SEC BOOKS

Ver.1.0

ESQR

Embedded System development Quality Reference guide

Written and edited by
Software Engineering Center,
Technology Headquarters,
Information-technology Promotion Agency, Japan

IPA

This document has been published as the English edition of ESQR (Embedded System development Quality Reference) Version 1.0 published by IPA/SEC* in Japan. ESQR provides evaluation metrics for visualizing the quality of software deliverables and work products to ensure systematic quality control of embedded software development.

The purpose of ESQR is to be used as the reference guide for introducing quantitative quality management concepts and practices to organizations and individual projects engaged in embedded software development that do not have satisfactory management system in place to measure and evaluate the quality of their software development.

August 2010
Software Engineering Center,
Information-technology Promotion Agency, Japan

* Software Engineering Center, Technology Headquarters, Information-technology Promotion Agency, Japan

Preface

The leaves on the trees in the Rikugien Garden, as seen from here on the 16th floor of the Bunkyo Green Coat Center Office, are again taking on their autumn hues of red and yellow. For us, this is the fourth time that we have been able to enjoy this beautiful view. Over the past four years, the Software Engineering Center (SEC) has been preparing the Embedded System Development Exemplar Reference (ESxR) series of guidebooks with the goal of cultivating the skills of Japan's embedded software developers. Our latest release is of the "Embedded System Development Quality Reference" (ESQR), the fourth guide of the series, which aims at assuring high-quality embedded system design. The situations surrounding systems and software are changing quickly and the enabling technologies are advancing quickly. On the other hand, the fundamentals of system and software development as "creative design and manufacturing" remain as before. The origins of modern science go back to when Sir Isaac Newton observed an apple fall from a tree, after which he went on to derive laws describing how it happened, and then the science and engineering based on it have developed to give us the creative design and manufacturing on which we depend today.

With this publication of the ESQR, we view system and software development as a kind of creative design and manufacturing and, by returning to basics, analyze the software itself and the work needed to create it objectively, visualizing and observing them to guide engineers to achieve better quality.

For this purpose, this guide uses original concepts throughout, such as system characteristics profiling to attain the quality required for the target system, project characteristics profiling to determine the characteristics of a project to develop the system, and ST-SEISMIC that is used to evaluate the influence of a system trouble. In addition, to set an actual quality goal, this guide provides a range of metrics (evaluation metrics) and corresponding reference values to measure and visualize quality, together with tips to improve software quality, some of which are unique.

These are defined as a means of materializing the concepts introduced in this guide and to "quantitatively control software quality at the development stage." We hope that you use them as a reference for establishing a quantitative quality control method. When reading this guide, you should select helpful reference information, adapt it to your own needs, and/or review it, to use it as an opportunity to start the discussion and deployment of a quantitative quality control scheme suitable for your own organization, project, and system.

2008 Fall

Embedded Software Engineering Section, SEC, IPA
Masayuki Hirayama, Satomi Yoshizawa, Sayuri Yamaguchi

Acknowledgements

ESQR grew out of many, many hearings, interviews, and questionnaire surveys mainly with SEC's research workers, as well as with many corporate partners and engineers. To create the draft of ESQR, experts from the SEC Software development technical committee and other arenas kindly gave us their valuable opinions and comments from the viewpoint of use in production environments. We would like to thank everybody who cooperated in the creation of this guide.

While this guide was being created, Yutaka Ukon, one of the most important members of ESQR creation group and an SEC's research worker, passed away. Mr. Ukon drew on his rich experience to give us many valuable ideas, opinions, and comments starting from the time of ESQR concept creation, greatly contributing to the completion of this guide.

Contents

Chapter 1

How to Read ESQR

Key to implementing high-quality embedded software is quality establishment in the development process. This "Embedded System Development Quality Reference" (ESQR) summarizes concrete methods of setting and achieving quality target values, which are essential to assuring the quality of an embedded system. This chapter is aimed at first-time readers of ESQR to show how this guide is positioned and how it should be utilized.

1.1 Purpose and Positioning of ESQR

Purpose of ESQR

With the rapid increase in the demand for embedded software over the last few years, the need for higher quality, reliability, and safety has arisen. Conventionally, a range of methods and concepts has been introduced and deployed to implement high-quality software, including well-defined review and testing, as well as the optimization of the software design structure. Besides, definitions of the software quality and measures for evaluating it, have been suggested in several different ways and introduced as quality concepts in the ISO/IEC9126 series. However, it still cannot be said that there are concrete and clear metrics, although there has been much debate on software quality and methodologies, such as "how many reviews and tests are required to ensure high quality" or "what level of quality is required for what system, and what methods should be used to achieve the required level." As a result, it is difficult to define a clear quality goal in an actual production environment, and therefore the concept of the quantitative control of quality (quality control) for achieving the goal is not sufficiently put into practice. This results in ongoing software errors and system troubles.

This guide was written based on this reality of quality control in software development, to hopefully deploy more systematic quality control methods in the embedded software development environment. In this sense, the purpose of this guide is to "visualize the qualities of software deliverables and work using metrics, and guide the reader to an appropriate development method which best fits the required quality level."

Putting this a little more concretely, this guide aims to help the reader to:

- Analyze and define the level quality required by the users of individual products,
- Define the evaluation metrics based upon the analysis result,
- Measure the suitability of the deliverable from each process and the related check work, and
- Link the check results to activities for ensuring quality.

At the start point is the concept of trying to represent "acceptable quality" with concrete values using metrics, rather than describing it abstractly or intuitively.

Positioning of ESQR

SEC has been developing the Embedded System Development Exemplar Reference (ESxR) series of reference guides to facilitate embedded software development. The "Embedded System Development Process Reference" (ESPR) summarizes the standard concepts behind the processes for developing embedded software. The "Embedded System Development Management Reference" (ESMR) describes how to create a development plan, as a first step to facilitating embedded software development project management. And, the "Embedded System Development Coding Reference" (ESCR) summarizes the way in which programs are coded to directly improve program quality.

This "Embedded System Development Quality Reference" (ESQR) proposes metrics for evaluating the sufficiency of the reviews and tests intended to ensure quality when putting the development processes defined in ESPR into practice, as well as for evaluating the qualities of the intermediate deliverables produced by these processes, and describes how each of these metrics should be implemented. ESMR creates development plans as a baseline for project management, including a plan for quality. This guide details the evaluation metrics used in such a quality plan and how to set corresponding target values. Basically, this guide refers to the names of intermediate deliverables such as work processes (e.g., unit test and requirements analysis) and the documents described in ESPR. Refer to ESPR as necessary.

This guide also includes specifications related to the quality evaluation of the source code. For the definitions of concepts such as deviations from coding rules, also refer to the description of the coding practices mentioned in ESCR.

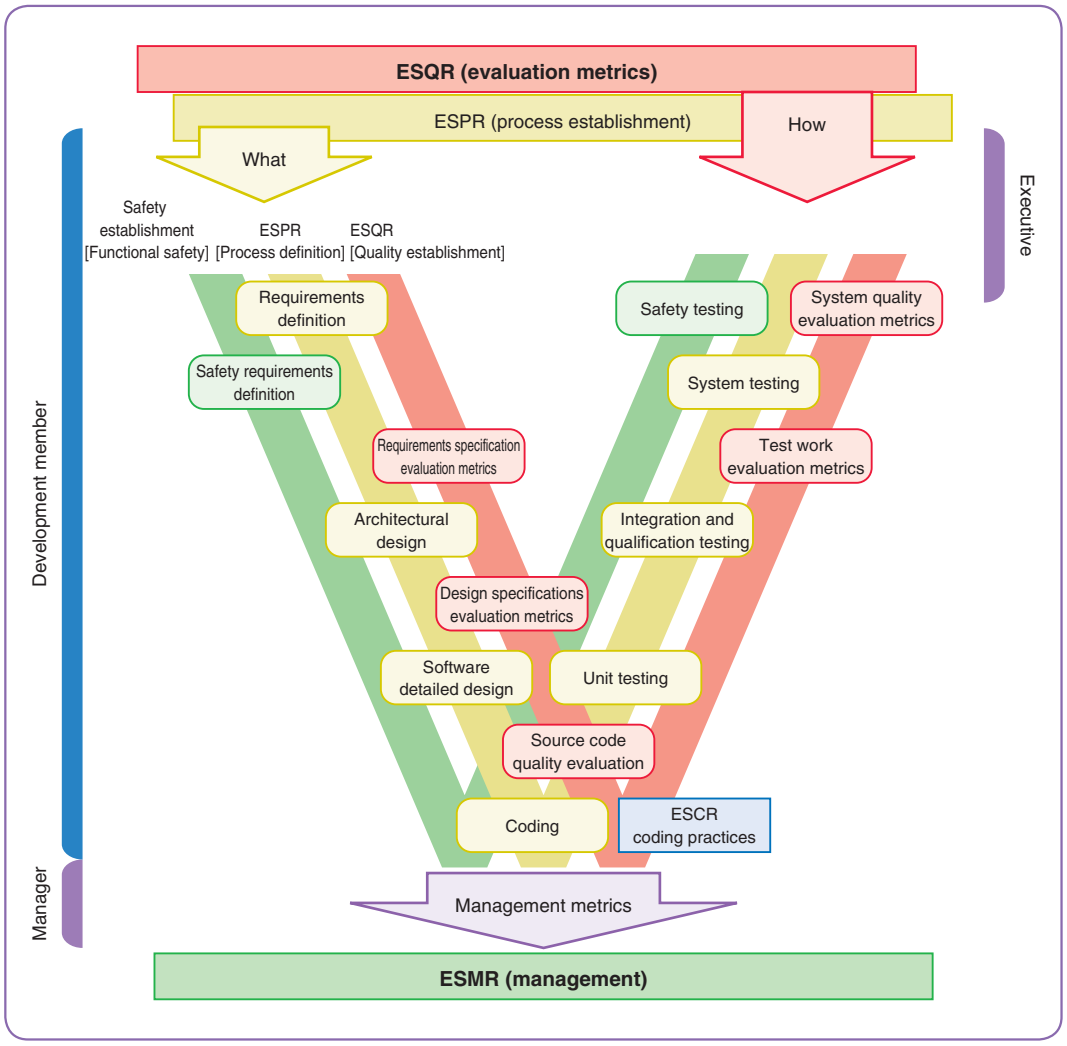


Figure 1-1: Positioning of ESQR

1.2 Embedded System Development Based upon Evaluation Metrics

This guide is centered around the concept of "setting a quantitative goal for software quality using evaluation metrics and quality control to achieve that goal." This section briefly explains this core concept of "evaluation metrics."

What are metrics?

We are surrounded by a huge variety of "things," and very often talk about the "characteristics" of these things. Suppose, for instance, that we think of an "apple." We evaluate the features of an apple by sensing its characteristics, such as its weight, size, color, etc. However, the phrase "large apple" is likely to mean different things to different people. If we were to attempt to determine the price of an apple according to its size, such sense-based evaluation would be very ambiguous, and would not work well. It is much more normal, therefore, to measure the diameter of the apple and represent it numerically, to eliminate personal differences and represent the features of the thing scientifically. Then, apples can be classified into large and medium sizes according to their diameters and a standard value. In this case, "the diameter of the apple" is a metric for evaluating the size of an apple. As such, everything in this world has characteristics, and metrics can be devised to measure and evaluate them.

Another example is "how long a customer is kept waiting when he or she goes to a bank to open an account." In this example, the target of the evaluation is not a "thing." But we nevertheless evaluate banks, saying that "Bank A offers a poor service because it makes us wait a long time, while Bank B offers a good service because it doesn't make us wait." Thus, we unwittingly compare the "bank account opening procedure" which is a non-material entity (service or work quality). Again, whether the service is good or poor varies from one person to another. So, to evaluate the service quality, more objective metrics are adopted, such as the measured average waiting time.

In this way, metrics can be used to measure the characteristics of non-material targets such as work. That is, metrics can be defined as follows:

> **Metrics:** A measure with a scientific basis for representing the characteristics of a material or non-material target (material, work, service, etc.).

Generally, regardless of the target, to determine the characteristics of the target, it is necessary to use a metric to measure some characteristics of the target and compare the measured value to the goal or some empirical value to determine the feature of the target.

Measurement method, units, and precision of metrics

Let's further consider the "diameter of an apple" metric, mentioned above. The diameter of an apple varies depending on where it is measured, as shown in Figure 1-2. For example, the diameter at the section across the center of the sphere of the apple differs from that at a section above the center of the sphere. A metric value that would vary depending on the measurement method would be meaningless. For a metric to be meaningful, therefore, it must be defined together with a valid measurement method.

In addition, the precision of the measurement must also be considered. Using a measure with a precision of 1 cm, values such as approximately 10 or 7 cm can be obtained. But, if we use a measure with a precision of 0.01 cm and obtain values such as 7.13 or 7.14 cm, the difference of 0.01 cm is meaningless. The point of this example is that we have to select a precision that is suitable for the characteristic to be measured with a metric.

In addition, length can be measured in cm, mm, or even inches. It is not strictly necessary to use the metric system to measure the diameter of an apple, but if the measurement units differ from person to person, any comparison of measured diameters will be confusing.

The following summarizes the requirements for the metrics, the units to be used, and the precision, based on the examples mentioned so far.

Metrics: Metrics must be defined with an appropriate measurement method to ensure that similar values are obtained regardless of who performs the measurement.

Metrics precision: Precision suitable to best represent the characteristic of the measurement target.

Metrics unit: The unit for a metric must be unified to a single set, as part of the definition of the metric.

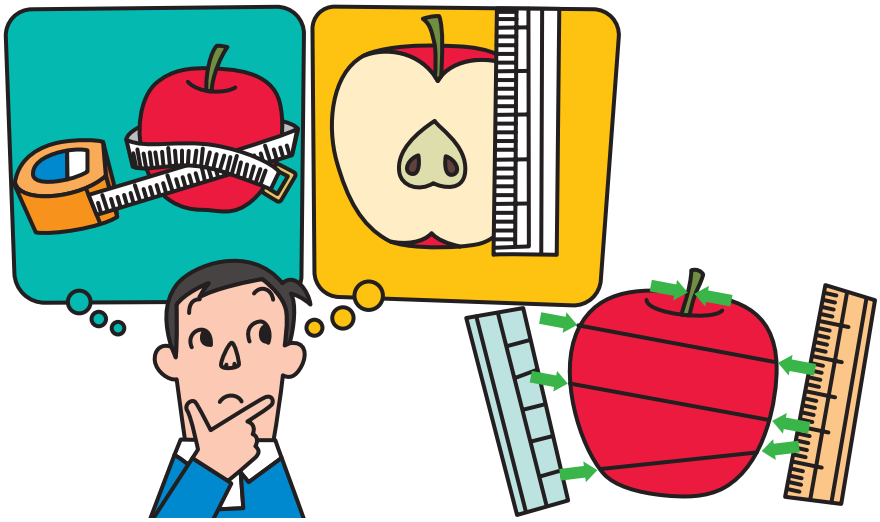


Figure 1-2: Meaning of metrics and measurement

How to use metrics

For what reasons are metrics determined and measured? If we again consider the "diameter of an apple" example, if the diameter of a given apple is larger than the standard value, then we could conclude that that apple is a "good apple for shipment." Or, an apple with a diameter that is much smaller than the standard could be classed as being too small and would be left on the tree to grow. A metric can be used in this way for judging whether the apple can be harvested or shipped. Let's think of another metric, namely, the "thickness of the stem of an apple." We could set the standard thickness to 2 mm and classify several apples into those having thicker stems and those with thinner

stems. Does this classification have any meaning? While we can easily measure the "thickness of the stem of an apple," its thickness probably has no influence on the value of the apple. As seen in this example, metric targets have different characteristics, all of which can be measured. Some metrics, however, although they can be measured, have little or no meaning. Generally, "measuring a metric" assumes that the measured metric value is utilized for some purpose. Therefore, when determining a metric, it is critically important to first clarify "what is required from the use of the metric, or for what purpose the metric is used" and then select the best metric for the purpose.

What is an evaluation metric?

This guide is intended to introduce methods to quantitatively control the characteristics of embedded systems in order to improve the quality of such systems.

The concept of "metrics" as a means of representing the characteristics of "materials" and "works," including software, has already been mentioned. This guide develops this concept to enable the deployment of evaluation metrics to represent "quality," a characteristic of software, and control this characteristic.

The "apple" used in the above examples is a physical object that we can pick up and measure. On the contrary, the embedded software targeted by this guide is very difficult to either hold or observe. That is, the embedded software implemented in products cannot actually be separated from those products, and even in the development environment, software can only be regarded as an object when in the form of an intermediate deliverable, such as the source code or a design document. In this sense, when considering software as the measurement target of a metric, it is very difficult to grasp.

On the other hand, failures have occurred with many commercial embedded systems or some of them are not easy to use in some way, and we often criticize such systems as being of "poor quality." As in this example, software is very difficult product to measure. But we do attach the concept of "quality" to software as well as other material objects. So, this guide clarifies the evaluation metrics applied to software to measure and represent its quality.

In software development, it is important to examine the fundamental factors determining software quality and then aim to control them. For example, to grow a "high-quality apple" with a large diameter, it does not make sense to merely measure the produced good apples with metrics. To obtain "high-quality apples" it is important to

determine whether the processes and farm work, such as "how much sunlight they get," "how much it rains," or "how much they are fertilized" are appropriate, and if not, to take measures such as "fertilize them more." In the case of software, because the final quality is very difficult to determine, such concepts are even more important. That is, in addition to measuring the ultimate software quality, it is necessary to evaluate how much effort was expended or how much necessary work was done during development. In this sense, evaluation metrics can be classified into the following two concepts:

Metrics for measuring the appropriateness of the work done in the process of obtaining the result => Process metrics

Metrics for measuring the quality of the resulting software => Product metrics

Process metrics and product metrics

As mentioned above, this guide deploys two metric concepts, process metrics and product metrics, to measure and evaluate that property of an embedded system that we call "quality."

(1) Process metrics

"The tasks that are required" in each phase of software implementation are described in the "Embedded System Development Process Reference" (ESPR). This guide summarizes the measures for checking if work involved in each process is done appropriately. While many different tasks are involved in software implementation, this guide focuses on ensuring software quality, and considers the metrics for evaluating whether the work central to securing and ensuring quality is done correctly from the aspects of both quality establishment and testing.

In general, the work related to software implementation can be regarded as being the repetition of three work elements, as shown below, "create properly, check, and fix."

- **Create properly:** Analyze the requirements, design the structure, and implement the program.
- **Check:** Check the software (e.g. through reviews) and ensure that it runs normally (e.g. through testing).
- **Fix:** Debug the software and modify the design and documents accordingly.

Among the above, "create properly and fix" has been discussed from many different viewpoints as a software engineering technique. This guide focuses on the second work element, namely, "check the software and ensure that it runs normally."

(2) Product metrics

When developing software used in industrial products, it is very important to measure and evaluate the deliverables created during the development, at appropriate timings, and then adjust (control) the quality according to the goal. This guide provides product metrics to measure and evaluate the deliverables created during development.

As mentioned above, software seems or does not seem to have some kind of form and is a hard-to-grasp "concept." Therefore, software quality is measured by directly measuring the intermediate deliverables created during the development process such as the specifications, design documents, source code, and executable code to evaluate the quality as a physical thing; Product quality.

Reference value for evaluation metrics

The basic idea behind this guide is that the development of embedded system should start with, as shown in Figure 1-3, requirements definition and then proceed through design, coding, and testing. Chapter 3 of this guide presents the metrics used to gradually ensure quality throughout each of these processes. These metrics are presented together with definitions, measurement methods, and guidelines needed to interpret the measured values. As you will see in Chapter 3 and the subsequent chapters of this guide, many of the evaluation metrics presented are selected because measurement is easy and they are not subject to personal differences. Along with these evaluation metrics (definition, measurement method, and interpretation), reference values, taking into consideration the quality level of the target system, are shown for determining validity of measured metric values. These reference values assume new development based upon waterfall-type processes. In actual embedded system implementation, in addition to entirely new system development, existing software assets are often used or reused. In these cases, modify the description according to the individual development conditions, based upon the metrics, reference values, and concepts presented in this guide. Note that the reference values presented for metrics are calculated from the data provided by the bodies approving SEC activities as well as through discussion, and should be used as references, or standards, and customized for your own product development.

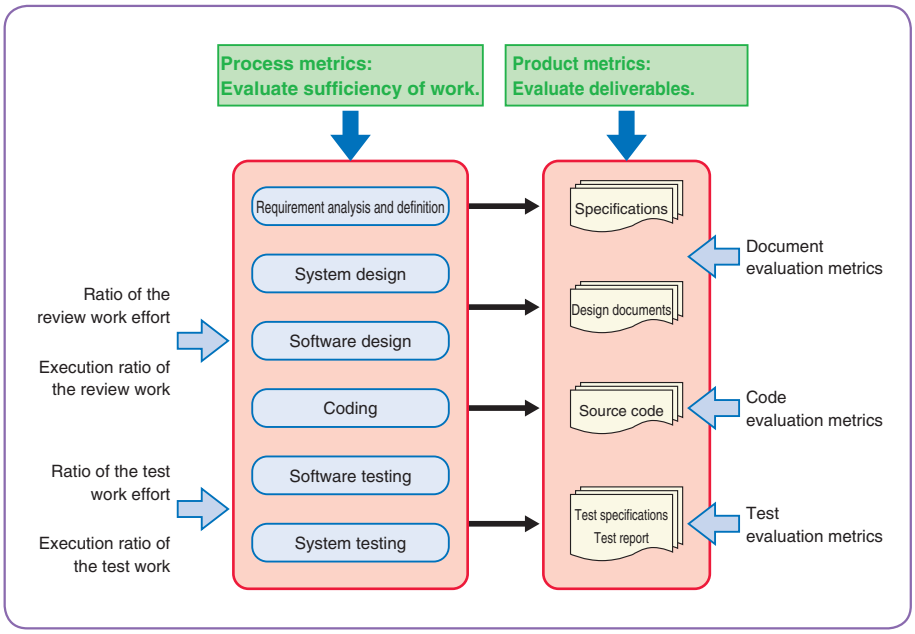


Figure 1-3: Positioning of process metrics and product metrics

1.3 Intended Users, Usage, and Effects of ESQR

Intended development and users

Intended development

This guide is aimed at the development of any embedded software. In an actual development environment, the following situations can be assumed:

- In the development of embedded system to be incorporated into a device, the software is created from scratch.
- Existing design assets may be utilized or reused in the development of the software, or some of the functions may be expanded.
- Partial (general-purpose) function or middleware is developed to implement the embedded software as requested by a business partner or for your own business.

Several development process patterns are possible for such software development, but the basic concept for quantitatively controlling the software quality in the development process remains consistent throughout. Therefore, the ideas or reference metrics presented in this guide may be applied to your own development by adapting them to suit your particular situation. Although the ideas in this guide are intended to be applied to the development of embedded software, they can also be applied to other types of software (component software of an enterprise information system, for example) with some modifications.

Intended users

This guide is aimed at those persons who develop software, manage quality, and who are responsible for the management in companies or projects for defining and using evaluation metrics.

This guide assumes that embedded software development involves the following users:

- Manager or leader who is responsible for managing a development project or development team and who examines and determines actual process control and quality management for individual development issues
- Member of an embedded software development organization who summarizes the basic concepts related to the development process standard or quality for the organization or department, and who supports its deployment
- Member of a support group for an embedded software development organization who indirectly supports the software development (e.g., quality assurance)

Needless to say, comprehension, support, and backup by the management are necessary because costs for process effort are involved in controlling system quality using quantitative evaluation metrics.

Intended usage

This guide summarizes the metrics and work flows for embedded software development that are designed to ensure and improve quality. The guide is intended to be used when preparing quality management for an individual organization or department, or for an individual project.

The preparation of quality management techniques can be assumed for the following scenarios:

- Analysis of product characteristics and definition of a quality goals suitable for the product.
- Determination of the standard to be attained prior to proceeding to the next software process.
- In a scenario where no measurements for software quality management are made, introduce evaluation metrics and perform active quality control (management).

To use this guide, as shown in Figure 1-4, first perform system characteristics profiling and project characteristics profiling, considering the business strategy of your organization or department, product strategy, and organization characteristics, and then reflect the results on the quality goal of the development. In addition, measure the defined evaluation metrics and, according to the results of gap analysis of the measurement result and goal, deploy the necessary methods and techniques to lead the

project in the desired direction. This guide describes system and project characteristics profiling in Chapter 2 and evaluation metrics in Chapter 3. For development and system quality control based upon the measured results, Chapter 4 provides tips on reviews and tests in checklists. For details on more direct methods and techniques such as design and implementation methods for ensuring higher quality, refer to other reference documents.

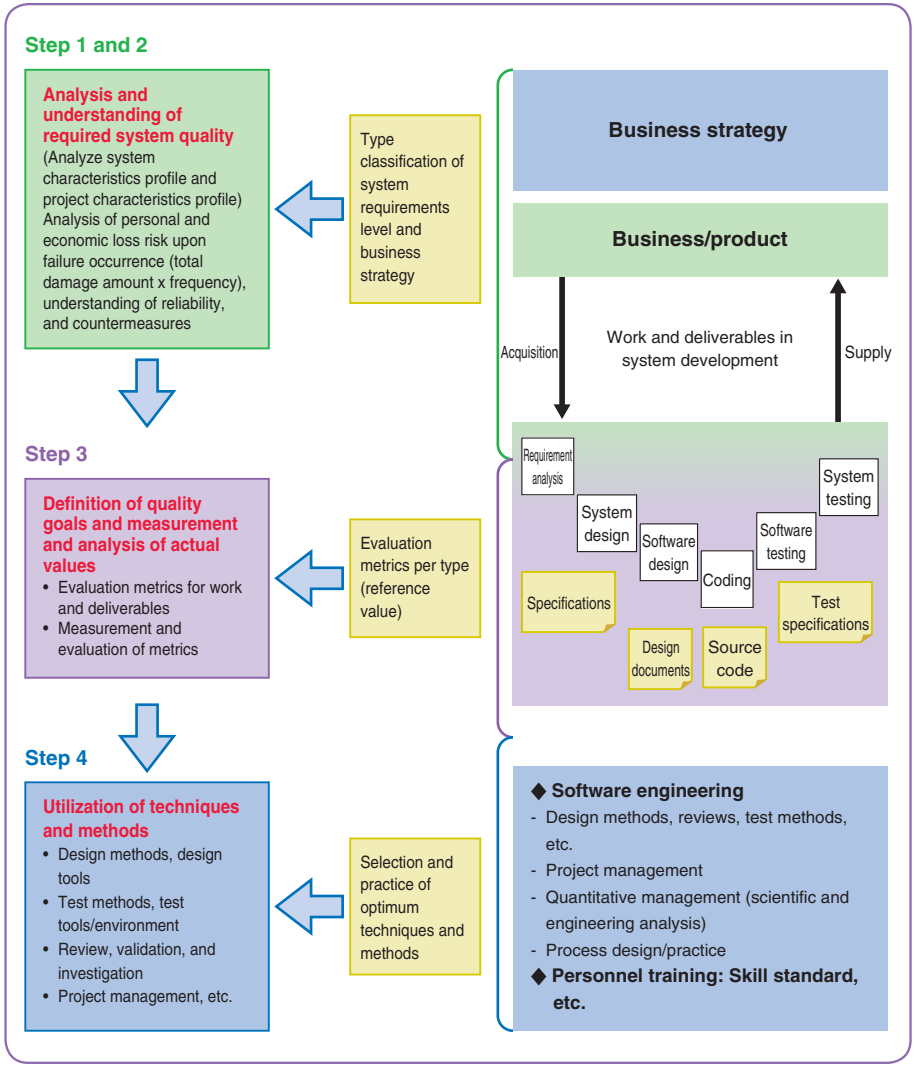


Figure 1-4: Usage flow in this guide

Expected effects

This guide sets out to control quality by applying the following steps, as shown in Figure 1-4.

Step 1: Analyze and understand the required quality, considering the characteristics of the system to be developed,

Step 2: Analyze and understand the characteristics of the development project,

Step 3: Define the quality goals according to the characteristics, and measure and evaluate the metrics in accordance with the actual development, and

Step 4: Perform quantitative quality control including the quality improvement methods required to approach the defined quality target value.

The application of these steps will produce the following effects.

Effects of analyzing the characteristics of the system to be developed (system characteristics profiling)

- You can analyze and consider the appropriate system quality level from the users' viewpoint.
- The quality level of a product may tend to be affected by the circumstances of the developer. By thinking of the product requirements from the user's viewpoint, in terms of both functionality and quality, you will be able to determine which work is necessary for ensuring quality, and which is unnecessary.

Effects of analyzing the characteristics of the project for developing the system (project characteristics profiling)

- You will be able to understand the characteristics of the project from the viewpoint of the system development team and then reflect them on the definition of the quality goal.
- You will gain an opportunity to objectively consider the influence of the project characteristics on quality.

Effects of controlling product quality according to a quality goal corresponding to the quality level

- Establishment of metrics to quantitatively evaluate the work required to ensure quality from the viewpoint of "sufficiency," according to the quality level determined through system and project characteristics profiling. In addition, by defining a goal value (quality target value) for these metrics and developing the product, you will only have to undertake the minimum required amount of work to ensure the desired quality.

1.4 Structure of ESQR

Structure of ESQR

This guide summarizes the work required to quantitatively control embedded system quality, applying a sequence of "system characteristics profiling," "project characteristics profiling," and "evaluation metric definition and procedure." According to the results of quantitative quality measurements, this guide introduces tips on "work required to establish high quality."

This guide consists of the following chapters.

Chapter 1: How to Read ESQR
Chapter 2: Defining Quality Target Values Using System Characteristics Profiling
Chapter 3: Definition and Reference Values for Evaluation Metrics
Chapter 4: Tips for High Quality Establishment

Chapter 1 explains the positioning and purpose of this guide and gives an overview of evaluation metrics, the core concept presented in this guide.

Chapter 2 describes the concept of system characteristics profiling needed to analyze and define the system, as well as the project characteristics profiling needed to evaluate the development project characteristics before using the evaluation metrics. System characteristics profiling systematically analyzes the level that is required for the system and characterizes the system from the user's viewpoints of quality and reliability. Project characteristics profiling focuses on the development project and evaluates the characteristics, so as to evaluate their influence on system quality and reliability.

Chapter 3 explains process metrics and product metrics which evaluate the sufficiency of the works and deliverables, respectively, explains the basic metrics measured for these metrics, and defines the concept and use of each evaluation metric. In addition, this chapter presents the reference values for the evaluation metrics for each system type defined in Chapter 2. The metrics presented in Chapter 3 can be measured easily by any organization. For example, this guide uses Lines of Code (LOC), which is the number

of physical source code lines, to represent the scale of the software. LOC can be easily determined by using a text editor and will not vary, regardless of who measures it.

Chapter 4 describes the checklist to be observed to ensure successful communication, specification and design documents writing, as well as review, and testing. It also provides tips on applying this guide throughout the organization.

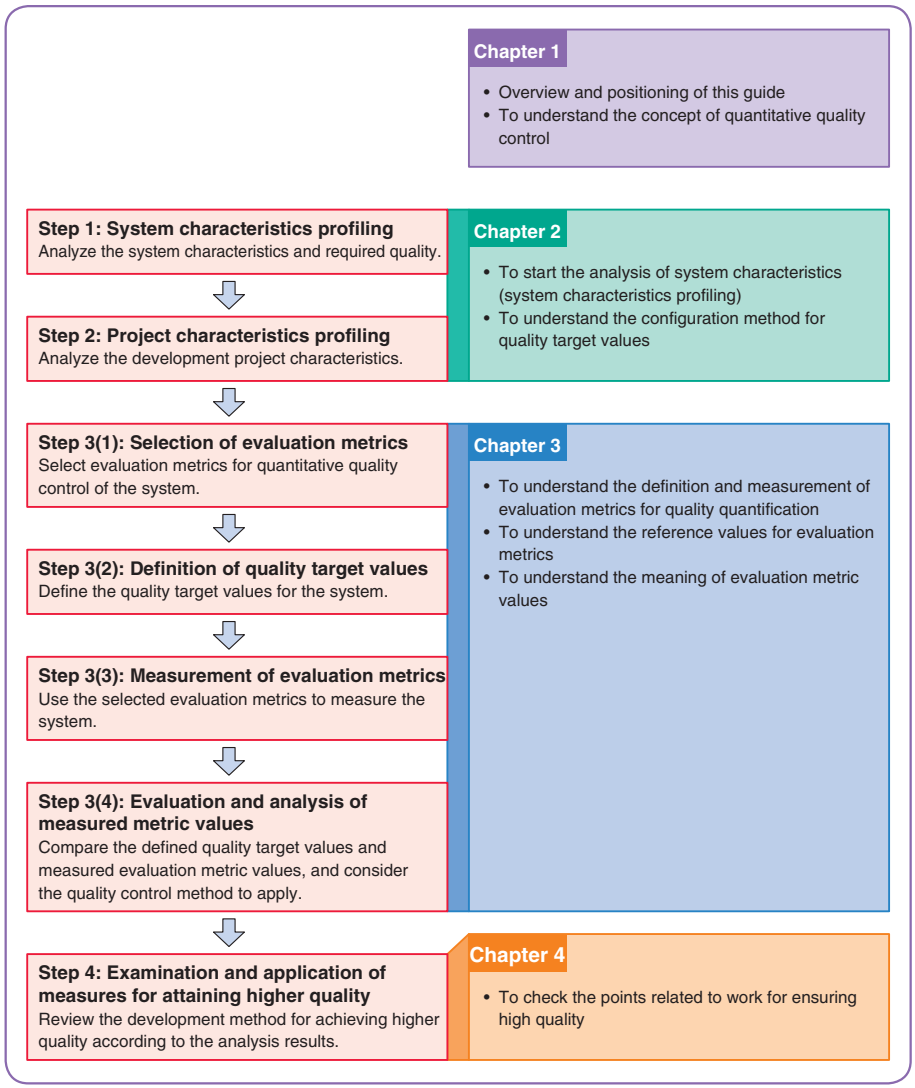


Figure 1-5: Structure of this guide

1.5 Notes on Using ESQR

Use as a reference

This guide summarizes the concept of system characteristics profiling, evaluation metrics definition and reference values, as well as methodology points for quality improvement, based upon quality control concepts using evaluation metrics during the development process, in order to improve the quality of embedded systems. These issues are based on a single concept for high-quality embedded system implementation, and are to be understood as reference information. On this assumption, use this guide by paying attention to the following points. Note that conformity to the contents of this guide does not imply any authentication or approval.

(1) System characteristics profiling concept

The concept of system characteristics profiling explained in this guide is presented from the user's viewpoint for assuring the quality and reliability required for the system. It may not be perfectly compatible with your system, depending on its characteristics. So, fully understand the meanings of this guide before attempting to apply them.

(2) Evaluation metrics definition and reference values as reference information

The concept of evaluation metrics presented in Chapter 3 is just one way of visualizing software quality – many other evaluation metrics, scales, and quality standards have been established around the world. We recommend that you also refer to them and select whichever best fits your development. In addition, the reference values for evaluation metrics presented in Chapter 3 should also be understood as some of the cases. These may take different values according to the development and system conditions and should be used by fully taking individual circumstances and conditions into consideration.

Note that the values presented in this guide (ESQR Ver1.0) were determined by examining data provided by cooperators of SEC during the creation of Ver1.0, and will

be changed to more accurate values in the next version, if necessary.

(3) Do not be a slave to numeric values

This guide provides various metrics and numeric values. The software development industry seems to be very sensitive to such values and tends to jump on any such bandwagon easily. As mentioned at the beginning of this guide, the quantification of software quality is merely a means of developing high-quality software and is not an end in itself. When we interviewed people in the industry at SEC, we often heard "although we've been gathering data, we can't see what it says" or "we don't know how to use the data that we've collected." For most such organizations, their goal is "gathering numeric data." After reading this guide, and before starting an examination of the metrics to quantify quality, thoroughly examine the reason why you are starting the activity.

Notes on using this guide

Observe the following precautions when using this guide.

(1) Referencing for improving quality within a company or development organization

When introducing quantitative quality control to a company or development organization in line with this guide, there is no need to obtain permission from SEC. In addition, there are no restrictions on quoting a part of this guide in an internal rule of a company or other document, or using it in a seminar for internal engineers; you must, however, name this guide to clarify the source.

If you have comments or opinions on such uses, SEC will welcome your feedback.

(2) Quotation in other books, papers, seminar texts, etc.

In such a case, you must ask SEC for permission in advance. When permitted, clarify the source, including the number(s) of the quoted page(s).

(3) Quotation for profit-making seminars, etc.

When quoting or introducing any part of this guide in a profit-making seminar for the general public, you must ask SEC for permission in advance.

Request for feedback to SEC

SEC is a public agency that aims to foster software development skills. And we believe that organizing and publishing documents like this guide will raise the standards of software development skills in Japan. For the metrics and reference values presented in this guide, actual values and feedback from the field are very important. We asked many companies to provide us with data while we were creating this guide, and to improve the quality of the guide as a reference, we need feedback from more organizations. We welcome every reader to provide feedback for actual data related to the metrics presented in this guide, including other metrics data.

1.6 Related Standards

International standards

(1) ISO/IEC9126 (JIS X0129)

This standard defines quality characteristics for software products. Based on the concept that a software product has a total of six aspects including reliability and functionality, this standard defines sub-characteristics for each aspect and alternative characteristics (external and internal characteristics) to represent them. This standard is often referenced in the software arena and can be said to be positioned at the deepest level of understanding regarding software quality.

Part of the product metrics concept presented in Chapter 3 is related to this standard, in particular the aspect of software product quality. In addition, the ratio of code control statement description is related to reliability (complexity), while the ratio of comment line description is related to maintainability. The creation of this guide was based on discussions with this standard in mind.

(2) IEC 61508 (JIS C0508)

An international standard to document functional safety policies related to electrical and electronic computer system (programmable systems). With a system such as a plant in mind, this standard evaluates its functional preparedness in terms of security and safety as a Safety Integrity Level (SIL) and recommends a development method corresponding to the SIL. This guide summarizes the concepts of system characteristics profiling to determine the levels of quality and safety required for the system in an easier way, based on the concept of SIL in this standard. Although not mentioned in the text, the four types of system characteristics profiling described in this guide are very closely related to the SIL levels.

(3) ISO/IEC 15939 (JIS X0141)

This standard summarizes the concepts of software measurement methods and processes. Regarding the flow and framework related to software measurement, this standard summarizes the concepts of the measurement methods for the target and basic measurement quantities, as well as the concepts of derived measurement quantities and metrics, comparing them with the standard values used for decision-making. Regarding these items, this standard also defines the terms related to measurement and evaluation.

This guide references this standard and unifies terms of scale to enable the measurement and evaluation of the target in terms of "metrics" from the viewpoint of familiarity with the field.

Chapter 2

Defining Quality Target Values Using System Characteristics Profiling

To set quality goals for embedded software and actually establish quality, the target values must be set by considering the characteristics of each target embedded system. This chapter introduces the concept of system characteristics profiling, which is required to set quality target values for embedded systems, and describes how to set the evaluation metrics for embedded software based on this concept.

2.1 Concept of Quality Target Value Setting Considering Embedded System Characteristics

Basic guidelines for setting quality target values

We are surrounded by many different embedded systems. Examples include information appliances such as cellular phones and TV sets, home appliance such as fridges and microwave ovens, transportation devices such as automobiles and elevators, as well as the components making up our social infrastructure, such as bank ATMs and station ticket barriers. The complicated controllers used in power plants are another example of an embedded system. When considering the qualities of embedded systems, it would be heavy-handed to group all these systems under the single category of "embedded systems." For example, when a trouble occurs with a cellular phone or information appliance, the user will almost always be inconvenienced, but this trouble is very unlikely to lead to the human cost such as injury to the user. If, on the other hand, a fault occurs in the control system of an automobile, elevator, or aircraft, it may lead to a serious accident, possibly involving human casualties. Thus, regarding the quality that an embedded system is to provide, or is expected to provide as a product, the viewpoint or level greatly depends on the characteristics of the given embedded system. Therefore, when developing an embedded system and establishing quality as part of the development process, it is necessary to consider the characteristics of the target system when setting the quality target value.

Flow of quality target value setting using system characteristics profiling

To set a quality target value in consideration of the characteristics of the target system using system characteristics profiling (SCP), apply the following three-step procedure.

Step 1: System characteristics profiling (SCP)

Consider the system troubles that could possibly occur during system use and operation, and select those scenarios in which the software to embed will be involved, and classify the target system into four system types according to potential economic and human damage and cost.

Step 2: Project characteristics profiling (PCP)

Evaluate the characteristics of the development project, considering the system characteristics, the implementation of the software embedded in the system, and the project characteristics. Use the evaluation result as an adjustment coefficient to apply to the four system types obtained in Step 1 as a reference to set the quality target value in Step 3. In project characteristics profiling, ten factors are checked and converted into the adjustment coefficient mentioned above according to the whole check status. These ten factors are just examples and should be reviewed or supplemented as necessary.

Step 3: Quality target value setting (QTVS)

Chapter 3 of this guide presents metrics and reference values for setting the quality target value that must be considered to establish the quality of the embedded software. Reference values are defined for each of the four system types classified in Step 1, and the adjustment base values are also shown. According to the results of system characteristics profiling and project characteristics profiling (Steps 1 and 2) and the evaluation metrics information described in Chapter 3, choose the evaluation metrics to be used for each development project and set quality target values. Note that the metric reference values given in Chapter 3 were produced by analyzing data collected through hearings and questionnaires undertaken by SEC and are provided for reference only. For an actual project, examine, review, and supplement these metrics as necessary.

Target and scope of quality target value setting

When setting quality target values by applying the three steps described above, it is necessary to determine the target and scope of the system characteristics profiling.

Guideline for system components

Generally, an embedded system consists of several "components," and consequently, the embedded software used in that system can also be divided into components. Each of these components may have very different characteristic as well as qualities required. The software used in an automobile system, for example, consists of many components and the quality, reliability, and performance demanded of the software

component controlling the braking system are very different from those required for the component that controls the air-conditioning system. In system characteristics profiling, "components" with different characteristics are profiled separately. This allows the setting of quality targets that reflect the characteristics of individual components. The way in which the target software is to be divided depends on the software functionality and structure. So, fully consider these factors and apply system characteristics profiling with appropriate granularity. Also note that, if profiling is too strict and focuses on the system components, you may fail to understand the characteristics of the entire system. The ideal situation is to profile the system components while being aware of the entire system.

Guideline on the use and reuse of software

Nowadays, it is commonplace to use and/or reuse legacy design assets in embedded software development. With this style of development, it is important to choose the adjustment targets for system characteristics profiling and quality target value setting (described below). In particular, when normalizing values within the overall volume or process effort of the target software, such as the execution ratio of the test work, targets can be chosen in different ways. As shown in Figure 2-1, when subsystem software is divided into files, the subsystems and files for which a program or design sheet has been changed or is possibly changed (by even just a single line) are targeted for system characteristics profiling or quality target value setting, regardless of the amount of volume changed. For software consisting of subsystems A, B, and C as shown in the figure, therefore, when subsystems B and C are partially modified or expanded, subsystem D is newly created, and subsystem A is used as is, subsystems B, C, and D are targeted in system characteristics profiling and the total number of source code lines for these three subsystems are used as the basis for setting the quality target value(1). This idea is also applied to the development of derived versions in a product family. That is, when developing a derived version of the software, components with functional additions or modifications are targeted (without regard to the total volume) for system characteristics profiling or quality target value setting.

(1): If subsystem A is a core part of the system and is targeted in the test, consider including its number of source code lines in the total number of source code lines, as necessary.

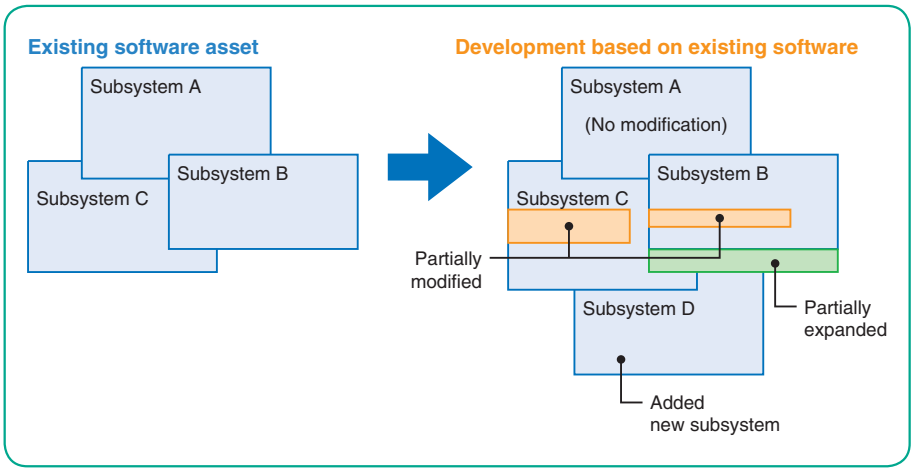


Figure 2-1: Guideline on the use and reuse of software

2.2 Step 1: System Characteristics Profiling

Role and positioning of system characteristics profiling

The embedded systems surrounding us are exposed to an unlimited range of scenarios because there are no restrictions on the way that users may apply them. The first step toward establishing quality for such a broad range of embedded systems, considering their characteristics, is to evaluate the characteristics of the target system. This guide uses system characteristics profiling (SCP[(2)]) to evaluate the characteristics of an embedded system. SCP is a system characteristics evaluation method developed by SEC, and is based on recent lively discussions on electronic and electrical functional safety and system dependability.

The most distinctive feature of system characteristics profiling is "assuming the use scenario of the target embedded system and classifying the system type from the user's viewpoint." That is, SCP classifies the system type by considering "when a system trouble occurs once the system is complete and actually being used, to what degree would the user be inconvenienced or how much damage would they incur?" For example, it estimates the degree to which typical users would be inconvenienced if the control system of a cellular phone were to fail, or how great the damage would be if the control system for a nuclear power station were to fail, causing the reactor to go out of control, and then classifies the system type according to the results.

In this way, to define and classify the type of the socially required system quality, the SCP described in this guide assumes the system troubles that could occur during system use and operation and classifies the target system (or its components) as one of the following four types according to the possible economic and/or human damage and cost.

(2) **SCP**: System characteristics profiling: Developed by M. Hirayama and S. Yoshizawa

Classification of system types

System type: SCP (system characteristics profiling)

Type 1: Normal: System for which the normal level of quality and reliability is required

Type 2: Normal Quality Required: System for which a higher level of quality and reliability than the normal level is required

Type 3: Critical: System for which high quality and reliability are required

Type 4: Highly Critical: System for which extremely high quality and reliability are required

This guide classifies a target system into these four types, defines the quality to be achieved by the product, and sets the quality target value for the development process. If we look at the two examples above (the cellular phone and nuclear power station control systems), it is easy to say that the latter will cause much greater problems upon the occurrence of a failure. The required quality level of a target system can be determined by applying SCP. The type of the quality level assigned to a target system can be classified; for example, the cellular phone control system is classified as Type 2: Normal Quality while the nuclear power station control system is classified as Type 4: Highly Critical.

Evaluation method for system characteristics profiling

In system characteristics profiling, the following two decisions are performed, in the order shown, to classify the target system into one of the four types above.

(1) Human cost calculation **(2) Economic cost calculation**

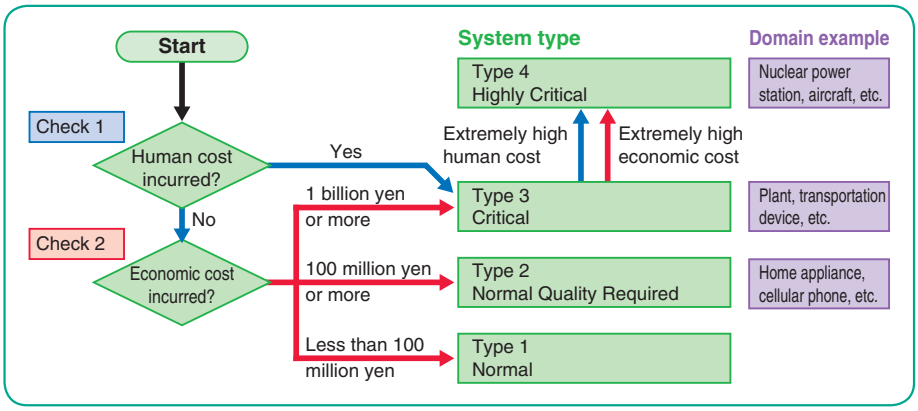


Figure 2-2: SCP check flowchart

(1) Calculating the human cost

Analyze whether direct or indirect users of the target system would be injured, killed, or otherwise fatally affected by an accident caused by a trouble of the system. The points to be considered when estimating human cost are as follows:

[Points to consider when estimating human cost]

- If a human cost (including serious injury) would be incurred as a result of an accident caused by a system trouble, high quality and reliability is required for the system as a means of ensuring safety (Type 3 or higher).
- If a very high human cost would be incurred as a result of an accident caused by a system trouble, the system is classified as Type 4 (Highly Critical).

Human cost	Type	Acronym	Meaning
No human cost	Type 2 or lower	NQ or lower	Normal Quality Required or lower
Human cost (including serious injury) may be incurred	Type 3	C	Critical
Very high human cost (including serious injury) may be incurred	Type 4	HC	Highly Critical

[Tips on estimation]

- **Direct or indirect damage acknowledgement**

  In a criticality accident such as the Chernobyl Nuclear Power Station accident, it is assumed that the engineers working there would lose their lives, while the health of residents around the power station would suffer considerably. In such a case, the damage incurred by the direct users and the indirect costs incurred by the residents would both be included in the human cost.
- **Possibility of multiple accidents being caused by a single product**

  If the engine control software of an automobile were to fail, resulting in a traffic accident, the number of passengers in the automobile would be regarded as being the direct human cost for a single accident. However, because as many automobiles as those manufactured featuring the same software would be distributed to the market, they could crash as a result of the same failure. In this case, consider the human cost accumulated the total number of automobiles manufactured when calculating the scale of the human cost.
- **Probability of accident**

  In the above examples, the probability of a power station accident is extremely low and human damage would rarely be caused. Almost certainly, automobile

engine control software failures would occur at higher (than the power station failure) frequencies. Therefore, the probabilities of system troubles occurring, as well as the human damage, should also be considered to calculate the scale of the damage more accurately. Since SCP is intended to easily classify the system type, it basically does not take failure occurrence and human damage occurrence probabilities into consideration. They may be considered, however, depending on the organization and project. In addition, for cases in which the participants (including neighbors) may suffer for a long time after the system trouble, as in the case of Chernobyl, also consider the term of the suffering in the evaluation.

(2) Calculating the economic cost

Next, analyze the economic cost incurred by the user in the event of the trouble of the target system. An embedded system alone would seem incapable of causing extreme economic cost. However, devices using embedded systems are nowadays used in larger systems or IT systems, such that the failure of the embedded system within a component could cause the entire system to stop functioning. In any such case, the evaluation of the possible economic cost is very important. The points to be considered when estimating the economic cost are as follows.

[Points to consider when estimating economic cost]

- When a system is judged as being of Type 4 in (1), calculation of the economic cost is not necessary.
- When economic cost is anticipated in the event of an accident, the system should be considered as being the cause of that economic cost. Calculate the economic loss that would be incurred by both direct and indirect users in the event of an accident.
- Classify the system into one of the following four types according to the economic cost incurred in the event of accident. Note that the type classification value is for reference only and can be revised according to the product area and the situation facing the company.

Economic cost	Type	Acronym	Meaning
Cost lower than 100 million yen	Type 1	N	Normal
Cost lower than 1 billion yen	Type 2	NQ	Normal Quality Required
Cost of 1 billion yen or higher	Type 3	C	Critical
Among Type 3 costs, those for which extremely large losses are anticipated	Type 4	HC	Highly Critical

[Tips on estimation]

- **Direct and indirect damage acknowledgement**

  If an airline's check-in system fails such that its passengers cannot check in, the damage to the airline's business (economic cost) due to the delay in providing a service can be the first item considered (direct cost). In addition, individual passengers may suffer their own business losses as a result of their flights being canceled (indirect cost). Consider both of these costs when estimating the economic cost.

  In the control system of a plant, any system trouble could suspend the manufacturing of products, preventing the manufacturer from providing those products to its customers. Such factors should also be considered when estimating the economic cost.

- **Estimating damage over time**

  If a system trouble occurs, it may take some time to repair and restart the system. The user continues to incur economic loss during this period. So, when estimating the economic cost, sum up that incurred while the system cannot provide services as a result of the trouble.

- **Exclusion of manufacturer's economic cost**

  When an embedded system suffers a failure, the manufacturer will also incur economic costs for recalls and repairs. In the first steps of SCP, however, these costs are excluded from the calculation and are left until the evaluation of the project characteristics profiling in the next step.

Applying feedback from failures in the field and experience with actual systems to assumptions

In SCP, the assumed human and economic costs incurred upon system troubles are used in calculations as mentioned above. However, it is often difficult to assume the type and scale of influence upon the occurrence of an actual system trouble caused by the failure of the developed system. In such a case, as mentioned in Section 2.6 of this guide, analyze any failures or accidents that have actually occurred with similar systems or products in the past and use the analysis result as a reference for making assumptions regarding damage.

2.3 Step 2: Project Characteristics Profiling

Meaning and positioning of project characteristics profiling

This guide uses an approach whereby the quality level of a target system is classified into one of four types as described in Step 1, after which the quality target value is set according to the type. For example, we could take "the review time for a Type 1 system for 10% of the process effort for the entire development project" as the quality target value. Suppose, however, that in an actual system development environment, a team of unskilled novices were to be assigned to the system development because the system is classified as being of Type 1, for which only normal quality is required. In such a case, it is doubtful that the recommended review time for Type 1 will be enough. Given the skills and experience of the development members, it would be safer to spend more time on the review. Therefore, this guide sets the final quality target values, based upon the system type obtained by system characteristics profiling and by adding the characteristics of the system development project. Step 2 focuses on the project for developing the target system and deploys the concept of project characteristics profiling (PCP[(3)]) to evaluate the project characteristics. As detailed below, PCP checks ten factors related to the system implementation characteristics and the development project circumstances. The result of the project characteristics profiling obtained here will be applied as adjustment coefficients for the system types when setting the quality target values in Step 3 of this guide.

Project characteristics profiling factors

Regarding the project characteristics as well as the system and software implementation characteristics in actual development, the ten profiling factors (hereafter referred to as "factors") listed in Table 2-1 are regarded as being adjustment factors for evaluating the project characteristics profile. Specifically, the project characteristics profiling factors listed in Table 2-1 and related to these ten factors are checked.

(3) **PCP**: Project Characteristic Profiling: Developed by M. Hirayama and S. Yoshizawa

Note that these ten factors used for project characteristics profiling are merely examples and it is recommended that they be reviewed as necessary depending on the circumstances of a target organization or project to be evaluated.

Table 2-1: Project characteristics profiling factors

(1) Software size	As the project becomes larger, software failure is more likely to occur and so more thoughtful development is required.
(2) Software complexity	As the project becomes more complicated, software failure is more likely to occur and so more thoughtful development is required.
(3) System constraint severity	As the system constraints become more severe, the number of matters to consider in design and implementation increases and testing for various cases becomes necessary.
(4) Specification clarity	A project with ambiguous system specifications or frequent changes to the specifications tends to result in redesign or failure, so cautious checking in the development phase is required.
(5) Quality of reuse software	When reusing existing assets for development, and when the quality of the reuse software is low, the quality of the new project will be affected and so more cautious review and testing are required.
(6) Degree of organization of development process	If a development process is not well organized and deployed, the development members will get out of step with each other, resulting in missing tasks and errors, and failures are more likely to occur.
(7) Labor division and hierarchicalization of development organization	If the development organization is large and deeply hierarchical and introduces division of labor, problems such as insufficient communications tend to occur.
(8) Skill of development members	If the skills of the development members vary from person to person or if poorly skilled persons hold a majority in the team, more cautious checks should be applied to the work and deliverables through the application of reviews.
(9) Experience and skill of project manager	If the project manager (PM) is inexperienced or poorly skilled, reviews and/or checks at appropriate timings may be insufficient, so sufficient review time should be reserved in advance.
(10) Damage to manufacturer upon occurrence of system trouble	When the damage to the manufacturer upon the occurrence of a system trouble (e.g., recall) would be considerable, thorough quality checks and quality establishment must be done in the development stage to lower the risk.

Project characteristics profile check table

Use Table 2-2 to check factors (1) to (10) and evaluate the project characteristics profile. For example, for the development target of the project, if "(1) Software size" is "extremely small," check the corresponding column. Once all of factors (1) to (10) have been checked, multiply the number of "-1" columns by -1 and then enter the result in the left subtotal column, and then multiply the number of "+1" columns by 1 and then enter the result in the right subtotal column. Finally, sum the subtotals and enter the result in the Total points column. This value will be the adjustment coefficient to apply to the system type for determining the quality target value in Step 3 of this guide.

Table 2-2: Project characteristics profile table

Factors			Negative adjustment (-1)		Basic		Positive adjustment (+1)
(1)	Software size	☐	Extremely small	☐	Average	☐	Extremely large
(2)	Software complexity	☐	Extremely simple	☐	Average	☐	Extremely complex
(3)	System constraint severity	☐	Loose	☐	Average	☐	Severe
(4)	Specification clarity	☐	Extremely clear	☐	Average	☐	Ambiguous
(5)	Quality of reuse software	☐	Extremely high quality	☐	Average	☐	Extremely low quality
(6)	Degree of organization of development process	☐	Well organized	☐	Average	☐	Not well organized
(7)	Labor division and hierarchicalization of development organization	☐	Organization is simple	☐	Average	☐	Organization is complex
(8)	Skill of development members	☐	Member skill is high	☐	Average	☐	Member skill is low
(9)	Experience and skill of project manager	☐	PM skill is high	☐	Average	☐	PM skill is low
(10)	Damage to manufacturer upon occurrence of system trouble	☐	Extremely low	☐	Average	☐	Extremely high
Subtotal							
Total points							

(Example)

The following shows an example of calculating the adjustment coefficient by using this check table.

In this example, the subtotal of the "-1" column is -3 points and the subtotal of the "+1" column is +1 point, so the adjustment coefficient is -2.

Table 2-3: Project characteristics profile table (example)

Factors			-1		0		+1
(1)	Software size	■	Extremely small	☐	Average	☐	Extremely large
(2)	Software complexity	☐	Extremely simple	☐	Average	☐	Extremely complex
(3)	System constraint severity	■	Loose	☐	Average	☐	Severe
(4)	Specification clarity	☐	Extremely clear	■	Average	☐	Ambiguous
(5)	Quality of reuse software	☐	Extremely high quality	■	Average	☐	Extremely low quality
(6)	Degree of organization of development process	☐	Well organized	■	Average	☐	Not well organized
(7)	Labor division and hierarchicalization of development organization	☐	Organization is simple	☐	Average	■	Organization is complex
(8)	Skill of development members	☐	Member skill is high	■	Average	☐	Member skill is low
(9)	Experience and skill of project manager	☐	PM skill is high	■	Average	☐	PM skill is low
(10)	Damage to manufacturer upon occurrence of system trouble	■	Extremely low	☐	Average	☐	Extremely high
Subtotal		-3		0		1	
Total points							-2

2.4 Step 3: Quality Target Value Setting

What is a quality target value?

The quality target values for the software to be developed are set by clarifying the measurement targets (system and development), using the type adjustment coefficient that is based on the system type determined in Step 1, and the project characteristics profile obtained in Step 2, and by referencing the individual evaluation metrics reference table given in Chapter 3 of this guide.

Chapter 3 defines the metrics for quantitatively controlling quality during embedded software development. These metrics are basically used as standards for setting quality targets during development, such as "to what degree should the design document be reviewed?" or "how complex should the source code be?" To this end, two types of metrics, as explained in Chapter 1, are provided, namely, process metrics and product metrics.

- Process metrics are used to evaluate the sufficiency of the work used for checking quality, such as the review of intermediate deliverables.
- Product metrics are used to evaluate the quality of intermediate deliverables during development.

In actual development, the target values for the process metrics and product metrics can be considered as being equal to the quality target values for the project. The quality target values, as already mentioned, should be set according to the quality and reliability levels required for the system or embedded software being developed. Uniform quality target values that are set without considering these factors would generate gaps between the required and actual quality levels of the system or software, resulting in excesses or shortfalls. This guide sets the target values for the individual process metrics and product metrics, based on the SCP and PCP results mentioned above.

Examination and determination of deployed process and product metrics

Before setting a quality target value, you must first determine the metrics to be used to set that quality target value. To this end, according to the evaluation metrics mentioned in Chapter 3 of this guide, examine and determine the metrics to be used in the target project. When examining the metrics to use, consider the development process for the project and the intermediate deliverables that are to be created, and determine the metrics that are to be measured and the timing at which the measurement is to be done. The metrics listed in Chapter 3 are for reference only and should be selected and deployed according to the target system or project characteristics or circumstances. They can be expanded as necessary. As mentioned in the tips for quality quantification given in Chapter 4 (Section 4.5) of this guide, you should also examine the metrics measurement cost and purpose sufficiently, as well as the actual use of the metrics.

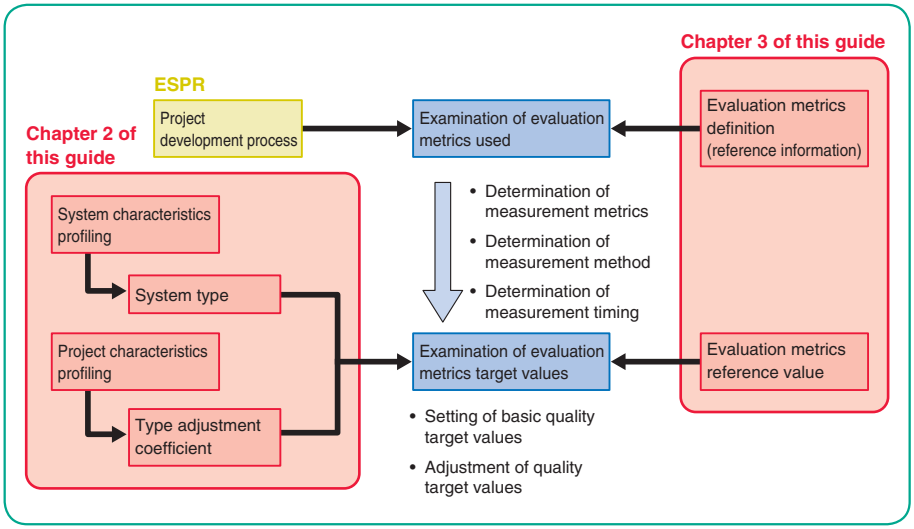


Figure 2-3: Quality target value setting flow

How to set a quality target value

Use the metrics reference values shown in Chapter 3 of this guide and the system type classified in Step 1, as well as the type adjustment coefficient obtained from the project characteristics profiling in Step 2, to set the quality target value for each metric. Also, consult with the project leader, manager, and/or actual engineers when setting the quality target values.

(1) Setting a basic quality target value

After determining the metrics to be used in a project, it is necessary to set target values for those metrics.

Chapter 3 of this guide provides reference values along with the definitions of individual process metrics and product metrics. For example, for the Ratio of the Design Review Effort (ID: PR11, RDRE), a representative process metric, the table shown below is provided. From this table, the basic value of the Ratio of the Design Review Effort should be 2.00% when the target system type is Normal, 6.00% for Normal Quality Required, 10.00% for Critical, and 14.00% for Highly Critical.

Table 2-4: Example of ratio of the design review effort

ID	PR11				
Name	Ratio of the Design Review Effort				
Abbreviation	RDRE				
Reference value	N	NQ	C	HC	Adjustment base value
	2.00	6.00	10.00	14.00	4.00
Reference value range	0.00 to 6.00	2.00 to 10.00	6.00 to 14.00	10.00 to 18.00	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• Represented as the balance between the process effort for the design review and process effort spent on design (design process effort). • As better safety and reliability are required, the design review process effort increases, as does the design process effort itself. Therefore, a higher value is not necessarily good but an appropriate value is required.				
Calculation method	Review Effort for DEsign/Process Effort for DEsign RDRE = REDE/PEDE				

(2) Adjusting the quality target value

Next, adjust the basic quality target value obtained in (1). Assume, for example, that the system characteristics profiling mentioned above is performed for a target system, and that it is classified as Type N in Step 1: SCP and Step 2: PCP resulted in +4 points. In this case, the ratio of the design review effort for the target system is as follows.

Ratio of the specifications review effort for Type N: 2.00%

Adjustment base value for the ratio of the design review effort: 4.00

Adjustment coefficient: +4

Therefore,

Ratio of the design review effort of the target system

= Ratio of the design review effort for the corresponding type + adjustment coefficient/10 × adjustment base value

= 2.00 + (4/10 × 4.00)

= 3.60%

Table 2-5: Quality target values setting table

Target system or project name	

System/project characteristics profiling result				
System Characteristics Profiling	Normal	Normal Quality	Critical	Highly Critical
Project Characteristics Profiling			point	

Process metrics			Reference value for corresponding type	Adjustment base value	Set target value
Ratio of the work effort					
	Specifications review			4.00	
	Design review			4.00	
	Code review			1.50	
	Testing review			1.50	
	Testing			5.00	
	Review			4.00	
Execution ratio of the work					
	Specifications review			2.40	
	Design review			2.40	
	Code review			1.20	
	Testing review			2.00	
	Testing			17.00	
	Review			8.00	
Product metrics			**Reference value for corresponding type**	**Adjustment base value**	**Set target value**
Document					
	Document volume				
		Requirements specification		4.00	
		Design document		10.00	
		Test specifications		10.00	
	Document balance				
		Requirements specification		-	
		Design document		-	
		Test specifications		-	
Code					
	Code volume				
		Number of lines	2 KLOC	Should be equal to or less than this value	
		Number of function lines	160 LOC	Should be equal to or less than this value	

	Code characteristics				
		Control statement description ratio		5.00	
		Comment line description ratio		5.00	
		Coding rule conformity		100	
Testing					
	Testing sufficiency				
		Testing density		25.00	
		Failure coverage		0.01	
	Operation completeness				
		Failure elimination ratio		3.00	

Document balance metrics				
	Item No.	Description	Reference value	Set target value
Requirements specification	R1.	Entire description volume	100	
	R2.	Target user and usage description	5	
	R3.	Description volume for operation environment conditions	10	
	R4.	Description volume for main functions	40	
	R5.	Description volume for safety and non-functional requirements	30	
	R6.	Description volume for overall system structure	10	
	R7.	Description volume for exception handling	5	
Design document	D1.	Entire description volume	R1×3	
	D2.	Description volume for overall system structure	5	
	D3.	Description volume for functional block structure	5	
	D4.	Description volume for functional block details	50	
	D5.	Description volume for interface data	20	
	D6.	Description volume for exception handling	20	
Test specifications	T1.	Entire description volume	R1×3	
	T2.	Description of test environment	5	
	T3.	Description of test procedure and conditions	10	
	T4.	Description of normal system	35	
	T5.	Description of abnormal system and exception handling	45	
	T6.	Description of test completion criteria	5	

Basic concepts of adjustment coefficient

As can be seen from the above calculations, when a system is classified as being N or NQ in Step 1: SCP, the adjustment effect in Step 2: PCP is a maximum of +/-10 (100%) and adjustment is performed by shifting the system characteristics profiling type up or down by one. That is, when the system is classified as being of Type 3: Critical in Step 1, the same quality target value as that for the Normal level (two levels below Critical) would be insufficient even if the development conditions obtained by PCP are considered for the maximum. Therefore, for a Critical system, it is recommended that the quality target value be set within the range from Normal Quality Required (one level below) to Highly Critical (one level above), considering the convenience of development.

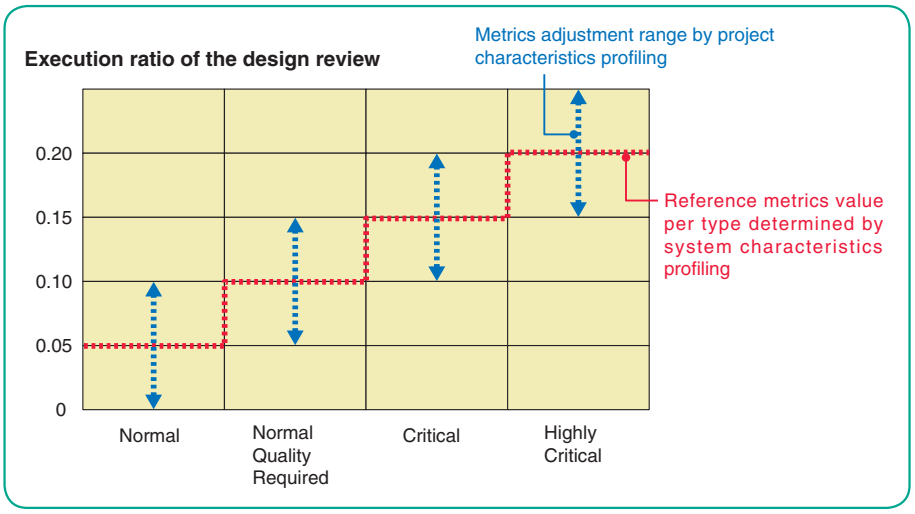


Figure 2-4: Basic concepts of adjustment coefficient

In actual quality target value setting, Table 2-5 is used to select the metrics and set the target value.

Continuous review of reference metric values within an organization

Chapter 3 of this guide lists the definitions of metrics and corresponding reference values as a standard for setting quality target values. These reference metric values were produced by SEC by analyzing data collected through hearings and questionnaire surveys with the corporate partners approving SEC's activities. The collected data came from organizations and projects in various areas and of different scales. So, these values should be used merely as references for examination.

In general, these standard metric values etc. are more precise for individual organizations when accumulated and used as values that reflect the characteristics of the organizations. Given this point, use the reference values given in this guide as a start point and modify them optimally based on the experience of your organization or corporation. Naturally, it is necessary to analyze the mid-to-long term metric trends and the results of their application to actual development projects and obtain optimum standard values, when preparing standard metric values for each organization. In the process, collect statistics on these values as necessary and process the data layers and out-of-scope values to prepare highly precise standard values that are well suited to each organization or corporation.

2.5 Profiling Example

This section introduces examples of system characteristics profiling and project characteristics profiling by using a simple case.

Assumed case: Small mobile communications device

Development condition

Major function: A palm-top size small, personal use mobile communications device for accessing e-mail and Web services. In addition to the communications function, the major functions include the display of e-mails and Web pages (with some animation), scheduling, and directory management.

Intended users: Young adults; Expected shipments of 10,000 units.

Software: The component software size is approximately three million LOC, developed using existing cellular phone control software. The device will be functionally and structurally simpler than a cellular phone. The software specifications will be relatively simple and easy to understand.

Development project: The development project team consists of existing cellular phone development project members and, because the team is temporarily and hurriedly formed, the relationship with the hardware team is not fixed. Therefore, the project is immature in terms of development processes. The project leader, however, has considerable experience.

Positioning of product: Because the shipment volume is not high and the product price is relatively low, the manufacturer expects that the recall cost in the case of product trouble is estimated as not being particularly high.

System characteristics profiling

Step 1: SCP (system characteristics profiling)

In this system, the probability of human damage due to a system trouble is presumed to be extremely low. Any economic cost caused by the system trouble can be calculated as follows.

- **NU**: Number of Users: Number of target users
- **RD**: Ratio of Damaged users: Ratio of total users to the number of target users affected by a system trouble
- **DI**: Damage of Impact: Estimated damage per user when the device is assumed to be unavailable to users for a day due to a system trouble
- **ND**: Non Service Days: Number of days during which the users cannot receive services due to a system trouble
- **ED**: Economic Damage: Conclusive economic cost caused by a system trouble

The relationship between the above elements can be expressed using the following formula:

$$ED = [ND \times (NU \times RD)] \times DI$$

When the following figures are assumed in this case:

NU = 10,000 (number of users)

RD = 1.0 (assuming that all users are affected by a system trouble)

ND = 2.0 days (system trouble will be repaired in about two days installing the software patch supplied by the vendor)

DI = 5,000 yen (assuming that each user is receiving a service equivalent to 5,000 yen using the device per day)

$$ED = [2.0 \times (10000 \times 1)] \times 5000$$
$$= 100{,}000{,}000 \text{ yen}$$

The economic cost can be estimated as shown above. Based on this result, this system can be classified as being of Type 2 (Normal Quality Required), given the assumed

damage due to the economic cost.

Note that the values given here for the number of users and the method of assuming damage to the users are just examples.

Step 2: PCP (project characteristics profiling)

Based on the development conditions listed at the beginning of this section, the type adjustment coefficient (turning point) is 0. So, the reference value obtained in Step 1 (Type 2) can be used as is.

Table 2-6: Project characteristics profiling (example of entry for assumed case)

			Negative adjustment (-1)		Basic		Positive adjustment (+1)
(1)	Software size	☐	Extremely small	■	Average	☐	Extremely large
(2)	Software complexity	☐	Extremely simple	■	Average	☐	Extremely complex
(3)	System constraint severity	☐	Loose	■	Average	☐	Severe
(4)	Specification clarity	☐	Extremely clear	■	Average	☐	Ambiguous
(5)	Quality of reuse software	☐	Extremely high quality	■	Average	☐	Extremely low quality
(6)	Degree of organization of development process	☐	Well organized	☐	Average	■	Not well organized
(7)	Degree of labor division and hierarchicalization of development organization	☐	Organization is simple	☐	Average	■	Organization is complicated
(8)	Skill of development members	☐	Member skill is high	■	Average	☐	Member skill is low
(9)	Experience and skill of project manager	■	PM skill is high	☐	Average	☐	PM skill is low
(10)	Damage to manufacturer upon occurrence of system trouble	■	Extremely low	☐	Average	☐	Extremely high
Subtotal		-2				2	
Total points							0

2.6 Evaluation of System Trouble and Reflection on System Characteristics Profiling

Reflection on system characteristics profiling according to feedback on system trouble

In the flow from system characteristics profiling to project characteristics profiling, and to a sequence of quality target value settings explained in this chapter, system characteristics profiling, which is the first step, assumes the occurrence of a system trouble before starting the development of the target system and analyzes the influence of that trouble. This analysis is very difficult, however, even if the specifications and use scenarios of the target system are thoroughly analyzed, and the evaluation result may differ depending on the analysis.

IEC61508, which defines system functional safety, includes a pre-safety plan and post-safety plan to describe the importance of feedback on actual system troubles and their reflection on subsequent development, as well as countermeasure planning considering assumed troubles when actually developing a system. Concerning this, Sections 2.2 to 2.4 in this chapter mainly describe the set up of quality target values in the pre-development stage related to the pre-safety plan. On the other hand, the evaluation of the influence of an actual system trouble and the feedback to quality control in the next project, equivalent to the post-safety plan mentioned in IEC61508, is also a very important factor. This section summarizes the concept of evaluating the influence of a system on the user or society when a system trouble occurs. The system trouble influence evaluation scale, ST-SEISMIC (System Trouble SEISMIC[(4)]), mentioned here, is newly considered by SEC and shall be understood as a sample measure for evaluating the influence of system troubles.

(4) **ST-SEISMIC**: System Trouble SEISMIC: Developed by M. Hirayama, S. Yoshizawa, and S. Yamaguchi

Concept of System Trouble SEISMIC Scale ST-SEISMIC

For correct system characteristics profiling, it is necessary to properly recognize that damage due to a system trouble or fault that is actually caused by a similar system, with respect to user health and economic well-being, as well as the influence on the system peripheral environment and society. Naturally, the level of damage and influence will depend on the influence of the actual system trouble. ST-SEISMIC classifies these levels into seven stages, as shown in Table 2-7, and evaluates the degree of the influence of actual system trouble. It is intended to be used as a reference for calculating the assumed amount of (1) human damage and (2) economic damage in SCP when developing subsequent or similar systems according to the evaluation of actual troubles.

Basically, evaluation according to the system trouble classes shown in Table 2-7 is reflected on the system type classification.

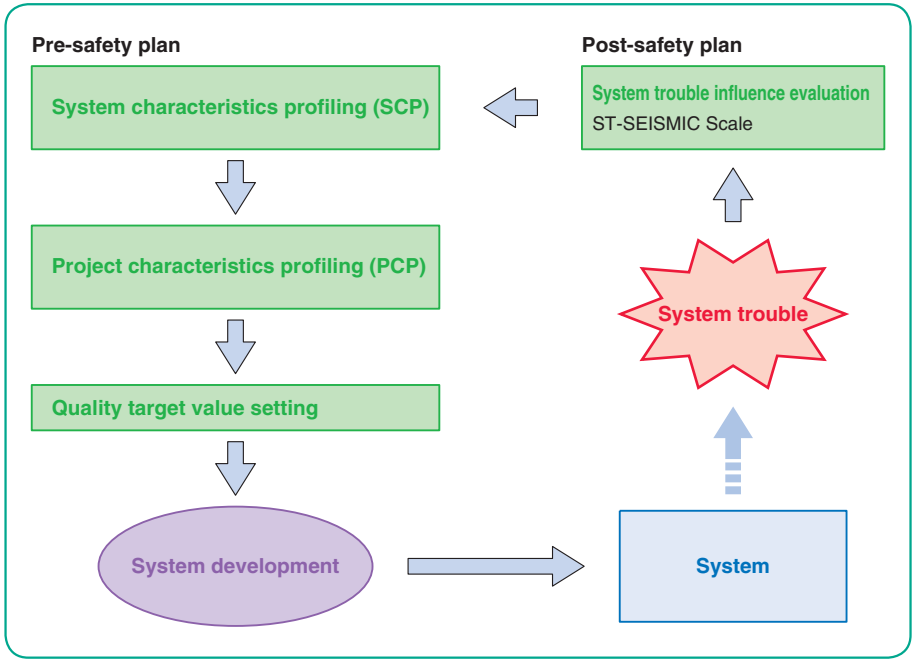


Figure 2-5: Using ST-SEISMIC

Table 2-7: ST-SEISMIC Scale (System Trouble SEISMIC Scale)

Class		Influence on user operation	Health hazard to user	Economic damage to user	Influence on system peripheral environment	System type
0	None	Failure is not noticed.				Normal
1	Minimal	Some users sense a problem with the system operation.				
2	Very slight	Many users sense a problem with the system operation.				Normal Quality Required
3	Slight	Most users sense a problem with the system operation and some make a complaint.	Some users may be exposed to a minimal health hazard.	Some services stop and some users may incur a slight economic loss.		
4	Moderate	Some of the user goals cannot be achieved.	Some users are exposed to a health hazard.	Some services stop and users may incur an economic loss.		Critical
5	Considerable	User goals cannot be achieved.	Some users are exposed to a serious health hazard.	Some or all services stop and some users may incur a serious economic loss.	System trouble leads to limited social unrest or inconvenience.	
6	Severe		Many users are exposed to a serious health hazard.	Some or all services stop and many users may incur a serious economic loss.	System trouble leads to social unrest.	Highly Critical
7	Very severe		Most users are exposed to a serious health hazard.	Some or all services stop and most users incur a serious economic loss	System trouble leads to serious social unrest.	

How to use the ST-SEISMIC Scale (System Trouble SEISMIC Scale)

As you may have already noticed with ST-SEISMIC, this scale is created by comparing the degree of seismic intensity (i.e., the "SEISMIC Scale") that is used in Japan, an earthquake country, to the evaluation of a system trouble. When a major earthquake occurs, its seismic intensity is quickly reported by the mass media. The intensity of anticipated earthquakes is also expressed using this scale. Recently, many system troubles have been reported. In the same way as with the seismic intensity, this scale can also be used as a guideline to indicate the countermeasures that are required for any given level of trouble or failure. Of course, the characteristics differ depending on the system, so this scale can also be used from the viewpoint of establishing quality and reliability in development by considering countermeasures suitable for the system characteristics or feeding them back to system characteristics profiling.

Chapter 3

Definition and Reference Values for Evaluation Metrics

To visualize the quality of embedded software, evaluation metrics are required. But if the definitions of evaluation metrics are ambiguous, the measured values will vary depending on who or which organization performs the measurement. In addition, to determine whether the measured value falls within the correct range, an evaluation standard for the measured value must be defined in advance. This chapter introduces the definition of evaluation metrics, corresponding reference values, and how to use them.

3.1 Definitions and Meanings of Evaluation Metrics and How to Use Them

As first explained in Chapter 1, the development of higher-quality software demands the precise monitoring of the actual development work and the resulting deliverables from the viewpoint of quality. To this end, it is essential that we have a system of quantitative quality control based on evaluation metrics to give us a means of objectively understanding the quality of the work being done and the resulting deliverables, while and then feeding back the results to improve the process as necessary. Evaluation metrics play a leading role in this quantitative quality control. This section introduces the concept of software evaluation metrics as used in this guide. In general, the following points need to be considered when examining evaluation metrics.

- Definitions and meanings of metrics and how to use them
- Measurement method and timing of metrics

Clarifying the definitions of metrics to be measured

One representative measure (evaluation metric) used to evaluate software is the number of lines of the source code, which is generally referred to as "Lines of Code" (LOC). If we think about this, the LOC value for the source code of any one process will vary depending on the language used to write the process, such as C, C++, assembly language, etc. Even if we limit the language to C, whether or not to count the comment and/or blank lines will result in very different LOC values. When the measuring and visualizing of software and its development work is to be done quantitatively using evaluation metrics, a clear definition of each metric is required.

Considering how to apply measured data to development

Numeric values, including evaluation metrics, that are acquired during development only have meaning when they are used in the actual development. An often heard phrase is "We have gathered numeric data, but we don't know what to do with it." Such a situation not only makes the effort of collecting the data useless, but also results in us

being allergic to collecting numeric data and unwilling to check even the required data. To prevent this situation from occurring and to visualize software and its development works using evaluation metrics, it is necessary to investigate, in advance, how to use the measured numeric data, in particular how it can be fed back into the development process.

For the process metrics and product metrics described in the subsequent sections, the aspects of the software or development work that are to be visualized by individual metrics, as well as the situations that can be assumed if the metric values are lower than the reference values, are described for reference purposes. Evaluation metrics described in this guide are selected as strictly as possible as those which can be measured easily and used for quality feedback during development. Using these metrics as a guide, consider the metrics that are to be used in your organization or project to visualize software development and how they are used for feedback into the development process, then narrow down the data to be measured and make full use of the measured data.

Considering the measurement method and timing of metrics

The gathering of data during product development places an extra burden on the development team. Therefore, it is necessary to thoroughly consider how and when the measurement should be done by applying metrics to the gathering of numeric data. Although the measurement method partly depends on the precision of metrics data to be measured, it is not desirable to demand an unreasonably long time and/or effort to gather unnecessarily precise data from the development team. It is important to always consider how meaningful data can be collected while keeping the amount of effort needed to acquire that data to a minimum.

3.2 Categorization of Evaluation Metrics

Usually in software development, software is created through a sequence of steps, including requirements analysis, design, coding, and testing. To control the quality of the final software, you should make multi-aspect quality checks as shown below.

(1) Checking the quality of each intermediate deliverable during the development process

(2) Checking each execution status of the work related to the development process, which may directly improve quality

(3) Checking the quality of the finally developed software

To ensure objectivity in such a quality check and quality control, this guide uses two types of metrics, namely, evaluation metrics and basic metrics which are numeric data to be measured to obtain evaluation metrics. The evaluation metrics consist of process metrics and product metrics.

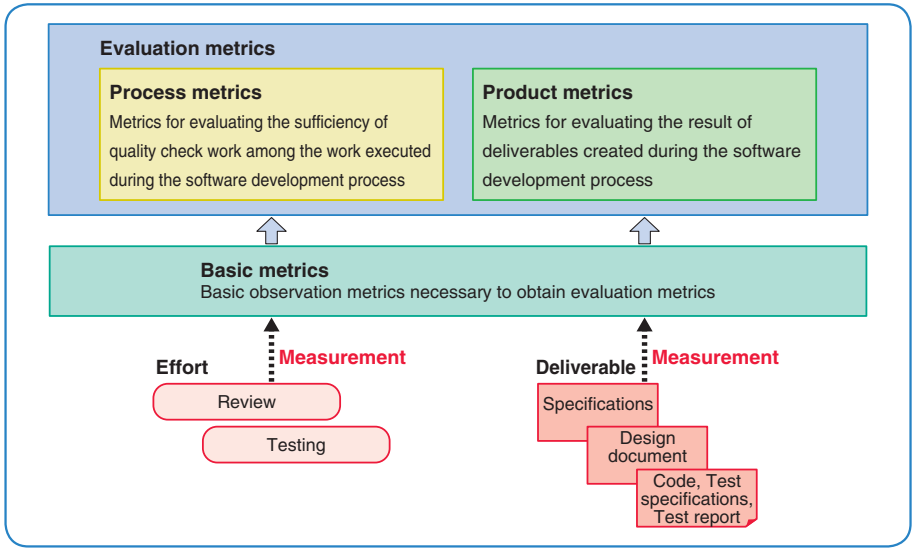


Figure 3-1: Types and meanings of evaluation metrics

Column

Efforts toward systematic organization of evaluation metrics in software engineering

In the field of software engineering, the subject of software measurement (software metrics) has been studied and software-related data has been collected. In a typical example, the ISBSG organization has collected software development data from over 3,000 IT projects conducted all over the world.

http://www.isbsg.org/japanese/index.htm

In Japan, SEC has collected development data from approximately 1,000 enterprise projects and analyzed and summarized that data in the "White Paper 2007 on software development projects in Japan." These trials were aimed at bringing project contours into sharp relief using various metrics, based mainly on software productivity.

On the other hand, such a data collection attempt has basically not been reported in the embedded software area, so there has been very little discussion as to the metrics that should be applied to each aspect of development. It is expected that data collection by ISBSG will be promoted in the embedded software area with the publication of this guide, and the understanding of software development will become possible with the adoption of numerical data in the near future.

Evaluation metrics

Evaluation metrics			
ID	**Abbreviation**	**Name**	**Measurement method or formula**
Process metrics: Ratio of the work effort			
PR10	RSRE	Ratio of the Specifications Review Effort	Review Effort for SPecification/Process Effort for SPecification
PR11	RDRE	Ratio of the Design Review Effort	Review Effort for DEsign/Process Effort for DEsign
PR12	RCRE	Ratio of the Code Review Effort	Review Effort for COde/Process Effort for COde
PR13	RTRE	Ratio of the Test Review Effort	Review Effort for Test Preparation/Process Effort for Test Preparation
PR14	RTWE	Ratio of the Test Work Effort	Process Effort for TEst/Process Effort in TOtal
PR15	RORE	Ratio Of the Review Effort	Review Effort in TOtal/Process Effort in TOtal
Process metrics: Execution ratio of the work			
PR20	ERSR	Execution Ratio of the Specifications Review	Review Effort for SPecification/Total Lines Of Code
PR21	ERDR	Execution Ratio of the Design Review	Review Effort for DEsign/Total Lines Of Code
PR22	ERCR	Execution Ratio of the Code Review	Review Effort for COde/Total Lines Of Code
PR23	ERTR	Execution Ratio of the Test Review	Review Effort for Test Preparation/Total Lines Of Code
PR24	ERTW	Execution Ratio of the Test Work	Process Effort for TEst/Total Lines Of Code
PR25	EROR	Execution Ratio Of the Review	Review Effort in TOtal/Total Lines Of Code
Product metrics: Document evaluation metrics			
Document volume evaluation metrics			
PD10	RSDV	Ratio of the Specifications Document Volume	Volume Of the Specifications Document/Total Lines Of Code
PD11	RDDV	Ratio of the Design Document Volume	Volume Of the Design Document/Total Lines Of Code
PD12	RTDV	Ratio of the Test Document Volume	Volume Of the Test Document/Total Lines Of Code
Document balance evaluation metrics			
PD20	BSDD	Balance of the Specifications Document Description	Number of pages of each part in requirements specification/Total number of pages in requirements specification
PD21	BDDD	Balance of the Design Document Description	Number of pages of each part in design document/ Total number of pages in design document
PD22	BTDD	Balance of the Test Document Description	Number of pages of each part in test specifications/ Total number of pages in test specifications
Product metrics: Code evaluation metrics			
Code volume evaluation metrics			
PD30	FLOC	File Lines Of Code	Same as File Lines Of Code of basic metrics
PD31	MLOC	Module Lines Of Code	Same as Module Lines Of Code of basic metrics
Code characteristics evaluation metrics			
PD32	ROCS	Ratio Of Control Statement	Number Of Control Statement/Total Lines Of Code
PD33	ROCL	Ratio Of Comment Line	Comment Lines Of Code/Total Lines Of Code
PD34	RDCR	Ratio of Deviation of Coding Rules	Number of Deviation of Coding Rules/Total Lines Of Code
Product metrics: Test evaluation metrics			
Test sufficiency evaluation metrics			
PD40	DOTI	Density Of Test Items	Number Of Test Items/Total Lines Of Code
PD41	ROFC	Ratio Of Fault detection in Comparison	Ratio Of Fault Detection during final 10% of test period/ Ratio Of Fault Detection during first 90% of test period
Operation completeness evaluation metrics			
PD42	ROFE	Ratio Of Fault Elimination	Number Of Eliminated Fault/Number Of Detected Fault

Basic metrics			
ID	**Abbreviation**	**Name**	**Measurement method or formula**
Source code volume basic metrics			
B10	TLOC	Total Lines Of Code	Total number of lines per file
B11	FLOC	File Lines Of Code	Scale of the source code described in each source file
B12	MLOC	Module Lines Of Code	Scale of the source code per function
B13	NOCS	Number Of Control Statement	Number of control statements per file
B14	CLOC	Comment Lines Of Code	Number of comment lines per file
B15	NDCR	Number of Deviation of Coding Rules	Accumulated number of deviations from coding rules
Document volume basic metrics			
B20	VOSD	Volume Of the Specifications Document	Total number of pages in documents related to specifications
B21	VODD	Volume Of the Design Document	Total number of pages in documents related to design
B22	VOTD	Volume Of the Test Document	Total number of pages in documents related to testing
Process effort basic metrics			
B30	RETO	Review Effort in TOtal	Total review process effort
B31	RESP	Review Effort for SPecification	Total specifications review process effort
B32	REDE	Review Effort for DEsign	Total design review process effort
B33	RECO	Review Effort for COde	Total code review process effort
B34	RETP	Review Effort for Test Preparation	Total test review process effort
B35	PETO	Process Effort in TOtal	Total development process effort
B36	PESP	Process Effort for SPecification	Total specifications creation process effort
B37	PEDE	Process Effort for DEsign	Total design process effort
B38	PECO	Process Effort for COde	Total code creation process effort
B39	PETP	Process Effort for Test Preparation	Total test preparation and check process effort
B3A	PETE	Process Effort for TEst	Total test process effort
Test volume basic metrics			
B40	NOTI	Number Of Test Items	Total number of test items (those executed only)
B41	NOET	Number Of Executed Test Items	Total number of executed test items (including duplicate items)
B42	NODF	Number Of Detected Fault	Total number of faults detected after unit testing
B43	NOEF	Number Of Eliminated Fault	Total number of faults corrected of those detected
B44	ROFD	Ratio Of Fault Detection	Number Of Detected Fault/Number Of Executed Test Items

Evaluation metrics normalization and basic metrics

This guide aims to visualize software using two types of evaluation metrics; process metrics and product metrics. The most important point requiring visualization is the size of the software or the scale of the development project (i.e., the process effort). For example, when the value of the review effort for design, a process metric, is 1 man-month, the project will be evaluated very differently to a 1000 man-month or 100 man-month project in terms of review sufficiency. We can also think of the number of design document pages as a product metric. The number of design document pages will be large when the size of the software to be developed is large. But in this case, "the number of design document pages" alone is not necessarily an appropriate value. Thus, some process metrics and product metrics must be evaluated relative to the software size of the measurement target or the total process effort to be spent on the target.

Therefore, when values are calculated for process metrics and product metrics in this guide, in addition to calculating direct values such as the review effort for design and the number of design document pages as basic metrics, values necessary for normalizing these values, such as the total number of lines of source code or the total process effort of development, are also measured. Note that the metrics needed for normalization are used as denominators in the formulas for calculating evaluation metrics, mentioned later.

3.3 Evaluation Metrics - Notes on Use

This guide describes the definition, measurement method, and usage of each process metric, product metric, and basic metric, as described later in this chapter. In principle, for these metrics we assume that the entire process, from system requirements definition through to testing, is done within a single organization. However, the actual development style will vary in practice, for example, part of the development work may be extracted or existing software may be reused. The following provides notes on each variation of the development style.

When part of the development work is extracted for outsourcing

In actual embedded system development projects, different development styles can be applied. For example, the entire flow from the initial requirements definition through to final testing may be done within a single project or organization, some of the work may be outsourced, your organization may be contracted to develop part of a project, or functions of the system may be distributed among several departments or companies for development. This guide is intended to visualize the quality of the software developed by the entire project, as described in Section 3.2, by using and measuring several evaluation metrics. These evaluation metrics ensure relatively precise measurement when the entire flow, from initial requirements definition through to final testing, is done or when all the functions are developed within a single project. However, the measuring of all of these evaluation metrics described in this guide may be difficult in projects in which some of the development work is extracted or outsourced, functions are distributed for development, or these projects are unified. Therefore, you should narrow down the measurable evaluation metrics in such cases according to the project scope or style, rather than using all of the evaluation metrics described as a reference for using this guide. Each organization, project, or department that is responsible for quality control should investigate how to use evaluation metrics according to its own situation.

Calculating basic metrics in partial development

As mentioned above, in projects in which design and coding, or requirements specification creation and final testing are contracted out, the evaluation metrics related to the work (review, etc.) done as part of each project and/or the deliverables produced by the work (design document, specifications, etc.) are only evaluated. In this case, reference the following for the basic metrics (total number of source code lines, development process effort, etc.) used as the basis for calculating the evaluation metrics.

- **Total Lines Of Code (total number of source code lines)**

  For Total Lines Of Code, use the number of lines of code for the entire software product to be developed, as a reference value, even if part of it is contracted out. If you are not sure of the entire source code volume because of the contract work, check with the client. The client should also cooperate with the contract vendor to ensure the creation of effective evaluation metrics.

- **Development process effort**

  Measure the development process effort for each process executed as part of the project. Because, however, the values for Ratio of the Test Work Effort and Ratio Of the Review Effort are relative to Process Effort in TOtal, measuring Process Effort in TOtal may not be possible if only the test work is contracted out. In this case, it is possible to regard the process effort for all of the work related to a project as Process Effort in TOtal, but the Ratio of the Test Work Effort may be 100%. For such a project, the reference values for the evaluation metrics given later in this guide are not appropriate. To evaluate the test work in this case, use Execution Ratio of the Test Work instead of Ratio of the Test Work Effort.

Measurement timing for evaluation metrics

The evaluation metrics mentioned in this guide should be measured at appropriate timings, according to the work status of each project. Figure 3-2 shows the general measurement timing.

This guide uses the evaluation metrics mentioned in Sections 3.4 to 3.6 to quantitatively control quality. Among them, the basic metrics that are actually measured should be estimated in advance, and their precision should be improved step by step as the development progresses. These metrics should be measured at the following timings.

- The document volume for the requirements specifications, design document, and test specifications should be measured upon the completion of the draft or the first edition of each deliverable.
- The code volume should be measured upon the completion of the source code.
- The required process effort should be measured by monitoring the start through the end of each work.
- The basic metrics for the test volume should be measured mainly in the test phase.

Concepts of Process Effort in TOtal and Total Lines Of Code according to measurement timing

The process metrics and product metrics actually used for quantitative quality control are obtained by processing the basic metrics values. For example, to obtain Execution Ratio of the Specifications Review, measure the process effort for the specifications review upon its completion, and then divide it by Total Lines Of Code. Coding has not normally been started, however, upon the completion of the specifications review, so the Total Lines Of Code value is not available for measurement at this point and calculation for Execution Ratio of the Specifications Review is not possible. The ratio of the work effort and execution ratio of the work, as used in this guide, are calculated using the development process effort and Total Lines Of Code, but these values are not available during development (i.e., in requirements definition, design, or coding). Therefore, to obtain the ratio of the work effort or execution ratio of the work during development, an inferred (estimated) value is used for the development process effort or Total Lines Of Code that is used as the denominator.

For these inferred values, approximate values should be obtained according to the values for similar developments in the past and taking the amount of specification changed in the system into consideration.

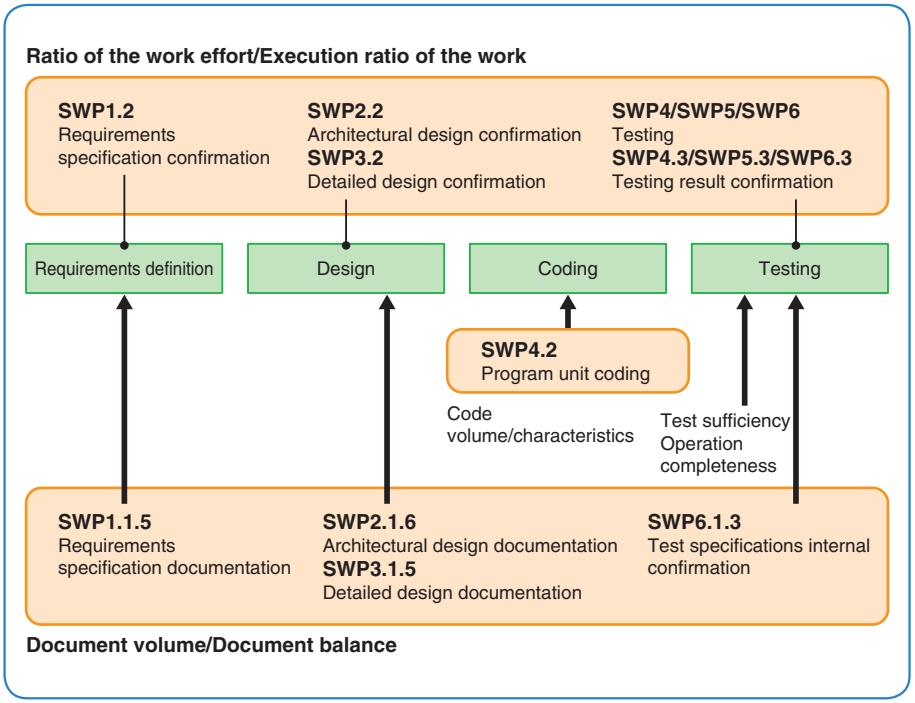


Figure 3-2: Metrics measurement timing

3.4 Process Metrics - Definition and Reference Values

What is a process metric?

A process metric is an evaluation metric for measuring and evaluating work to be done in the software development process. To create high-quality software, effective quality checks and confirmation must be performed during the development process. Process metrics are used to evaluate the sufficiency of this work.

Evaluation target for process metrics

The target to be evaluated with process metrics is the development work itself. Process metrics examine, for example, whether the specifications have been investigated properly or whether the design document has been reviewed sufficiently. This guide explains how to evaluate the appropriateness and sufficiency of the work related to quality checks, including those reviews and tests that are directly related to quality establishment, among the works to be done in the software development.

Viewpoint of process metrics evaluation

The actual meaning of "work," which is the target of a process metric, is in a way very difficult to grasp. Generally, the viewpoints by which work is evaluated are "whether the work was done properly," "whether the work was done sufficiently," and "whether any work has been omitted." For checking the appropriateness and sufficiency of the work, "Ratio of the work effort: Evaluation of relative sufficiency of the work by checking how much process effort (quantity) is assigned in the entire software development work" and "Execution ratio of the work: Evaluation of granularity of the work by checking whether appropriate work process effort (quantity) for the volume of the work target is assigned" are the two viewpoints used to evaluate the quantitative and qualitative sufficiency of the work.

Ratio of the work effort

The ratio of the work effort indicates the sufficiency of the work process effort

by comparing the process effort for reviewing and process effort for testing spent on intermediate and final deliverables against the work process effort spent on the process. For the specification, design, code creation, test creation (preparation), and result creation, the work process effort for each review is evaluated relative to the entire work process effort. For testing, the work process effort is evaluated relative to the entire development work process effort.

For example, Ratio of the Design Review Effort is obtained by dividing the design review work process effort by the process effort required for the entire design work process. Generally, if considerable process effort is spent on the design work, it can be said that the design complexity is correspondingly high, or that the design quantity is correspondingly large, with more design review process effort required. The concept of ratio of the work effort is that it is used as a metric to evaluate the sufficiency of any such work quantity. Of course, it is desirable to spend appropriate process effort on the review and testing to assure quality, relative to the development process effort. That is, a higher ratio of the work effort value is not necessarily good, and the target value must be determined in relation to the quality required for the target system and other factors.

Execution ratio of the work

Execution ratio of the work is a metric that indicates the qualitative sufficiency of the work by evaluating the process effort for reviewing and testing of the intermediate and final deliverables created in the software development process, relative to the software size. The work process effort spent on the review of the specification, design, source code creation, test creation (preparation and result creation), or the process effort spent on testing, is evaluated relative to the entire software size. The software size is represented by Total Lines Of Code (ID: B10, TLOC). For example, Execution Ratio of the Design Review is calculated by dividing the design review process effort by Total Lines Of Code for the system being developed. This value indicates how much process effort per line unit was spent on the design review and can be used as a metric for judging whether a design review appropriate for the software volume has been done. Of course, greater process effort is generally good for reviews and tests for ensuring quality, and a higher Execution ratio of the work value indicates higher quality. However, its target value should also be considered together with the required quality and development cost of the target system.

Evaluation target scope of process metrics

Software is created through roughly four processes including requirements specification investigation (specifications documentation), design, coding, and testing. For details on the process, refer to "Embedded System Development Process Reference" (ESPR). For the process metrics, two concepts, namely, ratio of the work effort and execution ratio of the work, are available. For these four processes, use the following metrics:

Ratio of the Specifications Review Effort/Execution Ratio of the Specifications Review

Ratio of the Design Review Effort/Execution Ratio of the Design Review

Ratio of the Code Review Effort/Execution Ratio of the Code Review

Ratio of the Test Review Effort/Execution Ratio of the Test Review

Ratio of the Test Work Effort/Execution Ratio of the Test Work

Ratio Of the Review Effort/Execution Ratio Of the Review

to check whether the review and testing have been done properly to ensure the quality of the deliverable for each process.

How to read the process metric definition table

The following describes the process metrics, one by one. Each table consists of the following fields:

Abbreviation: Abbreviation for the evaluation metric.

Reference value: Reference value for each system type.

Reference value range: Available range of reference values, taking the adjustment base value into consideration.

Tolerance: Tolerance of the value when used as an evaluation metric.

Calculation method: Formula for calculating the evaluation metric from basic metrics.

ID: ID proprietary to each metric. The ID of each process metric starts with PR.

Name: Name of the metric.

ID	PR10				
Name	Ratio of the Specifications Review Effort				
Abbreviation	RSRE				
Reference value	N	NQ	C	HC	Adjustment base value
	2.00	6.00	10.00	14.00	4.00
Reference value range	0.00 to 6.00	2.00 to 10.00	6.00 to 14.00	10.00 to 18.00	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• Indicates how much process effort is expended on the review of the specifications, relative to the process effort expended to document the specifications. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase, as will the process effort needed to investigate and document the specifications. Therefore, a greater review process effort is not necessarily good, but should always be an appropriate value.				
Calculation method	Review Effort for SPecification/Process Effort for SPecification RSRE = RESP/PESP				
Usage of the metric	• For the review of the specifications, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value (i.e., the process effort is too low relative to the entire project), it indicates that the design review process effort was less than the process effort for the specifications documentation, and so the amount of effort expended on the specifications review may be insufficient. In this case, it is highly likely that items may have been omitted from the review. • On the other hand, if the value of this metric is larger than the reference value (i.e., an unnecessarily long time was taken), it is possible that more effort was expended on the review than on the documentation of the specifications. In this case, it is probable that the project incorporates risk such as, for example, many of the requirements specification being left unfixed, requirements being too complicated, or there being too many functions.				
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review process effort can be somewhat greater than the specifications documentation process effort and will certainly be higher than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project and the number of functions.				

Adjustment base value: Adjustment base value for the reference value.

Measurement unit: Units of the evaluation metric.

Meaning of the metric: Meaning and interpretation of the evaluation metric.

Usage of the metric: Tips on quality control using the evaluation metric.

Remarks: Notes on and ideas of the reference value.

Figure 3-3: How to read the process metric definition table

Process metrics

ID	PR10
Name	Ratio of the Specifications Review Effort
Abbreviation	RSRE

Reference value	N	NQ	C	HC	Adjustment base value
	2.00	6.00	10.00	14.00	4.00
Reference value range	0.00 to 6.00	2.00 to 10.00	6.00 to 14.00	10.00 to 18.00	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• Indicates how much process effort is expended on the review of the specifications, relative to the process effort expended to document the specifications. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase, as will the process effort needed to investigate and document the specifications. Therefore, a greater review process effort is not necessarily good, but should always be an appropriate value.				
Calculation method	Review Effort for SPecification/Process Effort for SPecification RSRE = RESP/PESP				
Usage of the metric	• For the review of the specifications, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value (i.e., the process effort is too low relative to the entire project), it indicates that the design review process effort was less than the process effort for the specifications documentation, and so the amount of effort expended on the specifications review may be insufficient. In this case, it is highly likely that items may have been omitted from the review. • On the other hand, if the value of this metric is larger than the reference value (i.e., an unnecessarily long time was taken), it is possible that more effort was expended on the review than on the documentation of the specifications. In this case, it is probable that the project incorporates risk such as, for example, many of the requirements specification being left unfixed, requirements being too complicated, or there being too many functions.				
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review process effort can be somewhat greater than the specifications documentation process effort and will certainly be higher than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project and the number of functions.				

ID	PR11
Name	Ratio of the Design Review Effort
Abbreviation	RDRE

Reference value	N	NQ	C	HC	Adjustment base value
	2.00	6.00	10.00	14.00	4.00
Reference value range	0.00 to 6.00	2.00 to 10.00	6.00 to 14.00	10.00 to 18.00	

Measurement unit	%
Tolerance	Percentage (two high-order significant digits)
Meaning of the metric	• Indicates how much process effort is expended on the review of the design, relative to the design process effort (process effort required for the design process). • As higher levels of safety and reliability are required for the system, the process effort to be expended on the review will increase, as will the design work itself. Therefore, a greater process effort is not necessarily good, but should always be an appropriate value.
Calculation method	Review Effort for DEsign/Process Effort for DEsign RDRE = REDE/PEDE
Usage of the metric	• For the review of the design, an appropriate process effort, equivalent to or higher than a certain value, is required. • If the value of this metric is less than the reference value, it indicates that the design review process effort is less than the process effort for the design process, and it is highly likely that some items may be omitted from the design review. • On the contrary, if the value of this metric is larger than the reference value, then the amount of effort expended on design review process was relatively higher than the process effort for the design work. This may indicate that the design review required more effort due to project risks, for example, the specifications being ambiguous, the structure being too complicated, or there being too many functions.
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review process effort can be somewhat greater than the design work and it will certainly be greater than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project, the complexity of the structure, abnormality processing, etc.

ID	PR12
Name	Ratio of the Code Review Effort
Abbreviation	RCRE

Reference value	N	NQ	C	HC	Adjustment base value
	2.00	3.50	5.00	6.50	1.50
Reference value range	0.50 to 3.50	2.00 to 5.00	3.50 to 6.50	5.00 to 8.00	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• Indicates how much process effort is expended on the review of the source code, relative to the Process Effort for COde for the source code (coding process effort). • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase, as will the process effort needed to create the source code. Therefore, a higher ratio of the work effort value is not necessarily good, and instead the value should be appropriate.				
Calculation method	Review Effort for COde/Process Effort for COde RCRE = RECO/PECO				
Usage of the metric	• For the review of the source code, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value, the process effort expended on the source code review is too low relative to the coding process effort, and some problems may remain unsolved in the source code. • On the other hand, if the value of this metric is larger than the reference value, the process effort expended on the source code review is too great relative to the coding process effort, and the source code may incorporate project risks such as being difficult to understand or maintain. • A source code review tends to concentrate on particular parts. Do not, therefore, simply trust the value of this metric, but instead check the result with a coverage analysis, that is, by observing the per-module distribution to check that all important parts are reviewed and any items are not omitted.				
Remarks: Interpretation of the reference value	The reference value for this metric is calculated assuming that the software is being newly created. If a high proportion of the software is reused, the review process effort can be somewhat greater than the Process Effort for COde for the source code and the value can be higher than the reference value. Note that this metric value changes with the quality level required by the project, the complexity of the structure, abnormality processing, etc.				

ID	PR13
Name	Ratio of the Test Review Effort
Abbreviation	RTRE

Reference value	N	NQ	C	HC	Adjustment base value
	2.00	3.50	5.00	6.50	1.50
Reference value range	0.50 to 3.50	2.00 to 5.00	3.50 to 6.50	5.00 to 8.00	

Measurement unit	%
Tolerance	Percentage (two high-order significant digits)
Meaning of the metric	• Indicates how much process effort is expended on the review of the test work (i.e., review of the test specifications, items, and result creation), relative to the process effort for testing preparation and confirmation work. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase, as will the number of test review targets to be created. That is, a higher ratio of the work effort value is not necessarily good, and the value should always be appropriate.
Calculation method	Review Effort for Test Preparation/Process Effort for Test Preparation RTRE = RETP/PETP
Usage of the metric	• For the test work review, an appropriate process effort, equivalent to or greater than a certain value, is required. • There are two measuring points; review before executing the testing (test specifications and test script review, etc.) and review after executing the testing (test result and test report review, etc.). For each, apply the metric from the following viewpoints. • Point at which to apply the metric before executing the testing: The aim of performing the review sufficiently before executing the testing is to ensure that failures can be detected. If this metric value is low, the test itself is not checked for the quality and, therefore, failures may fail to be detected. On the contrary, if this metric value is high, the review method may not be good, the specifications review may be performed instead of the test specifications review, or the test specifications themselves are not of good quality, such that the review becomes unduly time-consuming. • Point at which to apply the metric after executing the testing: The aim of applying the metric at this point is to verify the sufficiency of the test. If this metric value is low, the test result judgment will be insufficient, such that failures may have been overlooked. On the contrary, if the value of this metric is high, the testing may incorporate project risks such as low product quality, leading to multiple problems that take much time to resolve. • In all cases, clarify why the review value is not appropriate and apply countermeasures.
Remarks: Interpretation of the reference value	The reference value for this metric is calculated assuming that the software is being newly created. If a high proportion of the software is reused, the review process effort can be relatively greater and the metric value can be higher than the reference value. Note that this metric value changes depending on the level of quality required by the project, the complexity, data variety, etc.

ID	PR14
Name	Ratio of the Test Work Effort
Abbreviation	RTWE

Reference value	N	NQ	C	HC	Adjustment base value
	30.00	35.00	40.00	45.00	5.00
Reference value range	25.00 to 35.00	30.00 to 40.00	35.00 to 45.00	40.00 to 50.00	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• Indicates how much process effort is expended on the testing of the executable code which forms the deliverable (all work from testing preparation, through execution, to confirmation), relative to the Process Effort in TOtal. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the testing will be greater, but it is only a proportion of the entire process effort. That is, a higher ratio of the work effort value is not necessarily good, and the value should be appropriate.				
Calculation method	Process Effort for TEst/Process Effort in TOtal RTWE = PETE/PETO				
Usage of the metric	• For the test, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value, the testing may be efficient, or the testing may be insufficient, such that some failures remain undetected. • On the other hand, if the value of this metric is greater than the reference value, it indicates that the work done before executing the testing (test specifications documentation, etc.) may be insufficient and the test design process effort is high, and that the product quality may be low such that the tests had to be repeated over and over. In this case, there may be project risks such as the test target product being of poor quality, the product being difficult to understand, or the testability being poor.				
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If the high proportion of the design or code is reused, the test process effort can be somewhat greater than the total development process effort and will be higher than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project, the complexity, data variety, etc.				

ID	PR15
Name	Ratio Of the Review Effort
Abbreviation	RORE

Reference value	N	NQ	C	HC	Adjustment base value
	4.00	8.00	12.00	16.00	4.00
Reference value range	0.00 to 8.00	4.00 to 12.00	8.00 to 16.00	12.00 to 20.00	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• Indicates how much process effort is expended on the all the reviews related to the development, relative to the Process Effort in TOtal. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase, but it is only a proportion of the entire process effort. That is, a higher ratio of the work effort value is not necessarily good, and the value should be appropriate.				
Calculation method	Review Effort in TOtal/Process Effort in TOtal RORE = RETO/PETO				
Usage of the metric	• For the review, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value, it indicates that the process effort expended on the review may be less than the total process effort expended on development, such that some failures remain undetected by the review. • On the other hand, if the value of this metric is greater than the reference value, there may be project risks such as the method of proceeding with the review being improper, the product being of poor quality, the product being difficult to understand, or the ease of review being poor. Even if the review process effort is appropriate, the review itself may be of bad quality. Also consider evaluating the quality of the review itself using the review record etc. • Also refer to the tips given in Chapter 4 on how to efficiently proceed with the review.				
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review process effort can be somewhat greater and the value can be higher than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project, complexity, data variety, etc.				

ID	PR20
Name	Execution Ratio of the Specifications Review
Abbreviation	ERSR

Reference value	N	NQ	C	HC	Adjustment base value
	7.20	9.60	12.00	14.40	2.40
Reference value range	4.30 to 9.10	6.70 to 11.50	9.10 to 13.80	11.50 to 16.30	
Measurement unit	Man-hours/KLOC				
Tolerance	Three high-order significant digits				
Meaning of the metric	• Indicates how much process effort is expended on the specifications review, relative to the project scale. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase.				
Calculation method	Review Effort for SPecification/Total Lines Of Code ERSR = RESP/TLOC				
Usage of the metric	• For the review of the specifications, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value (review process effort per scale is also too less), the specifications confirmation may be insufficient and the review may not have been done thoroughly. Therefore, the review details must be checked again. • On the other hand, if the value of this metric is greater than the reference value (process effort is unnecessarily high), there may be project risks such as a large portion of the requirements specification being unfixed, such that the specifications review turns into the investigation of the specifications, requirements being too complicated, or there being too many functions. Therefore, it is necessary to check whether the review results have, in fact, been reflected on the specifications.				
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review target may be small relative to the scale of the software, and the estimated value of the metric can be lower than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project and the number of functions.				

ID	PR21
Name	Execution Ratio of the Design Review
Abbreviation	ERDR

Reference value	N	NQ	C	HC	Adjustment base value
	7.20	9.60	12.00	14.40	2.40
Reference value range	4.80 to 9.60	7.20 to 12.00	9.60 to 14.40	12.00 to 16.80	

Measurement unit	Man-hours/KLOC
Tolerance	Three high-order significant digits
Meaning of the metric	• Indicates how much process effort is expended on the design review, relative to the project scale. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase.
Calculation method	Review Effort for DEsign/Total Lines Of Code ERDR = REDE/TLOC
Usage of the metric	• For the review of the design, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value, it indicates that the design review may not be sufficiently thorough, some items may have been missed from the reviewed design. • On the other hand, if the value of this metric is greater than the reference value, there may be project risks, for example, the design being difficult to review because of the ambiguous specifications, the design structure being too complicated, or there being too many functions.
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review target may be small relative to the scale of the software, and the estimated value of the metric can be lower than the reference value. • Note that this metric value changes depending on the level of quality required by the project, the complexity of the structure, abnormality processing, etc.

ID	PR22
Name	Execution Ratio of the Code Review
Abbreviation	ERCR

Reference value	N	NQ	C	HC	Adjustment base value
	3.60	4.80	6.00	7.20	1.20
Reference value range	2.40 to 4.80	3.60 to 6.00	4.80 to 7.20	6.00 to 8.40	
Measurement unit	Man-hours/KLOC				
Tolerance	Three high-order significant digits				
Meaning of the metric	• Indicates how much process effort is expended on the source code review, relative to the project scale. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase.				
Calculation method	Review Effort for COde/Total Lines Of Code ERCR = RECO/TLOC				
Usage of the metric	• For the review of the source code, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value, the required sections of the target source code are not thoroughly checked and some failures are likely to go undetected. • On the other hand, if the value of this metric is higher than the reference value, there may be project risks such as the product being difficult to understand, the maintainability being poor, or the review efficiency being low.				
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review target may be small relative to the scale of the software, and the estimated value of the metric can be lower than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project, the complexity of the structure, abnormality processing, etc.				

ID	PR23
Name	Execution Ratio of the Test Review
Abbreviation	ERTR

Reference value	N	NQ	C	HC	Adjustment base value
	6.00	8.00	10.00	12.00	2.00
Reference value range	4.00 to 8.00	6.00 to 10.00	8.00 to 12.00	10.00 to 14.00	

Measurement unit	Man-hours/KLOC
Tolerance	Three high-order significant digits
Meaning of the metric	• Indicates how much process effort is expended on the test work review, relative to the project scale. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase.
Calculation method	Review Effort for Test Preparation/Total Lines Of Code ERTR = RETP/TLOC
Usage of the metric	• If the value of this metric is less than the reference value, the test work review may be insufficient and the test may include undetected problems. • On the other hand, if the value of this metric is higher than the reference value, the test specifications or test report may be of bad quality and review may be time-consuming.
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • If a high proportion of the software is reused, the review target may be small relative to the scale of the project, and the estimated value of the metric can be lower than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project, the complexity, data variety, etc.

ID	PR24
Name	Execution Ratio of the Test Work
Abbreviation	ERTW

Reference value	N	NQ	C	HC	Adjustment base value
	34.00	51.00	68.00	85.00	17.00
Reference value range	17.0 to 51.0	34.0 to 68.0	51.0 to 85.0	68.0 to 102.0	
Measurement unit	Man-hours/KLOC				
Tolerance	Three high-order significant digits				
Meaning of the metric	• Indicates how much process effort is expended on the test work (from testing preparation, through execution, to confirmation) for a particular scale, relative to the overall scale of the project. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the testing will be greater.				
Calculation method	Process Effort for TEst/Total Lines Of Code ERTW = PETE/TLOC				
Usage of the metric	• For the testing, an appropriate process effort, equivalent to or greater than a certain value, is required. • If the value of this metric is less than the reference value, the test work may be insufficient relative to the scale of the development target, and some problems may go undetected consequently. • On the other hand, if the value of this metric is greater than the reference value, the testing will take more time and there may be project risks such as the target product being of poor quality, the product being difficult to understand, or the testability being low.				
Remarks: Interpretation of the reference value	• When the total productivity per man-month is 1 KLOC/man-month (1 man–month = 155 man-hours), NQ assumes that the test phase takes approximately 1/3 of the entire process effort and Execution Ratio of the Test Work is based on approximately 50 man-hours (155 × 0.3). • This metric is not influenced by the reuse ratio because the testing is done for the entire source code as a rule, regardless of whether it is reused. • This metric is normalized with "source code scale = total number of lines of the source code" for convenience, while measuring the relative quantity of the object code ensures more precise evaluation. When measurement is possible, it is a good idea to make an evaluation based on the value for total number of lines of the source code, excluding the total number of comments and blank lines.				

ID	PR25
Name	Execution Ratio Of the Review
Abbreviation	EROR

Reference value	N	NQ	C	HC	Adjustment base value
	24.00	32.00	40.00	48.00	8.00
Reference value range	16.0 to 32.0	24.0 to 40.0	32.0 to 48.0	40.0 to 56.0	
Measurement unit	Man-hours/KLOC				
Tolerance	Three high-order significant digits				
Meaning of the metric	• Indicates how much process effort is expended on all the development-related reviews relative to the project scale. • As higher levels of safety and reliability are demanded of a system, the process effort to be expended on the review will increase.				
Calculation method	Review Effort in TOtal/Total Lines Of Code EROR = RETO/TLOC				
Usage of the metric	• For the review, an appropriate process effort, equivalent to or higher than a certain value, is required. • If the value of this metric is less than the reference value, the process effort for review may be insufficient in comparison with the size of the software such that some failures go undetected. • On the other hand, if the value of this metric is greater than the reference value, there may be project risks such as the target product being of bad quality, the product being difficult to understand, or the ease of review being poor. Alternatively, the method of proceeding with the review may be improper. Even if the review process effort is appropriate, the review itself may be of bad quality. Also consider evaluating the quality of the review itself using the review record etc. • Also refer to the tips given in Chapter 4 on how to efficiently proceed with the review.				
Remarks: Interpretation of the reference value	• The reference value for this metric is calculated assuming that the software is being newly created. • For NQ software, the reference value is calculated assuming that 3 man-days are required for the review of the specifications to the code and 1 man-day is required for the test review for the software of 1 KLOC. • If a high proportion of the software is reused, the review process effort can be somewhat greater and the value can be higher than the reference value. • Note that the value of this metric changes depending on the level of quality required by the project, the complexity, data variety, etc.				

3.5 Product Metrics - Definition and Reference Values

Product metrics

In the software development process, deliverables (documents) such as design document, source code, and test specifications are created. Product metrics are used to evaluate the quality of these intermediate deliverables created in such a development process and deliverables created in the final stage.

Evaluation target for product metrics

As described above, in many of embedded software development projects, intermediate deliverables such as documents including specifications, design documents, source code, and test specifications, as well as test data, become the targets of product metrics. The final versions of materials almost the same as these are prepared as the deliverables in the final stage in most cases, so the deliverables are also targeted. In addition, software products as final deliverables and test work and test results as software products or system products are also targets of evaluation by the product metrics.

Viewpoints and types of product metrics evaluation

Though the deliverables which are the evaluation targets of the product metrics can be seen (i.e., visible), they quickly become difficult to comprehend when we try to evaluate their quality. This guide measures those factors that are relatively easy to measure (e.g., the volume such as the number of pages or lines, failure count, etc.) among the characteristics of the product and evaluates them from the viewpoint of whether a factor directly linked to quality is reflected.

The product metrics can be classified into those for documents, source code, and tests depending on the measurement target. ISO/IEC9126 classifies documents and source code into an internal quality measurement target and tests as part of the external quality measurement target with viewpoints for measurement. Refer to this as necessary.

Evaluation target scope of product metrics

Modern embedded software is a composite of multiple functional entities. For example, when an automobile is considered from the software aspect, various functions such as the engine, brakes, and air-conditioner controls all have integrated, dedicated software installed in the electronic computer unit (ECU) controlling each function. These software modules communicate with each other via a communications function, such as a dedicated network. The characteristics demanded of the software depend on the function and part within the same automobile. Therefore, in product metrics evaluation, it is necessary to identify the factors constituting the system and evaluate the scope that can be considered as having identical quality characteristics.

Overview of product metrics

(1) Document evaluation metrics

Document evaluation metrics are used to evaluate the quality of documents such as specifications and design documents among the created intermediate and final deliverables created during the software development. Documents created as part of embedded software development include requirements specification, design documents, and test specifications.

There are the following two types of document evaluation metrics:

(i) Document volume evaluation metric for purely evaluating the volume per document scale

(ii) Document balance evaluation metric for evaluating the balance of the contents described in a document

For details on the content to be described in the requirements specification, design documents, and test specifications targeted by the document evaluation metrics, as well as how these documents are positioned, refer to "Embedded System Development Process Reference" (ESPR).

Document volume evaluation metrics

As the first metric for evaluating whether a document is appropriate from the viewpoint of quality, a metric to determine whether the volume (the number of pages) is appropriate can be considered. However, the number of pages in such a document varies depending on the size of the software being developed as well as the document format.

Therefore, simply counting the number of pages in the document is meaningless.

The document volume evaluation metrics introduced in this guide obtain a rough count of the number of pages in a document, regardless of the format that is used, by assuming 2,000 characters (40 characters x 50 lines)(*) of text, figure, and table information as being equal to one page as a basic metric for the number of pages. And then, by dividing the counted number of pages by the software size, the volume of the document per unit scale (1 KLOC) is evaluated.

This guide normalizes the document volume per unit scale. However, the code size of the base system naturally depends on the system characteristics (e.g., whether the code consists primarily of the GUI or control logic). Therefore, when evaluating the document volume, the characteristics of each system must be fully considered.

Document balance evaluation metric

Even when the document volume is appropriate, if any necessary information is missing, the development will be considerable affected. This guide provides document balance evaluation metrics that indicate the items to be described as standard in the requirements specification, design document, and test specifications, and how much information is described for each item in a document, with the quantity of information in the entire document being assumed to be 100. For example, by measuring the percentage of description of the functional requirements as well as non-functional requirements in the requirements specification, the metrics aim to evaluate the document contents as described in the requirements specification. Note that the document description items used here are based upon the definitions in ESPR.

(2) Code evaluation metrics

The code evaluation metrics directly observe the source code which is the final deliverable from the coding of embedded software and evaluate its quality.

The evaluation metrics for measuring the source code volume by counting the number of lines include the logical number of lines which counts the lines composing the source program excluding comments, and the physical number of lines which simply counts all the lines. This guide uses the latter as the source code evaluation metric because an easier measurement method is preferred. With this counting method, however, the number of source code lines differs by a factor of two of three depending on how the code was written. To ensure the effectiveness of the source code metrics, for example, you can

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

create a coding rule for the project based on the ESCR to unify the writing style and control the project so that the ratio of comments is similar regardless of who writes the code.

The source code for a given product is rarely written in a single language. However, this guide basically provides the reference values assuming that the source code is written in C, which is popular for embedded software development. When using another language such as assembly language or C++, use a coefficient that is based on your organization's experience.

There are the following two types of code evaluation metrics:

(i) Code volume evaluation metric to evaluate the volume per file or function

(ii) Code characteristics evaluation metric to evaluate the balance of the source code description

Column

Measuring the software size

The size of the software can be measured as a number of function points (FP) or number of lines (LOC). The function point (FP) measurement focuses on the application functionality and measures the program size according to the I/O and application interface. The function point has some advantages that it can be used regardless of the programming language and that it can determine the size before coding. However, multiple measurement methods, all of which involve complicated measurement rules, are proposed, and the result may differ greatly depending on who makes the measurement and how. Therefore, special tools or experienced specialists are required. On the contrary, the number of lines (LOC) is visually clear and mechanically measurable, and does not vary regardless of who measures it, using the functions of a text editor or some similar means.

Code volume evaluation metrics

One method of determining whether the source code is appropriate from the viewpoint of quality is to check its volume against a metric. If the number of source code lines per unit exceeds a certain value, the maintainability and understandability may be affected.

There are the following two types of code volume evaluation metrics:

(i) File Lines Of Code: Scale (volume) of the source code described in each source file

(ii) Module Lines Of Code: Scale (volume) of the source code per function

The total of the File Lines Of Code values is used as the Total Lines Of Code which is an alternative metric to indicate the scale of the program itself.

Code characteristics evaluation metrics

Even when the source code volume is less than a certain value, if it is biased in description, the readability and maintainability in review is deteriorated and quality of the source code is affected. Therefore, this guide uses the code characteristics evaluation metrics focusing on the characteristics of the source code. For example, comment lines are a valuable information source and provide a supplementary explanation to the reader (reviewer or maintenance engineer) on what the source code function should do. By evaluating Ratio Of Comment Line to determine if any necessary supplementary information is missing or redundant, in accordance with the coding rule defined by the organization, any excess or shortfall of this information can be judged to some extent and the evaluation result can be used to instruct a re-check in the review.

There are the following three types of code characteristics evaluation metrics:

(i) Ratio Of Control Statement

(ii) Ratio Of Comment Line

(iii) Ratio of Deviation of Coding Rules

(3) Test evaluation metrics

The test evaluation metrics are used to evaluate sufficiency of the test actually running the object code, which is the final deliverable of coding.

The testing is executed for several different purposes, which can be roughly classified into the following two characteristics:

(i) Checking whether the object code runs differently from the specifications (function, performance, etc.) required for the system (fault detection)

(ii) Checking whether the object code satisfies the specifications (function, performance, etc.) required for the system (operation check).

These two characteristics seem to contradict each other but both are important. When designing a test, determine which is appropriate depending on the purpose and situation. This guide indirectly evaluates the quality of the software by checking the sufficiency of the test from this viewpoint.

There are the following two types of test evaluation metrics:

(i) Test sufficiency to evaluate whether enough tests have been done

(ii) Operation completeness to evaluate the repair status of any failures detected by the testing

Test sufficiency evaluation metrics

One of the metrics for evaluating a test from the viewpoint of sufficiency is the quantity of data produced as the result of the testing. For example, checking if the number of tested items is sufficient relative to the software scale is aimed at evaluating the test sufficiency by determining if the test volume is sufficient relative to the source code volume. Of course, the number of test items varies depending on the test phase, size and complexity of the software being developed, and the number of I/Os. Also, the definition of a "single item" also depends on the project or the person in charge. Therefore, simply counting the number of test items is meaningless.

This guide uses the item count at the general subsection or script level for the standard for the reference value. You should determine the standard in accordance with the project you are measuring.

There are the following two types of test sufficiency evaluation metrics:

(i) Density Of Test Items

(ii) Ratio Of Fault detection in Comparison

Operation completeness evaluation metrics

Even if failures are detected by the testing, whether to fix all of them depends on the project. The operation completeness evaluation metrics are used to evaluate the stability of the quality of the software after testing. These metrics evaluate how many of the failures detected in the test were fixed to estimate the software risk after shipment. Therefore, the operation completeness evaluation metrics only use a single metric; Ratio Of Fault Elimination.

How to read the product metrics definition table

The following describes the product metrics, one by one. Each table consists of the following fields:

Abbreviation: Abbreviation for the evaluation metric.

Reference value: Reference value for each system type.

Reference value range: Available range of reference values, taking the adjustment base value into consideration.

Tolerance: Tolerance of the value when used as an evaluation metric.

Calculation method: Formula for calculating the evaluation metric from basic metrics.

ID: ID proprietary to each metric. The ID of each process metric starts with PD.

Name: Name of the metric.

ID	PD10				
Name	Ratio of the Specifications Document Volume				
Abbreviation	RSDV				
Reference value	N	NQ	C	HC	Adjustment base value
	3	7	11	15	4.00
Reference value range	0.00 to 7.00	3.00 to 11.00	7.00 to 15.00	11.00 to 19.00	
Measurement unit	Page/KLOC				
Tolerance	Two high-order significant digits (e.g., 14 pages/KLOC for 14.8 pages/KLOC)				
Meaning of the metric	• This metric indicates requirements specification volume relative to the project scale. • If this metric value is low, the documentation may be insufficient.				
Calculation method	Volume Of the Specifications Document/Total Lines Of Code RSDV = VOSD/TLOC				
Usage of the metric	• If this metric value is less than the reference value, the document may be insufficient. A document may be insufficient because (1) the specifications were not investigated sufficiently, so the document is not ready, or (2) the specifications were investigated sufficiently but the work result was not sufficiently arranged. In all cases, review the document again to check the validity of its contents.				
Remarks: Interpretation of the reference value	• The reference value for this metric assumes the requirements specification for normal level software (N level) for which the required level of quality is not so high, and assumes that approximately three pages of specifications should be created per 1 KLOC of the source code, mainly for describing the information on software positioning and functionality, as well as the peripheral hardware. As the required quality level rises, a more detailed description is required for the specifications. Therefore, the reference value is defined so that the number of requirements specification pages increases as the system type rises to NQ, C, and HC. • In addition, the volume changes depending on the system characteristics. For example, a system with more user interfaces tends to require more volume for the requirements specification.				

Adjustment base value: Adjustment base value for the reference value.

Measurement unit: Units of the evaluation metric.

Meaning of the metric: Meaning and interpretation of the evaluation metric.

Usage of the metric: Tips on quality control using the evaluation metric.

Remarks: Notes on and ideas of the reference value.

Figure 3-4: How to read the product metrics definition table

Product metrics

ID	PD10
Name	Ratio of the Specifications Document Volume
Abbreviation	RSDV

Reference value	N	NQ	C	HC	Adjustment base value
	3	7	11	15	4.00
Reference value range	0.00 to 7.00	3.00 to 11.00	7.00 to 15.00	11.00 to 19.00	
Measurement unit	Page/KLOC				
Tolerance	Two high-order significant digits (e.g., 14 pages/KLOC for 14.8 pages/KLOC)				
Meaning of the metric	• This metric indicates requirements specification volume relative to the project scale. • If this metric value is low, the documentation may be insufficient.				
Calculation method	Volume Of the Specifications Document/Total Lines Of Code RSDV = VOSD/TLOC				
Usage of the metric	• If this metric value is less than the reference value, the document may be insufficient. A document may be insufficient because (1) the specifications were not investigated sufficiently, so the document is not ready, or (2) the specifications were investigated sufficiently but the work result was not sufficiently arranged. In all cases, review the document again to check the validity of its contents.				
Remarks: Interpretation of the reference value	• The reference value for this metric assumes the requirements specification for normal level software (N level) for which the required level of quality is not so high, and assumes that approximately three pages of specifications should be created per 1 KLOC of the source code, mainly for describing the information on software positioning and functionality, as well as the peripheral hardware. As the required quality level rises, a more detailed description is required for the specifications. Therefore, the reference value is defined so that the number of requirements specification pages increases as the system type rises to NQ, C, and HC. • In addition, the volume changes depending on the system characteristics. For example, a system with more user interfaces tends to require more volume for the requirements specification.				

ID	PD11
Name	Ratio of the Design Document Volume
Abbreviation	RDDV

Reference value	N	NQ	C	HC	Adjustment base value
	9	19	29	39	10.00
Reference value range	0.00 to 19.00	9.00 to 29.00	19.00 to 39.00	29.00 to 49.00	
Measurement unit	Page/KLOC				
Tolerance	Two high-order significant digits (e.g., 14 pages/KLOC for 14.8 pages/KLOC)				
Meaning of the metric	• This metric indicates the design documents volume relative to the project scale. • If this metric value is low, documentation may be insufficient.				
Calculation method	Volume Of the Design Document/Total Lines Of Code RDDV = VODD/TLOC				
Usage of the metric	• If this metric value is less than the reference value, the design document may be insufficient. A design document may be insufficient because (1) the design was not investigated sufficiently, so the document is not ready, or (2) the design was investigated sufficiently but the work result is not sufficiently arranged. In all cases, review the document again to check the validity of its contents.				
Remarks: Interpretation of the reference value	• The reference value of this metric assumes the requirements specification for normal level software (Type-1: N) for which the required level of quality is not so high, and defines the proper volume of the design document as being approximately three times that of the requirements specification if the contents of a single page of the requirements specification are properly reflected on the design. • In addition, the volume changes depending on the system characteristics. For example, a system with more user interfaces tends to require more volume for the design document.				

ID	PD12
Name	Ratio of the Test Document Volume
Abbreviation	RTDV

Reference value	N	NQ	C	HC	Adjustment base value
	9	19	29	39	10.00
Reference value range	0.00 to 19.00	9.00 to 29.00	19.00 to 39.00	29.00 to 49.00	
Measurement unit	Page/KLOC				
Tolerance	Two high-order significant digits (e.g., 14 pages/KLOC for 14.8 pages/KLOC)				
Meaning of the metric	• This metric indicates the test-related specifications volume relative to the project scale. • If this metric value is low, the documentation may be insufficient.				
Calculation method	Volume Of the Test Document/Total Lines Of Code RTDV = VOTD/TLOC				
Usage of the metric	• If this metric value is less than the reference value, then the test document may be insufficient. A test document may be insufficient because (1) the test items were not investigated sufficiently, so the document is not ready, or (2) the test items were investigated sufficiently but the work result is not sufficiently arranged. In all cases, review the document again to check the validity of its contents.				
Remarks: Interpretation of the reference value	• The reference value for this metric assumes the Requirements Specification for normal level software (Type-1: N) for which the required level of quality is not so high, and defines the proper volume of the test specifications as being approximately three times that of the requirements specification, assuming that variations including exceptions are tested for the software. • In addition, the volume changes depending on the system characteristics. For example, a system with more user interfaces tends to require more volume for the test specifications.				

Column

Influence of reuse ratio on metrics value

Recently, the development of software from scratch has become quite rare with the reuse of existing software assets and the deriving of new software from base software becoming common. In such software development, the quality must be controlled by considering both the reused part(s) and the entire software. Because the influence of the reuse ratio on the software development varies with each project or software, the reference value for Execution Ratio Of the Review is difficult to uniformly determine. Therefore, this guide provides reference values for entirely new software development as a guideline. It is recommended that the provided reference values be changed considering the project situation to reflect the influence of the reuse ratio on the software development.

ID	PD20
Name	Balance of the Specifications Document Description(1)
Abbreviation	BSDD

Reference value/reference value range	N	NQ	C	HC	Adjustment base value
	See the table on the next page.				None
Measurement unit	Percentage of description items of each document				
Tolerance	Percentage rounded to the nearest 10%				
Meaning of the metric	• This metric indicates how much information is described for each item which should be mentioned in the requirements specification relative to the entire volume of the requirements specification, to evaluate the sufficiency of the described contents of the document.				
Calculation method	• Number of pages of each part in requirements specification/total number of pages in requirements specification • For the contents of each part of the requirements specification, measure its percentage relative to the entire volume of the requirements specification. • To measure the description volume, for example, count the number of pages for "R2: Target user and usage description," and divide the measured number of pages by the total number of pages to calculate the percentage of the description for the item. Note that the items which should be mentioned in the requirements specification are selected according to "SYP1.1 System Requirements Specification Documentation" and "SWP1.1 Software Requirements Specification Documentation" in ESPR.				
Usage of the metric	For this metric value, if the value for a description item is extremely low compared with the description balance indicated by the reference values listed in the table on the next page, we can say that the description and investigation of that item is not sufficient, and the document must be reviewed or investigated again.				
Remarks: Interpretation of the reference value	• For measurement and evaluation of this metric, items which should be mentioned in the requirements specification, as well as the approximate description volume of each item, assuming the entire volume of the document to be 100 are shown. For example, the reference values for the requirements specification define that the volume of "R2: Target user and usage description" should be approximately 5% of the entire volume and the volume of "R3: Description volume for operation environment conditions" should be approximately 10% of the entire volume. According to these values, measure the items that are described in the target document (e.g., requirements specification) of whatever volume for each, and evaluate the appropriateness of the document contents. • Note that description ratio for each item varies according to the characteristics of the system. The reference values reflect the ratio for the standard system. For example, the ratio of R5 is higher for a system for which safety is required and the ratio of R7 is higher for a system in which there are many exceptions to handle. Check the items on which to focus in system characteristics profiling and reflect the check result on the target value setting.				

(1) **BDD**: Balance of the Document Description: Proposed by M. Hirayama and S. Yoshizawa

Document	Item No.	Contents	Reference %
Requirements specification	R1.	Entire description volume	100
	R2.	Target user and usage description	5
	R3.	Description volume for operation environment conditions	10
	R4.	Description volume for main functions	40
	R5.	Description volume for safety and non-functional requirements	30
	R6.	Description volume for overall system structure	10
	R7.	Description volume for exception handling	5

Column

Functional vs. non-functional requirements

When developing software, it is required to first clarify what is being developed. To this end, a task called requirements analysis and definition must be done. In requirements analysis and definition, the functional requirements that describe the software functions to be realized and non-functional requirements other than the functional aspects including performance, usability, and safety are clarified.

The functional and non-functional requirements that the software to be developed should have and the ratio between the two is highly dependent on individual systems. For example, for embedded software used in a product used by the general public, non-functional aspects such as usability and reliability must be sufficiently investigated in addition to the software functionality itself. For such software, matters that should be described as non-functional requirements increase and the balance of the description in the requirements specification should be adjusted.

ID	PD21
Name	Balance of the Design Document Description
Abbreviation	BDDD

Reference value/reference value range	N	NQ	C	HC	Adjustment base value
	See the table on the next page.				None
Measurement unit	Percentage of description items of each document				
Tolerance	Percentage rounded to the nearest 10%				
Meaning of the metric	• This metric indicates how much information is described for each item which should be mentioned in the design document relative to the entire volume of the design document, to evaluate sufficiency of the described contents of the design document.				
Calculation method	• Number of pages of each part in design document/Total number of pages in design document • For the contents of each part of the design document, measure its percentage relative to the entire volume of the design document. To measure the description volume, for example, count the number of pages for "D2: Description volume for overall system structure" and divide the obtained number of pages by the total number of pages to calculate the percentage of the description for the item. Note that the items to be mentioned in the design document are selected according to "SYP2.1 System Architectural Specifications Documentation" and "SWP2.1 Software Architectural Design Documentation" in ESPR.				
Usage of the metric	• For this metric value, if the value for a description item is extremely low relative to the description balance indicated by the reference values listed in the table on the next page, it can be judged that the description and investigation of that item is not sufficient, and the document must be reviewed or investigated again.				
Remarks: Interpretation of the reference value	• For the measurement and evaluation of this metric, items which should be mentioned in the design document, as well as the approximate description volume of each item, assuming the entire volume of the document to be 100 are shown. For example, the reference values for the design specifications define that the volume of "D2: Description volume for overall system structure" should be approximately 5% of the entire volume and the volume of "D3: Description volume for functional block structure" should also be approximately 5% of the entire volume. According to these values, measure the items mentioned in the document to be evaluated (e.g., design document) in whichever volume for each, and evaluate appropriateness of the contents of the document. • Note that the description ratio of each item varies according to the characteristics of the system. The reference values reflect the ratio for a standard system. If the balance target value is changed for Balance of the Specifications Document Description, you must also change the target value of this Balance of the Design Document Description accordingly.				

Document	Item No.	Contents	Reference %
Design document	D1.	Entire description volume	100
	D2.	Description volume for overall system structure	5
	D3.	Description volume for functional block structure	5
	D4.	Description volume for functional block details	50
	D5.	Description volume for interface data	20
	D6.	Description volume for exception handling	20

"D1: Entire description volume" is defined as being approximately three times "R1: Entire description volume" (of the requirements specification).

Column

How to represent design details

The design of software involves considering how the requirements (functional and non-functional) required for the software should be implemented, and investigating and determining the static structure and dynamic mechanism of the software. The static structure and dynamic mechanism of the software are often represented using diagrams and tables which can show them most clearly, rather than a natural language. Recently, the concept of design modeling is becoming popular in the software design world, and is one concept for abstractively arranging the software design using predefined diagram and table conventions. Using such diagrams and tables to represent the software design has the advantage of representing the design structure intuitively or logically, while it cannot represent the details and design basis to the full extent. It is required to investigate, in advance, the contents to be mentioned in the design document, including the representation systems that are used in order to precisely communicate the software design details.

ID	PD22
Name	Balance of the Test Document Description
Abbreviation	BTDD

Reference value/reference value range	N	NQ	C	HC	Adjustment base value
	See the table on the next page.				None
Measurement unit	Percentage of description items of each document				
Tolerance	Percentage rounded to the nearest 10%				
Meaning of the metric	• This metric indicates the number of items that should be mentioned in test specifications relative to the entire volume of the test specifications, to evaluate the sufficiency of the described contents of the document.				
Calculation method	• Number of pages in each part in test specifications/Total number of pages in the test specifications • For the contents of each part of test specifications, measure its percentage relative to the entire volume of the test specifications. • To measure the description volume, for example, count the number of pages for "T2: Description of test environment" and divide the obtained number of pages by the total number of pages to calculate the percentage of the description for the item. Note that the items to be mentioned in the test specifications are selected according to "SYP4.1 System Qualification Testing Preparation," "SWP5.1 Software Integration Testing Preparation," and "SWP6.1 Software Qualification Testing Preparation" in ESPR.				
Usage of the metric	• If the value for a description item is extremely low relative to the description balance indicated by the reference values listed in the table on the next page, we can say that the description and investigation of that item is not sufficient, and the document must be reviewed or investigated again.				
Measurement tips	• For the volume of each description item, a rough quantity with the number of pages (such as approximately 1.5 pages) or the number of lines is enough. In some cases, a percentage described with physical values such as the area or length of description area may be used.				
Remarks: Interpretation of the reference value	• For the measurement and evaluation of this metric, items which should be mentioned in the test specifications, as well as the approximate description volume for each item, assuming the entire volume of the document to be 100 are shown. For example, the reference values for the test specifications define that the volume of "T2: Description of test environment" should be approximately 5% of the entire volume and the volume of "T3: Description of test procedure and conditions" should be approximately 10% of the entire volume. According to these values, measure the items that are mentioned in the document to be evaluated (e.g., test specifications) in whichever volume, and evaluate the appropriateness of the contents of the document. • Note that the description ratio of each item varies with the characteristics of the system. The reference values reflect the ratio for a standard system. If the balance target value is changed for Balance of the Specifications Document Description, you must also change the target value of this Balance of the Test Document Description accordingly.				

Document	Item No.	Contents	Reference value
Test specifications	T1.	Entire description volume	R1*3
	T2.	Description of test environment	5
	T3.	Description of test procedure and conditions	10
	T4.	Description of normal system	35
	T5.	Description of abnormal system and exception handling	45
	T6.	Description of test completion criteria	5

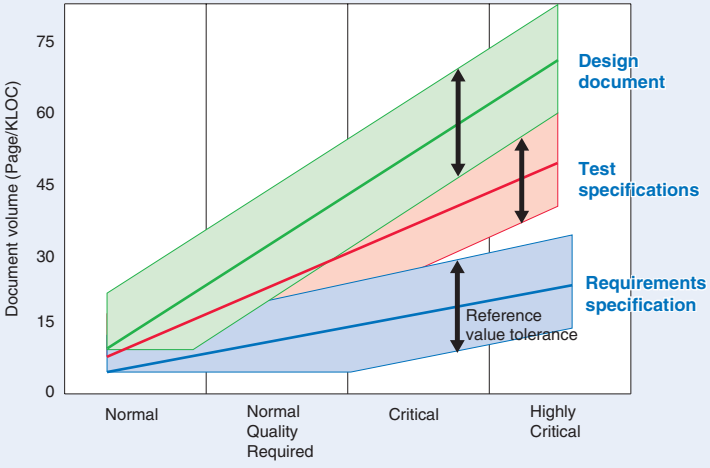
"T1: Entire description volume" is defined as being approximately three times "R1: Entire description volume" (of the requirements specification).

Column

Numeric metrics

The numeric metrics generally incur measurement errors for many reasons. It is desirable to use the reference values in this guide by modifying them as necessary in consideration of a tolerance of ±15%, as shown below.

Note that some metrics do not simply increase in proportion to the software size.



ID	PD30
Name	File Lines Of Code
Abbreviation	FLOC

Reference value/reference value range	N	NQ	C	HC	Must be equivalent to or lower than this value.
	2.00	2.00	2.00	2.00	
Measurement unit	KLOC				
Tolerance	Two high-order significant digits (e.g., 1.2 KLOC for 1231 LOC)				
Meaning of the metric	• This metric indicates the size of the software targeted for quality measurement and evaluation. • If this value is large, it indicates that the target software size is large. • Measure the number of lines per file and check if the file contains more than a certain number of lines. If the number of lines per file is too large, the readability and maintainability may suffer. Note that this metric is not influenced by the type of the system characteristics profile.				
Calculation method	File Lines Of Code: Use File Lines Of Code of the basic metrics as is. FLOC = FLOC				
Usage of the metric	• This metric targets a file with a large value for evaluation. • If the number of lines per file is much greater than the reference value, the maintainability may suffer and a source code review must be performed as early as possible. • Also, the total of the values for this metric (total File Lines Of Code) is used as the alternative metric to indicate the software size. If this measured value differs considerably from the software size (the total number of lines) grasped intuitively by the developer, project manager, or leader, the developer may have misunderstood the specifications or an error may have occurred with the internal processing, and a source code review needs to be performed as early as possible. • Note that this metric value assumes the use of C as the programming language. For details on how to convert the value when using multiple languages, refer to File Lines Of Code (ID=B11, FLOC) of the basic metrics.				
Remarks: Interpretation of the reference value	• There is a limit to the extent to which a human can understand the logic. For the text of a single chapter (be it a novel or a technical document), it is easy to follow the logic. The reference value for this metric assumes the text of a single chapter or two; 80 lines x 25 pages (2,000 lines).				

ID	PD31
Name	Module Lines Of Code
Abbreviation	MLOC

Reference value/reference value range	N	NQ	C	HC	Must be equivalent to or lower than this value.
	160.0	160.0	160.0	160.0	
Measurement unit	LOC				
Tolerance	Two high-order significant digits (e.g., 230 LOC for 231 LOC)				
Meaning of the metric	• This metric indicates the scale of a function, which is a unit for processing. • Measure the number of lines per function and check if the function contains more lines than a certain value. If the number of lines per function is too large, the readability and maintainability may suffer. Note that this metric is not influenced by the type of the system characteristics profile type.				
Calculation method	Module Lines Of Code: Use Module Lines Of Code of the basic metrics as is. MLOC = MLOC				
Usage of the metric	• This metric targets a function with a large value for evaluation. • Extract and measure those functions for which the number of lines exceeds the certain number of lines. If it is difficult to measure all the functions, it is a good idea to select and measure those functions which seem large. • If the number of lines per function is much greater than the reference value, the maintainability may suffer. If this is the case, it is required to consider measures such as dividing the function into multiple modules or performing a source code review as early as possible.				
Remarks: Interpretation of the reference value	• There is a limit to the extent to which a human can understand the logic. Two printed pages of source code are appropriate for review. The reference value for this metric assumes 80 lines x 2 pages.				

ID	PD32
Name	Ratio Of Control Statement
Abbreviation	ROCS

Reference value	N	NQ	C	HC	Adjustment base value
	35	30	25	20	5.00
Reference value range	30.00 to 40.00	25.00 to 35.00	20.00 to 30.00	15.00 to 25.00	
Measurement unit	%				
Tolerance	Two high-order significant digits				
Meaning of the metric	• This metric indicates the number of control statements in the measurement target file relative to the total number of lines of source code. Ratio Of Control Statement is proposed as an alternative metric because measuring the complexity of the source code is difficult. • High complexity infers that the source code description is complex, making the code difficult to maintain and understand, as well as of low quality. If this metric value is high, it indicates that the code may contain many branches and other logics and the design or code itself is complex, resulting in possible problems with reliability and maintainability.				
Calculation method	Number Of Control Statement/Total Lines Of Code ROCS = NOCS/TLOC				
Usage of the metric	• If the Number Of Control Statement value is high, the source code may be complicated, making it difficult to maintain and read, and may contain many bugs. When the value is too high, review the design, check for reliability in the source code review, and/or cover all the paths in the testing. • The complexity required of the software varies depending on the characteristics of the target system. Based on the reference value, define appropriate values for individual organizations and modules.				
Remarks: Interpretation of the reference value	• You can also use a method whereby a tool is used to measure the cyclomatic complexity. In this case, use a metric defined for each organization. • This guide proposes Ratio Of Control Statement as an alternative metric which allows for easier and lower-cost measurement. This metric measures the complexity by picking up those keywords that represent branches in the source code, and is easy to use. • This metric assumes that three or four control statements per 100 lines are used in N-type software.				

ID	PD33
Name	Ratio Of Comment Line
Abbreviation	ROCL

Reference value	N	NQ	C	HC	Adjustment base value
	20.00	25.00	30.00	35.00	5.00
Reference value range	15.00 to 25.00	20.00 to 30.00	25.00 to 35.00	30.00 to 40.00	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• This metric indicates the number of comment lines in the measurement target file relative to the total number of lines of source code. • Comments conforming to certain rules such as the coding rules provide the minimum amount of information needed to understand and maintain the source code. Source code containing appropriate lines of header comments and explanations of the functions and variables, provides higher readability. • If this metric value is too high, unnecessary comments may be included or unnecessary comments or code may be left undeleted in the source code, resulting in lower readability of the source code.				
Calculation method	Comment Lines Of Code/Total Lines Of Code ROCL = CLOC/TLOC				
Usage of the metric	• To use this metric effectively, it is important to define a rule for describing the comment (rule of the contents, location, update, etc.). By describing an appropriate number of comments within the source code according to this rule, its readability can be ensured. • However, because comments are written in a natural language except for some predefined items, the volume can vary greatly depending on who writes them. Therefore, use this metric with a 20 to 30% margin. If the value of this metric differs considerably from the target value, it is required to review the source code as soon as possible to check the situation. • You can also use Ratio Of Comment Line to see if the developer has written an appropriate number of comments. If he or she has not, you can use this metric value to aid in instructing him or her.				
Remarks: Interpretation of the reference value	• This metric assumes that comments are inserted according to a coding rule. • Evaluate this metric from viewpoints such as whether it varies for different files, and whether it is too high or low. • If this metric value is too high, the case in which useless and even dangerous comments are included is possible (e.g., an entire portion of an unused code section is commented out). • If this metric value is too low, necessary information (e.g., a header file) may be missing. Confirm that the comments have been written according to the coding rules.				

ID	PD34
Name	Ratio of Deviation of Coding Rules
Abbreviation	RDCR

Reference value	N	NQ	C	HC	Adjustment base value
	310	210	110	10	100.00
Reference value range	210.00 to 410.00	110.00 to 310.00	10.00 to 210.00	0.00 to 110.00	
Measurement unit	Number/KLOC				
Tolerance	Two high-order significant digits				
Meaning of the metric	• This metric indicates how much of the description deviates from the coding rules defined by the organization or project, relative to the total number of lines of source code.				
Calculation method	Number of Deviation of Coding Rules/Total Lines Of Code RDCR = NDCR/TLOC				
Usage of the metric	• When the deviation ratio is high, you should take more time for checking in the review to ensure quality. If a particular module shows a high deviation ratio, it is recommended to check if the module has any specialty or whether the deviation originates with the developer who wrote it.				
Remarks: Interpretation of the reference value	• If many required counteractions against deviations are not performed, or if there are many cases in which the source code is modified inappropriately to correct any deviation, you should consider a renew of the coding rule itself.				

ID	PD40
Name	Density Of Test Items
Abbreviation	DOTI

Reference value	N	NQ	C	HC	Adjustment base value
	25.00	50.00	75.00	100.00	25.00
Reference value range	0.00 to 50.00	25.00 to 75.00	50.00 to 100.00	75.00 to 125.00	
Measurement unit	Item/KLOC				
Tolerance	Two high-order significant digits				
Meaning of the metric	• This metric indicates the number of executed test items per the source code scale. This metric value shows sufficiency of the dynamic test and coverage of the test relative to the source code.				
Calculation method	Number Of Test Items/Total Lines Of Code DOTI = NOTI/TLOC				
Usage of the metric	• If the scale of the target is larger, the number of combinations is also larger and Density Of Test Items increases. A higher value for this metric is not necessarily good, and effective combinations should be considered when designing the test. Because Density Of Test Items may increase depending on the complexity and the number of I/Os of the target system, it is necessary to ensure the appropriate balance considering the adjustment coefficient and other factors. • If this value is lower than the reference value, the necessary number of tests have not been executed and some failures may remain undetected in the test. • If this value is higher than the reference value, some tests may be unnecessary and the test efficiency may be poor.				
Remarks: Interpretation of the reference value	• The reference value is calculated on the assumption that six test items are prepared for one function which consists of 120 lines for NQ-level software. • To further utilize this metric value, in addition to measuring Density Of Test Items for the entire software, it is recommended to check if Density Of Test Items is appropriate by considering highly difficult or complex portions and portions having many I/Os for individual modules and functions.				

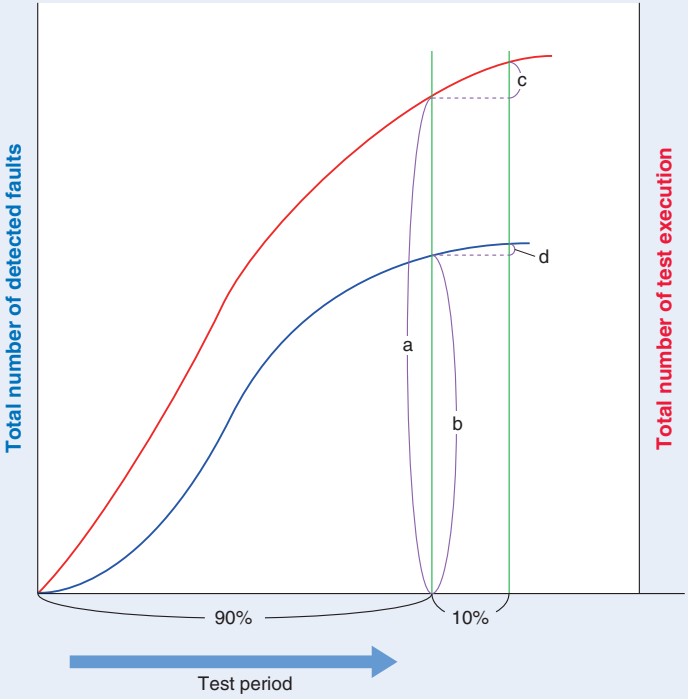
ID	PD41
Name	Ratio Of Fault detection in Comparison
Abbreviation	ROFC

Reference value	N	NQ	C	HC	Adjustment base value
	0.05	0.04	0.03	0.02	0.01
Reference value range	0.04 to 0.06	0.03 to 0.05	0.02 to 0.04	0.01 to 0.03	
Measurement unit	%				
Tolerance	Percentage (two high-order significant digits)				
Meaning of the metric	• There are many conditions for concluding the test. For example, the test is completed when the number of detected failures is likely to converge, that is, the failure ratio likely to fall below a certain value. This metric compares the fault detection ratios during the final and former stages of the test period to determine the stability of the software in the final stage. • During the final stage of the test period, if Number Of Executed Test Items increases and the number of faults detected gets closer to 0 (does not increase), it can be said that faults that may occur when the software is used are converging.				
Calculation method	(Ratio Of Fault Detection in final 10% of test period)/(Ratio Of Fault Detection in first 90% of test period)				
Usage of the metric	If this value is extremely high relative to the description balance as given by the reference value, the test is judged to be insufficient and the test items should be reviewed and/or the test execution status should be investigated again.				
Remarks: Interpretation of the reference value	• This metric assumes that (1) identified faults are fixed and (2) the number of test items increases as the regression tests are repeated. Therefore, the Number Of Executed Test Items value increases as development progresses, while the number of detected faults decreases. In the relationship between the period and the number of items, the situation of the organization should be considered (e.g., if the testing is not performed for a long time, omit this period from the calculations).				

Column

Ratio Of Fault detection in Comparison

In the world of software, as a concept used to forecast how faults converge, they often use the software reliability growth model called the "bug curve." Examples of this model include the logistic curve and Gompertz curve, which are used to forecast demand trends and economic growth. These curves have various models and choosing the wrong model can result in an unreliable evaluation result. In addition, they are statistics processes based upon probability and are very difficult to apply to software development. This guide uses a method for easily judging the convergence of faults by comparing the fault detection ratios during certain (first 90% and final 10%) periods of a test to see if all the faults have been detected in the final stage of the test.



Ratio Of Fault detection in Comparison
= (Ratio Of Fault Detection in final 10% of test period)/
(Ratio Of Fault Detection in first 90% of test period)
= (d/c)/(b/a)

ID	PD42
Name	Ratio Of Fault Elimination
Abbreviation	ROFE

Reference value	N	NQ	C	HC	Adjustment base value
	94.00	97.00	100.00	100.00	3.00
Reference value range	91 to 97	94 to 100	97 to 100.00	97 to 100.00	

Measurement unit	%
Tolerance	Percentage (two high-order significant digits)
Meaning of the metric	• This metric indicates how many of the detected faults were fixed. • With the upper limit being 100, this metric value should be as close as 100 as possible. That is, we should strive to fix every detected fault.
Calculation method	Number Of Eliminated Fault/Number Of Detected Fault ROFE = NOEF/NODF
Usage of the metric	• The Number Of Detected Fault metric indicates the total number of faults in all of the evaluation targets. If this value is high, the target may have quality problems. • By measuring this metric value during development, you can see how the faults are fixed and converged. If this metric value is high, it indicates that the faults are converging. On the contrary, if it is low, it indicates that many of the detected faults have been left unfixed and the status of the project is doubtful. • If this metric value is still low at the final stage of development, a large number of complaints can be feared and the release of the product should be reconsidered. • Because there are many types of faults, including serious and minor ones, do not just evaluate the number of faults, but also consider the details of the faults.
Remarks: Interpretation of the reference value	• Detected faults must not remain after development in a system for which a certain level of quality is required. Therefore, the target reference value for the system of Type 3: Critical or higher is set to 100. • If some faults are left unfixed, you can change the specifications or establish limitations (to assume that they were fixed) to increase this metric value, it is important to determine how to handle them by considering their frequency and danger. Also, to avoid operation errors or misunderstandings by the user, some devices are required (e.g., clarifying the workaround). • A metric similar to this metric is the number of unfixed bugs. Many projects may regard the number of unfixed bugs as merely being a numeric value. If there are projects of difference scales within a single organization, however, you can use this metric to normalize them for comparison.

3.6 Basic Metrics - Definition and Reference Values

What is a basic metric?

A basic metric is a metric that is used for calculating the above-mentioned process metrics and product metrics and which is actually measured at the various process of software development.

There are the following four types of basic metrics:

(1) Source code volume basic metrics
(2) Document volume basic metrics
(3) Process effort volume basic metrics
(4) Test volume basic metrics

Source code volume basic metrics

Total Lines Of Code

According to the concept of the code volume evaluation metrics of the product metrics mentioned above, measure the number of physical lines for each file, using a text editor or line count command. Also, Total Lines Of Code, the total number of lines of all files, is used as an alternative metric for the program scale. Creating a list of values measured for each of the files facilitates application to File Lines Of Code (ID: PD30, FLOC).

Number Of Control Statement

Measure the number of control statements contained in the source code for each file. In the case of the C programming language, a control statement is one of the following:

if, while, for, case, default, else

By using a tool such as an editor to search for these keywords, we can obtain a rough value for this metric. Create a list of values measured for each of the files as these measurements are done to use it for Ratio Of Control Statement (ID: PD32, ROCS).

Comment Lines Of Code

For each file, measure the number of comment lines in the source program. Comment lines can be measured using a tool, or for the C programming language, counting "/*" and "//" enclosing each comment produces a rough number of comments. In this case, however, it is necessary to apply a coding rule to control how to describe a comment spanning multiple lines.

Document volume basic metrics

According to the concept of the document volume evaluation metrics of the product metrics (i.e., approximately 2,000 characters per page)(*) mentioned above, measure the number of pages in each document created as a deliverable by each of the software development processes; requirements definition, architectural design, detailed design, and integration testing, to system qualification testing.

Creating a list of parts of each document with measured values facilitates application to Balance of the Specifications Document Description (ID: PD20, BSDD), Balance of the Design Document Description (ID: PD21, BDDD), and Balance of the Test Document Description (ID: PD22, BTDD).

Process effort volume basic metrics

Measure the total process effort for individual tasks. A rough value can be calculated from the work records.

Each review process effort

Calculate the total process effort expended on reviewing each process. Base this calculation on the review reports. In some cases, review records are not kept (e.g., in the case of a personal desktop review, peer review). In such a case you do not need to include such reviews in the review process effort.

Each creation process effort

Calculate the total process effort expended on the individual processes. Base this calculation on the work records. The measurement target process efforts are those involved with producing or reviewing deliverables. To avoid confusion, clarify the

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

measurement targets for each project. For example, the time needed for meetings is to be included but that for education is not.

Process Effort for TEst

Calculate the total process effort expended on testing the software or system. Include the process effort expended on test design, test data creation, test scenario creation, and test expected value creation. In addition, the fault record and test report documentation work process effort is included in Process Effort for TEst, but the unit testing process effort is not.

In this guide, the unit testing is positioned as testing at the program unit level, in accordance with the concept in ESPR SWP4.1 and SWP4.2.

Test volume basic metrics

Measure the volume of deliverables obtained through the test processes.

Number Of Test Items

The total number of test items including the integration testing, qualification testing, system qualification testing, etc. but excluding the unit testing. Because the number of items can be expressed in various granularities, ensure that the granularity is uniform within the target project. The measurement target test items are those that are effective as test items; that is, those which have actually been used in a test at least once. Test items that have been created but not used are not included in the measurement target.

Number Of Executed Test Items

The total number of test items executed. Similarly to Number Of Test Items, the unit testing is excluded. This value can be measured by multiplying the number of test items mentioned above by the number of test executions. Note that if a test is aborted then the test items after the abort point are not measured.

Number Of Detected Fault

Measure the number of faults detected after the completion of the unit test phase through the measurement time. Faults may be detected not only by the test team but also by stakeholders, including the programmers and other internal parties related to the project. Any fault that is detected should be recorded and managed with a incident

report per fault or database. When counting the incident reports to measure Number Of Detected Fault, it is necessary to differentiate those "faults" that are actually not faults (e.g., misunderstanding of the specifications, operation errors, manual errors, etc.) from actual faults. However, for the number of detected faults for this metric, it is also acceptable to use an easier method of calculating the ratio of actual faults relative to the total number of incident reports by sampling and then multiply the total number of incident reports by the calculated ratio, rather than precisely calculating the number of actual faults.

Number Of Eliminated Fault

Measure the number of faults that have been fixed, relative to the number of detected faults mentioned above. Whether faults have been fixed should be managed through the use of incident reports or a database, but note that for some projects it is difficult to reflect this result. You can also calculate this metric value by subtracting the number of remaining (unfixed) faults from Number Of Detected Fault.

Ratio Of Fault Detection

This basic metric is not measured but calculated by dividing Number Of Detected Fault by Number Of Executed Test Items, and indicates how much faults per tested scale could be detected. Calculate this value in segmented periods and compare the ratio between the period from the start to the latter stage of development and the final period of development to calculate Ratio Of Fault detection in Comparison (ID: PD41, ROFC) for the test sufficiency evaluation metrics.

Column

Rank of detected faults

Generally, a test reveals a variety of faults, from serious ones to very minor ones (some corporations classify faults into "bug ranks"). This guide does not mention fault ranks in detail in measurement of the number of faults which is required to calculate Ratio Of Fault Detection.

In actual deployment, it is necessary to define a rule to count faults by considering the fault rank.

How to read the basic metrics definition table

The following pages explain the basic metrics one by one. Each table consists of the following fields:

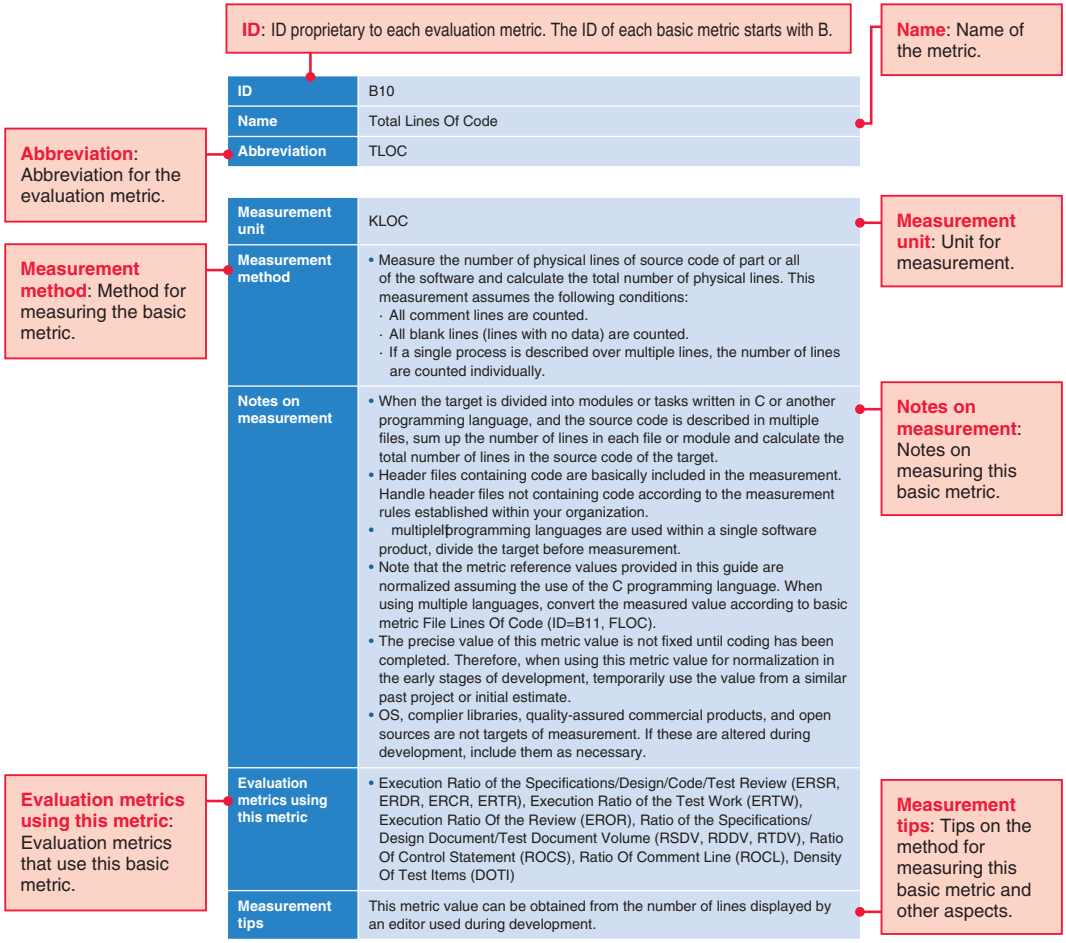


Figure 3-5: How to read the basic metrics definition table

Basic metrics

ID	B10
Name	Total Lines Of Code
Abbreviation	TLOC

Measurement unit	KLOC
Measurement method	• Measure the number of physical lines of source code of part or all of the software and calculate the total number of physical lines. This measurement assumes the following conditions: · All comment lines are counted. · All blank lines (lines with no data) are counted. · If a single process is described over multiple lines, the number of lines are counted individually.
Notes on measurement	• When the target is divided into modules or tasks written in C or another programming language, and the source code is described in multiple files, sum up the number of lines in each file or module and calculate the total number of lines in the source code of the target. • Header files containing code are basically included in the measurement. Handle header files not containing code according to the measurement rules established within your organization. • If multiple programming languages are used within a single software product, divide the target before measurement. • Note that the metric reference values provided in this guide are normalized assuming the use of the C programming language. When using multiple languages, convert the measured value according to basic metric File Lines Of Code (ID=B11, FLOC). • The precise value of this metric value is not fixed until coding has been completed. Therefore, when using this metric value for normalization in the early stages of development, temporarily use the value from a similar past project or initial estimate. • OS, complier libraries, quality-assured commercial products, and open sources are not targets of measurement. If these are altered during development, include them as necessary.
Evaluation metrics using this metric	• Execution Ratio of the Specifications/Design/Code/Test Review (ERSR, ERDR, ERCR, ERTR), Execution Ratio of the Test Work (ERTW), Execution Ratio Of the Review (EROR), Ratio of the Specifications/Design Document/ Test Document Volume (RSDV, RDDV, RTDV), Ratio Of Control Statement (ROCS), Ratio Of Comment Line (ROCL), Density Of Test Items (DOTI)
Measurement tips	This metric value can be obtained from the number of lines displayed by an editor used during development.

ID	B11
Name	File Lines Of Code
Abbreviation	FLOC

Measurement unit	KLOC
Measurement method	• Measure the number of lines for each of the files constituting part or all of the software. • This metric indicates the size of the software targeted for quality measurement and evaluation. If the value of this metric is high, it indicates that the size of the target software is large.
Notes on measurement	• Count the number of physical lines in each file. Count the number of comments and blank lines (follow the line counting rules for Total Lines Of Code (ID: B10, TLOC)). • For conversion when using multiple programming languages, for example, Capers Jones[(2)] demonstrates the ratio data from FP (function points) based on his experience as shown below. It is a good idea to create a conversion formula based on past programming results. 1FP = 320 statements: Assembly language 1FP = 128 statements: C 1FP = 53 statements: C++
Evaluation metrics using this metric	File Lines Of Code (FLOC)
Measurement tips	This metric value can be obtained from the number of lines displayed by an editor during development.

(2): Estimating Software Costs: Bringing Realism to Estimating, Capers Jones, McGraw-Hill Osborne Media, 2007

ID	B12
Name	Module Lines Of Code
Abbreviation	MLOC

Measurement unit	LOC
Measurement method	• Measure the number of lines for each function. The concept of a single line conforms to the measurement of Total Lines Of Code (ID: B10, TLOC). • If multiple programming languages are used within a single software product, measure the program unit equivalent to a function (subroutine, method, etc.) according to each programming language. Refer to File Lines Of Code (ID: B11, FLOC) for the conversion method.
Notes on measurement	• Count the number of physical lines in each function, which specifically indicates a function processing body enclosed by { (just after the function declaration) and }. • Comments and blank lines are also counted.
Evaluation metrics using this metric	Module Lines Of Code (MLOC)
Measurement tips	If it is difficult to measure the values for all the functions, it is also possible to select those functions which seem large.

ID	B13
Name	Number Of Control Statement
Abbreviation	NOCS

Measurement unit	LOC
Measurement method	• Measure the number of control statements contained in the source code of part or all of the software. In the case of the C programming language, a control statement refers to the following: if, while, for, case, default, else
Notes on measurement	• For a programming language other than C, determine the types of control statements to be measured within the project. • Comments are excluded from the measurement target. If performing a count while excluding comments would be cumbersome (see the measurement tips for this metric), a rough count is sufficient.
Evaluation metrics using this metric	Ratio Of Control Statement (ROCS)
Measurement tips	• By measuring the number of control statements contained in the source code using an editor, simple counting tool, or script, it is possible to obtain a rough value for Number Of Control Statement. Note that a conditional statement (e.g., for) may appear frequently within English comments. In this case, exclude the comments before attempting the measurement. If all of them cannot be excluded for convenience of measurement, check the source code visually and determine a rough ratio of such conditional statements within comments, and exclude the estimated number of such conditional statements from the total count.

ID	B14
Name	Comment Lines Of Code
Abbreviation	CLOC

Measurement unit	KLOC
Measurement method	• Measure the number of comment lines contained in the source code of part or all of the software. • For the C programming language, a rough value can be obtained by counting the total number of lines enclosed in /* and */ as well as those beginning with //. By determining a coding rule for writing comments, the measurement will be more precise or easier.
Notes on measurement	• Comments at the end of executable lines are not counted. • By unifying the counting rule for Comment Lines Of Code within the organization or project, efficient and stable measurement is possible. • You can use a tool to count the number of comment lines, if any. Note, however, that the measurement rules may differ from tool to tool. So, check the rules of the tool being used and change the rule to the method proposed by this metric if the tool supports changing of the rule. Or, you can define your own measurement rule for the project.
Evaluation metrics using this metric	Ratio Of Comment Line (ROCL)

ID	B15
Name	Number of Deviation of Coding Rules
Abbreviation	NDCR

Measurement unit	Location
Measurement method	• Count the total number of deviations from the coding rule.
Notes on measurement	• Even if the deviations are identical, count them separately if they appear in different locations in the source code. If, for example, the code deviates from the same rule at two locations, the count value is two. • Exclude any deviations that are processed according to the coding rule within the organization or project from the total number of deviations. • That is, the following deviations are included in Number of Deviation of Coding Rules: · Deviations not processed according to the coding rule that includes deviation processing · Deviations under a rule which does not permit deviations
Evaluation metrics using this metric	Ratio of Deviation of Coding Rules (RDCR)
Measurement tips	Summarizing the number of deviations for each coding rule is helpful in reviewing the coding rule itself.

Column

Coding rules

In general, each organization defines unique coding rules, which the developers are required to observe. Depending on the deployment rules within an organization, some coding rules may prevent a development target from being achieved because the quality characteristics on which to focus differ depending on the functionality demanded of the system. In this case, define a rule application exception procedure to deploy the coding rules (see Section 3.2 of Part.1 in ESCR). Among those deviations detected as violating the coding rules, exclude those permitted according to the rule application exception procedure from measurement of Number of Deviation of Coding Rules.

ID	B20
Name	Volume Of the Specifications Document
Abbreviation	VOSD

Measurement unit	Page
Measurement method	• This metric measures, among documents created as part of a project, the total number of pages in the document describing the requirements specification. • In software development, documents are created with Word and other text editors in various forms such as Excel and other spreadsheet tables or figures. Also, the number of lines or characters per page in documents created using Word varies greatly from one document to another. For this reason, when measuring this metric, text equivalent to 2000 characters (40 horizontal characters x 50 vertical lines)(*) is regarded as being one page. When a document is created using other tools, calculate the approximate number of pages using this standard.
Notes on measurement	• When the document contains figures, photos, or diagrams, consider what the equivalent number of pages would be if they were to be represented by text, and add the calculated number of pages to the total number of pages. • When existing documents are reused as a result of using existing software, include the number of reused pages in the total number of pages.
Evaluation metrics using this metric	Ratio of the Specifications Document Volume (RSDV)
Measurement tips	• Because the number of pages in a document can vary depending on how it is written, this metric does not require the value to be extremely precise. Instead, a rough value is used to evaluate the sufficiency of the document.
Supplement	(For measuring the amount of document balance description) • For the description volume for individual description items, it is acceptable to acquire a rough quantity as the number of pages (e.g., approximately 1.5 pages) or lines. In some cases, the area or physical dimensions (vertical and horizontal lengths) of the description text can be used to calculate the percentage of the description volume relative to the entire volume.

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

ID	B21
Name	Volume Of the Design Document
Abbreviation	VODD

Measurement unit	Page
Measurement method	• This metric measures, among documents created as part of a project, the total number of pages in the document describing the design. • In software development, documents are created using Word and other text editors in various forms such as Excel and other spreadsheet tables or figures. Also, the number of lines or characters per page in documents created using Word varies greatly from one document to another. For this reason, when measuring this metric, text equivalent to 2000 characters (40 horizontal characters x 50 vertical lines)(*) is regarded as being one page. When a document is created using other tools, calculate the approximate number of pages using this standard.
Notes on measurement	• When the document contains figures, photos, or diagrams, consider what the equivalent number of pages would be if they were to be represented by text, and add the calculated number of pages to the total number of pages. • When existing documents are reused as a result of using existing software, include the number of reused pages in the total number of pages.
Evaluation metrics using this metric	Ratio of the Design Document Volume (RDDV)
Measurement tips	• Because the number of pages in a document can vary depending on how it is written, this metric does not require the value to be extremely precise. Instead, a rough value is used to evaluate the sufficiency of the document.
Supplement	(For measuring the amount of document balance description) • For the description volume for individual description items, it is acceptable to acquire a rough quantity as the number of pages (e.g., approximately 1.5 pages) or lines. In some cases, the area or physical dimensions (vertical and horizontal lengths) of the description text can be used to calculate the percentage of the description volume relative to the entire volume.

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

ID	B22
Name	Volume Of the Test Document
Abbreviation	VOTD

Measurement unit	Page
Measurement method	• This metric measures, among documents created as part of a project, the total number of pages in the test specifications. • In software development, documents are created using Word and other text editors in various forms such as Excel and other spreadsheet tables and figures. Also, the number of lines or characters per page in documents created using Word varies greatly from one document to another. For this reason, when measuring this metric, text equivalent to 2000 characters (40 horizontal characters x 50 vertical lines)(*) is regarded as being one page. When a document is created using other tools, calculate the approximate number of pages using this standard.
Notes on measurement	• When the document contains figures, photos, or diagrams, consider what the equivalent number of pages would be if they were to be represented by text, and add the calculated number of pages to the total number of pages. • When existing documents are reused as a result of using existing software, include the number of reused pages in the total number of pages.
Evaluation metrics using this metric	Ratio of the Test Document Volume (RTDV)
Measurement tips	• Because the number of pages in a document can vary depending on how it is written, this metric does not require the value to be extremely precise. Instead, a rough value is used to evaluate the sufficiency of the document.
Supplement	(For measuring the amount of document balance description) • For the description volume for individual description items, it is acceptable to acquire a rough quantity as the number of pages (e.g., approximately 1.5 pages) or lines. In some cases, the area or physical dimensions (vertical and horizontal lengths) of the description text can be used to calculate the percentage of the description volume relative to the entire volume.

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

ID	B30
Name	Review Effort in TOtal
Abbreviation	RETO

Measurement unit	Man-hour
Measurement method	• This metric measures the total process effort expended on all reviews of the specifications, design, code, and testing. • The measurement is done as follows: · The process effort expended on specifications documentation, design, coding, testing preparation, and testing execution are not included. · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Execution Ratio Of the Review (EROR)
Measurement tips	Based on the review records and other information, calculate the metric value by multiplying "time for each review" x "number of times executed" x "number of attendees" ("∑ time" x "number of attendees").

ID	B31
Name	Review Effort for SPecification
Abbreviation	RESP

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on the review of the specifications, which are output from the specifications investigation. • The review target is the output from SYP1 (System requirements definition) and SWP1 (Software requirements definition) of the "Embedded System Development Process Reference" (ESPR), that is, the System Requirements Specification, Software Requirements Specification, and other specifications documents specific to the project. • The measurement is done as follows: · The process effort expended on the specifications documentation is not included. · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Ratio of the Specifications Review Effort (RSRE), Execution Ratio of the Specifications Review (ERSR)
Measurement tips	Based on the review records and other information, calculate the metric value by multiplying "time for each review" x "number of times executed" x "number of attendees" ("∑ time" x "number of attendees")

ID	B32
Name	Review Effort for DEsign
Abbreviation	REDE

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on the review of the design document, which is output from the design process. • The review target is the output from SYP2 (System architectural design) and SWP2 and SWP3 (Software architectural design and Software detailed design) of the "Embedded System Development Process Reference" (ESPR), that is, the System architectural design, System behavior design, System interface design, System architectural design, Software structure design, Software behavior design, Software interface design, Software architectural design, Software detailed design, Program unit functional/ structure design, and other documents specific to the project. • The measurement is done as follows: · The process effort expended on the design is not included. · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Ratio of the Design Review Effort (RDRE), Execution Ratio of the Design Review (ERDR)
Measurement tips	Based on the review records and other information, calculate the metric value by multiplying "time for each review" x "number of times executed" x "number of attendees" ("∑ time" x "number of attendees").

ID	B33
Name	Review Effort for COde
Abbreviation	RECO

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on the source code review. • The review target is the output from SWP4 (Coding and unit testing) of the "Embedded System Development Process Reference" (ESPR), that is, the program unit and source code. • The measurement is done as follows: · The process effort expended on the coding and unit testing is not included. · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Ratio of the Code Review Effort (RCRE), Execution Ratio of the Code Review (ERCR)
Measurement tips	Based on the review records and other information, calculate the metric value by multiplying "time for each review" x "number of times executed" x "number of attendees" ("∑ time" x "number of attendees").

ID	B34
Name	Review Effort for Test Preparation
Abbreviation	RETP

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on the review of the output related to the testing. The review target is the output from SYP3 (System integration testing), SYP4 (System qualification testing), SWP5 (Software integration testing) and SWP6 (Software qualification testing) of the "Embedded System Development Process Reference" (ESPR), that is, the test specifications, test environment, test data, test result, and test report. • The measurement is done as follows: · The process effort expended on the testing execution, environment preparation, document creation, etc. are not included. · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Ratio of the Test Review Effort (RTRE), Execution Ratio of the Test Review (ERTR)
Measurement tips	Based on the review records and other information, calculate the metric value by multiplying "time for each review" x "number of times executed " x "number of attendees" ("∑ time" x "number of attendees").

ID	B35
Name	Process Effort in TOtal
Abbreviation	PETO

Measurement unit	Man-hour
Measurement method	• This metric measures the entire process effort expended on the development of a part or all of the software targeted by the process metrics and product metrics. • The measurement is done as follows: · Sum up all the process effort, including direct work such as requirements definition, design, coding, and testing, and indirect work such as quality management related to the development. · To calculate the process effort, the work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day. • The precise value of this metric is not fixed basically until development has been completed. Therefore, when using this metric value for normalization during development, temporarily use either a value assumed based on a similar past project or estimate value.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort. • Any work that does not produce direct deliverables, such as training of engineers, is not included.
Evaluation metrics using this metric	Ratio Of the Review Effort (RORE), Ratio of the Test Work Effort (RTWE)
Measurement tips	It is also possible to calculate the entire process effort according to the number of persons (planned to be) involved in each process, while estimating the time needed for the process.

Column

Tip on interpreting metrics

If the entire process effort for software development as grasped intuitively by the developer, project manager, or leader differs greatly from the actually measured value for Process Effort in TOtal, the developer may have done unnecessary work or expended too much labor on the work. Therefore, the manager must check and review the work content as necessary.

ID	B36
Name	Process Effort for SPecification
Abbreviation	PESP

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on the specifications creation and investigation. • The target of this metric is the output from SYP1 (System requirements definition) and SWP1 (Software requirements definition) of the "Embedded System Development Process Reference" (ESPR), that is, the System requirements specification, Software requirements specification, and other specification documents specific to the project. • The process effort expended on the specifications review is also included. • The measurement is done as follows: · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Ratio of the Specifications Review Effort (RSRE)
Measurement tips	

ID	B37
Name	Process Effort for DEsign
Abbreviation	PEDE

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on the design work. • The target is the output from SYP2 (System architectural design) and SWP2 and SWP3 (Software architectural design and Software detailed design) from the "Embedded System Development Process Reference" (ESPR), that is, the System architectural design, System behavior design, System interface design, System architectural design, Software structure design, Software behavior design, Software interface design, Software architectural design, Software detailed design, Program unit functional/ structure design etc., and other design documents specific to the project. • The process effort expended on the design review is included. • The measurement is done as follows: · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Ratio of the Design Review Effort (RDRE)
Measurement tips	

ID	B38
Name	Process Effort for COde
Abbreviation	PECO

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on source code creation. • The target is the output from SWP4 (Coding and unit testing) of the "Embedded System Development Process Reference" (ESPR), that is, the program unit and source code. • The process effort expended on the source code review is included. • The measurement is done as follows: · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort.
Evaluation metrics using this metric	Ratio of the Code Review Effort (RCRE)
Measurement tips	

ID	B39
Name	Process Effort for Test Preparation
Abbreviation	PETP

Measurement unit	Man-hour
Measurement method	• This metric measures the total process effort expended on, among that work related to testing, test specifications documentation, test environment arrangement, test result confirmation, and test review. Note that the process effort expended on the testing execution is not included. • The target is the output from SYP3 (System integration testing), SYP4 (System qualification testing), SWP5 (Software integration testing), and SWP6 (Software qualification testing) of the "Embedded System Development Process Reference" (ESPR). • The measurement is done as follows: · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort. • Measure all the process effort for the work related to the testing, excluding the testing execution. • The SWP4 (Unit testing) items are not included.
Evaluation metrics using this metric	Ratio of the Test Review Effort (RTRE)
Measurement tips	

ID	B3A
Name	Process Effort for TEst
Abbreviation	PETE

Measurement unit	Man-hour
Measurement method	• This metric measures the process effort expended on all work related to the testing, for example, testing preparation, test environment arrangement, testing execution, test result confirmation, and test review. • The target is the output from SYP3 (System integration testing), SYP4 (System qualification testing), SWP5 (Software integration testing), and SWP6 (Software qualification testing) of the "Embedded System Development Process Reference" (ESPR). • The measurement is done as follows: · The work process effort is calculated, in units of man-hours, according to the work period and the total number of persons who worked on the project. · If the measurement unit used by your organization is "day" rather than "hour," multiply the value by the number of working hours per day.
Notes on measurement	• To obtain a precise measurement of the work time, stored records such as process effort records and work reports are used to obtain a value. However, considering the purpose and data precision of this metric, extremely strict measurement of the process effort is not necessary. As such, you can also use a rough estimate such as the work times stated by the personnel, or even assume the process effort. • If a reviewer has joined the review from outside the project, also include his or her process effort. • Measure all process effort related to the testing (Process Effort for Test Preparation + testing execution process effort). • Calculate the work process effort, in units of man-hours, according to the work period and the number of persons involved in the work. · SWP4 (Unit testing) is not included.
Evaluation metrics using this metric	Ratio of the Test Work Effort (RTWE), Execution Ratio of the Test Work (ERTW)
Measurement tips	A rough value can be obtained by multiplying "single process effort for test" x "number of persons involved in the test" x "number of test executions."

ID	B40
Name	Number Of Test Items
Abbreviation	NOTI

Measurement unit	Items
Meanings of metric value	• Number Of Test Items indicates the scale of the test (the number of items tested). • As the target software size is large and complicated, the value of this metric will be high.
Measurement method	• Measure the number of test items executed. • The tests targeted are SYP3 (System integration testing), SYP4 (System qualification testing), SWP5 (Software integration testing), and SWP6 (Software qualification testing) defined in the "Embedded System Development Process Reference" (ESPR). • The test item varies from a sub-item to a script, depending on the target product, organization, and/or test type. Even within the same organization, the test items may be described in different ways depending on the test purpose. For this reason, it is necessary to unify expressions within the project to ensure that the descriptions of the test items are uniform, before measuring this metric. • The measurement method for the test items in this guide conforms to the items in the test case details described in the test specifications/test report in "2.3 Sample Document Templates" on P.192 - 195 of ESPR. That is, the test details, test data, expected output, environmental condition, and test result are expressed on one line, and this one line is treated as a single item. You can count the number of items according to this information or devise a counting method for your organization.
Notes on measurement	• Pay attention to the following during measurement: · SWP4 (Unit testing) items are not included. · If test items are reused when existing software is reused, the reused test items should be included in the measurement. · Test items that have not been executed are not included in the measurement.
Evaluation metrics using this metric	Density Of Test Items (DOTI)
Measurement tips	• The number of test items may be precisely determined from the test items defined in the test specifications. It is also possible, however, to estimate the number of test items by multiplying the number of pages describing the test items by the number of liens per page. • The counting method for the test items may differ depending on the format or description granularity of the test specifications. Therefore, as a guide, it is also acceptable to assume 100 characters (40 characters x 2.5 lines)(*) of the test items described in the test specifications as a single unit and count that volume as a single test item regardless of its contents.

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

ID	B41
Name	Number Of Executed Test Items
Abbreviation	NOET

Measurement unit	Items
Measurement method	• Measure the total number of test items that are executed. • The tests targeted are SYP3 (System integration testing), SYP4 (System qualification testing), SWP5 (Software integration testing), and SWP6 (Software qualification testing) of the "Embedded System Development Process Reference" (ESPR). • The test item varies from a sub-item to a script, depending on the target product, organization, and/or test type. Even within the same organization, the test items may be described in different ways, depending on the test purpose. For this reason, it is necessary to unify expressions within the project to ensure that the descriptions of the test items are uniform, before measuring this metric.
Notes on measurement	• Pay attention to the following during measurement: · SWP4 (Unit testing) items are not included. · Count the total number of items executed. For example, if the same test set was executed for multiple times, measure the number of items as "Number Of Test Items x Number of times of test execution." · If the test was aborted, the subsequent (not executed) test items are not counted.
Evaluation metrics using this metric	Ratio Of Fault detection in Comparison (ROFC), Ratio Of Fault Detection (ROFD)
Measurement tips	• Accumulate the number of test items executed as mentioned in the test report. • For the execution results in the test report, you can also calculate a rough value for Number Of Test Items with 100 characters(*) as a single unit, just as with Number Of Test Items.

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

ID	B42
Name	Number Of Detected Fault
Abbreviation	NODF

Measurement unit	Faults
Measurement method	• Measure the number of faults detected in the review, testing, or otherwise, after the integration testing. • Faults detected in SWP4 (Unit testing) are not included.
Notes on measurement	• This metric value can be measured by counting the number of incident reports, etc. • Faults having the same cause and erroneous reports must be excluded. Ideally, faults should be managed and measured individually. If this is not possible, calculate a rough value by, for example, multiplying the total number of faults by a certain factor. • Because faults vary from a severe to slight, do not simply evaluate the number of faults, but also consider the details of those faults. For severe faults, adjust the number by, for example, multiplying the value by a weighting factor. • Exclude the number of faults detected in the tests, but finally released according to the specifications or limitations, from the total number of faults.
Evaluation metrics using this metric	Ratio Of Fault detection in Comparison (ROFC), Ratio Of Fault Elimination (ROFE), Ratio Of Fault Detection (ROFD)
Measurement tips	• You can measure this metric value from the number of pages of incident reports or number of entries in the fault list, if any. • Using a fault database facilitates measurement.

ID	B43
Name	Number Of Eliminated Fault
Abbreviation	NOEF

Measurement unit	Items
Measurement method	• This metric measures the number of fixed faults, among those faults detected in the review, testing, or otherwise, after the integration testing.
Notes on measurement	• You can measure this metric using the incident report. • The metric value is obtained by subtracting the number of remaining faults from the total number of faults. • Note that some fixed faults may not have been confirmed yet although they are reported as being "fixed" in the incident report, or may not yet have been reflected on the incident report although they were fixed, depending on the measurement timing.
Evaluation metrics using this metric	Ratio Of Fault Elimination (ROFE)
Measurement tips	• Number Of Eliminated Fault is 0 at the start of development, and increases following Number Of Detected Fault as the development proceeds. That is, the formula "Number Of Detected Fault ≥ Number Of Eliminated Fault" is always true. • Preparing a list of incident reports can facilitate measurement.

ID	B44
Name	Ratio Of Fault Detection
Abbreviation	ROFD

Measurement unit	%
Measurement method	Ratio Of Fault Detection = (Number Of Detected Fault/Number Of Executed Test Items) during a certain period
Notes on measurement	• This metric is not measured directly but calculated using the above formula. • This metric indicates the balance of the number of faults detected during a certain period against the number of tests executed during the same period.
Evaluation metrics using this metric	Ratio Of Fault detection in Comparison (ROFC)
Measurement tips	• Evaluation metrics using this metric: Ratio Of Fault detection in Comparison (ROFC) uses this metric in separate development periods, specifically, the first 90% and the final 10%. Therefore, it is necessary to match the development period, Number Of Detected Fault, and Number Of Executed Test Items. The incident report, work report, and other information are used for this matching, so prepare them so that you can trace the data chronologically when using this metric.

Column

I-Bottle and I-Glass

In the field of computing, bits and bytes are used to represent quantities of information. Since the design document and test specifications are expressed in characters, their sizes and the amount of information they contain could also be represented in bits or bytes. However, doing so does not present a clear image of the actual volume of the specifications or design information, as well as test item information (details). On the contrary, we tend to say "this is equal to ten Tokyo Domes" or "this is the same as three apples" to represent the size or weight of an object. This guide proposes a new means of representing the amount of description in the documents created during software development (e.g., the design document and test specifications); I-Bottle and I-Glass(3). I-Bottle mimics a bottle capable of holding 2000 characters(*) of information. According to this concept, the amount of information mentioned in a document such as the specifications documents is counted as "how many I-Bottles are required to hold the information." In addition, a smaller unit, I-Glass, is adopted for the test items in the test specifications. One I-Glass contains 100 characters of information. So, the number of test items is counted as "how many I-Glasses are required to accommodate the information." Note that twenty I-Glasses are equivalent to one I-Bottle.

(3) **I-Bottle**: Information Bottle/**I-Glass**: Information Glass: Proposed by M. Hirayama and S. Yoshizawa

(*): This reference value is based on Japanese, so it needs to be adjusted for other languages. For example, the value is nearly doubled in the case of English.

Chapter 4

Tips for High Quality Establishment

Chapters 2 and 3 discussed the concept of controlling quality by using quantitative metrics that can be measured objectively. This chapter offers a slightly different perspective. It contains tips for checking the quality of work qualitatively from the viewpoint of improving the quality of embedded software in each phase of its development process.
Achieving better quality requires that every stage of the development be done at the optimum timing using the best possible means. Regarding quality improvement technologies and techniques, there is already a wealth of knowledge publicly available. This chapter presents several sets of questions by which you can determine whether you are making effective use of these technologies and techniques for development, whether your organization is sufficiently motivated, and so on. Many of the tips contained herein come from people actually involved in development. We hope that these tips include information that you will find useful for your development work. This chapter groups the tips into five categories and provides a checklist for each category. Use these checklists to evaluate the vitality and other relevant aspects of your organization.
Also, at the end of the document, you will find a list of reference books.

4.1 Communication and Decision Making in Development

Importance of communication and decision making

When we analyze failed software development projects to determine the cause of failure, we realize that, in many cases, the failure is due to "human errors" rather than "technical problems." Such "human errors" include problems related to communication within the project team and with stakeholders, as well as those related to the decision-making process, and often lead to the failure of the project. Also, with the growth in the scale of software development over the last few years, it is becoming increasingly common for many people to work together in a development project, making communication even more important.

The types of communication in a software development can be roughly divided as follows:

- Communication for transmitting information (one-way)
- Communication for making decisions through consultation and discussion (interactive)

When software is developed by a one person, there is no need for communication. When multiple people are involved in the development, however, the transmission of information and the process of making decisions based on the situation facing each of those individuals play a particularly vital role.

Indispensable to achieving high productivity and high quality in software development involving many people is communication for building trust between the leader and the members of the project team, as well as among members and stakeholders, and for ensuring smooth information transmission and decision making.

Whether such smooth communication can be accomplished depends on the skills gained from experience and the communicative sense of each individual involved. To ensure smooth communication and a fast, trouble-free decision-making process, the following must be kept in mind:

- Find methods and mechanisms suited to the goal of communication.
- Communication is a human activity; think about the human aspects.

Communication checklist

NO.	Check item	Check
1	Does your organization have a work culture that encourages smooth communication?	
2	Do you respect those with whom you communicate?	
3	Are preparations and environments for meetings made?	
4	Do you spend enough time on expanding and discussing the necessary themes?	
5	Do you respect the past experience and knowledge of skilled individuals?	
6	Is the essence of a discussion shared?	
7	Is there a consensus on how decisions are made?	
8	Is necessary information correctly communicated to all project members?	
9	Are you over-dependent on asynchronous types of communication such as e-mail?	
10	Is communication done in ways to improve productivity and quality?	
11	Are the results of meetings and communication reflected on the development process?	
12	Do meetings leave a sense of futility?	

Explanation of each check item

Check 1 **Does your organization have a work culture that encourages smooth communication?**

Smooth communication is essential for sharing project objectives and goals and getting work done. To facilitate smooth communication, there needs to be a work culture (workplace environment and atmosphere) in which individual members of the project, project teams, and stakeholders can all communicate with one another openly for reporting, notification, and consultation. When judging whether such a culture exists, you should look at the human aspects as well. You may want to check: **(1) whether project members make proper greetings; (2) whether the project members are cheerful; (3) whether each project member knows what their coworkers are doing;** and **(4) whether there is open communication among organizations**.

Check 2 **Do you respect those with whom you communicate?**

To ensure smooth communication and maintain high quality, you also need to explore ways of ensuring that project members are more satisfied with how communication is done and that they better understand one another. It is not acceptable to overpower project members by taking advantage of your positional

superiority or to demoralize them by making personal attacks.

When communicating with project members, make sure **(1) that you understand their stance (background)** and **(2) that you use wording that can be understood by all of them**.

Check 3 **Are preparations and environments for meetings made?**

If attendees are not informed about the agenda or purpose of a meeting, they cannot make preparations or hold prior discussions. In such a case, you will need to waste precious time briefing the attendees on the purpose of the meeting and the like. Making proper preparations in advance of a meeting helps the attendees be more aware of the importance of the meeting and be more focused, thereby leading to productive discussions. To this end, you may want to check **(1) whether attendees are informed in advance about the purpose of a meeting; (2) whether the agenda of the meeting is made known beforehand;** and **(3) whether all necessary devices and tools (whiteboard, overhead projector, etc.) are prepared**.

Check 4 **Do you spend enough time on expanding and discussing the necessary themes?**

In order to share project objectives and goals and attain those goals, it is essential **(1) to spare enough time for each item on the necessary themes; (2) to have thorough discussions on the relevant topics including different opinions and methodologies;** and **(3) to ensure that all the members involved are convinced of the decisions made and stay motivated while getting work done**. Needless to say, the propagation of information and the holding of discussions are crucial not only for meetings but also when making routine reports and giving instructions. It is important that conflicting opinions and objections receive due consideration, including ideas about possible alternative measures.

Check 5 **Do you respect the past experience and knowledge of skilled individuals?**

To make good decisions, it is vital to make use of the knowledge and experience amassed within your project team or organization. To do so, you need to make sure **(1) that meetings are attended by the right people** and **(2) that there is a mechanism in place for leveraging the experience of individuals**.

Check 6 **Is the essence of a discussion shared?**

Software development entails a wide variety of processes from request identification and design to coding and testing. These processes often involve not only software engineers but also people with other diverse functions, including hardware engineers, management executives, and even legal staff. For this

reason, two things are necessary for communication in a software development project: **(1) you need to identify the stakeholders and precisely understand the needs of those individuals who play different roles in different stages of the development so that the software is designed and implemented consistently;** and **(2) meeting the first condition requires that the essence of any discussion be shared by all stakeholders**.

Check 7 **Is there a consensus on how decisions are made?**

Decision making is a process of selecting one course of action from all the available options, relative to a specific problem.

In communication for software development, decision making plays a significant role on which the success of a project hinges. When making a decision, therefore, you not only have to produce an outcome but also ensure **(1) that there is a consensus on how the decision is made; (2) that whoever is responsible is clarified;** and **(3) that objectives and goals are shared**. Even when a decision is made by the project leader in a top-down fashion, the rationale behind that decision and other relevant details must be explained to the project members.

Check 8 **Is necessary information correctly communicated to all project members?**

To develop high-quality software, there should be no insufficiencies or ambiguities in the information about the specifications and any changes that are made to them. Considering insufficiency or ambiguity in information from the viewpoints of communication and decision making, it is important **(1) that the necessary information is correctly communicated to all stakeholders; (2) that there is a system in place to ask for more information in case of any insufficiency or ambiguity in the information;** and **(3) that lines of communication are set up for transmitting information about a specification change, error, etc**. You must take these factors into account and ensure that the necessary information is adequately communicated to the pertinent project members.

Check 9 **Are you over-dependent on asynchronous types of communication such as e-mail?**

There are different methods and mechanisms of communication, each suited to a specific purpose. In recent years, e-mail programs, schedulers, electronic clearing systems, and other convenient tools have contributed to improving productivity and management accuracy. As these communication tools become more sophisticated, however, there is a growing tendency for them to replace human-to-human communication, resulting in lost opportunities to gain critical information from inter-human contacts or to make decisions based on such information. To ensure smooth communication, it is essential to **(1) adopt 'virtual**

communication' using electronic tools and other means or, alternatively, 'real communication' depending on the purpose.

Check 10 **Is communication done in ways to improve productivity and quality?**

The primary purpose of smooth communication in software development is to "improve productivity and quality." The holding of meetings is not the purpose. It is important to define roles properly and to make the right decisions quickly based on the experience and knowledge of your predecessors. Organizations with excellent communication have the following traits: **(1) when the need for a new task arises, the staff who will take charge are automatically assigned;** and **(2) younger members of the workforce are being nurtured through communication**.

Check 11 **Are the results of meetings and communication reflected on the development process?**

While the holding of meetings and the making of decisions based on smooth communication is vital, these meetings and decisions cannot make sense unless **(1) the results of the meetings and the decisions made are communicated to all the project members and reflected on the relevant stages of the development process**. To accomplish this, it is important to ensure **(2) that the meeting minutes contain sufficient information in a well organized manner; (3) that the meeting minutes and other necessary documents are distributed to the relevant project members;** and **(4) that any decisions made by higher-level staff are conveyed to lower-level staff**.

Check 12 **Do meetings leave a sense of futility?**

A "sense of futility" refers to the feelings of emptiness, resignation, or apathy. Meetings and any other communication tinged with a sense of futility do not contribute to maintaining high productivity or high quality. If you see a number of people leaving the meeting halfway, doing other work during the meeting, or raising issues after the meeting, that is a sign that you do not have a good communication environment. It is crucial to ensure that all attendees share the purpose of the meeting and stay focused. You may want to consider **(1) holding every meeting with a definite purpose in mind; (2) taking the leadership role to make sure a conclusion is reached; (3) ending the meeting within a reasonable timeframe;** and **(4) exploring ways to prevent meetings from becoming stale**.

4.2 Documents

Purposes and roles of documents

Software is developed by a number of people using a number of processes such as specification creation, design, coding, and testing. A task spanning multiple processes or done by multiple people requires a procedure that explains what to do and how to do it.

The role of a document is to identify the input and output of the information contained in it and tell readers (including yourself in the future) what is to be done and what has been done. The quality of a document can be judged by "whether it lets you understand everything you need to do next and everything that has been done." Possible problems with a document include:

- Missing or incorrect information
- Lying
- Misleading or obsolete information or incorrect quotation
- Illegible text
- Nonexistent document

A nonexistent document is out of the question, and an unreliable document is utterly meaningless. Avoiding these problems is a step toward creating a good document.

In addition, defining description rules for specifications and other documents offers the benefits mentioned below.

Take a design document for example. The potential benefits are as follows.

Accurate design

Creating a design document by using a description method that defines how to write and read information makes for more accurate designs with fewer omissions, ambiguities, overlaps, etc. This makes it possible to more strictly determine whether the resultant design is appropriate.

Visible design process and more efficient review

The software design process is easier to monitor, which allows for a more efficient design review. Also, the various measurements, which provide the basis for quality management, are easier to make.

Smooth communication

An accurate and easy-to-understand design document enables smooth communication among the people involved in software development.

Easier reuse of documents due to increased standardization

Using a standard description method promotes the standardization of documents. This makes it easier to reuse deliverables created in the software development process.

Document checklist

NO.	Check item	Check
1	Does the document define what to create and when to create it?	
2	Are there rules regarding revision?	
3	Is there a defined document approval procedure?	
4	Are the structure and content of the document appropriate?	
5	Are all the necessary documents at hand? Are any actually unnecessary?	
6	Is the document written properly in a predefined format?	
7	Are the terms defined?	
8	Is the document created taking into consideration that it may also be read by those not deeply involved in the project?	
9	Are the source document of this document and the next document that makes use of information in this document are clearly defined?	
10	Is the number of typos and omissions within the allowable limits? Are all abbreviations spelled out? Are there any expressions that are hard to understand?	
11	Are documents reviewed by anyone other than the author?	
12	Does the document contain all the necessary information? Does the volume of the document match the size of the system?	

Explanation of each check item

Check 1 **Does the document define what to create and when to create it?**

Various documents are created as part of the software development process, and each has its own specific purpose. "What documents to create and when to create them" should be defined based on the software development process. You need to create a document that matches the intended purpose, and at the right timing. Defining the creation timing of documents and their content is an essential requirement for creating high-quality documents. You should start by **(1) formulating a document creation plan that corresponds to the software development process**.

Check 2 **Are there rules regarding revision?**

Recent software development projects rarely entail developing an entirely new system from the ground up. Rather, functions of existing systems are reused in the design of a new system. In product development, it is not uncommon for a failure or non-conformity to result from the use of a slightly different version or a wrong part that performs a similar function or processing. When you develop a system by reusing existing functions, you need to **(1) define rules for revision** and **(2) create a revision history**. Also, ensure **(3) that lines of communication are established for making the revision known** and **(4) that traceability is achieved**.

Check 3 **Is there a defined document approval procedure?**

An approval procedure is a means of verifying that a document or deliverable has been finalized. It clarifies the responsibility supported by work standards and safety criteria.

Documents should be approved by pre-assigned qualified personnel performing inspection and verification from the aspects of quality, cost, and safety. It is important that **(1) the approval procedure be designed with quality, cost, and safety in mind**.

Check 4 **Are the structure and content of the document appropriate?**

A document having an inappropriate structure or content can be misleading to the reader.

For example, information on cost or the project progress is of little use if it is contained in a document entitled "Design Document." The title and structure of a document should be consistent with its content, and **(1) it is essential that the purpose of the document and what it intends to convey be clear**.

(2) Documents are a vital source of information. Keep in mind that the document is the only way for the reader to obtain information about the software.

Check 5 **Are all the necessary documents at hand? Are any actually unnecessary?**

In product development, it is a prerequisite that **(1) all the necessary documents are at hand**. The existence of unnecessary documents, on the other hand, confuses those involved in the development. Documents under review and those that have been turned down or scrapped should be managed separately from those that have been officially approved. **(2) The status of each managed document should be made clear**.

Check 6 **Is the document written properly in a predefined format?**

When creating a document, it is important **(1) that you have a common format defined by your organization or project** not only to boost development efficiency but also to ensure that all members achieve a common understanding. Every organization and project should prepare a document format or template containing basic information that includes, at least, the title of the document, the name of the author, a table of contents, the name of the person who gave approval, a revision history, and the creation date. In this way, **(2) developers will create documents according to description rules**, helping stabilize the quality of documents.

Check 7 **Are the terms defined?**

A design document described in a natural language may contain terms like "this system," "relevant system" and "target system," but it is often the case that the exact meaning of these terms is clear only to the author of the document. The use of undefined terms may mislead the reader, potentially resulting in design mistakes or degraded quality. It is essential to avoid the use of ambiguous terms that different readers may interpret differently. Also, technical terms and abbreviations familiar to the author may be incomprehensible to the designers and implementers who read the document and may invite misunderstanding. To accomplish smooth communication, it is desirable that the document **(1) does not contain ambiguous terms that different readers may interpret differently** and **(2) clearly defines the technical terms and abbreviations used**.

Check 8 **Is the document created taking into consideration that it may also be read by those not deeply involved in the project?**

One important aspect of a document is the appeal that makes the reader want to read it. Sentences should be readable and easy to understand. You may want to check **(1) whether expressions are appropriate; (2) whether sentences are precise and clear; (3) whether the passive voice is avoided where appropriate;** and **(4) whether the layout is reader-friendly**.

Check 9 **Are the source document of this document and the next document that makes use of information in this document are clearly defined?**

When creating a document, you need to identify its input and output. First, it is a prerequisite **(1) that you have all the necessary information - the information to be input (e.g., documents) - clearly defined and prepared**. Then, as the output, you need to **(2) identify the next process (document) with respect to the document you are creating and clarify the information that is to be provided for that next process and how it should be written**. Identifying the input and output of a document clarifies what should be written in it, preventing necessary information from being omitted and unnecessary information from being included.

Check 10 **Is the number of typos and omissions within the allowable limits? Are all abbreviations spelled out? Are there any expressions that are hard to understand?**

A document riddled with typos, omissions, and ambiguous expressions can give a sense of mistrust to the reader and end up being underused. If there are many typos and omissions, it is often difficult to review the document thoroughly, because merely pointing out those typos and omissions takes up most of the allocated time. Also, a document that does not spell out the abbreviations that it uses, or which contains ambiguous expressions that are difficult to understand, cannot convey essential information to the reader, which makes the existence of the document meaningless. However, demanding that a project team's internal documents be perfect may make document creation a time-consuming process. It is therefore necessary to **(1) define allowable limits for typos, omissions, abbreviations, and expressions and keep to those limits**.

Check 11 **Are documents reviewed by anyone other than the author?**

The review is an important task for quality establishment in the software development process. Reviewing the deliverables created during the development process at the relevant milestones in development improves the accuracy and quality of those deliverables. There are many ways of reviewing a document. One is a self review whereby the author reviews his or her own deliverable using a checklist or other means. Other review methods include inspection and walkthrough, which are done by a group of interested parties. Not only the engineers who create deliverables but also **(1) other interested parties such as relevant project members and stakeholders may participate in reviews**, which will help find more issues and problems. It is also important to have a system in place for **(2) retaining the issues and problems pointed out in reviews as review records with the date and staff in charge determined for reliable backup**.

Check 12 **Does the document contain all the necessary information? Does the volume of the document match the size of the system?**

Needless to say, the quality (content) of the document is far more important than its volume. It is important that the content to be written in the document be correct. Even when the document describes all the required items, it is highly likely that the document will lack the necessary information, if its volume is excessively small. As mentioned earlier in this guide in relation to the document volume evaluation metrics, **(1) whether the volume of the document matches the size of the system** should be checked.

Even if the volume condition is satisfied, you also need to check **(2) whether the document contains all the necessary information with a reasonable amount of information on every topic covered**.

4.3 Reviews

Purposes and roles of reviews

It is said that there is a fixed probability of a person making a mistake, no matter how hard they try to get things right. In product development, it is crucial to spot and prevent such mistakes through the eyes of multiple staff before the product is passed on to the next process. In the world of software development, reviews are employed in addition to tests in order to find mistakes, defects, and errors. Not only are reviews intended to improve the quality of software, but they also help in checking the software's internal structure and compliance with the project rules and standards, which cannot be evaluated by tests alone. What's more, reviews also serve as an educational tool that allows developers to learn from one another by explaining their deliverables to other people and obtaining feedback.

Reviews are often a more efficient way of detecting errors and defects than tests. Note, however, that the efficacy of a review largely depends on how it is done. For a review to be effective, it needs to be done:

- At the right timing;
- By appropriate participants; and
- Using an appropriate method.

Review methods and how they are done (examples)

Method	Description
Inspection	A systematic, rigid review method whereby a host called the moderator (a person other than the author of the deliverable) determines who will participate in the review, what roles they will play, and so on. The review involves fixing errors and checking that errors have been fixed.
Walkthrough	The author of the deliverable briefs review participants on it and asks for comments.
Pair review	An author-reviewer pair examines the deliverable under review.
Pass around	E-mail or copies of the deliverable under review are distributed to multiple reviewers for comments. The deliverable may be circulated among reviewers.

Review checklist (meeting-based reviews)

NO.	Check item	Check
1	Did the reviewee present his or her deliverable smoothly?	
2	Do participants know in advance what to review and how the review will be done?	
3	Do reviewers have a participation will?	
4	Has the deliverable under review been completed?	
5	Is a review record template available? Are reviews properly recorded?	
6	Has a conclusion been reached? Is there a system in place for informing the pertinent project members of the results?	
7	Are the opinions expressed during the review clear?	
8	Does the power relationship affect the review?	
9	Has an appropriate amount of time been allocated to the review? Was the review done properly?	
10	Are problems pointed out effectively?	
11	Is the review followed up properly?	
12	Is the review scheduled based on the progress of the entire project? Are the right people called and collected?	

Explanation of each check item

Check 1 **Did the reviewee present his or her deliverable smoothly?**

A software review begins with the reviewee presenting his or her deliverable. Therefore, the reviewee should full understand and organize the content of the deliverable and be prepared to give a concise presentation. When presenting a deliverable, the reviewee should avoid being ambiguous and ensure that the presentation is **(1) easy to understand so that it is interpreted in the same way by all the participants** and **(2) consistent with no contradictions**. Otherwise, different review participants may interpret the presentation in different ways, causing the discussion to wander from what should really be reviewed and debated.

Check 2 **Do participants know in advance what to review and how the review will be done?**

To perform an efficient and effective review, it is essential to **(1) make the subject and viewpoint of the review known** to the review participants. For example, when you will be doing a source code review, you should clearly inform the participants in advance about the subject of the review, such as functions and configuration files, and that the review will be done from the viewpoint of

reliability. Next, it is also important to **(2) share information about how the review will be done**. That way, the reviewee will know what to prepare before the review and the reviewers will understand their roles and to what extent to prepare themselves.

Check 3 **Do reviewers have a participation will?**

It is important that **(1) all participants in the review have a sense of ownership**. The level of motivation of the review participants greatly influences the outcome of the review. The greater the number of highly motivated people, the more the participants' brains are stimulated, prompting them to come up with more ideas. On the other hand, participants falling asleep or doing other work in the meeting can lower the motivation of the other participants, resulting in a lackluster review.

Check 4 **Has the deliverable under review been completed?**

Depending on the review method being used, the reviewers may need to **(1) check in advance the deliverable to be reviewed**. If the deliverable to be reviewed contains many blank pages or items marked TBD (To Be Determined), the motivation of the reviewers will be lost. This may lead to the reviewers failing to discuss the core part of the deliverable, or worse, the review itself may become meaningless. In such cases, **(2) the project leader may need to have the courage to judge the deliverable as not being ready and decide not to do the review** (reschedule it for later), even when the date of the review has been set.

Check 5 **Is a review record template available? Are reviews properly recorded?**

If you take no action based on what participants say during a review, the review will be meaningless. To ensure that appropriate action is taken after a review, it is important to **(1) categorize what reviewers have said into "suggestions," "requests," "questions," and "comments" and to then record those statements**. It is a good idea **(2) for the organization or project to prepare a review record template so that reviews can easily be recorded in a well organized format**. Also, be sure to **(3) select a recording secretary at the beginning of every review** to prevent the omission of any records.

Check 6 **Has a conclusion been reached? Is there a system in place for informing the pertinent project members of the results?**

When doing a review, you should always remember that you need to reach a conclusion. **(1) Identifying the action, deadline, and staff in charge for each suggested item helps clarify how to respond to the review results, and who is responsible**. One point requiring care is that if the reviewers' statements

consist of ambiguous requests or comments, the staff in charge may be at a loss as to what to do. It is necessary to **(2) discuss and reach a conclusion on how to respond to the review results** (not concrete steps to take for individual suggested items, but who will address them and how). And **(3) review results must be communicated to the relevant project members**.

Check 7 **Are the opinions expressed during the review clear?**

Lively discussion is important to a review. However, the staff in charge may not be able to understand abstract suggestions, such as that expressed by saying, "I feel something is wrong," misguided suggestions lacking an explanation of what the problem really is, and excessively wide-ranging suggestions. You should steer away from the abstract and make suggestions more concrete and specific so that the staff in charge can **(1) understand what the problem or issue really is** and **(2) consider ways to address the problem**.

Check 8 **Does the power relationship affect the review?**

While not peculiar to reviews, a specification or change policy may be decided based on the power relationship between people - e.g., user/developer or superior/subordinate. A review requires that all its participants discuss matters from a purely technical perspective on an equal footing. Often, however, the discussion is influenced by those who speak louder than the others. Review participants should maintain a sincere attitude when engaging in discussion. **(1) Do not decide on a policy based solely on the power relationship or compromise**.

Check 9 **Has an appropriate amount of time been allocated to the review? Was the review done properly?**

With a large-scale system, there may not be enough time to review all the deliverables for all the processes.

In some cases, therefore, you may want to review only those newly created parts of the design and specification documents while omitting the review of the other parts that are reused. If, however, you are not sufficiently careful when selecting the parts to omit, there is a chance that you will exclude the essential parts from the review. To prevent such inadvertent exclusion, you need to **(1) prioritize what to review** and **(2) allocate an appropriate amount of time to the review**. Note that merely prolonging the review process is not necessarily a good thing. From an efficiency perspective, it is also important that **(3) the review be no longer than necessary**. You should also avoid omitting a scheduled review due to lack of time or switching to a pass-around review without thoroughly considering the importance or content of the deliverable to be reviewed.

Check 10 **Are problems pointed out effectively?**

In cases where the document under review is poorly written, a review may end up with only formal errors, such as incorrect document format, typos and omissions, improper numbering, or inadequate indentation being pointed out, while more critical specification and design problems remain undiscovered. Of course, these formal errors should not be neglected (a pass-around review is effective for checking a document for formal errors). It is essential that **(1) the review be focused on the content of the specifications and design**. The reviewers should always ensure that their suggestions are effective, and the reviewees should strive to understand and fix the identified problems.

Check 11 **Is the review followed up properly?**

Even when the action, person in charge, and deadline have been decided, a review will be meaningless if it is not followed up by subsequent checking and implementation. You need to examine the review results and, if many problems are pointed out or if the result of making suggested corrections will be critical, take appropriate steps such as repeating the review. Regarding the action to be taken after the review, it is important that the review record or other appropriate document **(1) record the action/deadline check results** and **(2) indicate that the action has been checked by the leader and other members of the project team**.

Check 12 **Is the review scheduled based on the progress of the entire project? Are the right people called and collected?**

(1) A review should be scheduled in advance and done at the optimum timing. To ensure that your project proceeds smoothly, it is also important to **(2) manage the schedule so that the project can be implemented as planned**, by using the review as a milestone. **(3) Involving those people needed for the subject of the review** (e.g., legal experts in the case of safety issues) in addition to the development project members improves the quality of the review.

4.4 Tests

Purposes and roles of tests

In a broad sense, a test performed as part of software development is the process of running the software to see if it operates normally, with the aim of finding as many errors in the software as possible.

In the development phase of software development, tests are done for many different purposes. For example, a regression test is intended to check an added or modified program for any unexpected impact caused by the change. The purpose of a usability test is to check and evaluate software in terms of usability and other sensory elements.

To perform an effective test requires that the purpose and role of the test be identified, followed by the formulation of a test plan based on the identified purpose and role.

Test types and examples of purposes and roles

<Breakdown by phase>

Test type	Purpose and role
Unit testing	This test is intended to check each individual unit of software, such as a program function or class, to identify errors.
Functional testing	This test verifies whether each implemented function operates as intended.
Integration testing	This test verifies whether an execution unit combining some of the software components operates as intended and whether the interfaces function properly when handling exceptions.
System qualification testing	This test verifies whether the software operates normally in a finished product by building it into hardware, performing communication, etc.
Regression testing	This test verifies whether a change or changes to the program have had any unexpected impact on the software.
Documentation testing	This test verifies whether the product can be operated as described in the documentation (operation manual).

<Breakdown by purpose>

Test type	Purpose and role
Performance testing	This test verifies whether the processing speed and data throughput correspond to the expected values.
Stress testing	This test verifies whether the software operates normally when a load is applied to the I/O, CPU, communication, etc.
Usability testing	This test evaluates the usability of the software from the user's point of view.
Durability testing	This test verifies whether the software operates normally after prolonged product operation.
Recovery testing	This test evaluates the recoverability of the software after a failure.

<Breakdown by method>

Test type	Purpose and role
Structural testing	This test checks for logical inconsistencies, with the focus on the program's internal structure.
Black box testing	This test examines results based solely on the input and output, without considering the program's internal structure.
Path analyzing	This test analyzes and verifies the program behavior based on coverage (coverage ratio of written code).

The above lists software test types and examples of their purposes and roles. These tests may be given different names or positioned differently, depending on the company or product domain. The test staff must have a full understanding of the purpose of each test and perform the test using an appropriate method.

Tests checklist

NO.	Check item	Check
1	Do the test staff enjoy performing tests?	
2	Are the test staff independent?	
3	Is the test target clearly defined? Is an appropriate test method adopted for each test phase?	
4	Are errors recorded? Are fixed errors checked?	
5	Are sufficient test items covered for checking the system functions?	
6	Are tests performed on non-functional requirements (e.g., performance)?	
7	Is the test environment the same as or equivalent to the actual system operation environment?	
8	Is there a clear test plan? Is an appropriate amount of time allocated to the tests?	
9	Are test results checked based on clear judgment criteria?	
10	Is the cause of each detected error analyzed?	
11	Are there clear test completion criteria?	
12	Are you attempting to assure quality based on testing alone? Are you expecting too much from the tests?	

Explanation of each check item

Check 1 **Do the test staff enjoy performing tests?**

To software engineers, the most enjoyable part of programming is to "figure out how to make the program work." What about testing? A test involves two tasks: running the program to detect errors and confirming that the quality requirements for the product and software are fully satisfied. Both tasks require that a test strategy be developed and that a series of processes from test design to execution be implemented according to plan. However, there are still many organizations where tests are done on an ad hoc basis. Particularly, in the case of tests focused on the latter task, the menial work for functional checking tends to be stressed and excessive emphasis is sometimes placed on duty and responsibility, making the tests "no fun." Such "no fun tests" are not desirable from the perspectives of work efficiency and work quality. Effectively breaking down the test strategy helps to keep the test staff motivated as in development. To make the testing fun, it is necessary **(1) to make the test results visible so that the test staff remain highly motivated**, by appropriate means such as appreciating the work of staff members who have done excellent tests, and **(2) to establish a system in which the test staff enjoy doing tests**.

Check 2 **Are the test staff independent?**

Generally, there are two types of organization: one in which dedicated test staff work separately from developers and one in which developers perform all the work, including testing. In the former type of organization, products can be evaluated objectively in product tests, because test staff and developers are separated. This type of organization is also superior since it is easier to check compliance with standards and rules. The latter type of organization, in which developers are well versed in the whole series of work from specification creation and design to coding and testing, allows a flexible schedule and is also said to be highly productive. This guide does not adopt a dogmatic approach where one is claimed to be better than the other. What is important is to ensure that **(1) the test staff understand the purpose and role of the test when performing it** and that **(2) coding is not confused with testing**. In an organization with no independent test staff, time should be systematically allocated for testing and those engaged in testing should carry out tests from standpoints and viewpoints that differ from those of the developers.

Check 3 **Is the test target clearly defined? Is an appropriate test method adopted for each test phase?**

As shown in the tables provided at the beginning of this section, there are many types of test, each of which is suited to a specific phase, purpose, and method. The actual test work is done by using some of these tests in combination. This

means that numerous on-site test variations are available for actual development. To select the test combination that is most effective for the target system or software from these diverse test variations, you need to **(1) clearly define the test target and scope and gather the documents necessary for test design and test result judgment (system specifications, interface specifications, etc.)** for each test phase. Then, you need to identify the purpose of the test, select an appropriate method, and design the test. In other words, it is essential that, **(2) in each phase, you carry out the type of test appropriate for the purpose**. Also, you need to **(3) clearly define the test target and scope** for the test of each phase.

Check 4 **Are errors recorded? Are fixed errors checked?**

Needless to say, a test would be meaningless unless its results are recorded. It is necessary to **(1) create and maintain detailed records of detected errors**, including test repeat procedures, in preparation for future fixes and retests. Judgments and fixes made in response to such errors should also be recorded. Many projects reuse existing program source code or use derived versions. Therefore, you need to check the scope of the impact on related systems in addition to the information specific to the target product (version) and **(2) record and manage the action policies and measures if there is any impact on other versions**.

Check 5 **Are sufficient test items covered for checking the system functions?**

Those requirements that software needs to satisfy with regard to functions in order to satisfy user requirements are called functional requirements. With system functions increasing in recent years, the number of software functional requirements has been growing consistently. The functions that software is required to implement vary depending on the system operation context (operation environment, operating conditions, etc.). In some cases, you even need to consider function overlaps and parallel operation. From this perspective, it is necessary to **(1) prepare sufficient test items, taking into consideration the relevant factors such as possible operation variations** of the functions to be implemented by the system and software. It is important that, for each test, you prepare and execute test items appropriate for the purpose of the test. You should also ensure that test results are checked. As with software development deliverables (documents, source code, etc.), it is ideal to **(2) have test items reviewed by a third party for omissions and validity**. Also, you need to prepare and execute test items that are appropriate for the development target. In the case of those systems having many user interfaces, in particular, **(3) extensive test items should be prepared based on appropriate operations and scenarios, considering potential uses of the system by the user**.

Check 6 **Are tests performed on non-functional requirements (e.g., performance)?**

As opposed to the functional requirements explained in the preceding paragraph, all other elements (performance, capacity, data security, expandability, etc.) are called non-functional requirements. In many embedded systems, non-functional requirements are as significant as functional requirements, and it is not uncommon for their volume or importance to exceed that of the functional requirements. Developers need to consider and test not only functional requirements but also these non-functional requirements. Non-functional requirement tests include **(1) performance consideration and testing**, **(2) capacity consideration and testing**, **(3) data security consideration and testing,** and **(4) expandability consideration and testing**. These tests differ considerably depending on factors such as the target system's characteristics and system characteristics profile. You need to check the items regarding non-functional requirements in the specification creation and design stages and execute testing from the necessary non-functional aspects.

Check 7 **Is the test environment the same as or equivalent to the actual system operation environment?**

In cases where the operation/connection environment of the target system has many hardware components to be tested - e.g., when a cellular phone application is tested - it may be difficult to select the test target from among an enormous number of cellular phone models. Also, in embedded software development, software is developed in parallel with the hardware of the target system, which often leads to a thorny situation where you need to perform a software test before the corresponding hardware even exists. In a situation like this, it is important to **(1) prepare a test environment appropriate for each phase**, considering factors such as what is the optimal operation environment and what should be prepared and built as a temporary operation environment.

Check 8 **Is there a clear test plan? Is an appropriate amount of time allocated to the tests?**

In software development, it is a good idea to draw up a test plan together with a project plan. If creating a test plan independently is difficult, one solution is to include it in a project plan. In either case, you should ensure that **(1) the type, scope, and schedule of the test to be done are defined when a project plan is formulated**. Also, the time available for testing is often restricted because of a tight development schedule. It is essential to clarify a test plan when creating a project plan and **(2) allocate an appropriate amount of time for testing** in advance.

Check 9 **Are test results checked based on clear judgment criteria?**

Testing usually involves three tasks - preparation, execution, and result confirmation. Of these, you should check the content of the test items prepared in the test preparation task and those actually executed, while keeping the test target in mind. As with software development deliverables (documents, source code, etc.), it is ideal to **(1) have test items reviewed by a third party for omissions and validity**.

It is important to identify the applicable test items and test method, with the test target in mind, before conducting a test. The confirmation of test results entails making a judgment by comparing the actual test results with the expected test results indicated in the test procedures, test items, etc. To do so, you need to include clear judgment criteria for the test results when creating the test procedures, test items, etc. You should also ensure that, after a test is done, **(2) test results are recorded**. Particularly, **(3) in the case of those events that are difficult to reproduce, as much detail as possible, including the situations in which the events occurred, should be recorded** so that accurate information can be provided to developers and other related parties.

Check 10 **Is the cause of each detected error analyzed?**

It is essential to **(1) analyze the cause of each error detected** in testing. The analysis of error causes helps find similar errors and prevent errors in derived products under development. Furthermore, providing feedback on the true cause of an error to a subsequent product development project greatly contributes to improving the quality of the upcoming product. In the case of a critical error or an error having an extensive impact, in particular, it is vital to identify its underlying cause, share information on the error across the organization, and **(2) consider ways to prevent its recurrence on an organization-wide basis**.

Check 11 **Are there clear test completion criteria?**

One aspect of testing that is difficult to determine is how far to proceed with the test. A perfect test plan is difficult to create, and the amount of time available for testing is limited. You must, therefore, perform testing efficiently within a limited timeframe. It is necessary to understand that a software or system test is done by running the test target temporarily for testing purposes and, in this respect, it can be said to be a "snapshot" test that is focused on part of the life cycle of the system. This means that a software test is subject to limits. Therefore, it is essential to gain a full understanding of the quality and reliability requirements for the system and consider what to check in the test. You should **(1) decide on test completion criteria and draw up an appropriate test plan** in the test planning phase so as to **(2) avoid the wasteful repetition of tests**.

Check 12 **Are you attempting to assure quality based on testing alone? Are you expecting too much from the tests?**

Various test types were discussed at the beginning of this section, but none is superior to any of the others. While testing is the last chance to assure quality before the shipment of software, there is a limit to how much you can improve low-quality software, no matter how many times you test it. It is, of course, necessary to combine multiple tests when creating a test plan. To improve the quality of software, however, you also need to formulate and implement a comprehensive strategy, taking advantage of the merits of many different techniques such as testing, review, and specification checking. Of course, these are premised on more essential activities required for the improvement of software quality, including procurement and improvement of skills of personnel in charge of development or testing as well as process improvement. You need to keep all this in mind while **(1) creating and executing a comprehensive, integrated quality management plan**.

4.5 Quality Establishment Using Metrics

Ensuring that evaluation metrics are adopted by your organization

This guide has discussed how to understand the product to be developed (system characteristics profiling: SCP), how to understand the situation of the project (project characteristics profiling: PCP), and how to define evaluation metrics for quality establishment based on an understanding of these profiles. For quality establishment efforts using evaluation metrics to take effect, you not only need to define and measure evaluation metrics but also have the results adopted into your project. To ensure more effective quality establishment efforts, it is also essential to ensure that the contents of this guide are understood across the organization and that its concepts are adopted. Note that evaluation metrics are merely objective and indirect indicators. Applying various methods and improving the quality of the work itself is indispensable.

To summarize this guide, this section offers tips on how to define effective evaluation metrics and how to leverage these metrics to benefit your organization.

Checklist for quality establishment using metrics

NO.	Check item	Check
1	Are you obsessed exclusively with evaluating quantitative data?	
2	Is system characteristics profiling done properly?	
3	Is project characteristics profiling done properly?	
4	Do you select evaluation metrics based on their effectiveness? Is the selected measurement method appropriate?	
5	Are the target values for the evaluation metrics appropriate?	
6	Do you make effective use of existing proven methods in your project?	
7	Do you have a system in place for reflecting the measurement results on your project and providing feedback to subsequent development projects?	
8	Do all the project members understand the importance of using this guide and how to use it?	
9	Do the project members have the time and opportunities to learn?	
10	Are the project members clearly aware of their skill levels?	
11	Did you perform reviews during the process from system characteristics profiling to quality target value setting?	
12	Do you intend to make the findings of your quality quantification widely available to the public?	

Explanation of each check item

Check 1 **Are you obsessed exclusively with evaluating quantitative data?**

This guide has explained how to adapt the reference values to match your project through a series of processes of system characteristics profiling, project characteristics profiling, and evaluation metric setting. Once numerical targets are set, people tend to become obsessed with numbers. In this guide, however, we want to suggest the need to understand what those numerical values represent and then to use them for quality establishment. You should not react nervously to changes in numerical values; it is essential to act based on an understanding of what they really mean.

First, **(1) do not use reference values as is**. The reference values shown herein are purely for reference purposes and should not be interpreted as being absolute target values. There is a tendency for management staff to focus solely on numbers. When using the reference values, make sure that you understand the meaning of those values.

Next, **(2) do not make it a goal to achieve numerical targets alone**. In this guide, those elements that can be measured objectively are used as evaluation metrics. These metrics help you to evaluate the quantitative aspects, but not the quality of the work. For example, repeating any one test raises the value of the

test process effort, as well as the work effort ratio and work execution ratio of the test work. However, merely repeating a test does not change the quality of the software being tested. Similarly, if you do a system specification review slowly or have it done by more people than necessary, the value of the specification review process effort will increase, leading to greater work effort ratio and work execution ratio of the specification review. As demonstrated by these cases, manipulating numerical values is easy. However, doing so does not give you any control over quality, which was the original aim, and the target values you set will become meaningless. When attempting to control quality, keep in mind what you were attempting to do in the first place.

Check 2 **Is system characteristics profiling done properly?**

The most important aspect of system characteristics profiling is to **(1) identify the quality requirements from the viewpoint of the user** for what is supplied. What criteria does your organization use when setting quality targets? Do you rely solely on intuition? Are your decisions affected by development-side constraints, such as limited manpower, an imminent release date, or intolerance of bugs? While numerical values that are set in these ways often turn out to be correct as empirical values, they can still cause problems. Since quality targets are not appropriate, for example, the required level of quality cannot be secured and many bugs are not found until after shipment. Or, because more effort than necessary is put into quality assurance, the developers become exhausted or the software cannot be released on schedule. System characteristics profiling (SCP) is a means of supporting the rationale for the quality targets you regard as being appropriate.

Doing system characteristics profiling properly and gaining an objective understanding of the product to be developed allows you to think about quality targets rationally. It will also make members of those departments only partially involved in the development able to consider which part of the system will be impacted by the software they are developing, what kind of impact will be caused, and ultimately how the quality will be affected. **(2) Seeing quality as part of a bigger picture** can further improve quality.

Also, when doing system characteristics profiling, do you **(3) hold discussions with stakeholders such as the customer and hardware development members**? We believe that such discussions build a common understanding among the system stakeholders. Since system characteristics profiling involves checking not only the quality of the software but also the quality of the system as a whole, **(4) hardware quality targets can also be set** during this process. Use system characteristics profiling as an opportunity to think about the kind of quality the public expects from the product.

Column

Manufacturers' economic cost upon the occurrence of system trouble

There have been discussions within SEC as to whether "manufacturers' economic cost upon the occurrence of system trouble" - the 10th item for project characteristics profiling - should be covered by system characteristics profiling. We have also received public comments arguing that any recall cost should be added to the economic cost. After considering this matter from the user's point of view, however, we have come to the conclusion that system characteristics profiling is not applicable to this item. We suggest that system characteristics profiling should be done with a focus on the level of quality demanded by the user, rather than the convenience of the developers.

Check 3 **Is project characteristics profiling done properly?**

Project characteristics profiling involves identifying factors that hinder quality establishment and those that promote it. In other words, it is the process of **(1) determining and expressing weak points and strong points from the perspective of quality establishment**. As discussed in Chapter 2, for example, if the project is assigned entirely to new recruits, then it is necessary to step up the review process by having many people participate in checking whether the software has been created properly, so as to assure quality. If the software size is extremely small, the project may require much less process effort for review and testing than would normally be necessary for such a size. Project characteristics profiling is a means of expressing what a competent manager does naturally as a numerical value. It allows you to express what you normally decide as the project proceeds as a numerical value in advance, which makes it easer to gain a consensus on your decisions from other project members.

Whether you can perform effective project characteristics profiling depends on **(2) whether you have an accurate grasp of your project situation**. Holding hearings and discussions with developers for project characteristics profiling helps you recognize the problems that your project faces. Also, by considering what to use as the project characteristics profiling factors for calculating adjustment coefficients, you can find values that are better suited to your project. Note that the adjustment coefficients given in this guide are for reference purposes only. Note that not all of the project characteristics profiling factors will necessarily affect all the work. You may want to consider only those factors that greatly impact your work.

Check 4 **Do you select evaluation metrics based on their effectiveness? Is the selected measurement method appropriate?**

Gathering and analyzing data properly during a development project is difficult for an organization that is not familiar with such activities. If obtaining data for basic metrics places a huge burden on the developers, the project itself may fail. If you have never gathered data for a project, you first need to understand what each evaluation metric means, define the minimum evaluation metrics required for your project, and then begin by measuring the necessary basic metrics. **(1) Starting by doing what you can and then improving little by little** leads to long-term project success.

To make effective use of the gathered data, it is important to **(2) decide in advance how to gather data** (gathering method, timing, reporting method, etc.). To prevent the creation of irregular reports, it is a good idea to **(3) prepare a standard form**. Providing a means of gathering data with ease, such as having data gathered automatically by a tool, is also effective.

Check 5 **Are the target values for the evaluation metrics appropriate?**

Defining the evaluation metrics determines the basic metrics whose data is to be collected and the evaluation metrics to be evaluated. The next step involves setting quality target values. Initially setting targets that are too high, irrespective of the project situation, makes it difficult to continue the project. **(1) Are the results of project characteristics profiling properly reflected**? Considering what and how accurately you can measure, judge whether **(2) the set evaluation metrics are appropriate**. While it is essential to try to achieve the target values derived from the reference values, you should decide on a margin of error based on your experience and past data. After setting appropriately high target values, do not forget to establish a means of attaining them.

Check 6 **Do you make effective use of existing proven methods in your project?**

This guide describes how to define evaluation metrics for quality control and how to measure basic metrics for evaluating those evaluation metrics. If, however, your organization already has a method in place that is used for a similar purpose, you do not need to change those existing rules. **(1) If there is any gathered data, compare it to the data shown in this guide and select an effective method while replacing the metrics appropriately**.

For example, this guide uses the physical number of lines of source code as a metric that indicates the program scale. If your organization uses FP as a scale indication metric, you can set the target value of the metric normalized to the scale (execution ratio of the work, document volume, etc.) by comparing it with the actual value for your organization. You can also adopt the target value shown

in this guide by converting the FP to a code. The FP-to-code conversion ratio may be set based on the knowledge of your organization or the reference values presented in multiple reference books (e.g., those written by Capers Jones). Also, this guide sets the control statement description ratio for the source code as a metric for simple measurement of the source code complexity. If, however, your organization measures and evaluates complexity (e.g., cyclomatic complexity) by using a tool, it is not necessary to perform control statement measurement.

If you have gathered data but feel that it is not being used effectively for evaluation, you may want to review your evaluation process based on this guide.

Check 7 **Do you have a system in place for reflecting the measurement results on your project and providing feedback to subsequent development projects?**

The use of values obtained using evaluation metrics should not be limited to the evaluation of a single development project. Using those values for subsequent development projects ensures continuous quality improvement for products and allows project members to step up their skills. After the end of your development project, check the evaluation metrics to determine whether they have been achieved. If any metric falls short of or exceeds the target value, analyze the cause of the failure, the validity of the target value, etc. and **(1) provide feedback on the results to subsequent development projects**. At first, you may feel at a loss as to how to handle project characteristics profiling factors, reference values for evaluation metrics, and other data. We ask that you stay the course. Once the development is complete, look back and, if there are any evaluation metrics that you have failed to achieve or had difficulty in achieving, **(2) analyze the cause after the development and make sure the result is reflected on subsequent development projects**.

Furthermore, as described in Section 2.6, you can always compare the analysis results of system characteristics profiling with the results for the actual project by using the ST-SEISMIC scale and feeding back those results to the project so that the values become appropriate (this does not involve making compromises).

Accumulating data helps reinforce the project experience. It also means enhancing the capabilities of the organization. Do not be nervous when faced with changes in numerical values; take a long-term view and provide continuous feedback to the project.

Check 8 **Do all the project members understand the importance of using this guide and how to use it?**

The quality of software is invisible. As discussed in Section 1.2 using an example of apples, this guide uses a common measure, called evaluation metrics, in a project in order to make quality as visible as possible.

The use of a measure allows you to **(1) recognize the level of quality of what you are creating** and **(2) understand what you should be creating**. To use a measure properly, you need to know what that measure is. Also, by understanding how to use the measure (how to measure things) and what it is used for, you can use it more effectively.

If the project members understand the evaluation metrics and are committed to the use of those metrics, you can have the members **(3) accumulate data without forcing** them to do so. This will be even more effective.

Column

Processes and conventions

In software development, adopting appropriate processes and conventions leads to higher quality. The reference values for the evaluation metrics are rule-of-thumb values assumed to be obtained when "appropriate work" is done in the individual processes. Here, "appropriate work" refers to the act of creating something properly by means of a defined process according to a defined procedure. The "Embedded System Development Process Reference" (ESPR) published by SEC will help you learn about how to define and implement processes and how to write related documents. If you want to know about the coding practices, the "Embedded System Development Coding Reference" (ESCR) published by SEC will provide you with the necessary information. For information about the use of other technical methodologies, see the individual sections of Chapter 4 in this guide, as well as the list of reference books at the end.

Check 9 **Do the project members have the time and opportunities to learn?**

Of course, controlling quality through the application of evaluation metrics is necessary for quality improvement. It is more important, however, to meet required specifications and prevent errors or, in other words, to ensure quality establishment. To do so, the project members need to be constantly improving their technical skills. Creating a proper system or software requires not only software engineering technology but also a knowledge of domains, human skills, and many other kinds of knowledge and technology including management skills, quality management technology, and element technology. It is also necessary for your organization to be able to develop technologies needed to create new products.

One solution to this is to gather together people having all the necessary knowledge and skills, but that is difficult to achieve in most cases. There are

a number of ways for project members to gain the necessary knowledge and skills - e.g., reading books, holding study sessions in the workplace, participating in seminars, and asking customers for advice. On-the-job training is another effective method.

As is evident from the list of reference books given at the end of this document, many good books are available on the topic. Do your best to obtain and read some of these books. Also, **(1) the project members should keep improving their technical skills and abilities by taking part in seminars or by other means**.

Check 10 **Are the project members clearly aware of their skill levels?**

In order for the project members to make effective use of evaluation metrics and accomplish quality control in ways commensurate with their ability, it is vital that they have a basic understanding of what they can do. We think that going through the processes of project characteristics profiling and quality target value setting will help project members better understand their skill levels.

Implementing projects, conducting reviews, testing, measuring processes and products, and summarizing the measured data, which are techniques for developing products, all require different skills. It is necessary to **(1) identify what kind of skill is needed for each phase of development** and **(2) understand which skills you possess**. If you find that your skills or abilities is not enough to implement your project, you should consider taking adequate steps, such as introducing the necessary technology or holding study sessions.

Check 11 **Did you perform reviews during the process from system characteristics profiling to quality target value setting?**

In this guide, we have suggested that you check whether processes and deliverables have been reviewed thoroughly as a way of measuring quality. Of course, you should also perform reviews when using this guide. Ensure that **(1) a review is conducted whenever system characteristics profiling or project characteristics profiling has been done or quality target values have been set**. Going through the review process or holding discussions with stakeholders may bring you a new awareness. We think that what is even better is that **(2) reviews will promote communication among the project members, potentially helping them improve their skills and raise their awareness**.

Check 12 **Do you intend to make the findings of your quality quantification widely available to the public?**

In this world, there are many techniques for establishing quality. However, the knowledge of the target values adopted to evaluate how effectively those techniques are used and to determine the extent to which they are to be employed has been confined to individual organizations and has rarely been

made public. Through hearings and surveys conducted by SEC, we have also heard many people say that they want to know how other companies deal with evaluation metrics or that they are not sure about how to set targets. We therefore prepared this guide to suggest the metrics to measure, when to measure them, and their target values. We also understand, however, that these values, calculated based on the results of hearings with companies, are likely to change with advances in the rapidly evolving software industry and technology. We are aware that there is still room for scrutiny and that values in the field will play a critically important role. We asked a number of companies to submit such data when preparing this guide but, unfortunately, some declined our request. We would like the readers of this guide to **(1) provide feedback to SEC** on the actual data for the metrics suggested in this guide, as well as any other relevant metric data. Your feedback will be reflected in the next version of the guide. In addition to submitting data to the SEC, you are also encouraged to contribute to the academic community for the benefit of society. In recent years, there has been a significant drop in the number of research papers contributed to academic circles. Paper-driven information disclosure and discussion has been in the doldrums. Submitting research papers to the academic community not only makes a major technical contribution but also helps you organize your activities and provides you with opportunities to hear opinions from more people. It will benefit both your organization and yourselves.

Appendix A Reference Books

Listed below are the books that we referenced when writing this document, as well as those that we believe to be useful references for the process from software development to quality assurance. In recent years, a host of books on software engineering and other related topics have been published. Of these publications, the list includes books on the basics and classics that have been read for many years.

Title	Author	Published by	Published in
Engineering			
Software Engineering: A Beginner's Guide	Roger S. Pressman	McGraw-Hill Science/ Engineering/Math	1988
Software Engineering	Ian Sommerville	Addison Wesley	2006
SOFTWARE ENGINEERING: A Practitioner's Approach	Roger S. Pressman	McGraw-Hill Inc., US	
Software testing			
The Art of Software Testing (Business Data Processing)	Glenford J. Myers	John Wiley & Sons	2004
Software Testing Techniques	Boris Beizer	Intl Thomson Computer Pr (T)	1990
Metrics			
Software Quality Analysis and Guidelines for Success	Capers Jones	Intl Thomson Computer Pr (T)	2000
Applied Software Measurement	Capers Jones	Global Analysis of Productivity and Quality	2008
Practical Software Measurement: Objective Information for Decision Makers	John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, Fred Hall	Addison-Wesley Professional	2001
Metrics and Models in Software Quality Engineering	Stephen H. Kan	Addison-Wesley Professional	2002
Five Core Metrics: The Intelligence Behind Successful Software Management	Lawrence H. Putnam and Ware Myers	Dorset House	2003
Creating a software engineering culture	Karl E. Wiegers	Dorset House	1996
Reviews			
Software Inspection	Tom Gilb, D. Graham	Addison-Wesley Professional	1993
General			
Peopleware	Tom Demarco, Timothy Lister	Dorset House	1999

Editors

Masayuki HIRAYAMA	IPA Software Engineering Center
Satomi YOSHIZAWA	IPA Software Engineering Center
Sayuri YAMAGUCHI	IPA Software Engineering Center
Yutaka UKON	IPA Software Engineering Center

People who attended Article review

Sumio IZAWA	NEC Corporation
Naoko UEDA	FUJITSU LIMITED
Katsumi OHNO	Toyota Technical Development Corp.
Keiko KOGA	Hitachi Systems & Services, Ltd.
Fumio SHISHIDO	eSOL emBex Inc.
Makoto NONAKA	TOYO UNIVERSITY
Fusako MITSUHASHI	NEC Corporation
Taro YAMAZAKI	Nihon Unisys, Ltd.

ESQR

Embedded System development Quality Reference Guide

Ver.1.0

August 1, 2010
Written and edited by Software Engineering Center,
Information-technology Promotion Agency, Japan

http://www.ipa.go.jp/english/sec/