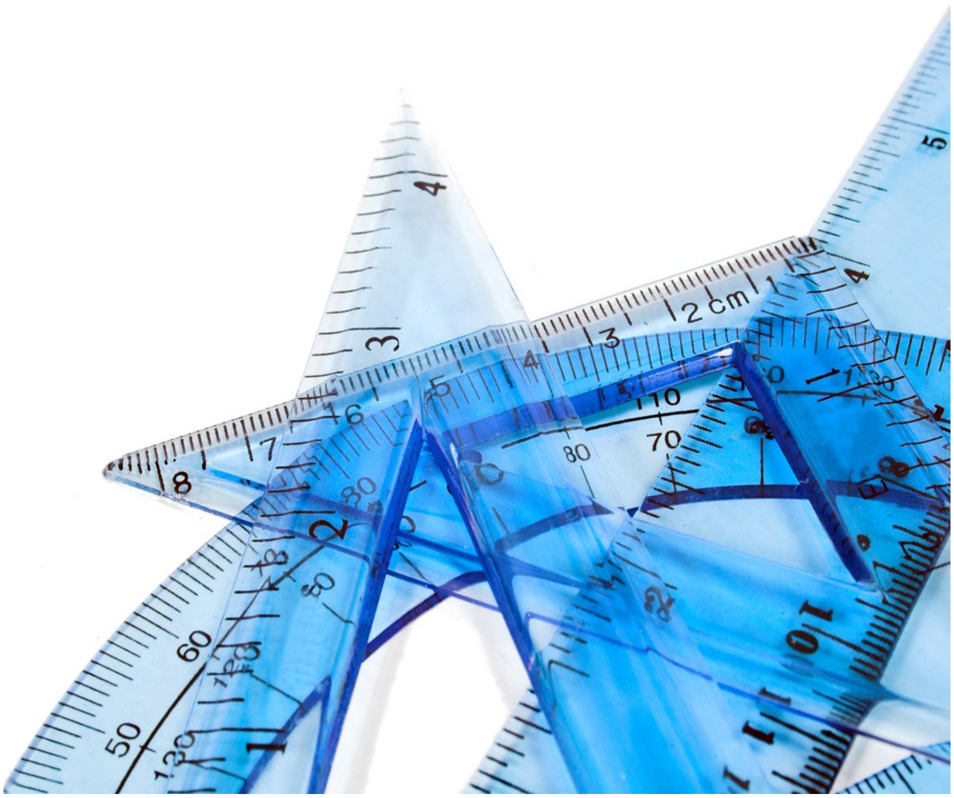
Luigi Buglione

Top10 Metrics - Metric Cards



White Paper

Version 1.0 – April 1, 2011

How to reference this document:
Luigi Buglione, *Top10 Metrics: Metric Cards, version 1.0,* WP-2011-01, White Paper, April 1 2011

For more information about other Process Improvement, Software Measurement & Quality issues, please visit:
< http://www.semq.eu > or contact the Author by email at luigi.buglione@computer.org

First Printing: April 2011

Table of Contents

1 Document Information....4
  1.1 Executive Summary....4
  1.2 History....4
  1.3 Acronyms....4
  1.4 References....5
2 Introduction....6
  2.1 Background and Rationale....6
  2.2 How Much to Measure?....6
  2.3 Metric Cards....7
3 Metrics cards....9
  3.1 SPS – Software Physical Size....9
  3.2 SFS – Software Functional Size....10
  3.3 CYC – McCabe Cyclomatic Complexity....12
  3.4 SDE – Software Development Effort....13
  3.5 SDD – Software Development Duration....14
  3.6 SDR – Software Defect Rate....15
  3.7 CPI – Cost Performance Index....17
  3.8 SPI – Schedule Performance Index....18
  3.9 EAV – Earned Value....19
  3.10 TEC – Test Coverage....20

1 Document Information

1.1 Executive Summary

The purpose of this document is to propose a list of balanced and selected set of measures [3] useful for being used in a measurement plan according 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 the Appendix A of ISO/IEC 15939 standard [2]. This document could represent a starting point for the MASP (Metrics in Automotive Software Projects) working group within the Automotive SPIN Italy (www.automotive-spin.it).

1.2 History

Revision	Date	Changes since last revision
1.00	April 1, 2011	• First issue

1.3 Acronyms

Acronym	Description
A-SPIN	Automotive SPIN Italia (www.automotive-spin.it)
BCWP	Budgeted Cost of Work Performed
BFC	Base Functional Component
BSC	Balanced Scorecard
CPI	Cost Performance Index
CYC	McCabe Cyclomatic Complexity
EAV	Earned Value
ENG	Engineering process group (ISO/IEC 15504)
EV	Earned Value
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
SDD	Software Development Duration
SDE	Software Development Effort
SDR	Software Defect Rate
SFS	Software Functional Size
SLC	Software Life Cycle
SPI	Schedule Performance Index
SPICE	Software Process Improvement Capability dEtermination (ISO/IEC 15504)
SPS	Software Physical Size
SUP	Support process group (ISO/IEC 15504)
TEC	Test Coverage

1.4 References

Ref	Title
[1]	ISO/IEC, *IS 15504-2:2003 – Information Technology – Process Assessment – Part 2: Performing an assessment*, International Organization for Standardization, October 2003, URL: www.iso.org
[2]	ISO/IEC, IS 15939:2007 – *Systems and Software Engineering – Measurement process*, International Organization for Standardization, February 2007, URL: www.iso.org
[3]	Buglione L., *Top metrics for SPICE-compliant projects*, Automotive-SPIN Italia, 5° Automotive SPIN workshop, Milan (Italy), June 4 2009, URL: www.automotive-spin.it
[4]	Buglione L. & Abran A., *Multidimensional Project Management Tracking & Control - Related Measurement Issues*, 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., *Automotive SPICE® Process Reference Model (PRM)*, v4.5, May 10 2010, URL: www.automotivespice.com

2 Introduction

2.1 Background and Rationale

One of the most known motto about measurement is that '*you cannot control what you cannot measure*' but moving a step before '*you cannot measure what you cannot define*'. This is fundamental because – even if a certain concept can be shared – not necessarily its definition can be applied exactly in the same way among different people. Often measures are simply cited and/or referred through a short title, without providing details that can clearly define what anybody should count in a consistent way. For instance, looking at one of the earliest measures adopted in Software Engineering – Lines of Code (LOC) – asking to few people it is not trivial to obtain the same answer about what must be counted or excluded. The same when dealing with defects (e.g. pre or post delivery? What is the boundary between debugging and testing, in order to record the right number of defects?) or the effort to be recorded, dealing also with a proper level of granularity (e.g. man-hours better than man-days). Thus, the solution can beside simply in a more granular and detailed definition for each measure of interest. The way suggested in several technical reports and studies is a 'metric card', showing few details helping people to apply the same definition for the same concept, reducing the probability to have historical data not comparable or needing a series of assumptions for deriving the 'numbers'.

2.2 How Much to Measure?

Another typical problem in measurement is about the 'how much' to measure. Of course the budget for the measurement process in a project/activity must be limited within a certain established percentage and some criteria for selecting and prioritizing those measures must be set. Referring to the ISO 15504 and Automotive SPICE PRM, in [3] a set of measures balanced against the measurable entity (project, resource, process, product) was proposed, as shown in next table. Those measures were classified according to the EAM (Entity-Attribute-Measure) taxonomy and associated to one (or more) processes from the Automotive SPICE PRM [5].

Entity (E)	Attribute (A)	Measure (M)	Threshold	A-SPICE
Project	Planning compliance	Effort (man/hrs) per SLC phase, per iteration (abs, %)	(profiles on historical data)	MAN.3
Resource	Time	% of open complaints / notes for delaying in providing the agreed furniture (tracked) per contract	≤10%	ACQ.4
Process*	Time performance	SPI (Schedule Performance Index)	ongoing	MAN.3
Process*	Cost performance	CPI (Cost Performance Index)	ongoing	MAN.3
Process	QA performance	% of non-conformances still open	≤15%	SUP.1
Process*	Maturity	Problem Reports (PR) by status (open, closed)	(profiles on historical data)	SUP.9
Process	Changeability	Avg Change Requests (CR) working time by status	(profiles on historical data)	SUP.8 - SUP.10
Process*	Planning reliability	Requirements Volatility of 'Scope Creep' Index (# of modified/new UR not formally traced / tot. # UR) by iteration	≤10%	ENG.4
Product*	Code Length	Kilo Lines of Code (KLOC) [system, function, module] *c.a 5 functions per module*	(abs, 100-150, 700-1000)	ENG.4
Product*	Functional Size	Functional Size (fsu) [system]	(abs)	ENG.4
Product*	Maintainability	Cyclomatic Complexity (of a function)	≤20	ENG.5, ENG.6

Entity (E)	Attribute (A)	Measure (M)	Threshold	A-SPICE
Product*	Maintainability	# of transfer parameters in a function	≤5	ENG.6
Product*	Maintainability	Average size of a function statement (operands + operators / # of executable statements)	≤10	ENG.6
Product*	Code Stability	# of exit points from a function	1	ENG.5, ENG.6
Product*	Code Stability	# of calling functions of a function (fan-out)	≤10	ENG.5, ENG.6
Product	Code Stability	# of execution paths in a function	≤1000	ENG.5, ENG.6
Product	Testability	Branch Coverage	100%	ENG.8
Product*	Testability	Max # nesting depth of the function control structure	≤4	ENG.8

Since these are only titles, a series of 'metric cards' will be proposed in Section 2. The information provided tries to answer to the "5Ws+H" rule (Who, What, Why, Where, When and How), with few, dedicated fields in the table structure.

The list of possible measures described here represents a suggestion and can be updated during time, adding or updating the existing cards. The idea behind the 'top 10 metrics' inserted in the document title would simply suggest to maintain the focus on few, core measures (possibly) representing more viewpoints and measurable entities in a project measurement plan. The further core concept suggested to follow is to maximize the informative value from the selected measures, selecting the measures taking care also to their cross-relationships along the different SLC phases, as done in a Balanced Scorecard (BSC). When dealing with a plenty of potential measures and need to reduce their amount to a core, vital, few ones, the BMP (Balancing Multiple Perspectives) technique can be applied [4].

In order to achieve this goal and make this document updated as much as possible, please send any comment/suggestion to the following email address:

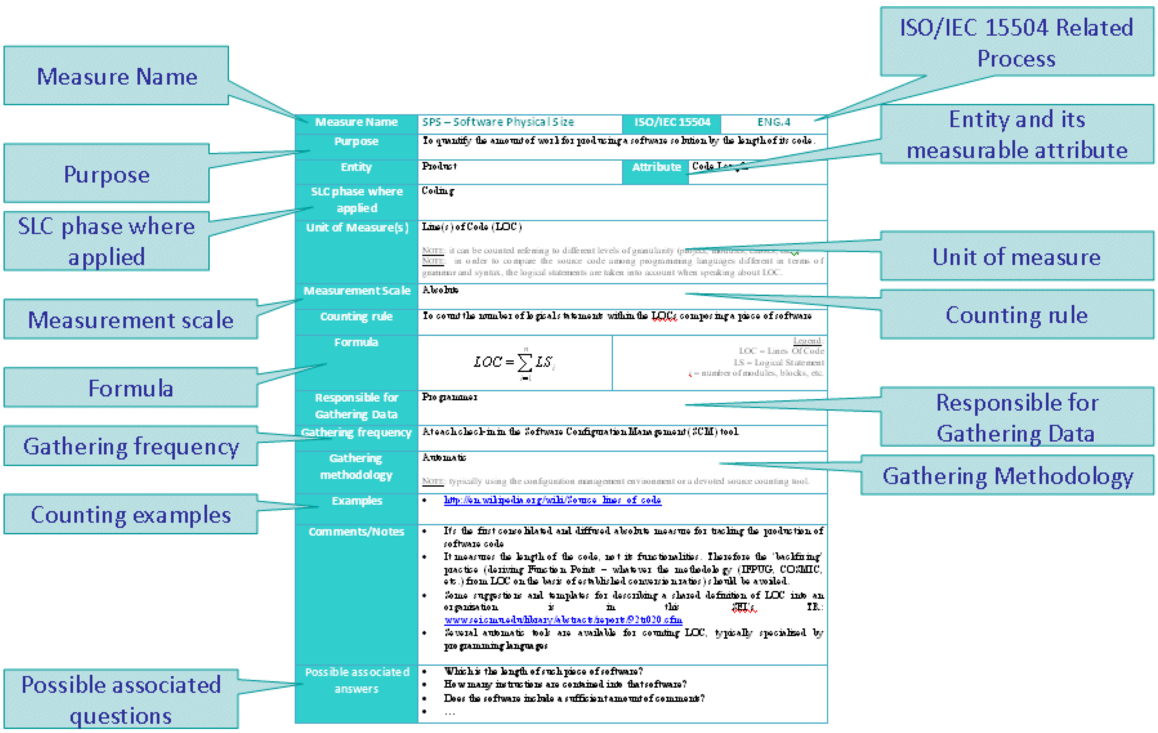
luigi.buglione@computer.org

2.3 Metric Cards

In Section 3.x, a series of 'metric cards' are proposed, with the following structure and fields:

- **Measure title/code**: title and (eventual) code for the measure
- **ISO/IEC 15504**: associated ISO/IEC 15504-2 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 needed data for computing that measure

- **Gathering Frequency**: the suggested frequency for gathering that measure
- **Gathering Methodology**: the suggested methodology/technique for gathering that measure
- **Counting examples**: one or more short calculation examples for showing the way the data must be applied for computing that measure
- **Comments/Notes**: eventual additional comments and/or notes for specifying or providing more information about that measure
- **Possible Associated Questions**: a list of possible associated questions in a sort of reverse-GQM analysis.



3 Metrics cards

3.1 SPS – Software Physical Size

Measure Name	**SPS – Software Physical Size**	**ISO/IEC 15504**	**ENG.4**
Purpose	To quantify the amount of work for producing a software solution by the length of its code.		
Entity	Product	**Attribute**	Code Length
SLC phase where applied	Coding		
Unit of Measure(s)	Line(s) of Code (LOC) NOTE: it can be counted referring to different levels of granularity (project, modules, classes, etc..). NOTE: in order to compare the source code among programming languages different in terms of grammar and syntax, the logical statements are taken into account when speaking about LOC.		
Measurement Scale	Absolute		
Counting rule	To count the number of logical statements within the LOCs composing a piece of software		
Formula	$LOC = \sum_{i=1}^{n} LS_i$	Legend: LOC = Lines Of Code LS = Logical Statement i = number of modules, blocks, etc.	
Responsible for Gathering Data	Programmer		
Gathering frequency	At each check-in in the Software Configuration Management (SCM) tool.		
Gathering methodology	Automatic NOTE: typically using the configuration management environment or a devoted source counting tool.		
Examples	• http://en.wikipedia.org/wiki/Source_lines_of_code		
Comments/Notes	• It's the first consolidated and diffused absolute measure for tracking the production of software code • It measures the length of the code, not its functionalities. Therefore the 'backfiring' practice (deriving Function Points – whatever the methodology (IFPUG, COSMIC, etc.) from LOC on the basis of established conversion ratios) should be avoided. • Some suggestions and templates for describing a shared definition of LOC into an organization is in this SEI's TR: www.sei.cmu.edu/library/abstracts/reports/92tr020.cfm • Several automatic tools are available for counting LOC, typically specialized by programming languages		
Possible associated questions	• Which is the length of such piece of software? • How many instructions are contained into that software? • Does the software include a sufficient amount of comments? • …		

3.2 SFS – Software Functional Size

Measure Name	**SFS – Software Functional Size**	**ISO/IEC 15504**	**ENG.4**
Purpose	To calculate the size of the functionalities to be added, changed, inserted in a software solution.		
Entity	Product	**Attribute**	Functional Size
SLC phase where applied	Bid (early-Stage) phase, Design phase, Project Closure.		
Unit of Measure(s)	*Fsu(Functional Size Unit)* <u>Note</u>: each *fsu* is composed by its own BFCs.		
Measurement Scale	Ratio		
Counting rule	To calculate the weighted sum by BFCs (Base Functional Components) considered in the chosen Functional Size Measurement (FSM) method.		
Formula	$fsu = \sum_{i=1}^{n} \sum_{j=1}^{m} BFC_i * w_j$	<u>Legend</u>: fsu = functional size unit BFC = Base Functional Component w = weight n = max number of BFC for that FSM method m = max number of complexity levels	
Responsible for Gathering Data	Functional Analyst		
Gathering frequency	Typically to be counted in three moments in time in the project lifetime: ▪ After the elicitation of high-level requirements (HLR) ▪ At the end of the Design phase ▪ At the Project closure		
Gathering methodology	Manual <u>Note</u>: Fsu cannot be automatically calculated from FURs expressed in natural language. There are tools able to make the count but moving from a pre-analyzed software object (e.g. expressed in UML diagrams/formats), that means to have yet performed the Analysis & Design phase(s).		
Examples	• <u>URL:</u> http://www.softwaremetrics.com/freemanual.htm • <u>URL</u>: http://www.semq.eu/leng/sizestfsm.htm		
Comments/Notes	• *Fsu* is the generic term for including all the possible units of measure related to the several FSM methods • BFC depends on the FSM method (e.g. for the IFPUG FPA, BFC are 5: ILF, EIF, EI, EO, EQ; for COSMIC are 4: Entry, Exit, Read, Write; etc.) • COSMIC is the solely FSM method without a weighting system: in such case, please consider the 'w' variable always equal to 1. • Any FSM method sizes only the FUR (Functional User Requirements) for a software product. Therefore NFR (Non-Functional Requirements) are out of scope from this measure. For instance, IFPUG is working on a new method called SNAP (Software Non-functional Assessment Process), to be released by 2011. Or the ISO/IEC 9126-1 Quality Model attributes can be considered, looking at their related metrics in parts 2-3-4. • For estimation purposes, it is very useful to maintain the data gathering in the project historical database (PHD) at the BFC level: a prediction model taking care of 2+ BFC in a multiple regression model is more efficient than using the whole *fsu* value.		
Possible associated questions	• How many functions are going to be implemented in the software solution? • Which is the value of functional requirements for such software? • …		

3.3 CYC – McCabe Cyclomatic Complexity

Measure Name	**CYC – McCabe Cyclomatic Complexity**	**ISO/IEC 15504**	**ENG.5** **ENG.6**
Purpose	To take under control the level of maintainability of a software program.		
Entity	Product	**Attribute**	Maintainability
SLC phase where applied	Coding		
Unit of Measure(s)	It can be applied to several levels of granularity (individual functions, modules, methods, classes of a program).		
Measurement Scale	Interval		
Counting rule	The v(G) is given by the summation of the number of edge minus the number of nodes plus the number of connected components in a function (or module, method, class – as stated in the 'Unit of Measure' field).		
Formula	$v(G) = e - n + p$		Legend: v(G) = Cyclomatic Complesity e = edge(s) n = node(s) p = connected component(s)
Responsible for Gathering Data	Programmer		
Gathering frequency	---		
Gathering methodology	Automatic		
Examples	• http://www.literateprogramming.com/mccabe.pdf (see in the paper)		
Comments/Notes	• Source: T.McCabe, *A complexity measure*, IEEE Transactions on Software Engineering, Vol. SE-2, No.4, December 1976, pp. 308-320, URL: http://www.literateprogramming.com/mccabe.pdf • Further variants and evolution of the initial concepts are reported in Wikipedia (http://en.wikipedia.org/wiki/Cyclomatic_complexity)		
Possible associated questions	• Which is the level of maintainability for such software? • Has the software need to be refactored? • ...		

3.4 SDE – Software Development Effort

Measure Name	**SDE – Software Development Effort** | **ISO/IEC 15504** | **MAN.3**
Purpose	To measure the time spent to complete a software development project or a single process/activity.
Entity	Project | **Attribute**: Effort
SLC phase where applied	All the SLC phases
Unit of Measure(s)	Man/days (or man/hours) <u>NOTE</u>: since different definitions and amount of hours for the working week are adopted worldwide, it is strongly suggested to apply the *man/hour* definition, in order to be consistent for benchmarking purposes.
Measurement Scale	Absolute
Counting rule	To sum the work effort by all the SLC phases defined and applied within an organization. <u>NOTE</u>: for instance, ISBSG in its repository defines the following phases: Plan, Specify, Design, Build, Test, Implementation, Unphased. <u>NOTE</u>: a consequence when applying for man/days as the counting unit, it is to pay attention in taking note of the extra-time spent per day from project resources. If not done, the risk is to historicize less working time. This could lead to underestimations for next projects, moving from low effort values recorded in historical databases.
Formula	$SDE = \sum_{i=1}^{n} LCPE_i$ <u>Legend</u>: SDE = Software Development Effort LCPE = Life Cycle Phase Effort i = number of LCP defined in the organization
Responsible for Gathering Data	Project Manager
Gathering frequency	At the end of each SLC phase
Gathering methodology	Semi-automatic <u>NOTE</u>: e.g. using internal time planning & tracking systems or e.g. MS-Project, Primavera
Examples	• http://csse.usc.edu/csse/TECHRPTS/2008/usc-csse-2008-836/usc-csse-2008-836.pdf • http://s3.amazonaws.com/publicationslist.org/data/a.abran/ref-2040/909.pdf
Comments/Notes	• It is preferable to use the more granular unit of measure as possible (e.g. man-hours) for allowing comparisons among organizations having different standards (e.g. in the UU.SS typically a working week is 128-hrs long, while in Europe is 160 hrs-long). • ISBSG- International Software Benchmarking Standards Group (www.isbsg.org) • A practical usage is to take into account the percentages among the different phases after classifying and clustering groups of projects with different characteristics (e.g. programming language, application type, development type, etc...) • ...
Possible associated questions	• How much time do we spend for Project Management? And for Analysis? • Is it proper the effort distribution among the SLC phases, compared with the defect density detected after the delivery of the software? • ...

3.5 SDD – Software Development Duration

Measure Name	SDD – Software Development Duration	ISO/IEC 15504	MAN.3
Purpose	To measure the elapsed time from project start date through to project finish date.		
Entity	Project	Attribute	Duration
SLC phase where applied	All the SLC phases		
Unit of Measure(s)	Man/days (or man/hours)		
Measurement Scale	Absolute		
Counting rule	To sum the calendar time by all the SLC phases defined and applied within an organization. NOTE: for instance, ISBSG in its repository defines the following phases: Plan, Specify, Design, Build, Test, Implementation, Unphased.		
Formula	$SDD = \sum_{i=1}^{n} LCPD_i$	Legend: SDE = Software Development Duration LCPD = Life Cycle Phase Duration i = number of LCP defined in the organization	
Responsible for Gathering Data	Project Manager		
Gathering frequency	At least at the project start and closure.		
Gathering methodology	Semi-automatic NOTE: e.g. using internal time planning & tracking systems or e.g. MS-Project, Primavera		
Examples	• http://us.generation-nt.com/answer/simple-project-duration-question-help-197334151.html • www.isbsg.org/ISBSGnew.nsf/WebPages/2471C311A3AF7549CA2574580022835D • www.tacticalprojectmanagement.com/attachments/049_IJPM%20Vandevoorde%20and%20Vanhoucke.pdf		
Comments/Notes	• It is preferable to use the more granular unit of measure as possible (e.g. man-hours) for allowing comparisons among organizations having different standards (e.g. in the UU.SS typically a working week is 128-hrs long, while in Europe is 160 hrs-long). • Note that well-known guides as the PMBOK (www.pmi.org) – the Project Management Body of Knowledge – refers to the *duration* more than *effort*. • Attention must be paid when an organization has extra-hours to be gathered in its effort historical database for calculating the % usage of the project team within the established schedule. • …		
Possible associated questions	• How many calendar-days are needed to complete the project? • Which is the ratio between project effort and its duration? Is it too high or low? • …		

3.6 SDR – Software Defect Rate

Measure Name	SDR – Software Defect Rate	ISO/IEC 15504	MAN.3 MAN.4
Purpose	To measure the quality of software product/item in terms of number of defects against its product size unit.		
Entity	Product	Attribute	Defectability
SLC phase where applied	Release phase		
Unit of Measure(s)	Defect NOTE 1: there are several ways and criteria for classifying defects. E.g. by severity/priority, or by typology, by origin, etc. NOTE 2: *"a problem which, if not corrected, could cause an application to either fail or to produce incorrect results"* *(ISO/IEC 20926:2003 Software engineering -- IFPUG 4.1 Unadjusted functional size measurement method -- Counting practices manual)*		
Measurement Scale	Ratio		
Counting rule	To calculate the ratio between the number of defects (delivered or discovered) and its product size (according to the product size unit used in the project monitoring). NOTE: for benchmarking purposes, it is suggested to split the values (both in the upper and lower part of the formula) according to the nature of the requirements originating them (functional; non-functional). If not done, the risk is to obtain higher values than expected.		
Formula	$SDR = \frac{DEF}{Size}$	Legend: SDR = Software Defect Rate DEF = no. of delivered defects Size = Unit of Product Size (e.g. LOC, FP, etc.)	
Responsible for Gathering Data	Test Manager		
Gathering frequency	At each agreed release to the customer		
Gathering methodology	Automatic NOTE: selecting a testing tool, the possibility of classification of defects would be a valuable feature.		
Examples	• www.pearsonhighered.com/assets/hip/us/hip_us_pearsonhighered/samplechapter/0201729156.pdf (from "Metrics and Models in Software Quality Engineering", S.Kan, Addison-Wesley, 2/ed., 2002)		
Comments/Notes	• It can be expressed as delivered defects (i.e. expected number of residual/latent defects after delivery) or actually discovered defects along the development life cycle • When dealing with a product functional size, the reported defects in the upper part of the ratio should be only the *functional* ones from black box testing. And so on, according to the product attribute intended to be measured. • A root-cause analysis (RCA) is suggested trying to detect the origin of a high SDR value. A well-known technique specifically devoted to Software Testing is e.g. ODC (Orthogonal Defect Classification): see www.chillarege.com/odc • A possible taxonomy for classifying is the one proposed by UKSMA in 2000 ('Quality Standards – Defect Measurement Manual): see www.uksma.co.uk		
Possible associated questions	• How much effort is it needed to fix the detected bugs? • Has the project planned a balanced number of test cases and related effort for the Testing phase within the SLC? • Which is the root-cause for a higher value of SDR than expected thresholds? • …		

3.7 CPI – Cost Performance Index

Measure Name	CPI – Cost Performance Index	ISO/IEC 15504	MAN.3
Purpose	To verify if the project is profitable along its lifetime.		
Entity	Project	Attribute	Cost Performance
SLC phase where applied	During the whole SLC		
Unit of Measure(s)	Activity value; activity cost		
Measurement Scale	Ratio		
Counting rule	To calculate the ratio between the Earned Value (EV) and Actual Costs (AC).		
Formula	$CPI = \frac{EV}{AC} = \frac{BCWP}{ACWP}$	Legend: CPI = Cost Performance Index EV =Earned Value BCWP = Budgeted Cost of Work Performed AC = Actual Cost AC = Actual Cost of Work Performed	
Responsible for Gathering Data	Project Manager		
Gathering frequency	When needed		
Gathering methodology	Semi-automatic NOTE: e.g. using internal time planning & tracking systems or e.g. MS-Project, Primavera		
Examples	• http://support.microsoft.com/kb/209115 (how to calculate CPI/SPI in MS-Project)		
Comments/Notes	• CPI ≥1 shows a favourable condition, while CPI<1 an unfavourable condition. • It can be useful to have a further split of main figures by profile (at least covering functional vs. non-functional processes) because their different average/median daily cost (e.g. a project manager or a technical architect will cost more than a programmer, but probably having different % of allocation during the project lifetime. • www.suu.edu/faculty/christensend/evms/CPIstabilityNCMJ.pdf		
Possible associated questions	• Is it the project respecting its planned budget? • Are we tracking at the proper level of granularity our internal cost figures for any profile? • …		

3.8 SPI – Schedule Performance Index

Measure Name	**SPI – Schedule Performance Index**	**ISO/IEC 15504**	**MAN.3**
Purpose	To measure the schedule efficiency of the project.		
Entity	Project	**Attribute**	Time Performance
SLC phase where applied	During the whole SLC		
Unit of Measure(s)	Activity value (actual vs. planned)		
Measurement Scale	Ratio		
Counting rule	To calculate the ratio between its Earned Value (EV) and Planned Value (PV).		
Formula	$SPI = \frac{EV}{PV} = \frac{BCWP}{PV} = \frac{BCWP}{BCWS}$	Legend: SPI = Schedule Performance Index EV =Earned Value BCWP = Budgeted Cost of Work Performed BCWS = Budgeted Cost of Work Scheduled PV = Planned Value	
Responsible for Gathering Data	Project Manager		
Gathering frequency	When needed		
Gathering methodology	Semi-automatic NOTE: e.g. using internal time planning & tracking systems or e.g. MS-Project, Primavera		
Examples	• http://support.microsoft.com/kb/209115 (how to calculate CPI/SPI in MS-Project) • www.pmboulevard.com/getFile.pmbx?fid=2156&cid=2798		
Comments/Notes	• SPI ≥1 shows a favourable condition, while SPI<1 an unfavourable condition. • Tracking SPI will allow to understand if the plan is going to follow the expectations • It can be useful to have a further split of main figures by requirement types (at least functional vs. non-functional) because their different % of involvement per any possible kind of project		
Possible associated questions	• Is it the project respecting its planned schedule? • Did we validate our effort data from the project historical database (PHD)? • …		

3.9 EAV – Earned Value

Measure Name	EAV – Earned Value	ISO/IEC 15504	MAN.3
Purpose	To measure project progress in an objective manner.		
Entity	Project	Attribute	Cost Progress
SLC phase where applied	During the whole SLC		
Unit of Measure(s)	Activity value		
Measurement Scale	Interval		
Counting rule	To calculate the value of work performed expressed in terms of the approved budget assigned to that work for a schedule activity or work breakdown structure component.		
Formula	$EAV = BCWP = \sum_{start}^{current} PV(comple$	Legend: EV = Earned Value BCWP = Budgeted Cost of Work Performed PV = Planned Value	
Responsible for Gathering Data	Project Manager		
Gathering frequency	When needed		
Gathering methodology	Semi-automatic NOTE: e.g. using internal time planning & tracking systems or e.g. MS-Project, Primavera		
Examples	• http://en.wikipedia.org/wiki/Earned_value_management		
Comments/Notes	• Also referred to as the budgeted cost of work performed (BCWP) • Tracking EAV will allow to understand if the plan is going to follow the expectations • It can be useful to have a further split of main figures by profile (at least covering functional vs. non-functional processes) because their different average/median daily cost (e.g. a project manager or a technical architect will cost more than a programmer, but probably having different % of allocation during the project lifetime.		
Possible associated questions	• Is the project progressing according to plans? • Are we tracking at the proper level of granularity our internal cost figures for any profile? • …		

3.10 TEC – Test Coverage

Measure Name	TEC – Test Coverage	ISO/IEC 15504	ENG.8
Purpose	To measure the level of testing depth on structural elements of the software (e.g. statement coverage).		
Entity	Process	Attribute	Testability
SLC phase where applied	Testing phase		
Unit of Measure(s)	Test Case; Requirements		
Measurement Scale	Ratio		
Counting rule	To calculate the ratio between the number of test cases planned and executed and the requirements from which they come from. NOTE: such ratio should be calculated maintaining proportionality between the upper and lower part of the formula. It can be done counting test case and requirements referred to the same project/product attribute (e.g. functionality, or complexity)		
Formula	$TEC = \frac{TC}{REQ}$	Legend: TEC = Test Coverage ratio TC = Test Cases REQ = no. of requirements	
Responsible for Gathering Data	Test Manager		
Gathering frequency	At each requirements change.		
Gathering methodology	Automatic NOTE: typically requirement and testing tools provide a traceability matrix for determining such TEC value.		
Examples	• http://qtest.qbilt.org/doc/qtest-manual.pdf • http://www.slidefinder.net/t/theory_predicate_complete_test_coverage/14849375		
Comments/Notes	• Two possible definitions of test coverage from ISO standards are: (1) the degree to which a given test or set of tests addresses all specified requirements for a given system or component *(ISO/IEC 24765:2009 Systems and software engineering vocabulary)* (2) extent to which the test cases test the requirements for the system or software product *(ISO/IEC 12207:2008 Systems and software engineering--Software life cycle processes, 4.51)* • Since requirements can be referred to different entities and related attributes, such traceability matrix should be classified by those proxies in order to check a proper coverage level also looking to a more detailed level (e.g. looking at the *product* entity, a first-level classification could be the one proposed in ISO/IEC 14143-1:2007 standard, classifying product requirements into functional, quality and technical. A second-level classification e.g. for the quality ones, can be the one provided by the quality model in ISO/IEC 9126-1:2001 with 6 main characteristics and 27 sub-characteristics, etc.)		
Possible associated questions	• Are the requirements sufficiently verified? • Which is the percentage of test cases against the project requirements? • Are the test cases properly balanced against the different requirement types (functional, quality, technical), according to ISO/IEC 14143-1:2007 taxonomy? • …		

--- End of the Document ---