Software Engineering Institute

A Data Specification for Software Project Performance Measures: Results of a Collaboration on Performance Measurement

Mark Kasunic

July 2008

TECHNICAL REPORT
CMU/SEI-2008-TR-012
ESC-TR-2008-012

Software Engineering Process Management
Unlimited distribution subject to the copyright.



Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2009 Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. This document may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Table of Contents

Acknowledgements iii

Abstract v

1 Introduction 1

2 Performance Measurement—Challenges 3

3 Performance Measures for Software Projects—Overview 9

3.1 Project Effort 10

3.2 Productivity 12

3.3 Project Duration 13

3.4 Schedule Predictability 15

3.5 Requirements Completion Ratio 17

3.6 Post Release Defect Density 18

4 Influence Factors for Software Projects—Overview 19

4.1 Size 22

4.2 Artifact Reuse 28

4.3 Project Type 38

4.4 Application Domain 39

4.5 Average Team Size 44

4.6 Maximum Team Size 47

4.7 Team Expertise 49

4.8 Process Maturity 51

4.9 Functional Requirements Stability 56

5 Using the Performance Measures and Influence Factors 59

6 Request for Feedback 63

Appendix: Benchmarks and Benchmarking 65

Glossary 71

References 81

Acknowledgements

This publication is the result of many incremental pieces of writing and could not have been accomplished without the involvement and contribution of our collaborators.

The following individuals participated on the team that developed the initial drafts of the definitions that appear in this report. The drafts were discussed through face-to-face meetings and by teleconference before being presented for additional review by other experts.

Kate Armel	(QSM)
Michael Bragen	(Software Productivity Research, LLC.)
Robert Floyd	(Raytheon)
David Garmus	(David Consulting Group)
Tim Hohmann	(Galorath Incorporated)
Mark Kasunic	(SEI)
Arlene Minkiewicz	(PRICE Systems)
Tony Rollo	(ISBSG)

In addition to the individuals above, the following collaborators provided feedback and participated in consensus-based document review workshops.

Carol Dekkers	(4SUM Partners)
Khaled El Emam	(University of Ottawa)
Eric Finch	(PRTM)
Pekka Forselius	(4SUM Partners)
Dennis Goldenson	(SEI)
Thomas Lienhard	(Raytheon)
Kristal Ray	(Oracle)
Bob Weiser	(Lockheed Martin)
David Zubrow	(SEI)

Thanks to Bill Novak of the SEI for his contribution to developing the definition for application domain. Thanks also to Linda Parker Gates of the SEI who provided helpful comments on an early version of the document.

The author would also like to thank Grant Bayne, Wolfhart Goethert (SEI), Zhang Hefei (Samsung), Ben Linders (Eriksson), Jim McCurley (SEI), Nandkumar Mishra (Patni Computer Systems), Yogesh Naik (Patni Computer Systems), James Over (SEI), David Rogers (EDS), and

Pradeep Waychal (Patni Computer Systems) for technical review of the final draft document. Their thoughtful comments improved the quality of the report.

Thanks to Dave Zubrow, manager of the Software Engineering Measurement & Analysis Group (SEMA), for developing the original idea and impetus that led to this work. Thanks to Bill Peterson, director of the SEI's program on Software Engineering Process Management (SEPM), who provided strong and visible support for this endeavor. Thanks to Bob Fantazier for his able graphic design support. Last but not least, thanks to Erin Harper and Barbara White for their excellent editorial support.

Abstract

This document contains a proposed set of defined software project performance measures and influence factors that can be used by software development projects so that valid comparisons can be made between completed projects. These terms and definitions were developed using a collaborative, consensus-based approach involving the Software Engineering Institute's Software Engineering Process Management program and service provider and industry experts in the area of software project performance measurement. This document will be updated over time as feedback is obtained about its use.

1 Introduction

Do you golf, jog, bowl, ride a bicycle, lift weights, or play basketball? If you do, then you likely keep track of your performance. Perhaps it is as simple as, "I knocked off three strokes from my game today," or "I lifted ten more pounds than I could last week," or "Our team had five less turnovers today compared with last week."

> "If winning isn't everything, why do they keep score."
>
> - Vince Lombardi

People keep score like this because most are performance- or achievement-driven. They want to know how well they are doing—whether their performance is improving or declining—and how their performance compares with their own personal best or with the performance of others. Performance feedback can provide the challenge and motivation for attaining higher levels of achievement.

In much the same way, companies, organizations, and software projects want to understand their overall performance, compare it to others, and find ways to become better.

Software organizations, whether they are just starting a measurement program or have a well-developed program, want a way to gauge the performance of their software projects against other organizations in their industry. Organizations just starting a measurement program do not have historical data on which to base their estimates, so they want to know what measures they should use and what reasonable targets for their measures are. Organizations that are more experienced in measurement want to compare their performance with competitors in their industry. Finally, organizations want to learn about the best practices used by industry leaders so they can adapt them for their own use through the improvement technique referred to as benchmarking. In each of these cases, the valid comparison of measurement data is an integral step in realizing these objectives. However, a widespread obstacle to valid measurement comparison is inconsistent terminology and a lack of common definitions for software project measurement terms.

In this document, we propose a set of defined software project performance measures and influence factors that can be used by software development projects so that valid comparisons of performance can be made. These terms and definitions were developed using a collaborative, consensus-based approach involving the SEI's Software Engineering Process Management (SEPM) program and service providers and industry experts in the area of software project performance measurement.

Section 6 of this document requests feedback regarding the use and value of the performance measures and influence factors described in this document. It is our intention to update this specification as we gain insight into the measures and influence factors useful for comparing and contrasting software project performance.

2 Performance Measurement—Challenges

What is performance measurement?

Performance measurement focuses on results. It asks, "What does success really mean?" In its simplest terms, performance measurement is a process of assessing the results of a company, organization, project, or individual to (a) determine how effective the operations are, and (b) make changes to address performance gaps, shortfalls, and other problems.

> "You can't manage what you can't measure."
>
> - Peter Drucker

Generally speaking, companies and organizations measure their performance using different methods and criteria. But the focus of a performance measurement system should be the key activities of the business. For each key activity, there are numerous possibilities for measurement. Measures must be selected carefully so that they address the specific goals and objectives of the activity.

Many progressive and leading organizations employ an enterprise-wide formal performance measurement system such as Goal-Driven Measurement [Park 1996, Basili 1994], Balanced Scorecard [Kaplan 1992], Six Sigma [Pyzdek 2003, Breyfogle 2003], Practical Software and Systems Measurement [McGarry 2001], and variations of Shewhart's Plan-Do-Check-Act Paradigm [Deming 1986]. Each of these approaches emphasizes the need to take the following steps:

1. Set clear and achievable performance goals or objectives.
2. Define quantitative measures and measurement indicators to characterize performance relative to the goals or objectives.
3. Establish measurement targets that reflect the desired condition or expectation for each performance measurement.
4. Collect the measurement data (i.e., results).
5. Evaluate the data and use the results to make adjustments in operations that will improve the probability of reaching the targets efficiently.

When selecting measures for software projects, organizations should always begin with a systematic measurement definition approach. However, we believe that the performance measures identified in this document are core measures that would be identified as part of the set of critical measures of success since they address important attributes of any software development project.

Why measure performance?

There are many reasons why organizations want to measure performance. Some of the reasons include the following:

Goal achievement. The purpose of performance measurement is to provide feedback about whether or not an organization is meeting its business or project goals. This feedback improves the likelihood of achieving these goals efficiently. Performance measurement enables a team, project, or organization to understand whether they are on track or whether adjustments need to be made to be successful.

Planning and Estimation. Historical measurement data can be used as a basis to forecast or estimate future performance. Because it ties activities to results, performance measurement is a long-term planning tool that can justify resource allocation for software projects.

Improvement. Performance data can be compared within and outside an enterprise to identify weak areas that can be addressed to improve overall performance.

Communication. The reporting of well-defined performance measures can enhance staff, stakeholder, and partner understanding and support of strategies and decisions.

Compliance. In some cases, companies measure performance in order to comply with regulations or other standards.

Performance measurement to support improvement

One important purpose for implementing a program of performance measurement is to support improvement. When data is available from multiple projects that possess similar characteristics, a project can compare its performance to others to determine areas of strength and weakness. When used in this way, measurement comparison serves as a motivator of process improvement.

A more powerful use of performance measurement is within the context of benchmarking. Benchmarking is a process that uses performance measurement to identify best-in-class achievement (i.e., the benchmark), but goes beyond mere comparison to determine *how* the best-in-class achievement was attained. Once the *how* is understood, the enablers (e.g., methods, procedures, tools) that led to the stellar performance are adapted by an organization or project that wants to improve and thereby achieve similar stellar performance [APQC 1993]. See the appendix for a more detailed description of benchmarks and benchmarking.

Performance measurement vs. appraisals

An organizational appraisal is a systematic study conducted periodically or on an ad hoc basis to assess how well an organization is working. For many years the SEI has used process appraisal results to provide information about the process maturity of the software engineering community. These appraisals are conducted by experts to examine aspects of organizational performance in the broad context in which it occurs. This type of assessment is very useful in that an in-depth examination of organizational processes allows for an overall assessment of whether the organization is employing good practices as part of its operations. These appraisals help the organization decide how it can change its practices to realize improvement. However, appraisals do not quantitatively address critical dimensions of success such as cost, schedule, quality, and customer satisfaction. Performance measurement, because of its ongoing nature, can serve to tell how an organization is performing over time. It serves as an early warning system and provides accountability for performance that is directly tied to the project or organization's critical success factors. Both types of assessment aim to support improvement. Quantitative data on project performance is needed to demonstrate actual improvement in areas critical to a company's success.

> "When you can measure what you are speaking about and express it in numbers, you know something about it."
>
> - Kelvin

The problem

When performance measurement is used for comparison purposes (either for simple comparison or for benchmarking), the measures to be compared *must* be commonly defined. However, in the software development world, measurement definition has by no means been standardized. Herein lays the major obstacle that has hampered effective software project performance comparison.

Consider the case of four different software projects that have similar characteristics. Each project measures productivity and uses the common term "productivity" to refer to the measure. However, the actual definitions that have been assigned to the term are different. Figure 1 (adapted from Kasunic [Kasunic 2006]) shows this problem conceptually. Although each project measures "productivity," the actual measures cannot be compared readily. Though a common term is used to refer to each measure, making comparisons is analogous to comparing apples and oranges.

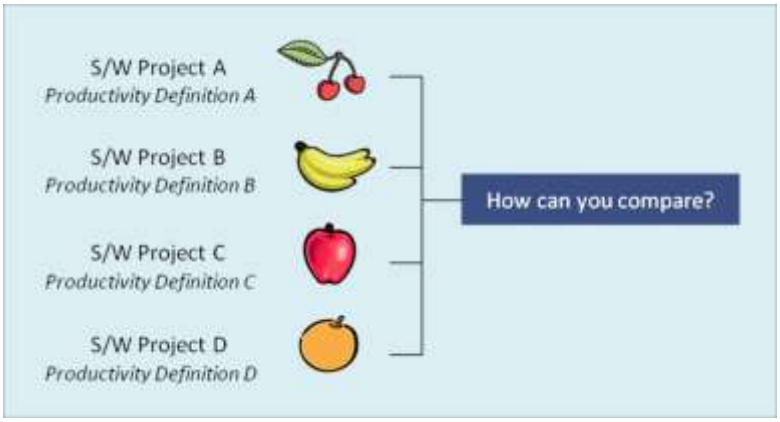


Figure 1. Common measurement definitions are required to make comparisons.

Addressing the problem

The SEI's Software Engineering Process Management (SEPM) program launched a collaborative research effort to investigate ways to improve the practice of software project performance measurement. Several types of organizations have been collecting project performance data, including development firms, cost estimation tool vendors, and service provider companies. Furthermore, there are organizations that focus specifically on performance measurement and possess repositories of software project performance data. To leverage this existing information, the SEI held a workshop with experts from a small set of organizations that were already working in or had a strong interest in the area of software project performance measurement. (See Table 1 for a list of collaborating organizations.)

The kick-off workshop was held at the SEI in Pittsburgh, Pennsylvania during April 2006. A key outcome of this workshop was a consensus-based acknowledgement that the lack of common definitions for performance measures makes measurement comparison difficult or impossible. The group of collaborators decided that a primary goal of the collaboration should be to work together to (1) define a small set of key performance measures and influence factors that should be used by software projects and (2) to develop consensus-based definitions for those measures and factors.

Table 1. Collaborating organizations.

David Consulting Group	PRTM
Galorath Incorporated	QSM
ISBSG	Raytheon
Lockheed Martin	SEI
Motorola	Software Productivity Research, LLC.
Oracle	4SUM Partners
PRICE Systems	University of Ottawa

Problem approach

Through a series of workshops and group work sessions, a set of software project performance measures and influence factors was identified and defined.

The group started with a large list of candidate measures and factors, then used multi-voting to whittle the list down to what was considered a manageable set of terms.[1] The selection process was guided by the following questions:

1. What are the key measures that best characterize project performance?
2. What factors impact performance in a significant way?
3. What measures and factors would be most useful for software project performance comparison?
4. How difficult would it be to collect the factor or measure?
5. Is the factor or measure currently being collected as part of your organization's data repository?[2]

The final list of performance measures is presented in Table 2, and the list of influence factors is presented in Table 3.

Table 2. Performance Measures
Project effort
Productivity
Project duration
Schedule predictability
Requirements completion ratio
Post-release defect density

Table 3. Influence Factors
Size
Artifact reuse
Project type
Application domain
Average team size
Maximum team size
Team expertise
Process maturity
Functional requirements stability

[1] As the group worked to define the terms in the list, some changes were made and a few terms were added (e.g., artifact reuse). Changes to the list were made through group consensus.

[2] Many of the collaborators contributing to this effort possess their own proprietary data repository of software project performance data.

Problem approach, cont.

Once the set of performance measures and influence factors was selected, individuals volunteered to draft the definitions of specific terms for group discussion and decision-making. To guide development of the draft definitions, authors were asked to use or align with already existing standards such as those available through ISO and IEEE when possible. Literature research was conducted to leverage previous work when standard definitions were unavailable.

A significant amount of discussion and multiple revision cycles occurred over a period of months before the definitions were approved by group consensus. Once approved, the definitions were rewritten for a broad audience and examples were added to clarify or augment the definitions. The evolving revisions were reviewed by each of the collaborators followed by redlining sessions during workshops where group members met face-to-face.

Specificity of the definitions

There is an unavoidable conflict between the level of specificity of a definition and its usability by organizations. On the one hand, having a very detailed definition ensures comparability of measures collected. However, this likely reduces the number of organizations that would use that exact definition. This trade-off was discussed at length among the collaborators. In the end, we decided to seek a middle level of specification that would define the attribute but allow some tailoring and variation in the actual operation definitions employed.

There was concern that over-specification would lead to overly complex, protracted definitions that are difficult or inappropriate for some organizations to implement. We also recognized that organizations operate within different business environments which may influence the type of measurement detail that is practical and useful. For these reasons, we leave it to organizations to specify the next level of definition detail to the definitions specified in this document if doing so is appropriate to their context.

In Section 6 of this document, readers are encouraged to provide feedback about their experiences with using the performance measures and influence factors that are specified in this document. In the future, we would like to publish case studies that describe how this specification was implemented by various organizations.

3 Performance Measures for Software Projects—Overview

Introduction

Software project performance measures, definitions, and examples are provided in this section.

The set of measures does not represent an exhaustive list. The measurement experts who collaborated on this project determined that they were key measures that every organization should collect and use as a basis to compare performance between projects.

In this section

In this section, measurement indicators for software development projects are defined and illustrated.

Performance measure	Subsection	See Page
Project effort	3.1	10
Productivity	3.2	12
Project duration	3.3	13
Schedule predictability	3.4	15
Requirements completion ratio	3.5	17
Post-release defect density	3.6	18

3.1 Project Effort

Aliases Team effort, work effort

Definition *Project effort* is the total project team time that is spent on project-related activities during the life cycle of the project.[3]

Activities that do not specifically contribute to the development and delivery of the software products are excluded from the calculation of project effort.[4]

Project effort should include all project-related effort, including compensated and uncompensated overtime.

$$Project\,Effort = \sum_{i=1}^{n} Team_Member_Hours_i$$

where

$Team_Member_Hours_i$	is the time spent on project-related activities for team member i; and
n	is the total number of individuals that contributed time to project-related activities over the life cycle of the project.

[3] Project-related activities are events in the software process life cycle for which effort data is collected and reported [IEEE 1992].

[4] Some examples of activities that address needs not directly related to the project include company-wide meetings, conference attendance, information seminars, and professional development training.

Team Member Hours

This table lists the types of activities included as part of team member hours.

Category	Description
Direct delivered team hours	Team hours that directly contribute to defining or creating outputs (e.g., software code, user guide) that are delivered to the customer.
Direct non-delivered team hours	Direct team hours resulting in production of outputs (e.g., requirements tracing document, risk management database, defect tracking logs) that are *not* delivered with the final product.
Support hours	Hours expended by members of the project team on work that does not directly define or create products but assists those who do.

Example

A project team of 10 individuals recorded their time spent on project-related activities and reported the information at the end of each week. When the project was completed, the cumulative hours for each team member were calculated and the following table was produced.

	Team member	Hours
1	Project Manager	590
2	Requirements Analyst	260
3	Software developer	450
4	Software developer	450
5	Software developer	450
6	Software developer	233
7	Software developer	100
8	Software tester	175
9	Software tester	150
10	Quality Assurance	35
	Total	**2893**

Therefore,

$$Project\,Effort = \sum_{i=}^{n} Team_Member_Hours_i = 2893 hours$$

3.2 Productivity

Aliases Efficiency, yield

Definition *Productivity* of a software project is calculated as follows:

$$Productivity = \frac{Size}{Project\,Effort}$$

where

Size	is defined as described in Section 4.1 of this document; and
Project Effort	is defined as described in Section 3.1 of this document.

Comment Productivity is expressed as

size per hours

where "size" depends on how size is measured by an organization (e.g., lines of code, function points, feature points, use cases, objects).

Example #1: FP A project developed 136 function points (FP). The project effort to accomplish this was 5,346 hours. Therefore,

$$Productivity = \frac{Size}{Project\,Effort} = \frac{136}{5,346} = 0.25\ FP\ per\ hour$$

Example #2 LLC A project developed 14,346 logical lines of code (LLC). The project effort to accomplish this was 5,346 hours. Therefore,

$$Productivity = \frac{Size}{Project\,Effort} = \frac{14,346}{5,346} = 2.7\ LLC\ per\ hour$$

3.3 Project Duration

Aliases Project cycle time, time-to-market

Definition *Project duration* is a measure of the length of a project in work days, excluding times when the project is not active due to work stoppages. Project duration includes non-work days such as weekend days and holidays.

Project start is the date when user requirements have been baselined.

Project end is the date of the first installation of the software application.

Project Duration is calculated as follows:

$$ProjectDuration = num_days - toppage_days$$

where:

num_days is the total # of calendar days between the project start and project end; and

stoppage_days is the number of days when project work was not executed due to work stoppage.

Example User requirements for a software project were baselined on November 3, 2006. The first installation of the software was completed on September 14, 2007. Due to funding issues, the project was suspended for 10 work days during June, 2007.

Example, continued

The following table summarizes the project duration information.

Month	# Calendar days	Stoppage days
November 3 [start]	28	
December	31	
January	31	
February	28	
March	31	
April	30	
May	31	
June	30	10
July	31	
August	31	
September 14 [end]	14	
Total	**316**	**10**

Therefore:

num_days = (total # calendar days)

num_days = 316

Project Duration is calculated as:

$$\begin{aligned} \textit{Project Duration} &= \textit{num_days} - \textit{stoppage_days} \\ &= 316 - 10 \\ &= 306 \textit{ days} \end{aligned}$$

3.4 Schedule Predictability

Aliases

Schedule estimation accuracy, schedule estimation variance, schedule underrun/overrun, schedule slippage

Definition

Schedule predictability is a measure of how much the original project duration estimate differs from the actual project duration that was achieved. Schedule predictability is defined as a percentage as

$$SP = \frac{(Project\ Duration) - (Estimated\ Project\ Duration)}{Estimated\ Project\ Duration} * 100$$

where:

SP	is schedule predictability;
Project Duration	is as defined as in Section 3.3 of this document; and
Estimated Project Duration	is the original estimate of project duration as documented in the baselined version of the project plan.

Note that schedule predictability is a positive value when there is a schedule overrun and a negative value when there is a schedule underrun.

Example #1: Overrun

The estimated project duration was documented as 278 days in version 1.0 of the project plan. However, the actual duration realized was 367 days.

Therefore, Schedule Predictability is calculated as

$$SP = \frac{(ProjectDuration) - (EstimatedProjectDuration)}{EstimatedProjectDuration} * 100$$

$$= \frac{367 - 278}{278} = \frac{89}{278} * 100$$

$$= 32.0\%$$

Example #2: Underrun

The estimated project duration was documented as 123 days in version 1.0 of the project plan. However, the actual duration realized was 111 days.

Therefore, Schedule Predictability is calculated as

$$SP = \frac{(Project\ Duration) - (Estimated\ Project\ Duration)}{Estimated\ Project\ Duration} * 100$$

$$= \frac{111 - 123}{123} = -\frac{12}{123} * 100$$

$$= -9.8\%$$

The value is negative, reflecting a schedule underrun.

3.5 Requirements Completion Ratio

Aliases

Requirements planned/delivered, features planned/delivered, scope satisfaction

Definition

The *requirements completion ratio* measures the extent to which planned functional requirements were satisfied in the final product implementation.

The requirements completion ratio (RCR) is expressed as a percentage as

$$RCR = \frac{Satisfied\,reqs}{Planned\,reqs} * 100\ \%$$

where:

Planned reqs is the number of requirements that were originally baselined at the beginning of the project[5] and those that have been added or modified through negotiation with the user; and

Satisfied reqs is the number of functional requirements that were satisfied in the delivered software product.

Definition: Functional requirements

Functional requirements describe what the system, process, product, or service must do in order to fulfill the user requirements.

Example

The original baselined functional requirements specification contained 90 requirements, and 87 of those requirements were satisfied.

Therefore,

$$RCR = \frac{Satisfied\,reqs}{Planned\,reqs} * 100\ \%$$

$$= \frac{87}{90} = 0.967 * 100 = 96.7\%$$

[5] This is R_T as defined in the term Functional Requirements Stability (FRS) on page 73 of this document.

3.6 Post Release Defect Density

Aliases Defect density after deployment, post-release defects

Definition[6] *Post-release defect density* is the number of unique defects per unit size discovered during the first six months after initial deployment of the software. Post-release defect density is defined as

$$PRDD = \frac{\sum}{Size}$$

where

$PRDD$	is post release defect density;
$D =$	is total number of unique defects discovered by users during the first six months after initial installment of the software; and
$Size$	is as defined in Section 4.1 of this document.

Example: Using FP A project delivered an application of size 236 function points (FP) to a customer. A tally was kept of the unique problem reports that were documented by users during the first six months after initial deployment of the software. The total number of unique defects that were discovered and reported was 15. Therefore,

$$PRDD = \frac{\sum}{Project\ Size} = \frac{15}{236} = 6.4 \text{ defects per 100 FP}$$

Example: Using LLC A project delivered an application of size 5,500 LLC to a customer. Three months after the application was installed, a tally was taken of the unique problem reports that had been documented by users. The total number of unique defects that were discovered and reported totaled 39. Therefore:

$$PRDD = \frac{\sum}{Project\ Size} = \frac{39}{5{,}500} = 7.1 \text{ defects per 1000 LLC}$$

[6] The ISBSG Glossary of Terms [ISBSG 2006] and IEEE Std 982.1-1988 [IEEE 1988a] were used as reference to develop this definition.

4 Influence Factors for Software Projects—Overview

Introduction

In this section, software project influence factors are defined with some examples for purposes of illustration.

What are influence factors?

Influence factors are aspects of the development environment that can impact the outcome of the software project. Some influence factors are controllable by management, while others are not. When making comparisons between software projects, influence factors can be used to facilitate the comparison of projects that are similar to each other (with respect to one or more influence factors). In a sense, influence factors can be considered as independent variables whereas the performance measures act as the dependent variables.

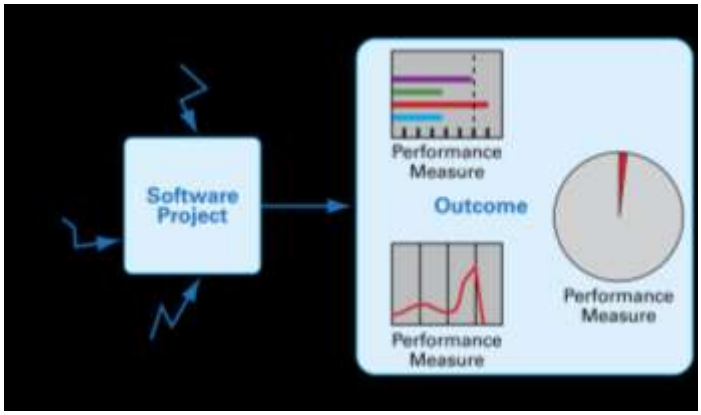


Figure 2. Distinguishing influence factors from performance measures.

Alternatives for some influence factors

While those collaborating on this project sought to develop a single common definition for each performance measure or influence factor, they recognized that in practice there are often different methods used for assessment of some factors. Alternatives are provided for the type of data that an organization may prefer to collect and use to characterize the following influence factors:

- size
- artifact reuse
- average team size
- process maturity

With regard to *size*, the collaborators of this definition effort acknowledged that various sizing methods have been embraced by different organizations in the software engineering community. There are strengths and weaknesses to any sizing approach for software, so various approaches have been identified for specifying size.

With regard to *artifact reuse*, the collaborators felt that there would be practical differences in the rigor that projects would be willing to apply in assessing this influence factor. Therefore, several methods for assessment have been identified and organizations can use the one they prefer.

Two methods are proposed for specifying *average team size* due to the variation in the way it is determined in the software community.

Finally, there are various approaches that are being used to assess *process maturity*. Therefore, various ways that this influence factor can be expressed are indicated.

In this section In this section, influence factors for software development projects are defined and illustrated.

Influence Factor	Subsection	See Page
Size	4.1	22
Artifact reuse	4.2	28
Project type	4.3	38
Application domain	4.4	39
Average team size	4.5	44
Maximum team size	4.6	47
Team expertise	4.7	49
Process maturity	4.8	51
Functional requirements stability	4.9	56

4.1 Size

Aliases

Project size, software size, functional size

Various size measures

Size is a measure of the extent of the software product that is developed and delivered by the project team. Understanding the size is a prerequisite to characterizing project performance, since productivity is defined as the amount of product delivered divided by the effort needed [IEEE 1992, ASQ 2007, Isixsigma 2007].

There are different approaches to measuring size. Two popular ways of measuring size are by counting (1) function points (FP) and (2) logical lines of code (LLOC).[7]

Since most projects report their size information in FP or LLC, we recommend using one of these counting methods if deemed appropriate by your organization. Doing so will make it possible to compare your project data to other projects that use one of these popular counting methods.

If your project uses a size measure other than FP or LLC (e.g., object points, web pages, screens), it may still be possible for you to make comparisons with projects that measure using LLC or FP. To do so, you need to develop a conversion factor that translates the project's size unit to LLC or FP. The conversion factor is the average number of LLC or FP required to generate one unit of the size measure being used by the project. The conversion factor would be similar to what QSM calls a *gearing factor* that is used to convert LLC into FP [QSM 2005].

[7] Other terms are sometimes used interchangeably with the term, logical lines of code (LLC) including but not limited to source lines of code (SLOC), logical lines of code (LLOC), logical source lines of code (LSLOC), and thousand lines of code (KLOC).

Example: Size measure other than FP or LLC

A project uses web pages as its size measure, and the size submitted is 47 web pages. Based on analysis conducted by the project, the average number of LLC required to produce a web page is 95.

a) the name of the measure is "web pages"

b) the software size in those units of measure is "47 web pages"

c) the conversion factor is "95 LLC = 1 web page"

d) the size in LLC using the conversion factor is "4,465 LLC"

Size-related terms

This table provides definitions of some important terms used in sizing methods.

Term	Definition
Logical lines of code (LLC)	A single software instruction, having a defined beginning and end independent of any relationship to the physical lines on which it is recorded or printed. Logical source statements are used to measure software size in ways that are independent of the physical formats in which the instructions appear [Park 1992].
Physical line of code	A single line of source code. Note that a logical line of code may consist of multiple physical lines of code [Park 1992].
Comment	Textual strings, lines, or statements that have no effect on compiler or program operations, usually designated or delimited by special symbols. Omitting or changing comments has no effect on program logic or data structures [Park 1992].
Blank lines	Lines in a source listing or display that have no visible textual symbols [Park 1992]. Blank lines are sometimes referred to as white space.

Table continues on next page

Size-related terms, continued

Table continues from previous page.

Term	Definition
Developed size	Developed size refers to statements (LLC method) or function points (FP method) that are *added* or *modified* for the specific application being developed. *Added size* is a count of those statements or function points that did not previously exist and were created specifically for the application. *Modified size* is a count of those statements or function points taken from a preexisting software product to which changes were made to make the software suitable for the application that is being measured [IEEE 1992].
Reused size	Source statements or function points that were not developed new for the application are referred to as reused size. Reused size is a count of unmodified LLC or FP obtained for the application from an existing source of software. The external source could be a previous version of the application, a reuse library, or acquired commercial off-the-shelf software [IEEE 1992].

FP method

The Function Point method sizes software by quantifying the tasks and services (i.e., functionality) that the software provides to the user based primarily on logical design. The objectives of function point analysis are to measure the functionality that the user requests and receives, delivered by software development and/or enhancements independent of the technology used for implementation.

To determine the size of a particular software release, five function types are measured: External Input, External Output, External Inquiry, Internal Logical File, and External Interface File.

When using the FP method, specify

1. the number of function points
2. whether function points are (a) unadjusted or (b) adjusted
3. the method followed to arrive at function point count (e.g., the Albrecht approach [Albrecht 1979], IFPUG guidelines [IFPUG 2005])

LLC method

The logical lines of code (LLC) method has been adapted from IEEE standard 1045-1992 [IEEE 1992]. This method entails counting computer instructions irrespective of the physical format in which they appear.

When using the LLC method,

- Count source statements for different programming languages separately; identify the programming language for each count.[8]
- Count source statements that are expanded within a software module (e.g., macro expansions) or supplied to the program (e.g., by an *include* statement); these statements shall be counted only once for all modules being measured.
- Count source statements that invoke, call, or direct inclusion of other source statements each time they are used.
- Count developed size statements and reused size statements.[9]
- Count only those source statements that were incorporated into the final product delivered to the customer; do not count software developed but not delivered to the customer such as software produced to support development of the final product (e.g., test software, tools, software aids).
- Do *not* count comment lines.
- Do *not* count blank lines.

[8] The programming language is identified since some languages are more efficient than other languages. For example, a fourth generation (4GL) language requires significantly fewer lines of code to accomplish the same functionality as a first generation language (1GL).

[9] Reuse is addressed in the section that follows titled, “Artifact Reuse.” The count of reused size statements can serve as an input to the artifact reuse measure.

Example #1: LLC method

Consider the following snippet of code using a simple Pascal-like language.

```
// Counter is updated

if X<Y
    then begin numcount := X-Y
    else begin numcount := X+Y;
```

When using the LLC method, this snippet would be counted as a single line of logical code.

There is also

- one comment line
- one blank line
- three lines of physical code

Explanation

- The single line of logical code begins with "if" and ends with the semi-colon; this is a single computer instruction although it is formatted in such a way that the single instruction wraps to three lines of physical code.
- The line beginning with "//" is a comment line.
- There is a blank space between the comment line and the line beginning with "if".

Example #2: LLC method

Consider the following snippet of C code.

```
for (x=0; x<50; ++x printf("Happy Birthday");  /* Print Greeting */
```

When using the LLC method, this snippet would be counted as two logical lines of code.

Explanation

- There is one physical line of code.
- There are two logical lines of code and a comment within the single physical line of code.
- The two logical lines of code include the *for* statement and the *printf* statement. Each of these is delimited by a semicolon.
- The text between the "/*" and "*/" is a comment.

Summary

Size information can be specified in one of the following ways:

1. number of function points (FP)

2. number of logical lines of code (LLC)
3. If using an alternative to FP or LLC, then provide size using one of the following ways:
   - by providing (a) the name of the size measure[10] and (b) the software size in those units of measure
   - by providing (a) the name of the size measure, (b) the software size in those units of measure , and (c) the conversion factor that translates the project's size unit to LLC or FP

[10] Examples of size measures other than LLC or FP include objects, web pages, use cases, implementation units (IUs) [Putnam 2005], screens, reports, database, tables, and scripts.

4.2 Artifact Reuse

Aliases

Asset reuse, software reuse

What is artifact reuse?

Artifact reuse is the use of existing software or software knowledge[11] to build new software or new documents for the project under consideration.

Reusable software knowledge items are referred to as reusable artifacts or reusable assets and may include requirement documents, designs, test cases, code, documentation or any other work product that is part of the project's development process.

Determining artifact reuse

An artifact reuse value is determined based on the reuse assessment method that is employed (assessment methods are described on page 29). In each of these methods, we are trying to find a proxy measure that provides a reasonable estimate of artifact reuse as defined by

$$Artifact\ Reuse = \frac{PE_{Saved}}{PE_{Total}} * 100$$

where

PE_{Saved}	is the project effort that was conserved or saved through the reuse of preexisting work products; and
PE_{Total}	is the total project effort that is calculated as described in Section 3.1 of this document.

Note: PE_{Saved} cannot be measured directly, so we preset an indirect way to approximate this value and to estimate artifact reuse. The remainder of this section describes how to accomplish this.

[11] Software knowledge is information that is captured in documents that are used within the software development process. These documents can include forms, templates, and instructions.

Developing an estimate of artifact reuse

Developing an estimate of artifact reuse relies on judgments made about

1. the percent of overall project effort required to develop the artifacts
2. the percent of effort savings realized by artifact reuse

Assessment methods

Project teams may vary in terms of the rigor they wish to apply to assessment of artifact reuse. To account for this, the following methods of assessment can be applied.

Assessment Method	Description
Gross	Artifact reuse that was realized during the project is estimated after project completion to obtain an estimate of PE_{Saved}. A low level of rigor is applied to how the estimate is derived.
Macro	Project effort is partitioned by project life-cycle phase. The degree of project reuse is estimated for each phase of the life cycle. These estimates are summed to obtain PE_{Saved}. The macro assessment applies more rigor than the gross assessment method.
Micro	Project artifacts are listed for each phase of the project's life cycle. The amount of effort conserved due to reuse is assessed for each of the artifacts within and across all life-cycle phases. These values are then summed to provide the estimate of PE_{Saved}. The micro method applies the highest degree of rigor to develop the estimate of PE_{Saved}.

When artifact reuse is reported, the data provider must also specify the assessment method used.

% Effort required to develop artifacts

This table provides guidance for estimating the percent of overall project effort required to develop the artifacts generated during each of the listed life-cycle phases.

The values in the right-most column are used as part of the computation to estimate artifact reuse which is described later in this section. These values are an estimate of the percent volume of artifacts developed for each of the life-cycle phases that are listed.

Assessment method	Life-cycle phase	Artifact type	% of phase effort to develop
Gross	n/a	n/a	100%
Macro	Design	n/a	40%
	Construction	n/a	25%
	Test	n/a	35%
Micro	Design	Concept/architecture	17%
		Detailed design description	60%
		Design validation	6%
		Other	17%
	Construction	Software code	95%
		Other	5%
	Test	Test plans	10%
		Test procedures	4%
		Test reports	13%
		Test drivers	25%
		Other	5%
		Other activities[12]	43%

Reuse attributes

When determining artifact reuse, consider

- reuse volume
- reuse effectiveness

Reuse volume

Reuse volume refers to the volume of reuse that was realized by the project.

[12] Since testing includes effort that must be performed regardless of available artifacts, testing artifacts account for less than 100% of the effort during that phase.

The reuse volume of software code is estimated by

(a) $\frac{Reused\,LLC}{Total\,LLC}$; or

(b) $\frac{Reused\,FP}{Total\,FP}$

A similar expression is used when the counting method is something other than LLC or FP.

For work products other than code, artifact reuse volume must be qualitatively estimated. In the case of non-code artifacts, the productivity gains from reuse will be different depending on the degree or extent to which project effort is conserved by reusing a given artifact.

In general, PE_{Saved} will be higher for *knowledge reuse* compared to reuse of simple templates or forms that have minimal knowledge content. While both templates and complex documents are examples of reuse, if boilerplate text from a complex document can be reused with minor modification, then PE_{Saved} will be significant compared to reuse of a document template that provides labeled categories only. In the latter case, the knowledge content that is reused is much less since the effort-intensive work is to complete sections of the template.

Assigning reuse volume for non-code artifacts

The reuse volume is obtained by assigning a percent value that represents the volume in terms of productivity savings for the artifact (if micro assessment is used) or group of similar artifacts (if gross or macro assessment method is used).

The assignment of volume is based on an individual's or team's best judgment.

Use the following table as guidance for making the assignment.

If the volume of reuse is perceived to be ...	Then use a percent in this range to assign reuse volume
High	67 – 100%
Medium	34 – 66%
Low	1 – 33%

Example

A project manager decided to use the micro method of assessment for determining artifact reuse.

As part of the assessment, she considered three documents that were part of the project repository.

The project manager considered the degree of reuse for each of the items and made the following assignments:

Document	Volume of reuse	Rationale
Change request template	5%	Simple form; would not take much to recreate. Contains only three category headings.
10-page legal form	90%	Most of the document text was from a boilerplate document that would have taken significant effort to create from scratch.
Peer review guidance doc	95%	No changes were made to document. It was used as-is by the project. Developing the process would have been very labor intensive.

Reuse effectiveness

Reuse effectiveness is the second attribute of artifact reuse. This attribute reflects the amount of modification required to make a preexisting artifact reusable for the project. A high value for effectiveness indicates that little or no modification is required. A low value indicates that the artifact or group of artifacts require a high degree of modification to be usable.

If the degree of modification is…	Use this value to characterize reuse effectiveness
None	100%
Low	66 – 99%
Moderate	33 – 65%
Significant	1 – 32%
Complete	0%

Example

A project manager decided to use the micro method of assessment for determining artifact reuse.

As part of the assessment, he considered several documents that were part of the project history folder.

The project manager considered the effectiveness of each artifact and assigned the values accordingly.

Document	Reuse effectiveness	Rationale
Change request template	100%	Complete reuse; no modification required.
10-page legal form	90%	Approximately 10% of the document required modification to be used.
Test plan	30%	Required some fairly significant changes so that it was appropriate for the project's testing approach.

Artifact reuse: gross assessment

When a gross assessment method is used, artifact reuse is estimated by the best judgment that can be brought to bear. In this case, the data provider specifies their best estimate of artifact reuse for the entire project.

This method is the least desirable since the estimate is made with little rigor.

Artifact reuse: macro assessment

When a macro assessment method is used, artifact reuse is estimated by

$$\begin{aligned} \textit{Artifact Reuse} = \; & (PE * R_Volume * R_Effect)_{Design} + \\ & (PE * R_Volume * R_Effect)_{Code} + \\ & (PE * R_Volume * R_Effect)_{Test} \end{aligned}$$

where

PE is the proportion of total project effort required to develop artifacts during a life-cycle phase, and

$PE_{Design} = 0.40$
$PE_{Code} = 0.25$
$PE_{Test} = 0.35$

R_Volume is the reuse volume that is assigned to all artifacts that were developed during a phase; and

R_Effect is the reuse effectiveness rating that is assigned to all artifacts that were developed during a phase.

Example: macro assessment

A project manager using the macro assessment developed the following estimates based on input from the project staff.

Phase	PE	Reuse Volume (*R_Volume*)	Reuse Effectiveness (*R_Effect*)
Design	0.40	0.6	0.3
Code	0.25	0.8	0.2
Test	0.35	0.5	0.2

Note: When using the macro assessment method, *R_Volume* and *R_Effect* refer to the entire suite of artifacts developed during that phase.

Artifact reuse is calculated as follows:

$$\begin{aligned} \textit{Artifact Reuse} &= [(PE * R_Volume * R_Effect)_{Design} + \\ &\quad (PE * R_Volume * R_Effect)_{Code} + \\ &\quad (PE * R_Volume * R_Effect)_{Test}] * 100 \\ &= [(0.4 * 0.6 * 0.3) + \\ &\quad (0.25 * 0.8 * 0.2) + \\ &\quad (0.35 * 0.5 * 0.2)] * 100 \\ &= [0.07 + 0.04 + 0.04] * 100 \\ &= 15\% \end{aligned}$$

Artifact reuse: micro assessment

When a micro assessment method is used, artifact reuse is estimated by

$$Artifact\ Reuse = \sum_{i=} \ tifact_PE_i * R_Volum\varepsilon * R_Effect_i * PE)_{Design} +$$
$$\sum_{i=} \ tifact_PE_i * R_Volum\varepsilon * R_Effect_i * PE)_{Code} +$$
$$\sum_{i=} \ tifact_PE_i * R_Volum\varepsilon * R_Effect_i * PE)_{Test}$$

where

PE_{Phase}	is the proportion of total project effort that is required to develop artifacts in a project life cycle phase and $PE_{Design} = 0.40$ $PE_{Code} = 0.25$ $PE_{Test} = 0.35$
$Artififact_PE_i$	is the proportion of the life-cycle phase effort that is required to develop Artifact i;
R_Volume_i	is the reuse volume that is assigned to Artifact i;
R_Effect_i	is the reuse effectiveness rating that is assigned to Artifact i;
m	is the number of artifacts being evaluated for the design phase;
n	is the number of artifacts being evaluated for the code phase; and
p	is the number of artifacts being evaluated for the test phase

Example: micro assessment

A project manager (with input from the project staff) used the micro assessment method for assessing artifact reuse. Each of the artifacts generated during the project was evaluated and assigned a reuse volume rating and a reuse effectiveness rating using this table as guidance.

Reuse Volume	If the volume of reuse is perceived to be ...	Use a percent in this range to assign reuse volume
	High	67 – 100%
	Medium	34 – 66%
	Low	1 – 33%

Reuse Effectiveness	If the degree of modification is ...	Use this value to characterize effectiveness
	None	100%
	Low	66 – 99%
	Moderate	33 – 65%
	Significant	1 – 32%

The results were tabulated and appear below.

Phase	Artifact	Artifact_PE	Reuse Volume (*R_Volume*)	Reuse Effectiveness (R_Effect)	PE
Design	Concept/architecture	0.17	0.2	0.3	0.4
	Detailed design doc	0.60	0.3	0.1	
	Design validation	0.06	0.3	0.1	
	Other docs	0.2	0.5	0.6	
Code	Reused code	0.95	0.25	1.0	0.25
	Other docs	0.05	0.2	0.3	
Test	Test plans	0.10	0.2	0.6	0.35
	Test procedures	0.04	0.2	0.7	
	Test reports	0.13	0.2	0.1	
	Test drivers	0.25	0.2	0.2	
	Other docs	0.05	0.3	0.3	
	Other activities	0.43	*n/a*	*n/a*	*n/a*

Example: micro assessment, continued

Overall artifact reuse for the project is calculated as follows:

$$\begin{aligned}
\textit{Artifact Reuse} &= \sum_{i=} \textit{tifact_PE}_i * \textit{R_Volum}\text{ę}_i * \textit{R_Effect}_i * \textit{PE})_{\textit{Design}} + \\
&\quad \sum_{i=} \textit{tifact_PE}_i * \textit{R_Volum}\text{ę}_i * \textit{R_Effect}_i * \textit{PE})_{\textit{Code}} + \\
&\quad \sum_{i=} \textit{tifact_PE}_i * \textit{R_Volum}\text{ę}_i * \textit{R_Effect}_i * \textit{PE})_{\textit{Test}} \\
&= (0.17 * 0.2 * 0.3 * 0.4) + \\
&\quad (0.60 * 0.3 * 0.1 * 0.4) + \\
&\quad (0.06 * 0.3) * 0.1 * 0.4) + \\
&\quad (0.2 * 0.5 * 0.6 * 0.4) \\
&\quad + \\
&\quad (0.95 * 0.25 * 1.0 * 0.25) + \\
&\quad (0.05 * 0.2 * 0.3 * 0.25) \\
&\quad + \\
&\quad (0.10 * 0.2 * 0.6 * 0.35) + (0.04 * 0.2 * 0.7 * 0.35) + \\
&\quad (0.13 * 0.2 * 0.1 * 0.35) + (0.25 * 0.2 * 0.2 * \\
&\quad 0.35) + (0.05 * 0.3 * 0.3 * 0.35) \\
&= (0.00408 + 0.0072 + 0.00072 + 0.024) + \\
&\quad (0.059375 + 0.00075) + \\
&\quad (0.0042 + 0.00196 + 0.00091 + 0.0035 + 0.001575) \\
&= 0.10827 \approx 0.11
\end{aligned}$$

4.3 Project Type

Alias Development type

Definition *Project type* is a classification that characterizes a project as belonging to one of the following type and subtype categories.

Type	Subtype	Description
New software	n/a	Newly developed software that does not include a preexisting base of previously developed software.
Modifications of existing software	Enhancement	Adding, changing, or deleting functionality to a preexisting application.
	Maintenance	Enhancement such as repairing defects, code restructuring, performance tuning, or other changes that are not directly related to changing the functionality of the application.
	Conversion	Conversion of source code so that application can be ported to a different platform. Functionality remains unchanged.
	Package implementation	Acquiring, modifying, configuring, and deploying a commercial off-the-shelf (COTS) software application. No changes made to delivered features or functionality.
	Package customization	Acquiring, modifying, configuring, and deploying a COTS software application. Results in changes to delivered features or functionality.
	Reengineering	Reconstructing an application based on formal design artifacts and a preexisting software base.

4.4 Application Domain

Aliases

Application purpose, application use

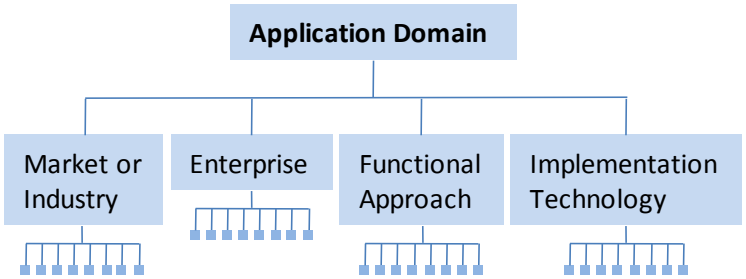
Definition

The application domain describes the environment and role of a software application.

The application domain of the software project is selected by choosing a category and various subcategories (if applicable) from the taxonomy that begins on the following page.

Diagram

This diagram shows the major dimensions of the application domain taxonomy. Each of the major dimensions is segmented into type and in some cases, subtype, categories.



Selecting your application domain

When categorizing your software project application, use one of the predefined alternatives for each of the major dimensions. In some cases, additional subcategory selections are available.

When you believe that your application does not adequately map to one of the predefined categories, specify "other" and list the category of your choice. In this way, the application domain taxonomy can be improved and expanded over time.

Application domain taxonomy

The table below lists an application domain taxonomy for categorizing software project application domains.

This taxonomy was generated by synthesizing and adapting previous work conducted in the area of application domain taxonomies [Glass 1995, Reifer 1990].

Dimension	Type	Subtype
Market/Industry	Agriculture	
	Computer/Software	
	Educational	
	Energy	
	Engineering/Construction/Architecture	
	Entertainment	
	Financial/Banking	
	Federal Government – non-military	
	State or Local Government	
	Home/Consumer	
	Industrial/Manufacturing	
	Insurance	
	Legal	
	Media/Publishing	
	Medical/Healthcare	
	Military/Defense	Air Force
		Army
		Navy
		Marines
		Coast Guard
		Other
	Nonprofit Institutions	
	Real Estate	
	Retail/Wholesale/Distribution	
	Scientific	
	Telecommunications	

Table continues on next page

Dimension	Type	Subtype
Market/Industry, continued	Transportation	Automotive
		Aviation
		Rail
		Naval
		Space
		Other
	Utilities	
	Other	
Enterprise	Sales	Contract Management
		Sales Readiness Software
		Other
	Marketing	Survey Management
		Other
	Management	Project Management
		Estimation
		Other
	Distribution/Supply Chain	
	Human Resources	
	Finance	
	Customer/Technical Support	Help desk software
		Other
	Information Technology	Data processing
		Business Systems
		Management Information Systems (MIS)
		Customer Relationship Management (CRM)
		Human Resource (HR) Systems
		Office Automation
		System (Administration) Software
	Education/Training	Classroom Management
		Training Management
		Other
	Manufacturing and production	

Table continues on next page

Dimension	Type	Subtype
Enterprise, continued	Command and Control (C2)	Air Defense Systems
		Air traffic control systems
		Mission-critical – ground
		Missile defense systems
		Operations/command center
		Satellite ground stations
		Tactical data systems
		Warning systems
		Other
	Intelligence, Surveillance, Reconnaissance (ISR)	Radar systems
		Satellite software
		Encryption
		Other
	Controls and Displays	Heads-Up Display (HUD)
		Other
	Weapons Systems	Electronic Warfare
		Fire control
		Other
	Navigation and Guidance	Identify Friend or Foe (IFF)
		Auto-pilot software
		Guidance systems
		Inertial navigation systems
		Other
	Telecommunications	Communications systems
		Modems/transmission systems
		Networking systems
		Switching systems
		Wireless phone software
		Teleconferencing
		Videoconferencing
		Other
	Automation/Process Control	Chemical
		Manufacturing
		Printing and publishing
		Other
	Simulation	Aircrew trainers
		Environmental simulators
		War-gaming simulators
		Other

Table continues on next page

Dimension	Type	Subtype
Enterprise, continued	Consumer	Entertainment
		Tax software
		Home office software
		Other
	Internet	Web-based training
		On-line shopping
Functional Approach	Signal Processing	
	Transaction Processing/Database	
	Knowledge-Based System	
	Rule-Based System	
	Virtual Reality	
	Robotics	
	Simulation and Modeling	
	Other	
Implementation Technology	Object-oriented	
	Client-server	
	Parallel processing	
	Distributed computing	
	Embedded	
	Firmware	
	COTS/GOTS	
	Other	

4.5 Average Team Size

Aliases Project staff size, average project staff size

Definition *Average team size* is the average number of individuals allocated to the project over the course of the project life cycle.

Average size may be calculated by (a) average headcount method, or (b) full-time equivalent (FTE) method.

(a) Average headcount method:

Average team size for a project of n months duration is calculated as follows

$$Average\ Team\ Size = \frac{\sum_{i=} \text{am Mem berCount}_i}{n}$$

where

$Team\ Member\ Count_i$	is the number of project staff members who work during month i of the project; and
n	is the duration of the project in months.

(b) Full-time equivalent method:

$$Average\ Team\ Size = \frac{Project\ Effort}{1776 * \frac{Project\ Length}{12}}$$

where

Project Effort	is as defined in Section 3.1 of this document;
Project Length	is the duration of the project in months.

When specifying a value for Average Team Size, the method for calculating must also be specified.

What is FTE? Full time equivalent (FTE) is a way to measure a worker's involvement in a project. An FTE of 1.0 means that the person is equivalent to a full-time worker.

In the U.S. Federal government, FTE is defined by the Office of Management and Budget (OMB) as a manpower measure used by the DoD to represent a year's worth of employee effort that equals 1,776 productive

work hours, excluding holidays and leave [OMB 2003].

For example, if the work year is defined as 1,776 hours, then one worker occupying a paid full time job all year would consume one FTE. Two employees working for 888 hours each would consume one FTE between the two of them.

Example: Average headcount method

Consider a five-month project with team members listed in the table below. An "X" denotes that the team member worked during that particular month.

		Month				
	Team member	**1**	**2**	**3**	**4**	**5**
1	Project Manager	X	X	X	X	X
2	Requirements Analyst	X	X			
3	Software Developer	X	X	X	X	X
4	Software Developer	X	X	X	X	X
5	Software Developer			X	X	X
6	Software Developer			X	X	X
7	Software Developer				X	X
8	Software Developer				X	X
9	Software Tester				X	X
10	Software Tester					X
11	Quality Assurance	X	X	X	X	X
	Total	5	5	6	9	10

Therefore,

$$Average\ Team\ Size = \frac{\sum_{i=} am\,MemberCount_i}{n}$$

$$= \frac{5+5+6+9+10}{5} = \frac{35}{5} = 7$$

Example: FTE method

Project staff members recorded how much time they worked on a four-month long project.

Staff member	March	April	May	June
Project Manager	165	157.5	165	157.5
Software Developer	165	157.5	165	206
Software Developer	40.5	80	120	206
Software Developer	38	80	110	206
Software Tester	4	4	30	157.5
Quality Assurance	4	4	15	15
Subtotals	416.5	483	605	948

Therefore, using the FTE method,

$$Average\ Team\ Size = \frac{Project\ Effort}{1776 * \frac{Project\ Length}{12}}$$

$$= \frac{416.5+483+605+948}{1776 * \frac{4}{12}} = \frac{2452.5}{592} = 4.1$$

4.6 Maximum Team Size

Alias Maximum staff size

Definition *Maximum team size* is the highest number of individuals that are allocated to the project over the course of the project life cycle.

Maximum team size for a project of n months duration is calculated as follows

$$Maximum\ Team\ Size = Max(x_i \ldots x_n)$$

where

x_i is the size of project team that worked at least 40 hours during month i of the project, where i is a positive integer in the range (1, n); and

n is the duration of the project in months.

Example Team members and their monthly resource allocations for a five-month project are listed in the table below.

		Month				
	Team member	**1**	**2**	**3**	**4**	**5**
1	Project Manager	X	X	X	X	X
2	Requirements Analyst	X	X			
3	Software Developer	X	X	X	X	X
4	Software Developer	X	X	X	X	X
5	Software Developer			X	X	X
6	Software Developer			X	X	X
7	Software Developer				X	X
8	Software Developer				X	X
9	Software Tester				X	X
10	Software Tester					X
11	Quality Assurance	X	X	X	X	X
	Total	5	5	6	9	10

Example, continued

The maximum team size is

$$Maximum\ Team\ Size = Max(x_i \ldots x_n)$$
$$= Max\ (5, 5, 6, 9, 10)$$
$$= 10$$

4.7 Team Expertise

Aliases Developers' expertise, personnel expertise, technical expertise.

Definition *Team expertise* is a 5-tuple of measures of the proficiency of the project team during each phase of the software development life cycle. The measure is a subjective one based on the informed expert judgment of those who perform the assessment.

The team expertise measure for each phase is an integer in the range (1 to 5) where 1 represents novice proficiency ability and 5 represents expert proficiency.

$$TE = (TE_{req}, TE_{arch}, TE_{dd}, TE_{code}, TE_{st})$$

where:

TE_{req} is expertise rating for team members who contribute to the Concept and Requirements Analysis Phase

TE_{arch} is expertise rating for team members who contribute to Architectural and/or High-Level Design Phase

TE_{dd} is expertise rating for team members who contribute to Detailed Design Phase

TE_{code} is expertise rating for Code Construction and Unit Testing Phase

TE_{st} is expertise rating for team members who contribute to System Test Phase

Rating scale This table provides simple guidance for mapping the integer rating scale.

Rating	Description of team expertise
5	All experts
4	Mostly experts & some novices
3	Some experts & some novices
2	Mostly novices & some experts
1	All novices

Guidance

The following steps provide guidance for assigning team expertise:

1. Select a life-cycle phase.
2. Consider the team in its entirety for that phase. Make an overall judgment of the team's expertise and assign the rating for that phase. (This is the rating for the life-cycle phase under consideration.)
3. Select the next life-cycle phase and repeat from #2.

Example

A project manager assessed the technical expertise of the project team during each phase of the project.

An "X" within a cell indicates that the individual was considered part of the project manager's assessment for a particular life-cycle phase.

The project manager considered the team that participated in each phase and assigned a rating for each. The rating is listed in the last row of the table.

		Phase				
	Team member	**Req.**	**Arch.**	**DD**	**Code**	**ST**
1	Project Manager	X	X	X	X	X
2	Requirements Analyst	X				
3	Software Developer 1	X	X	X	X	X
4	Software Developer 2	X	X	X	X	X
5	Software Developer 3	X		X	X	X
6	Software Developer 4			X	X	X
7	Software Developer 5			X	X	X
8	Software Developer 6			X	X	X
9	Software Tester 1				X	X
10	Software Tester 2				X	X
11	Quality Assurance					
	Team Expertise Rating	5	5	3	3	3

Example, continued

Therefore,

$$TE = (TE_{req}, TE_{arch}, TE_{dd}, TE_{code}, TE_{st})$$
$$= (5, 5, 3, 3, 3)$$

4.8 Process Maturity

Aliases

Software process maturity, capability, organizational capability

Definition

Process maturity is the extent to which a project's processes are explicitly defined, managed, measured, and controlled.

Explanation

In the broad arena of process maturity appraisal, a number of standards and models are used to assess process maturity. Some of the more popular approaches include the following:

- The Carnegie Mellon® SEI CMMI® framework
- ISO 9001
- ITIL
- ISO 15504 (SPICE)

The approaches listed above use different rating schemes to indicate the degree of process maturity. In ISO 9001, process maturity is assessed as (a) compliancy or (b) non-compliancy. For the other listed approaches, a level is assigned to an organization (or project) as an indicator of process maturity.

A maturity level is a defined evolutionary plateau for organizational process improvement. The maturity levels are measured by the achievement of the goals associated with each predefined set of process areas.

Quite often, process maturity is assessed for the organization to which a project belongs. When that is the case, it is assumed that the project processes share the maturity rating of the organization.

The remainder of this section describes the rating scheme for each of the process maturity models listed above.

® CMM and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce and is registered in the U.S. Patent and Trademark Office.

CMMI[13]

The maturity level is the degree of process improvement across a predefined set of process areas in which all goals in the set are attained. There are five maturity levels, each a layer in the foundation for ongoing process improvement, designated by the numbers 1 through 5 [SEI 2006, Chrissis 2006].

Maturity Level	Category
1	Initial
2	Managed
3	Defined
4	Quantitatively Managed
5	Optimizing

CMMI-based process maturity is defined as a 2-tuple, (x, y) where:

- x is the maturity level and $x \in \{1, 2, 3, 4, 5\}$;
- y is the year that an appraisal was conducted that established the maturity level

[13] CMMI replaced the CMM for Software which is no longer supported by the SEI and should not be used for rating process maturity. The CMM for Software included a rating scale similar to that of CMMI, but used different criteria. When using process maturity data, it is important to know which model was used as the criteria for making the rating. Some historical data on process maturity may reflect ratings against the CMM for Software.

ISO 9001 series

ISO 9001 is part of the ISO 9000 family of standards for quality management systems. ISO 9000 is maintained by ISO, the International Organization for Standardization and is administered by accreditation and certification bodies [ISO 2000].

ISO 9000 is composed of the following sections:

- ISO 9000:2000

  Quality management systems—fundamentals and vocabulary. Covers the basics of what quality management systems are and also contains the core language of the ISO 9000 series of standards. The latest version is ISO 9000:2005.

- ISO 9001 Quality management systems

  Requirements. Intended for use in any organization that designs, develops, manufactures, installs or services any product or provides any form of service. It provides a number of requirements that an organization needs to fulfill if it is to achieve customer satisfaction through consistent products and services that meet customer expectations.

- ISO 9004 Quality management systems

  Guidelines for performance improvements. Addresses continual improvement. This document provides advice on what can be done to enhance a mature system.

A company or organization that has been independently audited and certified to be in conformance with ISO 9001 may publicly state that it is "ISO 9001 certified" or "ISO 9001 registered." Certification to an ISO 9000 standard does not guarantee the compliance (and therefore the quality) of end products and services; rather, it certifies that consistent business processes are being applied.

ISO 9001-based process maturity is defined as a 2-tuple, (x, y) where

x is compliant if the project's organization is compliant with ISO 9000[14] and null if organization is either non-compliant or did not use this method for assessing process maturity.

y is the year that the project's organization was certified as being compliant with ISO 9000.

[14] It is usually sufficient to understand that when an organization claims to be "ISO 9000 compliant," it means they conform to ISO 9001.

ITIL

The Information Technology Infrastructure Library (ITIL) is a framework of best practice approaches intended to facilitate the delivery of high-quality information technology (IT) services. ITIL outlines an extensive set of management procedures that are intended to support businesses in achieving both quality and value, in a financial sense, in IT operations [OGC 2007].

IT Service Support Processes	IT Service Delivery Processes
Service Desk (SD)	Service Level Management
Configuration Management (CON)	Availability Management
Incident Management (IM)	Capacity Management
Problem Management (PM)	Financial Management
Change Management (CHA)	IT Service Continuity Management
Release Management (RM)	

Each of the above process areas can be assessed and assigned a maturity rating as follows:

Level	Maturity
0	Absence
1	Initiation
2	Awareness
3	Control
4	Integration
5	Optimization

ITIL certifications are managed by the ITIL Certification Management Board (ICMB) which is comprised of the OGC, IT Service Management Forum (ITSMF) International, and two examinations institutes. Organizations or a management system may not be certified as ITIL-compliant. However, an organization that has implemented ITIL guidance in IT service management (ITSM) may be able to achieve compliance with and seek certification under ISO/IEC 20000.

ITIL-based process maturity is defined for each of the IT Service Support Processes and IT Service Delivery Processes by assigning a maturity level rating x to each process area (i.e., IT Service Support Processes and IT Service Delivery Processes) where $x \in \{0, 1, 2, 3, 4, 5\}$. Also, the data-provider must provide the year that the project's organization was assessed using the ITIL-based assessment method.

ISO/IEC 15504 (SPICE)

ISO/IEC 15504 contains a reference model that defines a process dimension and a capability dimension [ISO 2004a, SEI 2008]

The *process dimension* divides processes into the following five categories:

- customer-supplier
- engineering
- supporting
- management
- organization

For each process, ISO/IEC 15504 defines a capability level on the following scale:

Level	Maturity
0	Incomplete
1	Performed
2	Managed
3	Established
4	Predictable
5	Optimizing

15504-based process maturity is defined for each of the process categories listed above by assigning a maturity level rating x where $x \in \{0, 1, 2, 3, 4, 5\}$.

Also, the data-provider must provide the year that the project's organization was appraised using the 15504-based appraisal method.

4.9 Functional Requirements Stability

Aliases Requirements volatility, scope creep, feature creep

Definition[15] *Functional requirements stability* is a measure that quantifies the cumulative degree to which the requirements changed throughout the life cycle of the project from the original requirements baseline.

Functional requirements stability (FRS) is defined as

$$FRS = \frac{R_T - R_C}{R_T}$$

where

R_T is the total number of requirements that were originally baselined at the beginning of the project; and

R_C is the total number of changes to the original baselined requirements and

$$R_C = R_a + R_m + R_r$$

where

R_a is the number of new requirements *added* to the original baselined requirements specification

R_m is the number of requirements *modified* in the original baselined requirements specification

R_r is the number of requirements *removed* from the original baselined requirements specification

Interpretation The maximum value of FRS is 1.0 and would indicate complete stability of the functional requirements. Decreasing values of FRS indicate increasing instability of the requirements.

[15] This definition has been adapted from Felici [Felici 2003].

Functional requirements vs. user requirements

This definition distinguishes functional requirements from user requirements. The measure addresses functional requirements only.

User requirements are sometimes referred to as business requirements and they are requirements governing the project's deliverable or product as expressed by the users. User requirements are typically expressed in terms of broad outcomes the user or business requires, rather than specific functions the software systems may perform.

Functional Requirements describe what the system, process, or product/service must do in order to fulfill the user requirements.

Example

The original baselined functional requirements specification contained 90 requirements.

Over the course of the project life cycle, two new requirements were added, ten of the original requirements were modified, and three of the requirements were removed. Therefore,

$R_T = 90$,

$R_a = 2$,

$R_m = 10$,

$R_r = 3$,

$R_C = R_a + R_m + R_r = 2 + 10 + 3 = 15$, and

$$FRS = \frac{R_T - R_C}{R_T} = \frac{90 - 15}{90} = \frac{75}{90} = 0.83$$

5 Using the Performance Measures and Influence Factors

Introduction

In this section we present several scenarios that demonstrate how the performance measures and influence factors can be used by an organization and the benefits derived from their use.

The need

Before valid measurement comparison and benchmarking can be conducted, common operational definitions for the measures must be in place. Herein lies one of the major obstacles that has prevented organizations from being able to effectively compare software project performance among projects within their organization and with projects outside of their organization.

Using the definitions

We believe that this data specification can be used by organizations that

- are beginning measurement programs
- want to standardize the way measures are defined across the enterprise
- want to compare their performance to projects that have submitted their data to proprietary and public project performance repositories
- want to conduct benchmarking studies between projects within and/or outside their organizations

Each of these cases is described in more detail in this section.

Standardization within an enterprise

Large organizations can benefit from adopting a standard set of software project performance measures. By doing so,

- personnel within the organization do not need to relearn new definitions as they move from one project to the next
- the organization could compare performance among projects

Beginning a measurement program

Organizations that are beginning a measurement program are often perplexed as to what they should begin to measure and how those measures should be defined. Organizations should always begin by using an approach such as the goal-driven measurement method [Park 1996, Basili 1994] to assure that measures are addressing the key goals of the organization. However, we believe that any organization that develops software will want to include the set of measures and factors identified in this report due to the intrinsic ability of these measures to characterize project performance.

Measurement comparison

Organizations using the definitions in this document can compare their software project performance measures to measures that are included in existing software project performance data repositories.

The definitions in this document were arrived at through a consensus-based process involving many of the leading organizations in the area of software project performance measurement. Many of the collaborators in this definition project have their own proprietary repositories that hold project performance measurement information from thousands of software projects [David 2008, Galorath 2008, Price 2008, QSM 2008, SPR 2008, 4Sum 2008]. Also, the International Software Benchmarking Standards Group (ISBSG) maintains a public repository of software project performance measures [ISBSG 2008]. However, each organization defines the measures differently, making it difficult or impossible to compare measures among these repositories.[16]

Collaborators that participated in this project and possess repositories were asked to characterize whether data from their repository could be matched with the performance measures and influence factors defined in this document. Table 4 is a summary of the responses.[17]

[16] For this reason, collaborators in this definition project believe that significant benefit would result for the software community if common definitions are adopted for performance measures.

[17] In some cases, collaborators reported that although they have not been collecting the particular influence factor or performance measure, they believe that there should be encouragement to do so because useful insights could be realized by capturing the information.

		Collaborating organizations that possess data repository					
	Data Item	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**
Influence Factors	Size	●	●	●	●	●	●
	Artifact reuse	●			◉	●	
	Project type	◉	●	●	●	●	●
	Application domain	●	◉	●	◉	◉	●
	Average team size		◒	●		●	●
	Maximum team size			●		●	●
	Team expertise	◉	●		●	●	
	Process maturity	◉		◒	●	●	
	Functional requirements stability			◉	◉	◉	
Indicators	Project effort	●	●	●	●	●	●
	Productivity	●	●	●	●	●	●
	Project duration	●	●	●	●	●	●
	Schedule predictability			●		◒	●
	Requirements completion ratio		●	●		◒	
	Post-release defect density	◉		●	◒	●	●

● Data collected
◒ Data partially collected
○ Don't know

Table 4. Repositories that possess data that map to the definitions specified in this document.

Support for benchmarking

This document provides a set of performance measures and influence factors that can be used as the nominal set of measures for performance comparison within a software project benchmarking study.

Benchmarking is an improvement approach that relies on common definitions of performance measures so that valid comparisons can be made between the projects that are part of the benchmarking exercise. The establishment of common performance measures is a prerequisite to conducting benchmarking. See the appendix for a more detailed discussion of benchmarking.

6 Request for Feedback

The SEI is interested in collecting feedback from and collaborating with organizations that intend to implement or are implementing the software project performance measures and influence factors that are specified in this document. If you would like to provide feedback or discuss collaboration, contact customer-relations@sei.cmu.edu.

Appendix: Benchmarks and Benchmarking

What are benchmarks?



Performance measurement and the concept of a benchmark go hand-in-hand. The term "benchmark" comes from geological surveying and means to take a measurement in comparison to a reference point.[18]

In the process improvement lexicon, a benchmark is a best-in-class performance or achievement. This achievement is then used as a reference point against which similar processes are compared or judged.

What is benchmarking?

There is a distinction between the term "benchmark" (noun), and the process of "benchmarking" (verb). While a benchmark is a measure, the process of benchmarking is an ongoing improvement process that compares a project's internal practices, processes, and methods to projects from other organizations. The purpose of benchmarking is to identify the best practices that led the project that owns the benchmark to achieve stellar performance. Once identified and characterized, these best practices are then adapted to achieve similar process improvements and concomitant enhanced performance.

Bench-mark-ing

The process of improving performance by continuously identifying, understanding, and adapting outstanding practices and processes found inside and outside the organization.

- American Productivity & Quality Center

The benchmarking approach to process improvement originated at Xerox during the early 1980s as part of the company's Total Quality Management (TQM) program called "Leadership Through Quality." Following their initial big successes using benchmarking, senior management required all organization within Xerox to pursue benchmarking. Robert C. Camp of Xerox is often referred to as the Father of Benchmarking as he is credited with developing the first formal, documented process for benchmarking [Camp 1989, Camp 1995, Camp 1998].

[18] The term originated as a surveyor's mark made on a stationary object of previously determined position and elevation and used as a reference point in tidal observations and surveys [AHD 2006].

What is benchmarking? –continued

Under the auspices of the American Productivity & Quality Center's International Benchmarking Clearinghouse, a guidebook has been developed that offers basic information about the benchmarking process [APQC 1993]. Many companies have adapted the generic benchmarking process model in their own ways. Recognizing that it is difficult to communicate among companies that use different approaches to benchmarking, four companies that are active benchmarkers created a four-quadrant model to explain what benchmarking is about. This template is adapted from APQC's Benchmarking Guidebook and is illustrated in Figure 3 [APQC 1993].

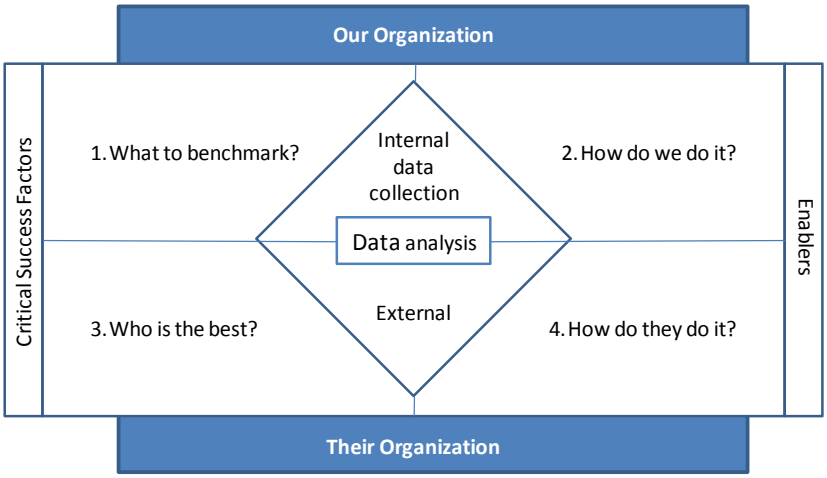


Figure 3. Benchmarking Process Template

This template establishes the general context model for a process that indicates the specific actions to complete the benchmarking process. The four quadrants are linked by the processes of data collection and analysis of performance measures. Enablers refer to the processes, practices, or methods that make possible the best-in-class performance. While performance benchmarks measure the successful execution of a process, enablers tell the reasons behind the successful implementation: the system, method, document, training, or techniques that facilitate the implementation of the process [APQC 1993]. Critical success factors are the characteristics, conditions, or variables that have a direct influence on your customer's satisfaction (and therefore your success).

Table 5 shows examples of the questions a team would ask for each of the quadrants.

What is benchmarking? —continued

Quadrant	Questions that are asked
1. What to benchmark?	• Have you identified critical success factors for your organization? • Have you selected the right thing to tackle (e.g., problem area to address, result to achieve)? • Will a change in the targeted process be perceived by customers as a benefit?
2. How do we do it?	• Have you mapped out your benchmarking process, and do you understand how you are doing it? • Will you be able to compare your measurements to others and make sense of the result?
3. Who is best-in-class?	• Which organizations perform this process better than you do?
4. How do they do it?	• What is their process? • What enables the performance of their process?

Table 5. Questions for each quadrant of the Benchmarking Process Template.

AQPC identifies four ways that benchmarking can be segmented according to the types of comparisons that are made during a particular study [APQC 1993].

- **Internal studies** compare similar operations within different units of an organization. While this simplifies implementation and data access, it yields the lowest potential for significant breakthroughs.
- **Competitive studies** target specific products, processes, or methods used by an organization's direct competitors. These types of studies are usually conducted by a third party to sanitize competitive information, nominalize performance to an agreed-upon base measure, and report case study information that has been approved by the contributing company. Competitive information is exceptionally difficult to obtain due to the concern about disclosure and antitrust issues.
- **Functional or industry studies** compare similar functions within the same broad industry or compare organization performance with that of industry leaders. This type of study has a good opportunity to produce breakthrough results and provide significant performance improvement. Because of the potential for industry studies to become available to direct competitors, these studies are typically conducted in the blind through a third party.

What is benchmarking? –continued

- **Generic benchmarking** compares work practices or processes that are independent of industry. This method is considered by some to be the most innovative and can result in changed paradigms for reengineering specific operations.

Various process models have been developed to describe benchmarking. APQC has studied companies that have strong benchmarking initiatives. Although there are differences between the models, they all follow a similar pattern. The one observation made is that most of the specific company models map into the Deming Cycle of Plan, Do, Check/Measure, Act.[19] The Xerox Benchmarking Process Model is summarized in Figure 4. In his book titled *The Benchmarking Book*, Spendolini describes a five-stage process that is very similar, but summarized at a higher level. The stages are (1) Determine what to benchmark, (2) Form a benchmarking team, (3) Identify benchmark partners, (4) Collect and analyze benchmarking information, and (5) Take action [Spendolini 1992].

Benchmarking is a well-established approach and there are many reference sources to help organizations get started. Benchmarking is recognized as an important tool in the process improvement toolbox of Six Sigma and other quality improvement approaches [Isixsigma 2007, Breyfogle 2003, Juran 1998].

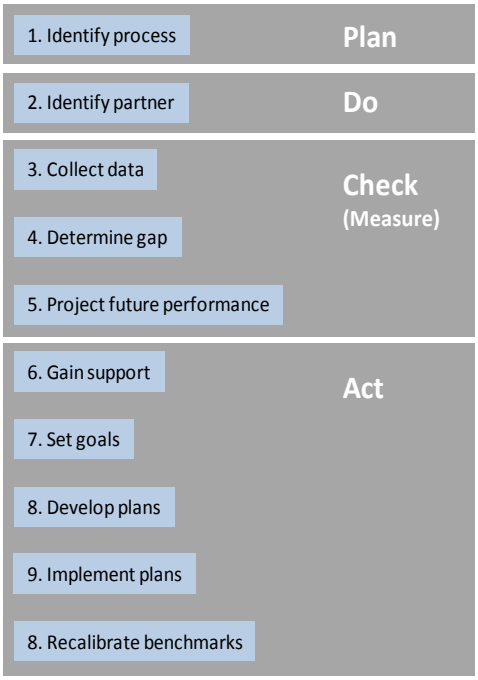


Figure 4. Xerox Benchmarking Process Model mapped to Deming Cycle.

[19] The Deming Cycle in quality is named after its leading proponent, Dr. W. Edwards Deming [Deming .1986].

Benefits of benchmarking

The purpose of benchmarking is to adapt stellar processes and practices from leading organizations so that the true potential of the organization can be realized. This is shown conceptually in Figure 5.

A research study conducted by APQC's International Benchmarking Clearinghouse demonstrated benchmarking's tremendous leverage. More than 30 organizations reported an average $76 million first-year payback from their most successful benchmarking project. Among the most experienced benchmarkers, the average payback soared to $189 million [APQC 2008a][20].

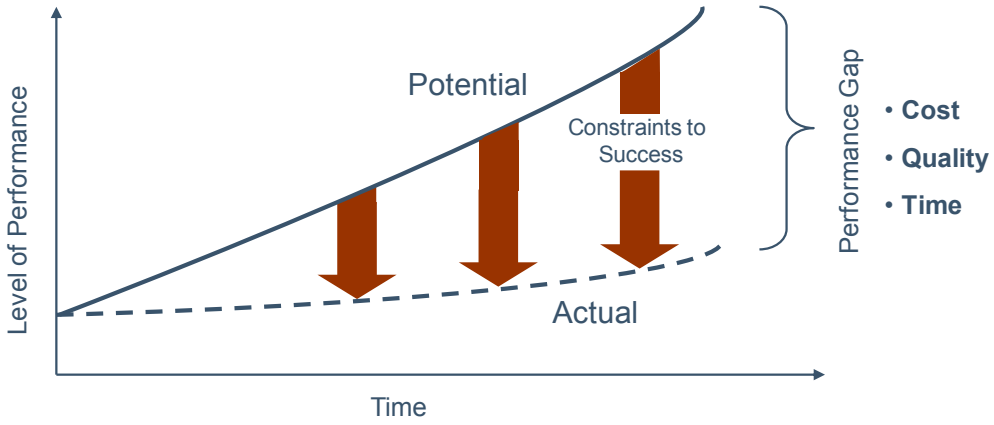


Figure 5. Purpose of benchmarking is to close the gap between actual performance and potential performance.

[20] Obtaining this document through downloading requires registration. However, registration is free.

Glossary

Adjusted function point count (AFP)	The unadjusted function point count multiplied by the value adjustment factor [ISO 2003a].
Application	A cohesive collection of automated procedures and data supporting a business objective [ISO 2003a].
Application software	Software designed to help users perform particular tasks or handle particular types of problems, as distinct from software that controls the computer itself [ISO 2004b].
Architectural design phase	The life-cycle phase in which a system's general architecture is developed, thereby fulfilling the requirements laid down by the software requirements document and detailing the implementation plan in response to it [ISO 2007].
Artifact	Any piece of software (i.e., models/descriptions) developed and used during software development and maintenance. Examples are requirements specifications, architecture and design models, source and executable code (i.e., programs), configuration directives, test data, test scripts, process models, project plans, various documentation etc. [Conradi 2003].
Assessment process	A determination of the extent to which the organization's standard processes contribute to the achievement of its business goals and to help the organization focus on the need for continuous process improvement [ISO 2004a].
Benchmark	A measured, best-in-class achievement; a reference or measurement standard for comparison; this performance level is recognized as the standard of excellence for a specific business process [APQC 2008b].
Benchmarking	The process of identifying, learning, and adapting outstanding practices and processes from any organization, anywhere in the world, to help an organization improve its performance. Benchmarking gathers the tacit knowledge—the know-how, judgments, and enablers—that explicit knowledge often misses [APQC 2008b].
Logical line of code (LLC)	A single software instruction, having a defined beginning and ending independent of any relationship to the physical lines on which it is recorded or printed. Logical source statements are used to measure software size in ways that are independent of the physical formats in which the instructions appear [Park 1992].

Code	In software engineering, computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler, or other translator [ISO 2007].
Physical line of code	A single line of source code. Note that a logical line of code may consist of multiple physical lines of code [Park 1992].
Blank lines	Lines in a source listing or display that have no visible textual symbols [Park 1992].
Comment	Textual strings, lines, or statements that have no effect on compiler or program operations. Usually designated or delimited by special symbols. Omitting or changing comments has no effect on program logic or data structures [Park 1992].
Commercial-off-the-shelf (COTS)	Software defined by a market-driven need, commercially available, and whose fitness for use has been demonstrated by a broad spectrum of commercial users [ISO 2006a].
Computer instruction	A statement in a programming language, specifying an operation to be performed by a computer and the addresses or values of the associated operands [ISO 2007].
COTS	Commercial-off-the-shelf [ISO 2000c].
CPM	Counting Practices International Standard [ISO 2005a].
Data	A representation of facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automatic means [ISO 2007].
Data provider	An individual or organization that is a source of data [ISO 2002a].
Defect	A problem which, if not corrected, could cause an application to either fail or to produce incorrect results [ISO 2003a].
Design	The process of defining the software architecture, components, modules, interfaces, and data for a software system to satisfy specified requirements [ISO 2007].
Design architecture	An arrangement of design elements that provides the design solution for a product or life-cycle process intended to satisfy the functional architecture and the requirements baseline [IEEE 1998a].
Design phase	The period in the software life cycle during which definitions for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements [ISO 2007].
Detailed design	The process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented [ISO 2007].

Detailed design description	A document that describes the exact detailed configuration of a computer program [ISO 2007].
Detailed design phase	The software development life cycle phase during which the detailed design process takes place, using the software system design and software architecture from the previous phase (architectural design) to produce the detailed logic for each unit such that it is ready for coding [ISO 2007].
Development	The specification, construction, testing, and delivery of a new information system [ISO 2003a].
Development project	A project in which a completely new application is realized [ISO 2005a].
Direct delivered team hours	Team hours that directly contribute to defining or creating outputs (source statements, function points, documents, etc.) that are delivered to the customer.
Direct non-delivered team hours	Direct team hours resulting in production of outputs (source statements, function points, documents, etc.) that are *not* delivered with the final product.
Documentation	A collection of documents on a given subject ; any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results; the process of generating or revising a document [ISO 2007].
Domain	A distinct scope, within which common characteristics are exhibited, common rules observed, and over which a distribution transparency is preserved [ISO 2003b].
Enhancement	The modification of an existing application [ISO 2003a]. The activities carried out for an application that change the specifications of the application and that also usually change the number of function points as a result [ISO 2005a].
Enhancement project	A project in which enhancements are made to an existing application [ISO 2005a].
Enterprise	A company, business, firm, partnership, corporation, or governmental agency. An organization may be involved in several enterprises and an enterprise may involve one or more organizations [PMI 2004].
Environment	The circumstances, objects, and conditions that surround a system to be built [IEEE 1988b].

Expert judgment	Judgment provided based upon expertise in an application area, knowledge area, discipline, industry, etc. as appropriate for the activity being performed. Such expertise may be provided by any group or person with specialized education, knowledge, skill, experience, or training, and is available from many sources, including: other units within the performing organization; consultants; stakeholders, including customers; professional and technical associations; and industry groups [PMI 2004].
Function point (FP)	A measure that represents the functional size of application software [ISO 2003a].
Function point analysis (FPA)	A standard method for measuring software development and maintenance from the customer's point of view [ISO 2003a].
Function point count	The function point measurement of a particular application or project [ISO 2003a].
Functional requirements	Description of what the system, process, or product/service must do in order to fulfill the user requirements.
Indicator	A measure that provides an estimate or evaluation of specified attributes derived from a model with respect to defined information needs [ISO 2005b]
Installation phase	The period of time in the software life cycle during which a software product is integrated into its operational environment and tested in this environment to ensure that it performs as required [ISO 2007].
Life cycle	Evolution of a system, product, service, project or other human-made entity from conception through retirement [ISO 12207].
Logical line of code (LLC)	Source statement that measures software instructions independently of the physical format in which they appear. Synonym is logical source statement.[IEEE 1992].
Maintenance	The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment [ISO 2007].
Maintenance enhancement	A modification to an existing software product to satisfy a new requirement. There are two types of software enhancements: adaptive and perfective. A maintenance enhancement is not a software correction [IEEE 2006a].
Maintenance project	A software development project described as maintenance to correct errors in an original requirements specification, to adapt a system to a new environment, or to enhance a system [ISO 2007].

Measure	A variable to which a value is assigned as the result of measurement [ISO 2005b].
Measurement	The act or process of assigning a number or category to an entity to describe an attribute of that entity[IEEE 1994].
Measurement method	A logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale [ISO 2005b].
Organization	A group of persons organized for some purpose or to perform some type of work within an enterprise [PMI 2004].
Previously developed software	Software that has been produced prior to or independent of the project for which the plan is prepared, including software that is obtained or purchased from outside sources [IEEE 1994].
Process	A set of interrelated actions and activities performed to achieve a specified set of products, results, or services [PMI 2004].
Process improvement	Actions taken to change an organization's processes so that they more effectively and/or efficiently meet the organization's business goals [ISO 2004a].
Product	A complete set of computer programs, procedures and associated documentation and data designed for delivery to a user [ISO 1999].
Productivity	The ratio of work product to work effort (ISO/IEC 20926:2003 - Software Engineering) [ISO 2003a].
Project	A temporary endeavor undertaken to create a unique product, service, or result [PMI 2004].
Project life cycle	A collection of generally sequential project phases whose name and number are determined by the control needs of the organization or organizations involved in the project. A life cycle can be documented with a methodology [PMI 2004].
Project Management Body of Knowledge (PMBOK)	An inclusive term that describes the sum of knowledge within the profession of project management. As with other professions, such as law, medicine, and accounting, the body of knowledge rests with the practitioners and academics that apply and advance it [PMI 2004].

Project phase	A collection of logically related project activities, usually culminating in the completion of a major deliverable. Project phases (also called phases) are mainly completed sequentially, but can overlap in some project situations. Phases can be subdivided into subphases and then components; this hierarchy, if the project or portions of the project are divided into phases, is contained in the work breakdown structure. A project phase is a component of a project life cycle. A project phase is not a project management process group [PMI 2004].
Schedule	The planned dates for performing schedule activities and the planned dates for meeting schedule milestones [PMI 2004].
Project team	All the project team members, including the project management team, the project manager, and, for some projects, the project sponsor [PMI 2004].
Project team members	The persons who report either directly or indirectly to the project manager, and who are responsible for performing project work as a regular part of their assigned duties [PMI 2004].
Quality	The degree to which a system, component, or process meets specified requirements; the degree to which a system, component, or process meets customer or user needs or expectations [ISO 2007].
Requirement	A condition or capability needed by a user to solve a problem or achieve an objective [ISO 2007].
Requirements phase	The period of time in the software life cycle during which the requirements for a software product are defined and documented [ISO 2007].
Reusability	The degree to which an asset can be used in more than one software system, or in building other assets [IEEE 1999].
Reusable software product	A software product developed for one use but having other uses, or one developed specifically to be usable on multiple projects or in multiple roles on one project. Examples include, but are not limited to, COTS software products, acquirer-furnished software products, software products in reuse libraries, and preexisting developer software products. Each use may include all or part of the software product and may involve its modification. This term can be applied to any software product (for example, requirements, architectures), not just to software itself [IEEE 1998b].
Reuse	Building a software system at least partly from existing pieces to perform a new application [ISO 2007].

Reused source statement	Unmodified source statement obtained for the product from an external source [IEEE 1992].
Sizing	The process of estimating the amount of computer storage or the number of source lines required for a software system or component [ISO 2007].
SLCP	Software Life Cycle Processes [ISO 2004d].
SLOC, Source Lines of Code	The number of lines of programming language code in a program before compilation [ISO 2000b].
Software	Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system; for example, command files, job control language; includes firmware, documentation, data, and execution control statements [ISO 2007].
Software design	The use of scientific principles, technical information, and imagination in the definition of a software system to perform pre-specified functions with maximum economy and efficiency [ISO 2007].
Software life cycle (SLC)	The period of time that begins when a software product is conceived and ends when the software is no longer available for use [ISO 2007].
Software maintenance	The totality of activities required to provide cost-effective support to a software system [ISO 2006a].
Software product	The set of computer programs, procedures, and possibly associated documentation and data [ISO 2007].
Software project	The set of work activities, both technical and managerial, required to satisfy the terms and conditions of a project agreement. A software project should have specific starting and ending dates, well-defined objectives and constraints, established responsibilities, and a budget and schedule. A software project may be self-contained or may be part of a larger project. In some cases, a software project may span only a portion of the software development cycle. In other cases, a software project may span many years and consist of numerous subprojects, each being a well-defined and self-contained software project [IEEE 1998c].
Software project life cycle (SPLC)	The portion of the entire software life cycle applicable to a specific project; it is the sequence of activities created by mapping the activities of IEEE Std 1074 onto a selected software project life-cycle model (SPLCM) [IEEE 2006b].

Software requirement	A software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document [ISO 2007].
Software requirements phase	The software development life-cycle phase during which the requirements for a software product, such as functional and performance capabilities, are defined, documented, and reviewed [ISO 2007].
Software testing	The dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior [ISO 2005c].
Source code	Computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler, or other translator; a source program is made up of source code [ISO 2007].
SPLC	Software project life cycle [IEEE 2006b].
Staff-hour	An hour of effort expended by a member of the project staff [IEEE 1992].
Statement	In a programming language, a meaningful expression that defines data, specifies program actions, or directs the assembler or compiler [ISO 2007].
Subtype	A subset of a data type, obtained by constraining the set of possible values of the data type [ISO 2007].
Team member	All the project team members, including the project management team, the project manager and, for some projects, the project sponsor. Synonym is *project team member* [PMI 2004].
Test	An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component [ISO 2007].
Test case	A documented instruction for the tester that specifies how a function or a combination of functions shall or should be tested [ISO 1994].
Test phase	The period of time in the software life cycle during which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied [ISO 2007].
Unadjusted function point count (UFP)	The measure of the functionality provided to the user by the project or application [ISO 2003a].

Use case	In UML, a complete task of a system that provides a measurable result of value for an actor [ISO 2007].
Use case specification	A document that describes a use case; a use case specification's fundamental parts are the use case name, brief description, precondition, basic flow, postcondition, and alternate flow [ISO 2007].
User requirements	Description of the set of user needs for the software [ISO 2006b].
Web page	A digital multimedia object as delivered to a client system. A web page may be generated dynamically from the server side, and may incorporate applets or other elements active on either the client or server side [IEEE 2000].

References

URLs are valid as of the publication date of this document.

[Albrecht 1979]
Albrecht, Allan J. "Measuring Application Development Productivity." *Proceedings of the SHARE/GUIDE IBM Applications Development Symposium.* Monterey, CA, Oct. 1979.

[AHD 2006]
Editors of the American Heritage Dictionaries. *The American Heritage Dictionary of the English Language, 4th Ed.* Houghton Mifflin, 2006.

[ASQ 2007]
American Society for Quality. *Quality Glossary*. http://www.asq.org/glossary (2007).

[APQC 1993]
American Productivity & Quality Center. *The Benchmarking Management Guide.* Productivity Press, 1993.

[APQC 2008a]
American Productivity & Quality Center. *Benchmarking: Leveraging Best-Practice Strategies.* http://www.apqc.org/portal/apqc/ksn?paf_gear_id=contentgearhome&paf_dm=full&pageselect=include&docid=112421 (1994-2008).

[APQC 2008b]
American Productivity & Quality Center. *Glossary of Benchmarking Terms.* http://www.apqc.org/portal/apqc/ksn?paf_gear_id=contentgearhome&paf_dm=full&pageselect=detail&docid=119519 (1994-2008).

[Basili 1994]
Basili, V., Caldiera, G., & Rombach, H. D. "The Goal Question Metric Approach," 528-53. *Encyclopedia of Software Engineering*, John Wiley & Sons, Inc., 1994.

[Breyfogle 2003]
Breyfogle, Forrest W., III. *Implementing Six Sigma: Smarter Solutions® Using Statistical Methods,* 2nd ed. John Wiley & Sons, 2003.

[Camp 1989]
Camp, Robert C. *Benchmarking: The Search for Industry Best Practices That Lead to Superior Performance.* ASQC Quality Press, 1989.

[Camp 1995]
Camp, Robert C. *Business Process Benchmarking: Finding and Implementing Best Practices.* ASQC Quality Press, 1995.

[Camp 1998]
Camp, Robert C. *Global Cases in Benchmarking: Best Practices from Organizations Around the World.* ASQ Quality Press, 1998.

[Chrissis 2006]
Chrissis, M. B., Konrad, M., & Shrum, S. *CMMI: Guidelines for Process Integration and Product Improvement*, 2nd ed. New York: Addison-Wesley, 2006.

[Conradi 2003]
Conradi, Reidar. *Software Engineering Mini Glossary*. http://www.idi.ntnu.no/grupper/su/publ/ese/se-defs.html (2003).

[David 2008]
David Consulting Group. *Performance Benchmarking.* http://www.davidconsultinggroup.com/measurement/benchmarking.aspx (2008).

[Deming 1986]
Deming, W. Edwards. *Out of the Crisis.* MIT Center for Advanced Engineering Study, 1986 (ISBN: 0911379010).

[Felici 2003]
Felici, Massimo. *A Formal Framework for Requirements Evolution.* http://www.dirc.org.uk/publications/techreports/papers/20.pdf (2003).

[Florac 1992]
Florac, William, A. *Software Quality Measurement: A Framework for Counting Problems and Defects.* (CMU/SEI-92-TR-022, ADA258556). Software Engineering Institute, Carnegie Mellon University, 1992. http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.022.html

[Galorath 2008]
Galorath, Inc. *SEER Project Management Tool Overview.* http://www.galorath.com/index.php/products/ (2008).

[Glass 1995]
Glass, R. L. & Vessey, Iris. "Contemporary Application Domain Taxonomies." *IEEE Software*, (July 1995).

[IEEE 1988a]
The Institute of Electrical and Electronics Engineers. *IEEE Std 982.1-1988—IEEE Standard Dictionary of Measures to Produce Reliable Software.* The Institute of Electrical and Electronics Engineers, 1988. http://ieeexplore.ieee.org/iel1/2752/1441/00035033.pdf

[IEEE 1988b]
The Institute of Electrical and Electronics Engineers. *IEEE 1362—IEEE Guide for Information Technology—System Definition—Concept of Operations (ConOps)Document.* The Institute of Electrical and Electronics Engineers, 1988.
http://ieeexplore.ieee.org/iel4/6166/16486/00761853.pdf

[IEEE 1992]
The Institute of Electrical and Electronics Engineers. *IEEE Std 1045-1992—Standard for Software Productivity Metrics*. The Institute of Electrical and Electronics Engineers, 1992.
http://ieeexplore.ieee.org/iel1/2858/5527/00211732.pdf?arnumber=211732

[IEEE 1994]
The Institute of Electrical and Electronics Engineers. *IEEE Standard for Software Safety Plans.* The Institute of Electrical and Electronics Engineers, 1984.
http://ieeexplore.ieee.org/iel1/3257/9808/00467427.pdf

[IEEE 1998a]
The Institute of Electrical and Electronics Engineers. *IEEE 1220-1998—IEEE Standard for Application and Management of the Systems Engineering Process.* The Institute of Electrical and Electronics Engineers, 1998. http://www.techstreet.com/cgi-bin/detail?product_id=22369

[IEEE 1998b]
The Institute of Electrical and Electronics Engineers. *IEEE Std 1012-1998—IEEE Standard for Software Verification and Validation.* The Institute of Electrical and Electronics Engineers, 1998.
http://ieeexplore.ieee.org/iel4/5672/15185/00702332.pdf

[IEEE 1998c]
The Institute of Electrical and Electronics Engineers. *IEEE Std 1058-1998—IEEE Standard for Software Project Management Plans.* The Institute of Electrical and Electronics Engineers, 1998.
http://ieeexplore.ieee.org/iel4/5978/16012/00741937.pdf

[IEEE 1999]
The Institute of Electrical and Electronics Engineers. *IEEE 1517-1999—IEEE Standard for Information Technology - Software Life Cycle Processes—Reuse Processes*. The Institute of Electrical and Electronics Engineers, 1999.
http://www.techstreet.com/cgi-bin/detail?product_id=224066

[IEEE 2000]
The Institute of Electrical and Electronics Engineers. *IEEE Std 2001™-2002—IEEE Recommended Practice for the Internet—Web Site Engineering, Web Site Management, and Web Site Life Cycle*. The Institute of Electrical and Electronics Engineers, 2000. http://ieeexplore.ieee.org/iel5/8449/26606/01185571.pdf

[IEEE 2006a]
The Institute of Electrical and Electronics Engineers. *IEEE 14764-2006. Software Engineering—Software Life Cycle Processes—Maintenance.* The Institute of Electrical and Electronics Engineers, 2006. http://ieeexplore.ieee.org/iel5/11168/35960/01703974.pdf?tp=&isnumber=35960&arnumber=1703974

[IEEE 2006b]
The Institute of Electrical and Electronics Engineers. IEEE 1074-2006—IEEE Standard for Developing a Software Project Life Cycle Process. The Institute of Electrical and Electronics Engineers,2006. http://www.techstreet.com/cgi-bin/detail?product_id=1277365

[IFPUG 2005]
IFPU\G Counting Practices Manual. International Function Point Users Group. http://www.ifpug.org/publications/manual.htm, 2005.

[ISBSG 2006]
Glossary of Terms. International Software Benchmarking Standards Group (ISBSG). Glossary of Terms, v5.9.1. http://www.isbsg.org/isbsg.nsf/weben/Glossary%20of%20Terms (2006).

[ISBSG 2008]
International Software Benchmarking Standards Group (ISBSG). *Software Development and Enhancement Repository.* http://www.isbsg.org/Isbsg.Nsf/weben/Development%20&%20Enhancement (2008).

[Isixsigma 2007]
iSixSigma.*iSixSigma Dictionary*. http://www.isixsigma.com/dictionary/Productivity-523.htm (2007).

[ISO 1994]
International Organization for Standardization. *ISO/IEC 12119:1994—Information technology—Software packages—Quality requirements and testing.* http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=1308 (1994).

[ISO 1995]
International Organization for Standardization. *ISO/IEC 12207:1995—Information technology—Software life cycle processes.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21208 (1995).

[ISO 1999]
International Organization for Standardization. *ISO/IEC 15910:1999—Information technology—Software user documentation process.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29509 (1999).

[ISO 2000]
International Organization for Standardization. *ISO 9001:2000 Quality Management Systems–Requirements.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21823 (2000).

[ISO 2002]
International Organization for Standardization. *ISO/IEC 15939:2002—Software engineering—Software measurement process.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29572 (2002).

[ISO 2002b]
International Organization for Standardization. *ISO/IEC 20968:2002—Software engineering—Mk II Function Point Analysis—Counting Practices Manual.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35603 (2002).

[ISO 2003a]
International Organization for Standardization. *ISO/IEC 20926:2003—Software engineering—IFPUG 4.1 Unadjusted functional size measurement method—Counting practices manual*.
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35582 (2003).

[ISO 2003b]
International Organization for Standardization. *ISO/IEC 19500-2:2003—Information technology—Open Distributed Processing—Part 2: General Inter-ORB Protocol (GIOP)/Internet Inter-ORB Protocol (IIOP).*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32619 (2003).

[ISO 2004a]
International Organization for Standardization. *ISO/IEC 15504: Information technology—Process assessment.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38932 (2004).

[ISO 2004b]
International Organization for Standardization. *ISO/IEC 18019:2004—Software and system engineering—Guidelines for the design and preparation of user documentation for application software.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=30804 (2004).

[ISO 2004c]
International Organization for Standardization. *ISO/IEC 90003:2004—Software engineering—Guidelines for the application of ISO 9001:2000 to computer software.*
http://www.iso.org/iso/catalogue_detail?csnumber=35867 (2004).

[ISO 2004d]
International Organization for Standardization. *ISO SO/IEC TR 9126-4:2004—Software engineering—Product quality—Part 4: Quality in use metrics.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39752 (2004).

[ISO 2005a]
International Organization for Standardization. *ISO/IEC 24570:2005—Software engineering—NESMA functional size measurement method version 2.1—Definitions and counting guidelines for the application of Function Point Analysis.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37289 (2005).

[ISO 2005b]
International Organization for Standardization. *ISO/IEC 25000:2005—Software Engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35683 (2005).

[ISO 2005c]
International Organization for Standardization. *ISO/IEC TR 19759:2005—Software Engineering—Guide to the Software Engineering Body of Knowledge (SWEBOK).*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33897 (2005).

[ISO 2006a]
International Organization for Standardization. *ISO/IEC 25051:2006—Software engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing*.
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37457 (2006).

[ISO 2006b]
International Organization for Standardization. *ISO/IEC 14143-6:2006—Information technology—Software measurement—Functional size measurement—Part 6: Guide for use of ISO/IEC 14143 series and related International Standards.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39499 (2006).

[ISO 2007]
International Organization for Standardization. *ISO/IEC WD 24765—Systems and software engineering vocabulary.*
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50518 (2007).

[Juran 1998]
Juran, Joseph M., co-ed. & Godfrey, Edward G., co-ed. *Juran's Quality Handbook,* 5th ed. New York: McGraw-Hill, 1998 (ISBN: 007034003X).

[Kaplan 1992]
Kaplan, R.S. & Norton D.P. "The Balanced Scorecard Measures That Drive Performance." *Harvard Business Review* (Jan-Feb): 71-80.

[Karlöf 1993]
Karlöf, Bengt & Östblom, Svante. *Benchmarking: A Signpost of Excellence in Quality and Productivity.* John Wiley & Sons, 1993.

[Kasunic 2006]
Kasunic, Mark. "Performance Benchmarking Consortium," *NDIA CMMI Technology and Users Conference*. Denver, CO, 2006.

[McGarry 2001]
McGarry John, Card David, Jones Cheryl, Layman Beth, Clark Elizabeth, Dean Joseph, & Hall, Fred. *Practical Software Measurement: Objective Information for Decision Makers.* Addison-Wesley Professional, 2001.

[OGC 2007]
Office of Government Commerce, United Kingdom. *IT Infrastructure Library.*
http://www.itil-officialsite.com/home/home.asp (2007).

[OMB 2003]
Office of Management and Budget. *DLA Commercial Activities Guidebook for OMB Circular A-76 dated May 29, 2003.*
http://www.dla.mil/j-3/a-76/A-76Guidebook29May04AppendA.html (2003).

[Park 1992]
Park, Robert E. *Software Size Measurement: A Framework for Counting Source Statements.* (CMU/SEI-92-TR-020, ADA258304). Software Engineering Institute, Carnegie Mellon University, 1992. http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.020.html

[Park 1996]
Park, Robert E., Goethert, Wolfhart B., Florac, & William A. *Goal-Driven Software Measurement–A Guidebook. (*CMU/SEI-96-HB-002, ADA313946). Software Engineering Institute, Carnegie Mellon University, 1996.
http://www.sei.cmu.edu/publications/documents/96.reports/96.hb.002.html

[Paulk 1993]
Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth,& Weber, Charles V. *Capability Maturity Model℠ for Software, Version 1.1.* (CMU/SEI-93-TR-024, ADA263403). Software Engineering Institute, Carnegie Mellon University, 1993.
http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.024.html

[PMI 2004]
Project Management Institute. *A Guide to the Project Management Body of Knowledge, Third Edition*. Project Management Institute, 2004 (ISBN-10: 193069945X, ISBN-13: 978-1930699458).

[Price 2008]
PRICE Systems. *Price Research.* http://www.pricesystems.com/products/price_research.asp (2008).

[Putnam 2005]
Putnam, Lawrence H., Putnam, Douglas T., & Beckett, Donald M. "A Method for Improving Developers' Software Size Estimates." *CrossTalk* (April 2005): 16-18.
http://stsc.hill.af.mil/crosstalk/2005/04/0504Putnam.pdf

[Pyzdek 2003]
Pyzdek, Thomas. *The Six Sigma Handbook: The Complete Guide for Greenbelts, Blackbelts, and Managers at All Levels, Revised and Expanded Edition,* 2nd ed. New York: McGraw-Hill, 2003 (ISBN: 0071372334).

[QSM 2005]
QSM. *QSM Function Point Programming Languages Table, Version 3.0.*
http://www.qsm.com/FPGearing.html (2005).

[QSM 2008]
QSM. *Project Management for Software Development: Metrics and Measurements.*
http://www.qsm.com/met-analyze_overview.htm (2008).

[Reifer 1990]
Reifer Consultants. *Productivity and Quality Survey.* El Segundo, CA, 1990.

[SEI 2006]
Software Engineering Institute. "Capability Maturity Model Integration (CMMI)."
http://www.sei.cmu.edu/cmmi/ (2006).

[SEI 2008]
Software Engineering. "ISO/IEC 15504."
http://www.sei.cmu.edu/cmmi/faq/15504-faq.html (2008).

[Spendolini 1992]
Spendolini, Michael J. *The Benchmarking Book.* AMACON, American Management Association, 1992.

[SPR 2008]
SPR. *Benchmarking & Assessment.* http://www.spr.com/benchmark/default.shtm (2008).

[4SUM 2008]
4SUM Partners. http://northernscope.com/contact_us (2008).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	July 2008	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
A Data Specification for Software Project Performance Measures: Results of a Collaboration on Performance Measurement	FA8721-05-C-0003

6. AUTHOR(S)

Mark Kasunic

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2008-TR-012

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2008-012

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This document contains a proposed set of defined software project performance measures and influence factors that can be used by software development projects so that valid comparisons can be made between completed projects. These terms and definitions were developed using a collaborative, consensus-based approach involving the Software Engineering Institute's Software Engineering Process Management program and service provider and industry experts in the area of software project performance measurement. This document will be updated over time as feedback is obtained about its use.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Operational definitions, data specification	99

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL