$ $Nf_{(0,t)}$ = number of failed execution during the time interval $(0,t)$ N= total number of executions													
Responsible for Gathering Data	Tester														
Gathering frequency	<ul style="list-style-type: none"> At the end of Testing phase. At fixed interval time after the release of software. 														
Gathering methodology	Manual														
Examples	<table border="1"> <thead> <tr> <th>Time interval after release (days)</th> <th>Number of failed execution, Nf</th> <th>Number of execution, N</th> <th>REI_i</th> </tr> </thead> <tbody> <tr> <td>0-15</td> <td>3</td> <td>100</td> <td>0.97</td> </tr> <tr> <td>0-30</td> <td>25</td> <td>500</td> <td>0.95</td> </tr> </tbody> </table>			Time interval after release (days)	Number of failed execution, Nf	Number of execution, N	REI _i	0-15	3	100	0.97	0-30	25	500	0.95
Time interval after release (days)	Number of failed execution, Nf	Number of execution, N	REI _i												
0-15	3	100	0.97												
0-30	25	500	0.95												
Comments/Notes	<ul style="list-style-type: none"> REI can be applied at different levels of granularity (module, functions, methods as well as software components). Values close to one indicate that the software is high reliable over a given interval. The Reliability Index can also be used in reliability growth models that estimate the reliability of the software (http://rmod.ee.duke.edu). A reliability growth model can be used to plan when to release software. 														
Possible associated questions	<ul style="list-style-type: none"> How much reliable is the observed software? Or how much is it prone to fail? When should be stopped the testing phase? Which parts of the software seem to be less reliable? For software components that expose the same functionalities and cost can we choose the more reliable? 														

3.13 RES – Requirement Stability

Measure Name	RES – Requirement Stability	ISO/IEC 15504	ENG.1, ENG.4												
Purpose	To evaluate the affordability of the estimation process														
Entity	Process	Attribute	Req. elicitation capability												
SLC phase where applied	Analysis														
Unit of Measure(s)	Requirements														
Measurement Scale	Ratio														
Counting rule	<p>Compute the ratio between the number of software requirements changed (added, modified and deleted) and the existing software requirements</p> <p><u>Note:</u> (1) this ratio should be calculated with respect to a time interval or different software versions; (2) the software requirement can be functional as well as not functional, it is also possible to evaluate both of them with the same metric</p>														
Formula	$RS_{k-1,k} = \frac{RC}{ER}$	<p><u>Legend:</u></p> <p>RS_{k,k-1}= Requirement Stability ratio evaluated over the time interval [k-1,k] or from the version k-1 to version k</p> <p>RC = number of requirements changed</p> <p>ER= number of existing requirement at time k-1 or for the version k-1</p>													
Responsible for Gathering Data	Analyst														
Gathering frequency	<ul style="list-style-type: none"> At the end of the design phase 														
Gathering methodology	<ul style="list-style-type: none"> Manual 														
Examples	<table border="1"> <thead> <tr> <th>Software Version</th> <th>ER-Total number of existing requirements</th> <th>RC-Number of requirements changed</th> <th>RES_{k-1, k}</th> </tr> </thead> <tbody> <tr> <td>v1</td> <td>10</td> <td>5</td> <td>0.50</td> </tr> <tr> <td>v2</td> <td>25</td> <td>9</td> <td>0.36</td> </tr> </tbody> </table>			Software Version	ER-Total number of existing requirements	RC-Number of requirements changed	RES _{k-1, k}	v1	10	5	0.50	v2	25	9	0.36
Software Version	ER-Total number of existing requirements	RC-Number of requirements changed	RES _{k-1, k}												
v1	10	5	0.50												
v2	25	9	0.36												
Comments/Notes	<ul style="list-style-type: none"> A low value for RES ratio denotes a little change in the software development because the existing requirements have not been changed. Software requirements can be functional (FUR) as well as not functional (NFR). Better to consider separately both types as well as the root-cause analysis for mid-high RES values. ‘Scope creep’ is the term adopted in the Functional Size Measurement (FSM) and Requirement Engineering communities for a project scope continuously modifying during the project lifecycle 														
Possible associated questions	<ul style="list-style-type: none"> How many times has the software product changed due to changing in the requirements? What software versions have been subjected to a high number of changes? Is the customer continually changing his/her requirements? Why? 														

3.14 SRD – Software Robustness Distribution

Measure Name	SDR – Software Robustness Distribution	ISO/IEC 15504	ENG.5, ENG.6
Purpose	To improve the management of costs and risks associated to the construction or acquisition from third parties of reliable and reusable software.		
Entity	Product	Attribute	Reliability
SLC phase where applied	Coding		
Unit of Measure(s)	Statistical distribution		
Measurement Scale	Ratio		
Counting rule	Compute for each function/method (f), the ratio between the number of invariant/variant checks and the complexity of the function/method.		
Formula	$SRD_f = \frac{IC_f}{CMPLX_f}$	Legend: SRD _f = Robustness Index of f IC _f = Invariant Checks in f CMPLX _f = Complexity of f	
Responsible for Gathering Data	<ul style="list-style-type: none"> • Programmer (during development, to enhance the above mentioned product attributes) • Assessor (to evaluate reliability and reusability) 		
Gathering frequency	<ul style="list-style-type: none"> • Monthly during main development • When needed afterwards. 		
Gathering methodology	Automatic Note: it is possible to use a static analyzer that is flexible enough (such as ECLAIR, http://bugseng.com) to be configured for the detection of all kinds of invariant checks, or semiautomatic, using manual recognition of the invariant checks and automatic computation of the complexity measure.		
Examples	<ul style="list-style-type: none"> • A software component needs to be developed or acquired that is meant to be highly reliable and/or it has to be combined with other software components that are not fully specified, or not fully developed, or that may change in the future due to updates to the overall product. SRD allows to measure the “robustness” of such a piece of software, intended as its ability to continuing function and/or appropriately signals the anomaly in presence of unforeseen usage conditions. 		
Comments/Notes	<ul style="list-style-type: none"> • Invariant/variant checks are programming devices that are meant to handle all situations that are at least conceivably possible. In a setting where maximum reusability and reliability are of concern, the number of assumptions made while coding an individual function are kept to an absolute minimum: the caller of a function is not trusted to respect the contract, all externally provided data is tainted until proven secure, all system calls that may conceivably go wrong will sooner or later go wrong. • Invariant/variant checks may be <i>active</i> or <i>inactive</i> in production runs. Active checks may trigger the execution of “plan B” code and system operation continues (possibly in some “degraded mode”) or stop the system in a way that is as graceful as possible. In both cases, these checks will usually ensure appropriate data about the failure is stored and/or communicated so as not to go unnoticed. Checks that are inactive in production runs can be activated in various kind of testing runs for the purpose of debugging and assessing robustness. • Ordinary assertions as provided in C by <assert.h> are a well-known (though very rudimentary) device to specify inactive invariants. • Typical invariant checks will test the range of numerical values, the return value of system calls, the presence of string terminators and so forth. Typical variant checks will test that loops do make progress and that their termination is reached before a certain number of iterations. • The SRD distribution should of course be studied using well-known statistical measures (average, variance, ...) across the space of all program functions/methods. • The complexity measure adopted for functions/methods is a parameter. It will typically combine cyclomatic complexity with the number and type of implicit and explicit parameters, here included the accessed non-deterministic data sources (such as sensor 		

	<p>input).</p> <ul style="list-style-type: none"> • Each function/method is analyzed in isolation. The number of invariant checks (such as assertions and tests that result into the possible registration of anomalies) is counted and divided by the complexity of the function/method.
<p>Possible associated questions</p>	<ul style="list-style-type: none"> • How is this software dependent on the particular usage conditions it has been designed for? • How costly will be the reuse of this software due to the presence of invariants we will have to guess (as opposed to finding them embodied in explicit tests)? • How likely is that a run-time anomaly will go undetected?

3.15 SFIN – Structural Fan-In

Measure Name	SFIN – Structural Fan-In	ISO/IEC 15504	ENG.5, ENG.6
Purpose	It measures the complexity of the static (design-time) structure of code.		
Entity	Product	Attribute	Maintainability
SLC phase where applied	Coding		
Unit of Measure(s)	Procedure(s)		
Measurement Scale	Absolute		
Counting rule	FIN (procedure) = number of procedures that call this procedure		
Formula	$SFIN_p = \sum P_{inp}$	Legend: p = procedure under measure P_inp = procedure that call the one under measure	
Responsible for Gathering Data	Developer		
Gathering frequency	At the end of each build/release		
Gathering methodology	Automatic <ul style="list-style-type: none"> E.g.: SciTools (http://www.scitools.com) 		
Examples	See a counting example here : http://goo.gl/ChTZZ		
Comments / Notes	<ul style="list-style-type: none"> A high SFIN value indicates a heavily used procedure, while a low SFIN is the opposite. A high value for fan-in means that X is tightly coupled to other modules, as a consequence a change of X will have extensive propagation effects. Please refer also to IFC as an aggregated measure using SFIN and SFOUT as inputs 		
Possible associated questions	<ul style="list-style-type: none"> What are the most used functions, which performance improvement can impact on the overall system performance? What's the reusability level for such application? Can a called procedure often be optimized (e.g. inlining)? When SFIN=0 can the procedure be removed, since never invoked? 		

3.16 SFOUT – Structural Fan-Out

Measure Name	SFOUT – Structural Fan-Out	ISO/IEC 15504	ENG.5, ENG.6
Purpose	It measures the complexity of the static (design-time) structure of code.		
Entity	Product	Attribute	Maintainability
SLC phase where applied	Coding		
Unit of Measure(s)	Procedure		
Measurement Scale	Absolute		
Counting rule	SFOUT (procedure) = number of procedures that are called by this procedure		
Formula	$SFOUT_p = \sum P_outp$	Legend: p = procedure under measure P_outp = procedure that are called by the one under measure	
Responsible for Gathering Data	Developer		
Gathering frequency	At the end of each build/release		
Gathering methodology	Automatic E.g.: SciTools (http://www.scitools.com)		
Examples	See a counting example here : http://goo.gl/ChTZZ		
Comments / Notes	<ul style="list-style-type: none"> • A high SFOUT value indicates a procedure calling many others, while SFOUT=0 means it is a leaf procedure. • It can be applied at different levels of granularity (individual functions, modules, methods, classes of a program). • Please refer also to IFC as an aggregated measure using SFIN and SFOUT as inputs 		
Possible associated questions	<ul style="list-style-type: none"> • Is the code strongly coupled or self-sufficient (=more maintainable)? • Can the procedure/function be used “as-is” in another project, providing the same expected results? 		

3.17 WPU – Work Product Usage

Measure Name	WPU – Work Product Usage	ISO/IEC 15504	PA 2.2, 3.2, 4.2, 5.2
Purpose	To determine the rate of usage of Work Products (WP) across the project lifetime.		
Entity	Project	Attribute	Effectiveness
SLC phase where applied	All SLC Phases		
Unit of Measure(s)	Work Product (WP)		
Measurement Scale	Ratio (see WPU), absolute (see WPU')		
Counting rule	Two possible formulas: (a) Compute the ratio between the number of WPs per WP-type released in a certain period of time (WPU); (b) Compute the number of WPs per WP-type released by SLC phases (WPU')		
Formula	(a) $WPU = \frac{NOU}{T}$	<u>Legend</u> WPU = Frequency of same WP Updates NOU = No. of same WP updates in period T T = period of time	
	(b) $WPU' = NOU$	WPU' = count of the same WP updates in the project phase / in the whole project	
Responsible for Gathering Data	Project Manager		
Gathering frequency	Regularly during a project – Across projects (process measure)		
Gathering methodology	Manual <u>Note: it can be partly automated</u> being supported by a document management tools (e.g.: for CM or Intranet document managers)		
Examples	• ---		
Comments / Notes	<ul style="list-style-type: none"> Such measures can be estimated, jointly with other measures, the usefulness of a Work Product referring to other Work Products or to the same Work Product in other projects Such measures can be used in conjunction with the number of accesses per period (also in reading mode) performed by <i>all need-to-know people</i> Much higher values are expected for record/report types documents than for guides/procedures/templates. Further stats could be derived grouping WP by type, according to the organizational QMS (Quality Management System) definitions It can be used in process improvement and process optimization policy RATIONALE: From experience, various situation typically occur, and among them: <ul style="list-style-type: none"> Case 1: There were instances of WP's that were mostly produced for complying to contractual constraints or to applicable standards, and not for their recognized impact into project goals (the final product is achieved in time, within budget and with the expected qualities). Such WP's were never or scarcely used by their nominal users during the SLC, and sometimes they were produced at the end of the SLC with no effects on the final product. Case 2: There were also WP's that were thought of as useful at the beginning of the SLC but were never used after. HOW TO MEASURE: Direct automatic usage detection would be the best way, but is difficult, not generally applicable (excepting for WP's bound to a database) and unpractical. Then, on the assumption that a commonly used WP is frequently updated, the metric, quite an indirect one, is that of the frequency of updates of WP instances. NOTE: The measure is expected to be rather sensitive depending on the type of WP and on the SLC phase. Examples of extreme cases are SQA Plan (a few updates, if any) and code (continuous updates). Also the meaning of the measure changes with the type 		

	of WP and with the SLC phase.
Possible associated questions	<ul style="list-style-type: none"> • Is the WP worth to be kept and maintained? • Can the WP be updated just to be used or to get easier to use? • Is the relevance of such Work Product understood or made known by the responsible to the need-to-knows? • How much does it cost (and should be properly foreseen at the estimation stage) to maintain project WPs across the project lifetime? Which skills should be involved?

--- End of the Document ---