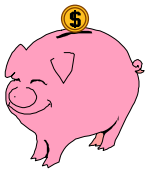
Chapter 4. Core Measures

Chapter Highlights

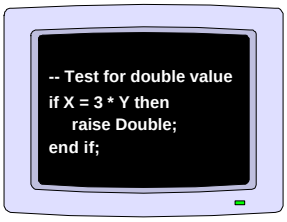


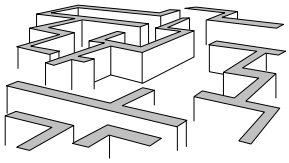
COST

- Reporting period dates
- Total effort
- Effort by development and maintenance activity

ERRORS

- Dates error reported and corrected
- Effort to isolate and correct the error
- Source and class of error



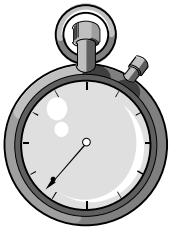


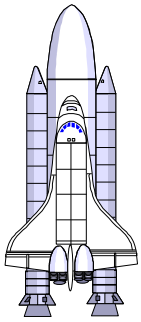
PROCESS CHARACTERISTICS

- Identification of programming languages
- Indication of the use of significant processes
- Description of measurement study goals

PROJECT DYNAMICS

- Changes to requirements
- Changes to code
- Growth of code
- Predicted characteristics





PROJECT CHARACTERISTICS

- Development dates
- Total effort
- Product size
- Component information
- Software classification

This chapter describes a set of core measures that any organization can use to begin a measurement program. There is no universal, generally applicable collection of measures that will satisfy the needs and characteristics of all organizations. However, on the basis of the experiences of mature measurement programs throughout NASA, a set of measures in the following five categories will typically be required by any software development and maintenance organization:

1. Cost
2. Errors
3. Process characteristics
4. Project dynamics
5. Project characteristics

Although organizations beginning a measurement program may want to use the core set as a baseline, they will soon find that additional information is required to satisfy their specific goals and that some of the core measures are not required. Each organization should use those measures that reflect its own goals. As its measurement program matures, the organization will recognize which measures support those goals and which provide no added value.

The recommended core measures in each of the categories exhibit the following important attributes. They

- Address the three key reasons for measurement
  1. Understanding
  2. Managing
  3. Guiding improvement
- Support both software development and software maintenance activities
- Are easy to collect and archive
- Are based on the experience of mature NASA measurement programs

The following sections provide further information on the core measures.

4.1 Cost

Cost is the most universal and commonly accepted measure for understanding and managing software processes and products. Consequently, cost data represent the most essential part of any measurement program. Although many development organizations assume that the cost data must be extensive and detailed to capture the overall cost characteristics of a software project adequately, the cost data should actually be easy to capture. If a programmer needs more than a few minutes each week (on the average) to record his or her effort, then the forms require too much data. As long as the managers are aware of the total amount of effort required for the software projects, an organization can gain a significant amount of insight by observing the trends

over time. The simplest, yet most critical, cost measure is the record of the total expenditures for a project.

4.1.1 Description

Collect effort data at least monthly.

Every project must capture staff effort data on a consistent, periodic basis. A monthly schedule is recommended, at a minimum; however, many major NASA measurement programs capture effort data biweekly or even weekly. The higher frequency requires little additional work and provides more project characterization detail.

Clarify the scope of effort data collection.

The scope of the effort data collection depends on the organization's goals. Each organization must determine precisely who will supply effort data, at what point during the software life cycle measurement will begin, and when data collection will terminate. Typically, effort data must be collected for all personnel who charge their time to the software project, specifically, technical, management, secretarial, and publications staff.

For every data reporting period, each individual must minimally report the total number of hours of effort and a breakout of the number of hours per activity (e.g., design, code, test, or other).

A decision concerning the reporting of unpaid extra hours of effort must be based on whether the intent is to measure the actual effort expended or the actual effort charged. Some organizations maintain separate records of unpaid overtime hours.

Within the SEL, every programmer and every first- or second-line manager provide effort data. Data collection starts when the functional requirements have been completed and the software life cycle begins with the requirements analysis phase.[6] For *development* projects, data collection continues until the system is turned over for operational use. For *maintenance* projects, data collection starts at the beginning of the operations phase and continues until the analysts determine that no additional value will be gained from further collection. Each maintenance project is judged on its own merits. Some may provide data for 1 year only, whereas others provide data until the software is retired.

4.1.2 Data Definition

When the measurement program is first established, personnel from the analysis component must define the activities to ensure clarity and internal consistency. Focus should be on using locally

[6] For all five categories of measures, the SEL begins to capture data no earlier than the beginning of the software requirements analysis phase. System requirements definition is normally performed by a different organization from the one that develops the software.

developed definitions for the activities. Excessive time should not be spent trying to be consistent with outside organizations.

All project personnel (e.g., programmers, managers, QA staff, CM staff, and testers) provide the data listed in Table 4-1. Additional resource data on the documentation effort (total hours by publications) and the clerical effort (total hours charged by secretarial support) may be extracted from project management accounting records, as long as there is a definition of scope and characteristics. The data must be consistent from project to project and should provide an accurate history of the cost required to produce and to maintain the software product.

Table 4-1. Data Provided Directly by Project Personnel

Data	Descriptions
All Effort	
Date	Date of the end of the reporting period
Total effort	Total hours charged to the project during that period
Development Activity Only	
Hours by development activity	Predesign Create design Read and review design Write code Read and review code Test code units Debugging Integration test Acceptance test Other
Maintenance Only	
Hours by maintenance class	Correction Enhancement Adaptation Other
Hours by maintenance activity	Isolation Change design Implementation Unit test and system test Acceptance test and benchmark test Other

The SEL Personnel Resources Forms (see Figures A-5 and A-6 in Appendix A) and the Weekly Maintenance Effort Form (see Figure A-13) are examples of forms used to capture effort data for development and maintenance projects, respectively. Programmers and managers typically complete a form every week. Both forms provide space for recording total hours and the distribution of hours by activities. To reduce questions and confusion, the definitions of the

activities are supplied on the forms. Other organizations may use different definitions as long as they are applied consistently throughout the organization's measurement program.

Figure 4-1 summarizes the life-cycle phases, sources, and frequency for cost data collection. Typically, organizations separate the costs of development and maintenance activities.

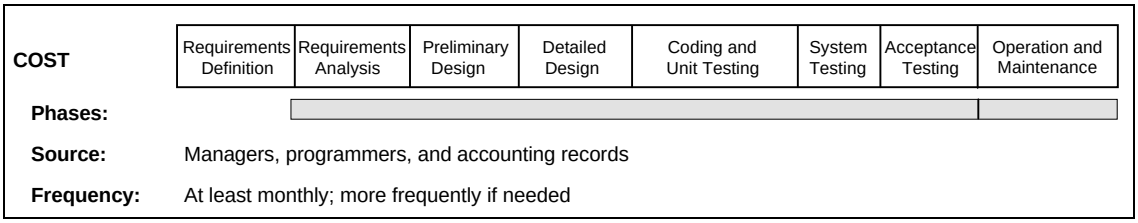


Figure 4-1. Cost Data Collection Summary

4.2 Errors

Error data make up the second most important category of core measures. A better understanding of the characteristics of software defects is necessary to support a goal of higher quality and greater reliability. Error data may minimally include only counts of defects detected during a specific life-cycle phase; at the other extreme, error data may include detailed descriptions of the characteristics of the errors and information on where the errors came from, how they were found, and how they were corrected. The level of detail must be driven by the goals and needs of the particular organization. This section recommends core error measures based on those collected within a successful measurement program in a medium-sized NASA organization.

4.2.1 Description

The core error measures consist of the

- Date the error was found
- Date the error was corrected
- Effort required to isolate and correct the error
- Source of the error
- Error class

When the measurement program is first established, the measurement analysts must define the scope of the error reporting activity.

Collect error data only for controlled software.

Error data should be captured only after a unit of software has been placed under configuration management control. This recommendation, which is based on 17 years of experience, may seem

counterintuitive. However, until CM checkout and checkin procedures have been established as prerequisites for making changes, consistent error reporting cannot be guaranteed. Within the SEL, a unit is turned over for configuration control only after it has been *coded*. Other NASA organizations (e.g., JPL) have reported significant improvements from collecting and analyzing data about defects detected and corrected during formal inspections of requirements documents (see Reference 26).

Do not expect to measure error correction effort precisely.

Programmers focusing on their technical activities may not be able to report the exact amount of time required for a particular change. Forms should allow them to estimate the approximate time expended in isolating and correcting an error.

4.2.2 Data Definition

After completing a software change, a programmer submits the appropriate change form with the data shown in Table 4-2. A change form is required whenever a controlled software component is modified, whether or not the detection of an error necessitated the change. Experience has shown that the process of reporting such changes enhances configuration management and that the information proves useful in modeling the dynamics of the software in an organization. In addition to the measures already cited, a maintenance change form must include the type of modification. As always, it is important to focus locally when defining the error classes.

Table 4-2. Change Data

Data	Descriptions
All Changes	
Date error reported	Year, month, and day
Date error corrected	Year, month, and day
Source of error	Requirements, specification, design, code, previous change, other
Class of error	Initialization, logic/control, interface, data, computational
Effort to isolate error	Approximate number of hours
Effort to implement change	Approximate number of hours
Maintenance Changes Only	
Type of modification	Correction, enhancement, adaptation

The SEL Change Report Form and the Maintenance Change Report Form (see Figures A-1 and A-4 in Appendix A) are examples of forms used to capture error data for development and maintenance projects, respectively. In either case, a single form is used to report both software errors detected and software changes to correct the errors. Programmers use only one form to report one error that requires changes to multiple components.

Figure 4-2 summarizes the life-cycle phases, sources, and frequency for error data collection.

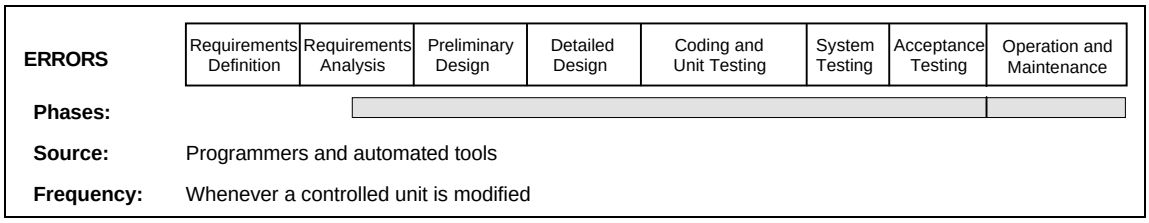


Figure 4-2. Error Data Collection Summary

4.3 Process Characteristics

> ***Do not expect to find generalized, well-defined process measures.***

Focusing on the process characteristics category of software measures allows investigation into the effectiveness of various software engineering methods and techniques. Looking at process characteristics also provides insight into which projects use related processes and can thus be grouped together within the measurement program to derive models and relationships or to guide improvements.

Because few process features are consistently defined and can be objectively measured, few core measures are recommended in this category. Rather than capturing extensive process characteristics, it is suggested that some basic information be collected about the development process used for the project being measured.

4.3.1 Description

The recommended core process measures are limited to the following three:

1. Identification of development language(s)
2. Indication of the use of specific processes or technology [e.g., the Cleanroom method or a particular computer-aided software engineering (CASE) tool]
3. Description of measurement study goals

Common descriptions of measures do not exist for such fundamental software engineering process elements as methodology, policies, automation, and management expertise. Therefore,

recommending that such measures be included in the core set is not useful. Measures such as these must be defined and analyzed locally for consistency with the organization's goals.

Do not expect to find a database of process measurements.

Detailed process descriptions cannot be stored in a database. Instead, important process information is often provided in papers and reports. For example, if an organization is studying the impact of using different testing strategies, the analysts must capture the detailed information about the results of applying different techniques and report on the results.

Understand the high-level process characteristics.

Before attempting to capture advanced process measurement data, an organization must have a clear understanding of the core process measures. Experience within the SEL has shown that the most important process characteristic is the choice of programming language; the availability of this information may provide further insight during the analysis of other measurement data.

4.3.2 Data Definition

Table 4-3 summarizes the core process characteristics measures. Figure 4-3 summarizes the life-cycle phases, sources, and frequency for process characteristics data collection.

Table 4-3. Process Characteristics Data

Data	Descriptions
Development language	Language name: percentage used Language name: percentage used ...
Important process characteristics (if any)	One-line textual description (e.g., "used Cleanroom")
Study goals	Brief description of the goals and results of the measurement study associated with the project

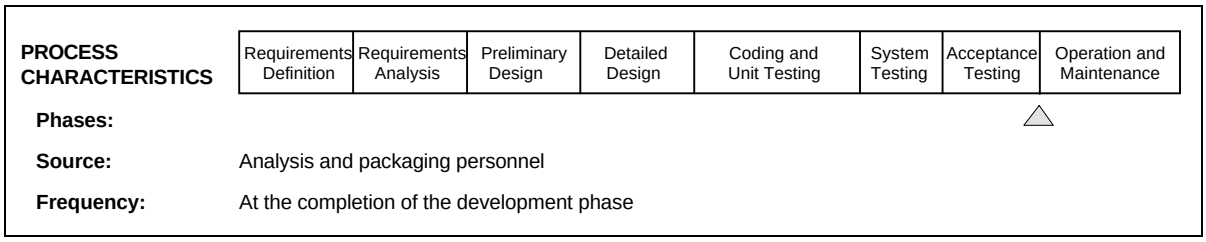


Figure 4-3. Process Characteristics Data Collection Summary

4.4 Project Dynamics

The next category of core measures—project dynamics—captures changes (to requirements, to controlled components, and in the estimates for completion) during the software life cycle. Experience has shown that such information aids management and improves understanding of the software process and product.

4.4.1 Description

The core measures in this category characterize observed changes in the project requirements and the product code, as well as updated estimates of the final project characteristics (see Section 4.5). These measures consist of

- Changes to requirements
- Changes to baseline code
- Growth in baseline code
- Predicted project characteristics

Requirements changes represent the overall stability of the software requirements and can be used effectively to manage the development effort and to improve understanding of the characteristics of the software problem definition in the local environment.

Records of changes to the code and the growth of the code provide insight into how the various phases of the life cycle affect the production of software, the most tangible product that a development process generates. Change measures are useful in managing ongoing configuration control processes, as well as in building models of the development process itself.

The measures of predicted project characteristics are excellent management aids and are useful for studying the cause and effect of changes, as well as process and problem complexity. The characteristics should be captured on a regular basis, at least monthly.

4.4.2 Data Definition

The Project Estimates Form (see Figure A-8 in Appendix A) is an example of a form used to provide predicted project characteristics at the start of the project and periodically throughout the life cycle. Table 4-4 summarizes the core project dynamics measures, and Figure 4-4 summarizes the life-cycle phases, sources, and frequency for project dynamics data collection.

Table 4-4. Project Dynamics Data

Data	Descriptions
Changes to requirements	Count and date of any change made to the baselined requirements specifications
Changes to code	Weekly count of the number of software components changed
Growth of code	Biweekly count of the total number of components and total lines of code in the controlled library
Predicted characteristics	Monthly record of the estimated completion dates and software size
Dates	End design End code End testing System completed
Size	Total components Total lines of code (new, reused, modified)
Effort	Total staff months (technical, management, support services)

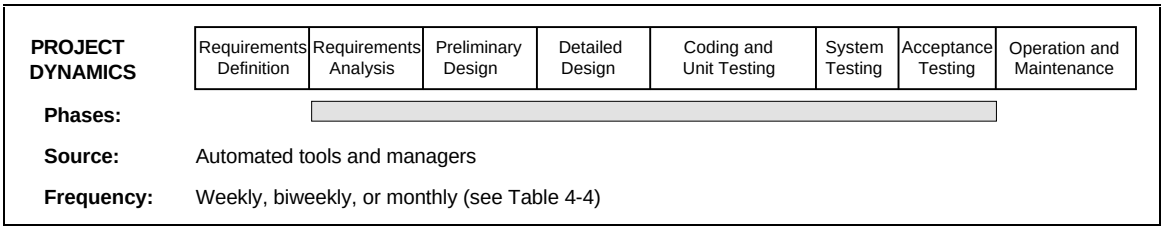


Figure 4-4. Project Dynamics Collection Summary

4.5 Project Characteristics

The core measures that characterize the completed project constitute another essential part of the measurement program. Organizations derive models and relationships from project characteristics in the historical database. Without a basic description of the overall software project effort, it is difficult to apply the other measurement information in a meaningful manner.

4.5.1 Description

The project characteristics can be broken down into five categories of core measures:

1. Development dates
2. Total effort
3. Project size
4. Component information
5. Software classification

Use simple definitions of life-cycle phases.

The important dates are the beginning and the end of each life-cycle phase and the final project completion date. If the organization is using a strict waterfall life cycle with nonoverlapping phases, then the end of a nonterminal phase is defined by the beginning of the subsequent phase. When a different life-cycle methodology is applied, the organization will have to adjust the structure of the project characteristics data. Each organization must determine how it wants to capture details of the key phase dates within the software life cycle. The simplest approach is to use the classical phase definitions of a standard life-cycle methodology. However, as long as an organization has its own consistent internal definitions, there is no overwhelming reason to adopt an external standard. Multiple releases can be treated as multiple projects or as a single project followed by maintenance enhancements.

The total effort expended on the project should be divided into hours used by programmers, managers, and support services. At the conclusion of the project, the totals should be determined from accounting information or another official source. The sum of the effort data collected during the development or maintenance project should be compared with the value obtained from the alternative source to cross-check the accuracy.

The core size measures are the total size of the software product and the total number of components within the product. NASA experience shows that archiving additional details about the origin of the code (e.g., whether it is new, reused, or modified) can lead to useful models.

Use lines of code to represent size.

NASA programs typically measure software size in terms of lines of code. Some authorities recommend other size measures [e.g., function points (see Reference 17)]. However, no other measure is as well understood or as easy to collect as lines of code.

This guidebook also recommends collecting size and origin information for software components and defines a software component as a separately compilable unit of software for the project being measured. Some organizations define components as subprograms or subsystems, which is fine as long as the organization applies that definition consistently and derives useful results. The SEL

captures the basic information for each separately compilable unit of source code and has found that the overhead required to extract the information using an automated tool is trivial. As a result, programmers can be freed from expending additional effort in providing that information.

The final category of project characteristics core measures is software classification. This measure is abstract and of limited value. Consequently, most organizations are advised to spend only limited effort collecting and analyzing classification data. Nevertheless, several NASA organizations have found a high-level classification scheme to be both adequate and useful. These organizations use three broadly defined classes:

1. Business or administrative applications
2. Scientific or engineering applications
3. Systems support

Other organizations may want to record more detailed classification data, such as

- Embedded versus nonembedded
- Real-time versus nonreal-time
- Secure versus nonsecure

4.5.2 Data Definition

The recording of project characteristics data can often be substantially automated to minimize the burden on the development and maintenance organization. Dates and effort, for example, are normally available from management accounting reports; automated tools frequently can be used to report size and component information, and the time and effort needed to indicate software classification is minimal. Table 4-5 summarizes the project characteristics data.

No universally accepted definition exists for the start and stop times of various phases, such as when a project starts or when a design ends. Experience within NASA has led to the use of phase dates as follows:

- *Start of software development*—delivery of system requirements documents
- *End of requirements analysis*—completion of specifications review
- *End of design*—completion of design review
- *End of coding*—completion of code and unit test
- *End of testing*—delivery to acceptance testing
- *End of development*—delivery to operations

Table 4-5. Project Characteristics Data

Data	Descriptions
Dates	
Phase start dates (year, month, and day)	Requirements analysis Design Implementation System test Acceptance test Cleanup Maintenance
End date	Project end
Effort	
Total hours	Project total Management personnel Technical personnel Support personnel (e.g., publications), if applicable
Size	
Project size (lines of code)	Delivered Developed Executable Comments New Extensively modified Slightly modified Reused
Other (count)	Number of components Pages of documentation
Component information (for each component)	
Component size (lines of code)	Total Executable
Component origin	New Extensively modified Slightly modified Reused
Software classification	Business/administrative Scientific/engineering Systems support

The effort data, compiled at the conclusion of the project, are used as part of the high-level summary information for the project. The information represents the total cost of the project broken down among developers, managers, and support services.

Table 4-5 lists several measures for lines of code. Consensus may never be reached on what constitutes a line of code. Therefore, to facilitate various forms of comparison and analysis, this guidebook recommends recording multiple values. The core measures include counts of

- *Total lines delivered*—every logical line, including comments, blanks, executable, and nonexecutable
- *Developed lines*—total lines with a reuse factor
- *Executable statements*—total number of executable statements
- *Comment lines*—total number of lines containing only comments or blanks

The SEL captures source lines of code in four categories:

1. *New*—code in new units
2. *Extensively modified*—code for reused units in which 25 percent or more of the lines were modified
3. *Slightly modified*—code for reused units in which fewer than 25 percent of the lines were modified
4. *Reused verbatim*—code for units that were reused with no changes

For estimation purposes, lines of code are often classified into two categories that combine newly written and extensively modified units as *new* code and slightly modified and verbatim code as *reused* code. Consequently, the SEL relationships (see Reference 9) for estimating developed lines are

FORTRAN developed lines = new lines + 20% of reused lines

Ada developed lines = new lines + 30% of reused lines

(See Sections 2.2.1 and 6.1.2 for more discussion of developed lines of code.)

Specify which software is to be counted.

It is important to be specific about which software is to be included in the size counts. For example, it is usually appropriate to exclude throw-away prototypes, test harnesses, and commercial off-the-shelf (COTS) software from the reported totals.

Component information can provide insight into the overall development characteristics. Although the total amount of information may be extensive, it should be easy to compile at the conclusion of the project and can be almost completely retrieved via automated software tools such as code counters, auditors, or analyzers.

The Project Completion Statistics Form (see Figure A-7 in Appendix A) is an example of a form used for collecting project characteristics at the completion of a project. Figure 4-5 summarizes the life-cycle phases, sources, and frequency for project characteristics data collection.

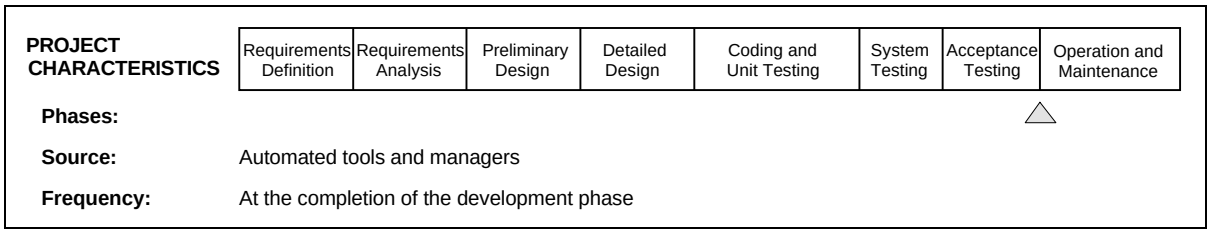


Figure 4-5. Project Characteristics Collection Summary