(3) What type of progress-related information would you recommend be reported to divisional management?

Topic for discussion

20.1 The "Golden Bridge" software development project was scheduled to be completed in about 12 months. Two to six team members were planned to work on the project. Project progress control was based on a monthly report that would refer to each of the 32 activities to be performed and to the components (1) risk item management, (2) timetable, and (3) project's human resources utilization.

The first three monthly progress reports submitted to management did not indicate any deviation from the plan. The fourth progress report presented a substantial overrun in terms of human resources utilization (overtime, etc.) as well as a one-month delay in the expected completion dates for some of the activities.

(1) Can you suggest possible reasons for the relatively late detection of the deviations from the plan?
(2) For each of the above three components, describe the measures that could have prevented deviations and their adverse effects.
(3) Suggest some interventions that management could have introduced to compensate for the project's failures, including the assumptions behind each intervention.

Chapter 21

Software quality metrics

Chapter outline

"You can't control what you can't measure"
Tom DeMarco (1982)

Tom DeMarco's statement has become the motto for software quality experts trying to develop and apply quality metrics in the software industry.

Two alternative but complementary definitions (IEEE, 1990) describe quality metrics as a category of SQA tools:

(1) A quantitative measure of the degree to which an item possesses a given quality attribute.
(2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

That is, the second definition refers to the process that produces quality metrics whereas the first refers to the outcome of the above process.

It is commonly believed that quality metrics should be included in software, as in other industries, among the fundamental tools employed to assist management in three basic areas: control of software development projects and software maintenance, support of decision taking, and initiation of corrective actions. Statistical analysis of metrics data is expected to pinpoint (descriptively and statistically significant) changes initiated as a result of application of new development tools, changed procedures, and other interventions.

The scope of software quality metrics (hereinafter "metrics") has expanded considerably over the past few decades. We will review some of those most pertinent software development and maintenance issues at the heart of this chapter. Metrics as a quality assurance tool has not, unfortunately, been applied at an adequate level in the software industry. Nor have they provided benefits at the anticipated levels. Only a small portion of software development organizations apply software quality metrics systematically, and few of these report successful use of the results of their efforts. Some of the reasons for this situation and prospects for the future are discussed in the last part of the chapter.

Experience with metrics in the field has not threatened its potential significance, attested to by the ISO 9000-3 standard (see ISO/IEC (1997) Sec. 4.20 and ISO (2001) Sec. 8). Software quality metrics are also an important part of the CMM guidelines (see Paulk *et al.*, 1995).

Several books, chapters in books, and numerous journal as well as conference papers have been dedicated to this subject. Some are listed in the bibliography at the end of this chapter. In addition, IEEE offers some software quality metrics criteria within its software engineering standards (IEEE 1988, 1998). A comprehensive discussion of metrics for software reuse, including their economic implications, is presented by Poulin (1997).

After completing this chapter, you will be able to:

- Explain the objectives of software quality metrics.
- List the requirements to be fulfilled by successful software quality metrics.
- Explain how software quality metrics are categorized.
- Compare the KLOC and function point measures for the size of a software system.
- Describe the process of defining a new software quality metrics.
- Explain the reasons for limitations characterizing some software quality metrics.

21.1 Objectives of quality measurement

Software quality and other software engineers have formulated the main objectives for software quality metrics, presented in Frame 21.1.

Frame 21.1 Main objectives of software quality metrics

(1) To facilitate management control as well as planning and execution of the appropriate managerial interventions. Achievement of this objective is based on calculation of metrics regarding:

- Deviations of actual functional (quality) performance from planned performance
- Deviations of actual timetable and budget performance from planned performance.

(2) To identify situations that require or enable development or maintenance process improvement in the form of preventive or corrective actions introduced throughout the organization. Achievement of this objective is based on:

- Accumulation of metrics information regarding the performance of teams, units, etc.

Comparison provides the practical basis for management's application of metrics and for SQA improvement in general. The metrics are used for comparison of performance data with *indicators*, quantitative values such as:

- Defined software quality standards
- Quality targets set for organizations or individuals
- Previous year's quality achievements
- Previous project's quality achievements
- Average quality levels achieved by other teams applying the same development tools in similar development environments

- Average quality achievements of the organization
- Industry practices for meeting quality requirements.

In order for the selected quality metrics to be applicable and successful, both general and operative requirements, as presented in Frame 21.2, must be satisfied.

Frame 21.2 Software quality metrics – requirements

General requirements	Explanation
Relevant	Related to an attribute of substantial importance
Valid	Measures the required attribute
Reliable	Produces similar results when applied under similar conditions
Comprehensive	Applicable to a large variety of implementations and situations
Mutually exclusive	Does not measure attributes measured by other metrics

Operative requirements	Explanation
Easy and simple	The implementation of the metrics data collection is simple and is performed with minimal resources
Does not require independent data collection	Metrics data collection is integrated with other project data collection systems: employee attendance, wages, cost accounting, etc. In addition to its efficiency aspects, this requirement contributes to coordination of all information systems serving the organization
Immune to biased interventions by interested parties	In order to escape the expected results of the analysis of the metrics, it is expected that interested persons will try to change the data and, by doing so, improve their record. Such actions obviously bias the relevant metrics. Immunity (total or at least partial) is achieved mainly by choice of metrics and adequate procedures

21.2 Classification of software quality metrics

Software quality metrics can fall into a number of categories. Here we use a two-level system.

The first classification category distinguishes between life cycle and other phases of the software system:

- Process metrics, related to the software development process (see Section 21.3)
- Product metrics, related to software maintenance (see Section 21.4).

The second classification category refers to the subjects of the measurements:

- Quality
- Timetable
- Effectiveness (of error removal and maintenance services)
- Productivity.

These items are dealt with in the respective sections.

A sizeable number of software quality metrics involve one of the two following measures for system size:

- **KLOC** – this classic metric measures the size of software by thousands of code lines. As the number of code lines required for programming a given task differs substantially with each programming tool, this measure is specific to the programming language or development tool used. Application of metrics that include KLOC is limited to software systems developed using the same programming language or development tool.
- **Function points** – a measure of the development resources (human resources) required to develop a program, based on the functionality specified for the software system (see Appendix 21A).

Customer satisfaction metrics are excluded from our presentation; the reader can find wide coverage of this topic in the marketing literature.

21.3 Process metrics

Software development process metrics can fall into one of the following categories:

- Software process quality metrics
- Software process timetable metrics
- Error removal effectiveness metrics
- Software process productivity metrics.

21.3.1 Software process quality metrics

Software process quality metrics may be classified into two classes:

- Error density metrics
- Error severity metrics.

Another group of indirect metrics that relates to software process quality is the McCabe's cyclomatic complexity metrics (see Section 9.4.4).

A discussion of the above three classes follows.

Error density metrics

This section describes six different types of metrics. Calculation of error density metrics involves two measures: (1) software volume, and (2) errors counted.

Software volume measures. Some density metrics use the number of lines of code while others apply function points. For a comparison of these measures, see Section 21.2.

Errors counted measures. Some relate to the number of errors and others to the weighted number of errors. Weighted measures that ascertain the severity of the errors are considered to provide more accurate evaluation of the error situation. A common method applied to arrive at this measure is classification of the detected errors into severity classes, followed by weighting each class. The weighted error measure is computed by summing up multiples of the number of errors found in each severity class by the adequate relative severity weight. Department of Defense standard MIL-STD-498, presented in Table 8.1, describes a commonly used five-level severity classification system. It should be noted that application of weighted measures can lead to decisions different than those arrived at with simple (unweighted) metrics: weighted measures are assumed to be better indicators of adverse situations.

Example 1. This example demonstrates the calculation of the number of code errors (NCE) and the weighted number of code errors (WCE). A software development department applies two alternative measures, NCE and WCE, to the code errors detected in its software development projects. Three classes of error severity and their relative weights are also defined:

Error severity class	Relative weight
Low severity	1
Medium severity	3
High severity	9

The code error summary for the department's Atlantis project indicated that there were 42 low severity errors, 17 medium severity errors, and 11 high severity errors. Calculation of NCE and WCE gave these results:

	Calculation of NCE	Calculation of WCE	
Error severity class *a*	(number of errors) *b*	Relative weight *c*	Weighted errors $D = b \times c$
Low severity	42	1	42
Medium severity	17	3	51
High severity	11	9	99
Total	70	—	192
NCE	70	—	—
WCE	—	—	192

Table 21.1: Error density metrics

Code	Name	Calculation formula
CED	Code Error Density	$CED = \frac{NCE}{KLOC}$
DED	Development Error Density	$DED = \frac{NDE}{KLOC}$
WCED	Weighted Code Error Density	$WCED = \frac{WCE}{KLOC}$
WDED	Weighted Development Error Density	$WDED = \frac{WDE}{KLOC}$
WCEF	Weighted Code Errors per Function point	$WCEF = \frac{WCE}{NFP}$
WDEF	Weighted Development Errors per Function point	$WDEF = \frac{WDE}{NFP}$

Key:
- NCE = number of code errors detected in the software code by code inspections and testing. Data for this measure are culled from code inspection and testing reports.
- KLOC = thousands of lines of code.
- NDE = total number of development (design and code) errors detected in the software development process. Data for this measure are found in the various design and code reviews and testing reports conducted.
- WCE = weighted code errors detected. The sources of data for this metric are the same as those for NCE.
- WDE = total weighted development (design and code) errors detected in development of the software. The sources of data for this metric are the same as those for NDE.
- NFP = number of function points required for development of the software. Sources for the number of function points are professional surveys of the relevant software.

Table 21.1 (above) displays six error density metrics.

Example 2. This example follows Example 1 and introduces the factor of weighted measures so as to demonstrate the implications of their use. A software development department applies two alternative metrics for calculation of code error density: CED and WCED. The unit determined the following indicators for unacceptable software quality: CED > 2 and WCED > 4. For our calculations we apply the three classes of quality and their relative weights and the code error summary for the Atlantis project mentioned in Example 1. The software system size is 40 KLOC. Calculation of the two metrics resulted in the following:

Measures and metrics	Calculation of CED (Code Error Density)	Calculation of WCED (Weighted Code Error Density)
NCE	70	—
WCE	—	192
KLOC	40	40
CED (NCE/KLOC)	1.75	—
WCED (WCE/KLOC)	—	4.8

The conclusions reached after application of the unweighted versus weighted metrics are different. While the CED *does not* indicate quality *below* the acceptable level, the WCED metric *does* indicate quality *below* the acceptable level (in other words, if the error density is too high, the unit's quality is not acceptable), a result that calls for management intervention.

Error seveity metrics

The metrics belonging to this group are used to detect adverse situations of increasing numbers of severe errors in situations where errors and weighted errors, as measured by error density metrics, are generally decreasing. Two error severity metrics are presented in Table 21.2.

Table 21.2: Error severity metrics

Code	Name	Calculation formula
ASCE	Average Severity of Code Errors	$ASCE = \frac{WCE}{NCE}$
ASDE	Average Severity of Development Errors	$ASDE = \frac{WDE}{NDE}$

21.3.2 Software process timetable metrics

Software process timetable metrics may be based on accounts of success (completion of milestones per schedule) in addition to failure events (non-completion per schedule). An alternative approach calculates the average delay in completion of milestones. The metrics presented here are based on the two approaches illustrated in Table 21.3.

The TTO and ADMC metrics are based on data for all relevant milestones scheduled in the project plan. In other words, only milestones that were designated for completion in the project plan stage are considered in the metrics' computation. Therefore, these metrics can be applied throughout development and need not wait for the project's completion.

21.3.3 *Error removal effectiveness* metrics

Software developers can measure the effectiveness of error removal by the software quality assurance system after a period of regular operation (usually 6 or 12 months) of the system. The metrics combine the error records of the development stage with the failures records compiled during the first year (or any defined period) of regular operation. Two error removal effectiveness metrics are presented in Table 21.4.

Table 21.3: Software process timetable metrics

Code	Name	Calculation formula
TTO	Time Table Observance	$TTO = \frac{MSOT}{MS}$
ADMC	Average Delay of Milestone Completion	$ADMC = \frac{TCDAM}{MS}$

Key:
- MSOT = milestones completed on time.
- MS = total number of milestones.
- TCDAM = total Completion Delays (days, weeks, etc.) for All Milestones. To calculate this measure, delays reported for all relevant milestones are summed up. Milestones completed on time or before schedule are considered "O" delays. Some professionals refer to completion of milestones before schedule as "minus" delays. These are considered to balance the effect of accounted-for delays (we might call the latter "plus" delays). In these cases, the value of the ADMC may be lower than the value obtained according to the metric originally suggested.

Table 21.4: Error removal effectiveness metrics

Code	Name	Calculation formula
DERE	Development Errors Removal Effectiveness	$DERE = \frac{NDE}{NDE + NYF}$
DWERE	Development Weighted Errors Removal Effectiveness	$DWERE = \frac{WDE}{WDE + WYF}$

Key:
- NYF = number of software failures detected during a year of maintenance service.
- WYF = weighted number of software failures detected during a year of maintenance service.

21.3.4 Software process productivity metrics

This group of metrics includes "direct" metrics that deal with a project's human resources productivity as well as "indirect" metrics that focus on the extent of software reuse. Software reuse substantially affects productivity and effectiveness.

An additional term – "benchmarking software development productivity" – has recently entered the list of metrics used to measure software process productivity (see Maxwell, 2001; Symons, 2001).

Four process productivity metrics, direct and indirect, are presented in Table 21.5.

21.4 Product metrics

Product metrics refer to the software system's operational phase – years of regular use of the software system by customers, whether "internal" or "external" customers, who either purchased the software system or con-

Table 21.5: Process productivity metrics

Code	Name	Calculation formula
DevP	Development Productivity	$DevP = \frac{DevH}{KLOC}$
FDevP	Function point Development Productivity	$FDevP = \frac{DevH}{NFP}$
CRe	Code Reuse	$CRe = \frac{ReKLOC}{KLOC}$
DocRe	Documentation Reuse	$DocRe = \frac{ReDoc}{NDoc}$

Key:
- DevH = total working hours invested in the development of the software system.
- ReKLOC = number of thousands of reused lines of code.
- ReDoc = number of reused pages of documentation.
- NDoc = number of pages of documentation.

tracted for its development. In most cases, the software developer is required to provide customer service during the software's operational phase. Customer services are of two main types:

- **Help desk services (HD)** – software support by instructing customers regarding the method of application of the software and solution of customer implementation problems. Demand for these services depends to a great extent on the quality of the user interface (its "user friendliness") as well as the quality of the user manual and integrated help menus.
- **Corrective maintenance services** – correction of software failures identified by customers/users or detected by the customer service team prior to their discovery by customers. The number of software failures and their density are directly related to software development quality. For completeness of information and better control of failure correction, it is recommended that all software failures detected by the customer service team be recorded as corrective maintenance calls.

Commonly, all customer services – namely, HD and corrective maintenance services – are provided to customers/users by a software support center (the "customer service center", among the many titles given to this service). It is expected that very few customer calls will be related to identified failures. In other words, most of the software support center's customer calls will be "non-failure" calls. For those calls that deal with an identified failure and for cases where the maintenance team has detected a failure, a failure report is expected.

HD metrics are based on all customer calls while corrective maintenance metrics are based on failure reports. Product metrics generally rely on performance records compiled during one year (or any other specified period of time). This policy enables comparisons of successive years in addition to comparisons between different units and software systems.

The array of software product metrics presented here is classified as follows:

- HD quality metrics
- HD productivity and effectiveness metrics
- Corrective maintenance quality metrics
- Corrective maintenance productivity and effectiveness metrics.

It should be remembered that software maintenance activities include:

- Corrective maintenance – correction of software failures detected during regular operation of the software.
- Adaptive maintenance – adaptation of existing software to new customers or new requirements.
- Functional improvement maintenance – addition of new functions to the existing software, improvement of reliability, etc.

In the metrics presented here we limit our selection to those that deal with corrective maintenance. For other components of software maintenance, the metrics suggested for the software development process (*process metrics*) can be used as is or with minor adaptations.

21.4.1 HD quality metrics

The types of HD quality metrics discussed here deal with:

- HD calls density metrics – the extent of customer requests for HD services as measured by the number of calls.
- Metrics of the severity of the HD issues raised.
- HD success metrics – the level of success in responding to these calls. A success is achieved by completing the required service within the time determined in the service contract.

HD calls density metrics

This section describes six different types of metrics. Some relate to the number of the errors and others to a weighted number of errors. As for size/volume measures of the software, some use number of lines of code while others apply function points. The sources of data for these and the other metrics in this group are HD reports. Three HD calls density metrics for HD performance are presented in Table 21.6.

Severity of HD calls metrics

The metrics belonging to this group of measures aim at detecting one type of adverse situation: increasingly severe HD calls. The computed results may contribute to improvements in all or parts of the user interface (its "user friendliness") as well as the user manual and integrated help menus. We have

Table 21.6: HD calls density metrics

Code	Name	Calculation formula
HDD	HD calls Density	$HDD = \frac{NHYC}{KLMC}$
WHDD	Weighted HD calls Density	$WHDD = \frac{WHYC}{KLMC}$
WHDF	Weighted HD calls per Function point	$WHDF = \frac{WHYC}{NMFP}$

Key:
- NHYC = number of HD calls during a year of service.
- KLMC = thousands of lines of maintained software code.
- WHYC = weighted HD calls received during one year of service.
- NMFP = number of function points to be maintained.

selected one metric from this group for demonstration of how the entire category is employed. This metric, the **Average Severity of HD Calls** (**ASHC**), refers to failures detected during a period of one year (or any portion thereof, as appropriate):

$$ASHC = \frac{WHYC}{NHYC}$$

where WHYC and NHYC are defind as in Table 21.6.

Success of the HD services

The most common metric for the success of HD services is the capacity to solve problems raised by customer calls within the time determined in the service contract (*availability*). Thus, the metric for success of HD services compares the actual with the designated time for provision of these services.

For example, the availability of help desk (HD) services for an inventory management software package is defined as follows:

- The HD service undertakes to solve any HD call within one hour.
- The probability that HD call solution time exceeds one hour will not exceed 2%.
- The probability that HD call solution time exceeds four working hours will not exceed 0.5%.

One metric of this group is suggested here, **HD Service Success** (**HDS**)**:**

$$HDS = \frac{NHYOT}{NHYC}$$

where NHYOT = number of HD calls per year completed on time during one year of service.

21.4.2 HD productivity and effectiveness metrics

Productivity metrics relate to the total of resources invested during a specified period, while effectiveness metrics relate to the resources invested in responding to a HD customer call.

HD productivity metrics

HD productivity metrics makes use of the easy-to-apply KLMC measure of maintained software system's size (see Table 21.6) or according to function-point evaluation of the software system. Two HD productivity metrics are presented in Table 21.7.

HD effectiveness metrics

The metrics in this group refer to the resources invested in responding to customers' HD calls. One prevalent metric is presented here, **HD Effectiveness (HDE):**

$$\text{HDE} = \frac{\text{HDYH}}{\text{NHYC}}$$

where HDYH and NHYC are as defined in Tables 21.7 and 21.6 respectively.

21.4.3 Corrective maintenance quality metrics

Software corrective maintenance metrics deal with several aspects of the quality of maintenance services. A distinction is needed between software system failures treated by the maintenance teams and failures of the maintenance service that refer to cases where the maintenance failed to provide a repair that meets the designated standards or contract requirements. Thus, software maintenance metrics are classified as follows:

- **Software system failures density metrics** – deal with the extent of demand for corrective maintenance, based on the records of failures identified during regular operation of the software system.
- **Software system failures severity metrics** – deal with the severity of software system failures attended to by the corrective maintenance team.

Table 21.7: HD productivity metrics

Code	Name	Calculation formula
HDP	HD Productivity	$\text{HDP} = \frac{\text{HDYH}}{\text{KLMC}}$
FHDP	Function point HD Productivity	$\text{FHDP} = \frac{\text{HDYH}}{\text{NMFP}}$

Key:
- HDYH = total yearly working hours invested in HD servicing of the software system.
- KLMC and NMFP are as defined in Table 21.6.

- **Failures of maintenance services metrics** – deal with cases where maintenance services were unable to complete the failure correction on time or that the correction performed failed.
- **Software system availability metrics** – deal with the extent of disturbances caused to the customer as realized by periods of time where the services of the software system are unavailable or only partly available.

Software system failures density metrics

The software system failures density metrics presented here relate to the number and/or weighted number of failures. The size of the maintenance tasks is measured by the total number of code lines of the maintained software as well as by the function point evaluation. The sources of data for these metrics are software maintenance reports. Three software system failures density metrics are presented in Table 21.8.

Software system failures severity metrics

Metrics of this group detect adverse situations of increasingly severe failures in the maintained software. Results may trigger retesting of all or parts of the software system. The events measured relate either to the disturbances and damages caused to the customer (representing the customer's point of view) or to the resources required to resolve the failure (representing the interests of the maintenance team). The metric presented here can be used for both purposes, that is, to apply weights that refer to the severity of the disturbances and damages experienced by the customer, or to the extent of resources required by the maintainer. This metric, the **Average Severity of Software System Failures (ASSSF)**, refers to software failures detected during a period of one year (or alternatively a half or a quarter of a year, as appropriate):

$$\text{ASSSF} = \frac{\text{WYF}}{\text{NYF}}$$

Table 21.8: Software system failures density metrics

Code	Name	Calculation formula
SSFD	Software System Failure Density	$\text{SSFD} = \frac{\text{NYF}}{\text{KLMC}}$
WSSFD	Weighted Software System Failure Density	$\text{WSSFD} = \frac{\text{WYF}}{\text{KLMC}}$
WSSFF	Weighted Software System Failures per Function point	$\text{WSSFF} = \frac{\text{WYF}}{\text{NMFP}}$

Key:
- NYF = number of software failures detected during a year of maintenance service.
- WYF = weighted number of yearly software failures detected during a year of maintenance service.
- KLMC = thousands of lines of maintained software code.
- NMFP = number of function points designated for the maintained software.

Failures of maintenance services metrics

As mentioned above, maintenance services can fail either because they were unable to complete the failure correction on time or when the correction performed failed and a repeated correction is required. The metrics presented here relate to the second type of maintenance failure.

A customer call related to a software failure problem that was supposed to be solved after a previous call is commonly treated as a maintenance service failure. For practical purposes, many organizations limit the time frame for the repeat calls to three months, although the period can vary by type of failure or some other organizational criterion. The metric, **Maintenance Repeated repair Failure** (**MRepF**), is defined as follows:

$$\text{MRepF} = \frac{\text{RepYF}}{\text{NYF}}$$

where RepYF is the number of repeated software failure calls (service failures).

Software system availability metrics

User metrics distinguish between:

- Full availability – where all software system functions perform properly
- Vital availability – where no vital functions fail (but non-vital functions may fail)
- Total unavailability – where all software system functions fail.

The source for all availability metrics is user failure records. The latter specify the extent of damage (non-vital failure, vital failure and total system failure) as well as duration (hours) for each failure. Three software system availability metrics are presented in Table 21.9.

21.4.4 Software corrective maintenance productivity and effectiveness metrics

While corrective maintenance productivity relates to the total of human resources invested in maintaining a given software system, corrective maintenance effectiveness relates to the resources invested in correction of a single failure. In other words, a software maintenance system displaying higher productivity will require fewer resources for its maintenance task, while a more effective software maintenance system will require fewer resources, on average, for correcting one failure. Three software corrective maintenance productivity and effectiveness metrics are presented in Table 21.10.

Table 21.9: Software system availability metrics

Code	Name	Calculation formula
FA	Full Availability	$FA = \frac{NYSerH - NYFH}{NYSerH}$
VitA	Vital Availability	$VitA = \frac{NYSerH - NYVitFH}{NYSerH}$
TUA	Total Unavailability	$TUA = \frac{NYTFH}{NYSerH}$

Key:
- NYSerH = number of hours software system is in service during one year. For an office software system that is operating 50 hours per week for 52 weeks per year, NYSerH = 2600 (50×52). For a real-time software application that serves users 24 hours a day, NYSerH = 8760 (365×24).
- NYFH = number of hours where at least one function is unavailable (failed) during one year, including total failure of the software system.
- NYVitFH = number of hours when at least one vital function is unavailable (failed) during one year, including total failure of the software system.
- NYTFH = number of hours of total failure (all system functions failed) during one year.
- $NYFH \geq NYVitFH \geq NYTFH$.
- $1 - TUA \geq VitA \geq FA$.

Table 21.10: Software corrective maintenance productivity and effectiveness metrics

Code	Name	Calculation formula
CMaiP	Corrective Maintenance Productivity	$CMaiP = \frac{CMaiYH}{KLMC}$
FCMP	Function point Corrective Maintenance Productivity	$FCMP = \frac{CMaiYH}{NMFP}$
CMaiE	Corrective Maintenance Effectiveness	$CMaiE = \frac{CMaiYH}{NYF}$

Key:
- CMaiYH = total yearly working hours invested in the corrective maintenance of the software system.
- KLMC = thousands of lines of maintained software code.
- NMFP = number of function points designated for the maintained software.
- NYF = number of software failures detected during a year of maintenance service.

21.5 Implementation of software quality metrics

The application of software quality metrics in an organization requires:

- Definition of software quality metrics – relevant and adequate for teams, departments, etc.
- Regular application by unit, etc.
- Statistical analysis of collected metrics data.

- Subsequent actions:
  - Changes in the organization and methods of software development and maintenance units and/or any other body that collected the metrics data
  - Change in metrics and metrics data collection
  - Application of data and data analysis to planning corrective actions for all the relevant units.

The technical aspects of Nokia's experience in applying metrics are summarized by Kilpi (2001). Unfortunately, this paper does not explore Nokia's application of metrics to decision making in such areas as productivity, effectiveness and so forth.

21.5.1 Definition of new software quality metrics

The definition of metrics involves a four–stage process:

(1) Definition of attributes to be measured: software quality, development team productivity, etc.
(2) Definition of the metrics that measure the required attributes and confirmation of its adequacy in complying with the requirements listed in Frame 21.2.
(3) Determination of comparative target values based on standards, previous year's performance, etc. These values serve as *indicators* of whether the unit measured (a team or an individual or a portion of the software) complies with the characteristics demanded of a given attribute.
(4) Determination of metrics application processes:
  - Reporting method, including reporting process and frequency of reporting
  - Metrics data collection method.

The new metrics (updates, changes and revised applications) will be constructed following analysis of the metrics data as well as developments in the organization and its environment. The software quality metrics definition process is described in Figure 21.1.

21.5.2 Application of the metrics – managerial aspects

The process of applying a metric or a set of metrics is similar to the implementation of new procedures or methodologies. It involves:

- Assigning responsibility for reporting and metrics data collection.
- Instruction of the team regarding the new metrics.
- Follow-up includes:
  - Support for solving application problems and provision of supplementary information when needed.
  - Control of metrics reporting for completeness and accuracy.
- Updates and changes of metrics definitions together with reporting and data collection methods according to past performance.

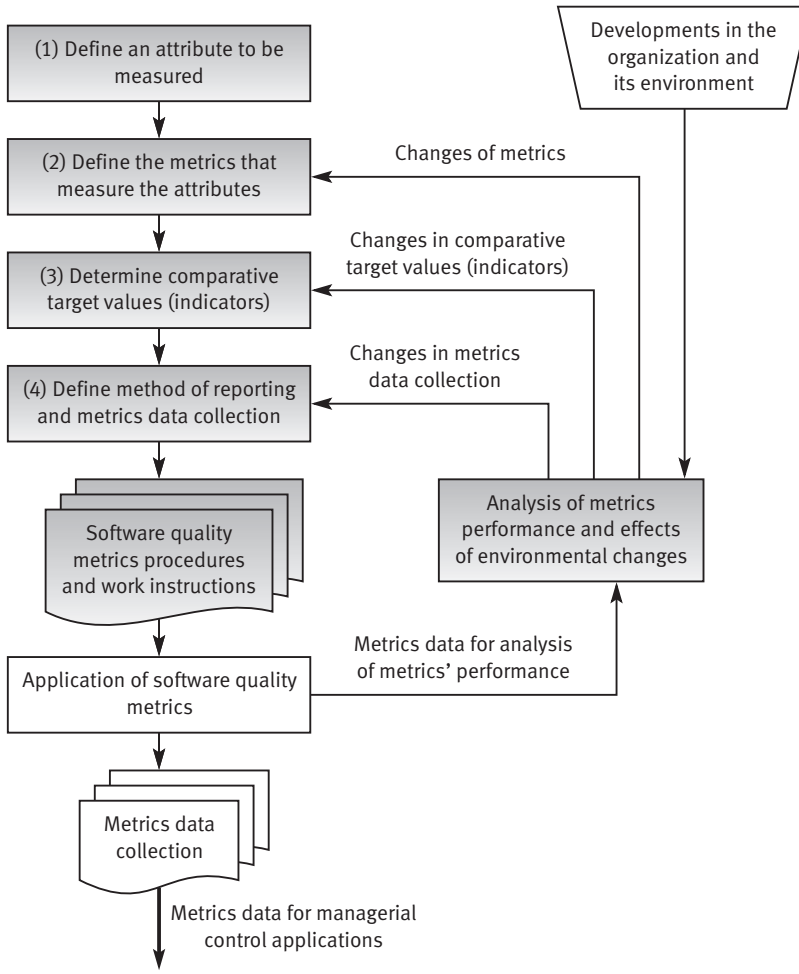

Figure 21.1: The process of defining software quality metrics

Implementation tip

Not a few of the currently applied software quality metrics procedures and work instructions omit the third stage of the metrics definition process: setting target values (indicators). In other words, no target values for the metrics are to be found in the procedure or its appendices, the accompanying work instructions or any other document. In most cases this situation reflects a lack of serious commitment to metrics use in managerial control, the major reason for applying metrics in the first place. When application of metrics goes beyond lip service, target values should be set even if updates of these values are expected soon after their first application.

An interesting application of software quality metrics for comparison of national software industries is presented in the following example.

Example – Comparison of US and Japanese software industries
Cusumano (1991) makes use of three metrics in a comparison of the US and Japanese software industries:

- Mean productivity
- Failure density (based on measurements during the first 12 months after system delivery)
- Code reuse.

These metrics are presented in Table 21.11, and Cusumano's results are presented in Table 21.12.

21.5.3 Statistical analysis of metrics data

Analysis of metrics data provides opportunities for comparing a series of project metrics. These certainly include comparison of metrics results against

Table 21.11: US and Japanese software industries – metrics

Name	Calculation formula
Mean productivity (similar to DevP, Table 21.5)	$\frac{\text{KNLOC}}{\text{WorkY}}$
Failure density (similar to SSFD, Table 21.8)	$\frac{\text{NYF}}{\text{KNLOC}}$
Code reuse (similar to CRe, Table 21.5)	$\frac{\text{ReKNLOC}}{\text{KNLOC}}$

Key:
- KNLOC = thousands of non-comment lines of code.
- WorkY = human work-years invested in the software development.
- ReKNLOC = thousands of reused non-comment lines of code.

Source: Based on Cusumano (1991)

Table 21.12: US and Japanese software industries – comparison of three software quality metrics

Software quality metrics	United States	Japan
Mean productivity	7290	12447
Failure density	4.44	1.96
Code reuse	9.71%	18.25%
N – (number of companies)	20	11

Source: Cusumano (1991)

predefined indicators, as well as comparisons with former projects or team performance at different periods of time, and so on. Another important comparison relates to the effectiveness with which the metrics themselves fulfill their respective aims. The following questions are just a sample of those that can be asked with respect to the metrics portion of the SQA process.

- Are there significant differences between the HD teams' quality of service?
- Do the metrics results support the assumption that application of the new version of a development tool contributes significantly to software quality?
- Do the metrics results support the assumption that reorganization has contributed significantly to a team's productivity?

For the metrics data to be a valuable part of the SQA process, statistical analysis is required of the metrics' results. Statistical tools provide us with two levels of support, based on the type of statistics used:

- Descriptive statistics
- Analytical statistics.

Descriptive statistics

Descriptive statistics, such as the mean, median and mode as well as use of graphic presentations such as histograms, cumulative distribution graphs, pie charts and control charts (showing also the indicator values) to illustrate the information to which they relate, enable us to quickly identify trends in the metrics values. Exaggerated trends, identified by the appearance of acute deviations from target values, may indicate the need for corrective actions or, alternatively, to continue or expand application of a successful innovation. Because these statistics are so basic to quality assurance, the majority of the popular software statistical packages (*statistical tools*) available generally provide a rather complete menu of graphic presentations, including the ones mentioned above. However, it should be stressed that descriptive statistics, however sophisticated, are not intended to analyze the statistical significance – how much the observed trends are the result of chance rather than substantive processes – of the events.

Analytical statistics

Description is not always enough. To determine whether the observed changes in the metrics are meaningful, whatever the direction, the observed trends must be assessed for their significance. This is the role of analytic statistics (e.g., regression tests, analysis of variance, or more basic tests such as the T-test and Chi-square test). However, the application of statistical analysis to software system performance metrics is relatively difficult, which is one outcome of the complexity of software systems development and maintenance, discussed in the introduction to this book. For further study of the subject the reader is referred to the literature on statistical analysis.

What should be noted here is the fundamental difference between statistical analysis of production line metrics by application of classical SPC (statistical process control) methods and of software development and maintenance. While production line activities are repetitive, the development activities, by definition, vary from one project to the next; they are never repetitive in the SPC sense. Although the statistical methods applied may be similar, the subject matter differs, as may the implications of the statistical results.

21.5.4 Taking action in response to metrics analysis results

The actions taken in response to metrics analysis can be classified as direct actions if initiated by the project or team management or indirect actions if initiated by the Corrective Action Board. The CAB indirect actions are a result of analysis of metrics data accumulated from a variety of projects and/or development departments.

Examples of the direct changes initiated by management include reorganization, changes in software development and maintenance methods, and revision of the metrics computed. For a comprehensive discussion of indirect actions as initiated by the Corrective Action Board, see Chapter17.

21.6 Limitations of software metrics

Application of quality metrics is strewn with obstacles. These can be grouped as follows:

- Budget constraints in allocating the necessary resources (manpower, funds, etc.) for development of a quality metrics system and its regular application.
- Human factors, especially opposition of employees to evaluation of their activities.
- Uncertainty regarding the data's validity, rooted in partial and biased reporting.

These difficulties are fairly universal and, as such, apply to software quality metrics too. However, additional obstacles may appear that are uniquely related to the software industry. These are discussed in this section. (For an up-to-date discussion in the literature see Rifkin (2001), McGarry (2001), Maxwell (2001) and Symons (2001), who discuss the difficulties in applying software quality metrics, especially for decision making in the context of software development.)

The unique barriers associated with the application of software quality metrics are rooted in the attributes measured. As a result, most commonly used metrics suffer from low validity and limited comprehensiveness. Examples of software parameters metrics that exhibit severe weaknesses are:

- Parameters used in development process metrics: KLOC, NDE, NCE
- Parameters used in product (maintenance) metrics: KLMC, NHYC, NYF.

The main factors affecting development proces parameters, especially their magnitude, are:

(1) Programming style: strongly affects software volume, where "wasteful" coding may double the volume of produced code (KLOC).
(2) Volume of documentation comments included in the code: affects volume of the code. The volume of comments is usually determined by the programming style (KLOC).
(3) Software complexity: complex modules require much more development time (per line of code) in comparison to simple modules. Complex modules also suffer from more defects than simple modules of similar size (KLOC, NCE).
(4) Percentage of reused code: the higher the percentage of reused code incorporated into the software developed, the greater the volume of code that can be produced per day as well as the lower the number of defects detected in reviews, testing and regular use (NDE, NCE).
(5) Professionalism and thoroughness of design review and software testing teams: affects the number of defects detected (NCE).
(6) Reporting style of the review and testing results: some teams produce concise reports that present the findings in a small number of items (small NCE), while others produce comprehensive reports, showing the same findings for a large number of items (large NDE and NCE).

The main factors affecting the magnitude of the product (maintenance) parameters are:

(1) Quality of installed software and its documentation (determined by the quality of the development team as well as the review and testing teams): the lower the initial quality of the software, the greater the anticipated software failures identified and subsequent maintenance efforts (NYF, NHYC).
(2) Programming style and volume of documentation comments included in the code: as in the development stage, both strongly affect the volume of the software to be maintained, where wasteful coding and documentation may double the volume of code to be maintained (KLMC).
(3) Software complexity: complex modules require investment of many more maintenance resources per line of code than do simple modules, and suffer from more defects left undetected during the development stage (NYF).
(4) Percentage of reused code: the higher the percentage of reused code, the lower the number of defects detected in regular use as well as the fewer required corrective maintenance and HD efforts (NYF).
(5) Number of installations, size of the user population and level of applications in use: affect the number of HD calls as well as the number of defects detected by users during regular use (NHYC, NYF).

By affecting the magnitude of the parameters, these factors distort the software product quality metrics on which they are based. The inevitable result is that a major portion of the metrics we have discussed do not reflect the real productivity and quality achievements of development or maintenance teams in what may be the majority of situations. In other words, many of the metrics reviewed here, like the metrics applied in other industries, most of which are characterized by relative simplicity of application, are characterized by low validity and limited comprehensiveness.

Substantial research efforts are needed in order to develop metrics appropriate to the software industry. The function point method is an example of a successful methodological development aimed at replacing the problematic KLOC metric. For a comprehensive discussion of the function point method, see Appendix 21A.

Summary

(1) Explain the objectives of software quality metrics.

Software quality metrics are implemented:

- To support control of software development projects and software maintenance contracts. Their aim is to provide management with information regarding:
  – Compliance with functional (quality) performance requirements
  – Compliance with project timetable and budget.
- To deliver the metrics accumulated and analyzed by the CAB. Use of these metrics data is aimed at enabling preventive and corrective actions throughout the organization.

(2) List the requirements for successful software quality metrics.

Applicability of quality metrics is determined by the degree to which the following general and operative requirements are fulfilled:

General requirements

- Relevant – measures an attribute of considerable importance
- Valid – measures the required attribute
- Reliable – produces similar results when applied in similar conditions
- Comprehensive – applicable to a large variety of situations
- Mutually exclusive – does not measure attributes already measured by other metrics.

Operative requirements

- Easy and simple – data collection is implemented with minimal resources
- Does not require independent data collection – metrics data collection is based on currently employed data collection systems, e.g. employee attendance records, cost accounting methods
- Immune to biased interventions by interested parties (team members and others).

(3) Explain how software quality metrics are categorized.

A two-level system of categories is used here. The first level distinguishes between two categories:

- Process metrics, related to the software development process
- Product metrics, related to software maintenance.

Each first-level category is broken down into one of three sub-categories :

- Software process quality metrics
- Software process timetable metrics
- Software process productivity metrics.

The software product metrics are classified into four HD and corrective maintenance sub-categories:

- HD quality metrics
- HD productivity and effectiveness metrics
- Software corrective maintenance quality metrics
- Software corrective maintenance productivity and effectiveness metrics.

(4) Compare the KLOC and function points measures for the size of a software system.

A significant number of the metrics presented here use one of two measures for software system size, which are compared according to the following criteria:

- **Dependency on the development tool, programming language, or programmer style.** KLOC depends heavily on the development tool's characteristics and on the programmer's style. Alternatively, although the function point method does not depend on either of these factors, it does depend to some extent on the function point instruction manual used. It should also be noted that most successful implementations and research supporting the results of the function point method are related to data processing systems, whereas only limited experience has been gained in other areas of software systems.
- **Professional experience required for implementation.** Relatively little experience is required for counting KLOC, while relatively great experience is needed to evaluate function points.
- **Amount of professional work required.** Relatively little for KLOC; far more work for evaluation of function points.
- **Subjective factors.** Estimation of KLOC requires little subjective judgment, whereas the opposite is true for function points because subjective evaluations are required for determining the weight and relative complexity factors for each software system component, as required by the function point method.
- **Pre-project estimates.** Pre-project estimates are unavailable for KLOC but available for function points as the latter can be based on requirement specification documents.

(5) Describe the process of defining a new software quality metric.

The definition of metrics involves a four-stage process:

(a) Definition of attributes to be measured: software quality, development team productivity, etc.

(b) Formulation of the metric and assessment of its adequacy with respect to metrics requirements.
(c) Determination of comparative target values (indicator) to enable the evaluation of the performance measured by the metrics.
(d) Determination of the metrics application process:
  – Reporting method
  – Metrics data collection method.

(6) Explain the reasons for limitation characterizing some software quality metrics.

A unique difficulty faced by use of software quality metrics is rooted in the measures (parameters) that comprise many software quality metrics. As a result, a large proportion of software metrics, including most of the commonly used metrics, suffer from low validity and limited comprehensiveness. Examples of metrics that exhibit severe weaknesses are:

- Software development metrics that are based on measures such as KLOC, NDE and NCE
- Product (maintenance) metrics that are based on measures such as KLMC, NHYC and NYF.

For example, the KLOC measure is affected by the programming style, the volume of documentation comments included in the code and the complexity of the software. NYF is affected by the quality of the installed software and its documentation as well as the percentage of reused code, among the other factors affecting maintenance.

Selected bibliography

1. Albrecht, A. J. (1979) "Measuring Application Development Productivity", in *Process Joint SHARE/GUIDE/IBM Application Development Smposium*, October 1979, 34–43.
2. Albrecht, A. J. and Gaffney, J. E. (1983) "Software Functions, Source Lines of Code and Development Efforts Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering*, SE–9, Nov. 1983, 639–648.
3. Caldiera, G., Antoniol, G., Fiutem, R. and Lokan, C. (1998) "Definition and experimental evaluation of function points for object-oriented systems", in IEEE Computer Society, *Proceedings of the Fifth International Software Metrics Symposium, Metrics 1998*, 20–21 November 1998, Bethesda, MD, IEEE Computer Society Press, Los Alamitos, CA, pp. 167–178.
4. Cusumano, M. A. (1991) *Japan's Software Factories – A Challenge to U.S. Management*, Oxford University Press, New York.
5. Davis, D. B. (1992) "Develop applications on time, every time", *Datamation*, 1 Nov, 85–89.
6. DeMarco, T. (1982) *Controlling Software Projects: Management, Measurement and Estimation*, Yourdon Press, New York.
7. Fenton, N. E. (1995) *Software Metrics – A Rigorous Approach*, International Thomson Press, London.
8. Fenton, N. E. and Pfleger, S. L. (1998) *Software Metrics – A Rigorous and Practical Approach*, 2nd edn, International Thomson Press, London.

9. Grable, R., Jernigan, J., Pogue, C. and Davis, D. (1999) "Metrics for small projects: experience at the SED", *IEEE Software*, 16(2), 21–29.
10. Gramus, D. and Herron, D. (1996) *Measuring the Software Process – A Practical Guide to Functional Measurements*, Yourdon Press, Prentice Hall, Upper Saddle River, NJ.
11. Henderson-Sellers, B. (1996) *Object-Oriented Metrics – Measures of Complexity*, Prentice Hall, Upper Saddle River, NJ.
12. IEEE (1988) "IEEE Std 982.1-1988 – IEEE Standard Dictionary of Measures to Produce Reliable Software", in *IEEE Software Engineering Standards Collection*, The Institute of Electrical and Electronics Engineers, New York.
13. IEEE (1990) "IEEE Std 610.12-1990 – IEEE Standard Glossary of Software Engineering Terminology", in *IEEE Software Engineering Standards Collection*, The Institute of Electrical and Electronics Engineers, New York.
14. IEEE (1998) "IEEE Std 14143.1-2000 – Implementation Note for IEEE Adoption of ISO/IEC 14143:1998 Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concept", in *IEEE Software Engineering Standards Collection*, The Institute of Electrical and Electronics Engineers, New York.
15. IEEE (2000) "IEEE Std 1061-1998 – Standard for Software Quality Metrics Methodology", in *IEEE Software Engineering Standards Collection*, The Institute of Electrical and Electronics Engineers, New York.
16. IEEE Computer Society (1994) *Proceedings of the 2nd International Software Metrics Symposium*, IEEE Computer Society Press, Los Angeles, CA.
17. IEEE Computer Society (1998) *Proceedings of the Fifth International Software Metrics Symposium, Metrics 1998*, Bethesda, MD, IEEE Computer Society Press, Los Alamitos, CA.
18. ISO (1997) *ISO 9000-3:1997(E), Quality Management and Quality Assurance Standards – Part 3: Guidelines for the Application of ISO 9001:1994 to the Development, Supply, Installation and Maintenance of Computer Software*, 2nd edn, International Organization for Standardization (ISO), Geneva.
19. ISO/IEC (2001) "ISO 9000-3:2001 Software and System Engineering – Guidelines for the Application of ISO 9001:2000 to Software, Final draft", International Organization for Standardization (ISO), Geneva, unpublished draft, December 2001.
20. Jeffery, D. R., Low, G. C. and Barnes, M. (1993) "A comparison of function point counting techniques", *Transactions on Software Engineering*, 19(5), 529–532.
21. Jones, C. (1996) *Applied Software Measurement – Assuring Productivity and Quality*, 2nd edn, McGraw-Hill, New York, Sec. 3.
22. Jones, C. (1998) *Estimating Software Costs*, McGraw-Hill, New York.
23. Kautz, K. (1999) "Making sense of measurement for small organizations", *IEEE Software* 16(2), 14–20.
24. Kan, S. H. (1995) *Metrics and Models in Software Quality Engineering*, Addison Wesley, Reading, MA.
25. Kilpi, T. (2001) "Implementing a software metrics program at Nokia", *IEEE Software*, 18(6), 72–77.
26. Lowe, G. C. and Jeffery, D. C. (1990) "Function Points in the Estimation and Evaluation of the Software Process", *IEEE Transactions on Software Engineering*, 16(1), 64–71.

27. Maxwell, K. D. (2001) "Collecting data for comparability: benchmarking software development productivity", *IEEE Software*, 18(5), 22–25.
28. McGarry, J. (2001) "When it comes to measuring software, every project is unique", *IEEE Software*, 18(5), 19, 21.
29. Mendes, E., Mosley, N. and Counsell, S. (2001) "Web metrics – estimating design and authoring efforts", *IEEE Multimedia*, 8(1), 50–57.
30. Moller, K. H. and Paulish, D. L. (1993) *Software Metrics – A Practitioner's Guide to Improved Product Development*, IEEE Computer Society Press and Chapman & Hall, London.
31. Oman, P. and Pfleeger S. L. (eds) (1997) *Applied Software Metrics*, IEEE Computer Society Press, Los Alamitos, CA.
32. Paulk, M. C., Weber, C. V., Curtis, B. and Chrissis, M. B. (1995) *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Reading, MA.
33. Poulin, J. S. (1997) *Measuring Software Reuse – Principles, Practices and Economic Models*, Addison-Wesley, Reading, MA.
34. Pressman, R. S. (2000) *Software Engineering – A Practitioner's Approach*, European adaptation by D. Ince, 5th edn, McGraw-Hill, International, London, Chs 4, 19 and 24.
35. Rifkin, S. (2001) "What makes measuring software so hard?", *IEEE Software*, 18(3), 41–45.
36 Schulmeyer, G. G. (1999) "Software quality assurance metrics", in G. G. Schulmeyer and J. I. McManus (eds), *Handbook of Software Quality Assurance*, 3rd edn, Prentice Hall, Upper Saddle River, NJ, 403–443.
37 Sedigh-Ali, S., Ghafoor, A. and Paul, R. (2001) "Software engineering metrics for COTS-based systems", *IEEE Computer*, 34(5), 44–50.
38 Shoval, P. and Feldman, O. (1997) "A combination of the Mk-II function points software estimation method with the ADISSA methodology for systems analysis and design", *Information and Software Technology*, 39, 855–865.
39 Simmons, P. (1994) "Measurement and the evaluation of I.T. investment", in IEEE Computer Society, *Proceedings of the 2nd International Software Metrics Symposium*, 24–26 October, 1994, IEEE Computer Society Press, Los Angeles, CA.
40 Symons, C. R. (1991) *Software Sizing and Estimating – Mk II FPA (Function Point Analysis)*, John Wiley, Chichester, UK.
41 Symons, C. (2001) "Software benchmarking: serious management tool or a joke", *IEEE Software*, 18(5), 18, 20.

Review questions

21.1 Section 21.3.1 describes the following three code-error density metrics: CED, WCED and WCEF.

(1) Compare CED and WCED including references to their managerial application characteristics as well as to their validity.
(2) Compare WCED and WCEF including references to their managerial implementation characteristics as well as to their validity.
(3) Which of the above metrics would you prefer? List your arguments.

21.2 Section 21.3.3 describes the following two development productivity metrics: DevP and FDevP.

(1) Compare DevP and FDevP including references to their managerial implementation characteristics as well as to their validity.
(2) Which of the above metrics would you prefer? List your arguments.

21.3 Section 21.4 lists metrics for HD and corrective maintenance services.

(1) Explain the difference between these services.
(2) Justify the separate metrics categories and the actions based on their differences.

21.4 Section 21.4.3 describes two maintenance failure density metrics – WSSFD and WSSFF.

(1) Evaluate each of the above metrics as to the degree they fulfill the requirements for software quality metrics as listed in Frame 21.2.
(2) Indicate the expected direction of distortion for each of the metrics.

21.5 HD services are vital for successful regular use of a software system.

(1) Suggest situations where the HD service is a failure.
(2) What metrics can be applied for the failure situations mentioned in (1)?

21.6 Section 21.3 describes several measures used to construct the software development metrics presented in this section.

Based on the listed measures, suggest two new process quality metrics and two new process productivity metrics.

21.7 Section 21.4 describes several measures used to construct the HD and corrective maintenance metrics presented in this section.

Based on the listed measures, suggest three new product quality metrics and two new product productivity metrics.

21.8 Choose one of the product metrics described in Section 21.4 that includes NYF as one of its measures.

(1) Examine the five factors affecting the maintenance measures listed in Section 21.6, indicate in what direction each of them might bias the metrics you have chosen, and indicate how that bias affects the metric's validity.
(2) Examine the above five factors and indicate how each of them may limit the comprehensiveness of the metrics you have chosen.

21.9 A human resources software system requires 15000 lines of Visual Basic code and 5000 lines of SQL code.

(1) Estimate the number of function points required for the software system.
(2) Estimate the number of lines of C code required for the software system.

21.10 Analysis of the requirement specifications for a tender for development of *The Buyers Club CRM System* has been publicized in a professional journal.

ABC Software Labs is considering participating in the tender. The team appointed to prepare the tender analyzed its requirement specifications and obtained the following results:

- Number of user inputs – 28
- Number of user outputs – 36
- Number of user online queries – 24
- Number of logical files – 8
- Number of external interfaces – 12.

The team estimated that 50% of the components are simple, 25% average and 25% complex. The team also evaluated the project's complexity, with an estimated RCAF = 57.

(1) Compute the function points estimate for the project.
(2) Mr Barnes, the Chief Programmer, estimated that 3500 lines of C++ code will be required for the project. Based on the result for (1), do you agree with his estimate?

Topics for discussion

21.1 Two versions for the measure of software system size – KLOC – are applied: one version counts every code line, while the other counts only the non-comment lines of code.

(1) Discuss the advantages and disadvantages of each version. Refer to the validity of both versions.
(2) Try to suggest an improved version that will comply with the arguments you mentioned in your answer to (1).

21.2 *Money-Money*, a software package for financial management of medium-to-small businesses developed by Penny-Penny Ltd, captured a substantial share of the market. The Penny-Penny help desk (HD) has gained a reputation for its high level of professional service to customers who use the software package. During the third and fourth quarters of 2002, the company invested substantial efforts in preparing an improved user manual. Its distribution to customers was completed during December 2002.

The following table presents HD data summarizing the firm's HD activities for the first quarters of 2002 and 2003.

Data	Code	1st Quarter 2002	1st Quarter 2003
Number of customers	A	305	485
Total number of calls received during the quarter	B	2114	2231
Number of HD calls requiring visit to customer's site	C	318	98
Average time for customer calls served by phone (in minutes)	D	9.3	8.8
Average time for customer calls served by visits to customer's site (in minutes)	E	95	118
Number of customer complaints	F	38	41

(1) Propose three quality metrics for the HD services, based on the HD data.

No.	Quality metrics	Calculation formula

(2) Calculate the value of the quality metrics according to the data presented in the above for each quarter, under the following headings.

No.	Quality metrics	Quality metrics for 1st Quarter 2002	Quality metrics for 1st Quarter 2003

(3) Evaluate the changes in the service quality according to the metrics you suggested.
(4) Can the investments made to improve the user's manual be justified? List your arguments.

21.3 The selection of quality metrics presented in Sections 21.3 and 21.4 include several severity metrics for errors and failures (e.g., ASCE and ASHC).

(1) Explain the importance of these metrics and list the managerial needs not covered by the other metrics.
(2) Suggest situations where such metrics are unjustified.

21.4 Examine the metrics described in Sections 21.3 and 21.4.

(1) Analyze the measures (parameters) that comprise the respective metrics and decide whether they are objective or subjective, where objective measures are based on reliable counts and subjective measures are partly or totally determined by professional evaluation.
(2) Compare the attributes of objective and subjective measures.
(3) List the advantages and disadvantages of the two types of measures.

21.5 The two Software Development Department teams have recently completed their projects. Both applied the same development tool and similar programming style, where comments comprise about a quarter of the total number of lines of code. The following metrics were supplied:

	Team A	Team B
NCE	15.4	9.1
NDE	22.3	20.6

(1) What additional data would you require to determine which of the teams achieved better quality results?
(2) After examining the metrics, what differences in software quality conception held by the team leaders may be concluded from the results?

21.6 Choose one of the process metrics described in Section 21.3 that includes KLOC as one of the constituent measures.

(1) Examine the three factors listed in Section 21.6 affecting KLOC (as a measure of the software development process) and indicate in what direction each of them might bias the metrics you have chosen and how this would affect its validity.
(2) Examine the above three factors and indicate the way by which each of them may limit the comprehensiveness of the metrics you have chosen.

21.7 Comparison revealed 188 errors detected during the development process for a team's recently completed project compared with 346 errors during the team's previous project.

(1) What additional data would you require to determine whether real progress in software quality has been achieved (as claimed by the team leader)?
(2) Which software quality metrics would you use to examine the team leader's claim?

21.8 Statistical analysis software packages enable the user to calculate descriptive and analytical statistics.

(1) Explain in your own words the difference between descriptive statistics and analytical statistics.
(2) Explain the differences in the making and implementing decisions based on each type of statistical tool.

Appendix 21A The function point method

21A.1 Introduction

An important attribute of the function point method is its capacity to provide pre-project estimates of project size, stated in terms of required development resources. These estimates represent one major basis for the resource estimates a firm uses in preparing its tender proposals and project plans. Use of such a tool prevents or at least reduces substantially the risk of managerial failure incurred by underestimating (or overestimating, which is less likely) the expected project costs.

It is clear that KLOC measurement for software size does not possess this attribute as the number of code lines may be counted only after programming completion, which occurs at a very late stage of the project. An alternative measure – the *function point method* – possesses the desired attribute. It measures project size by functionality, indicated in the customer's or tender requirement specification. More accurate estimates are produced as the analysis phase progresses and the software system functions and components are thoroughly studied.

An inherent attribute of KLOC use (not shared by the function point method) is dependence on the programming language or development tool. This attribute limits the comprehensiveness of the KLOC measure as it limits its applicability to comparisons based on the same development tool – unless a conversion factor is used. The extreme differences in the number of lines of code needed for function points is illustrated by the averages presented by Jones (1998).

The estimates for the average number of lines of code (LOC) required for programming a function point are the following:

Programming language/development tool	Average LOC
C	128
C++	64
Visual Basic	32
Power Builder	16
SQL	12

It should be noted that the number of function points for a given software system depends to some extent on the function point counting instruction manual used (the ones most commonly used currently are IFPUG 3, IFPUG 4 and Mark II).

The methodology was first presented in 1979 (Albrecht, 1979; Albrecht and Gaffney, 1983). The function point method is already in wide commercial use but is still considered experimental by many professionals. A wide range of research and tool development activity has been carried out. The main research efforts were directed to validating the method, to improve and adapt it to special areas of software such as real-time software systems and object-oriented software systems. Tool development efforts focus on function point application manuals (especially on function point counting methods) and applications to large-scale software systems. Here we mention just a few of the numerous publications: Gramus and Herron (1996), IEEE (2000), Jeffery *et al.* (1993), Low and Jeffery (1990), Symons (1991), Davis (1992), Caldiera *et al.* (1998), Henderson-Sellers (1996) and Shoval and Feldman (1997).

21A.2 The function point method

The function point method for estimating project size is conducted as follows:

- **Stage 1:** Compute crude function points (CFP). The number of software system functional components are first identified, followed by evaluation of each component as "simple", "average" or "complex". At this point we are able to apply weighting factors to the system components and compute their weighted value. The sum of the weighted values for the software system is the CFP.

- **Stage 2:** Compute the relative complexity adjustment factor (RCAF) for the project. The RCAF varies between 0 and 70.
- **Stage 3:** Compute the number of function points (FP):

$$FP = CFP \times (0.65 + 0.01 \times RCAF)$$

Stage 1: Calculation of crude function points

The method relates to the following five types of software system components:

- Number of user inputs – distinct input applications, not including inputs for online queries.
- Number of user outputs – distinct output applications such as batch processed reports, lists, customer invoices and error messages (not including online queries).
- Number of user online queries – distinct online applications, where output may be in the form of a printout or screen display.
- Number of logical files – files that deal with a distinct type of data and may be grouped in a database.
- Number of external interfaces – computer–readable output or inputs transmitted through data communication, on CD, diskette, etc.

The function point method applies weight factors to each component according to its complexity; the form shown in Table 21A.1 can assist in computation of the CFP.

Table 21A.1: Crude Function Points (CFP) – calculation form

Software System Component	Complexity level									Total CFP
	Simple			Average			Complex			
	Count	Weight factor	Points	Count	Weight factor	Points	Count	Weight factor	Points	
	A	B	C= A×B	D	E	F= D×E	G	H	I= G×H	J=C+F+I
User inputs		3			4			6		
User outputs		4			5			7		
User online queries		3			4			6		
Logical files		7			10			15		
External interfaces		5			7			10		
Total CFP										

Stage 2: Calculating the relative complexity adjustment factor (RCAF)
The relative complexity adjustment factor (RCAF) summarizes the complexity characteristics of the software system by assigning grades (0 to 5) to the 14 subjects that substantially affect the required development efforts. The list of subjects is presented in the RCAF calculation form; see Table 21A.2.

Table 21A.2: Relative Complexity Adjustment Factor (RCAF) – calculation form

No.	Subject	Grade
1	Requirement for reliable backup and recovery	0 1 2 3 4 5
2	Requirement for data communication	0 1 2 3 4 5
3	Extent of distributed processing	0 1 2 3 4 5
4	Performance requirements	0 1 2 3 4 5
5	Expected operational environment	0 1 2 3 4 5
6	Extent of online data entries	0 1 2 3 4 5
7	Extent of multi-screen or multi-operation online data input	0 1 2 3 4 5
8	Extent of online updating of master files	0 1 2 3 4 5
9	Extent of complex inputs, outputs, online queries and files	0 1 2 3 4 5
10	Extent of complex data processing	0 1 2 3 4 5
11	Extent that currently developed code can be designed for reuse	0 1 2 3 4 5
12	Extent of conversion and installation included in the design	0 1 2 3 4 5
13	Extent of multiple installations in an organization and variety of customer organizations	0 1 2 3 4 5
14	Extent of change and focus on ease of use	0 1 2 3 4 5
	Total = RCAF	

Stage 3: Computing the number of function points (FP)
The function point value for a given software system is computed according to the results of stages 1 and 2, by applying the following formula:

$$FP = CFP \times (0.65 + 0.01 \times RCAF)$$

21A.3 Example – the *Attend-Master* software system

Attend-Master is a basic employee attendance system that is planned to serve small to medium-sized businesses employing 10–100 employees. The system is planned to have interfaces to the company's other software packages: *Human-Master*, which serves human resources units, and *Wage-Master*, which serves the wages units. Attend-Master is planned to produce several reports and online queries. The scheme of the planned software system is found in the data flow diagram (DFD) shown in Figure 21A.1.

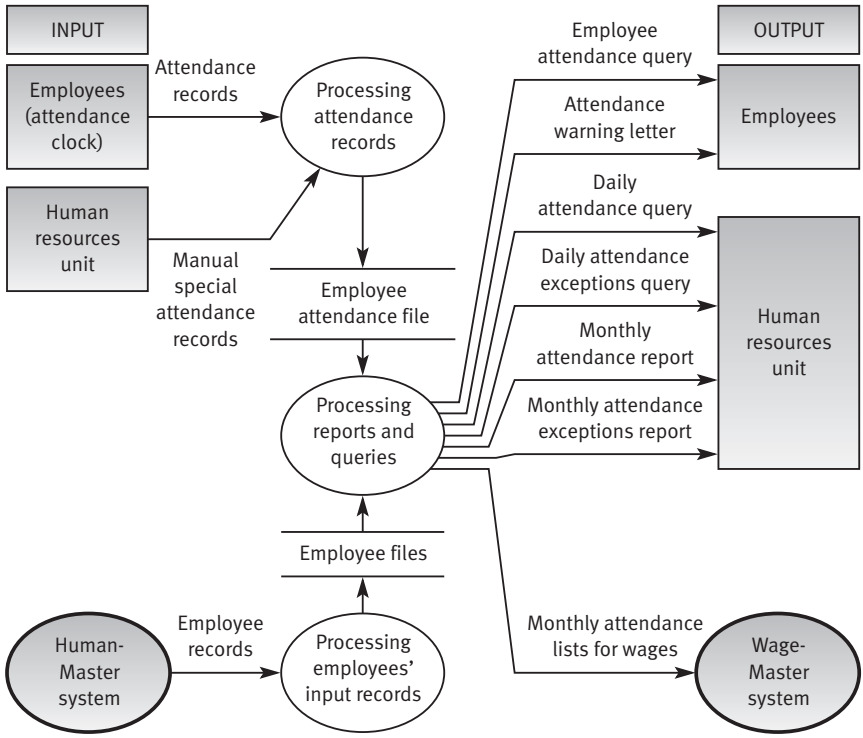

Figure 21A.1: The *Attend-Master* data flow diagram

Let us now compute the function point value for the proposed Attend-Master software system.

Stage 1: Calculation of crude function points

Analysis of the software system as presented in the DFD summarizes the number of the various components:

- Number of user inputs – 2
- Number of user outputs – 3
- Number of user online queries – 3
- Number of logical files – 2
- Number of external interfaces – 2.

The degree of complexity (simple, average or complex) was evaluated for each component (see Table 21A.3), after which CFP calculations were performed.

Table 21A.3: Attend-Master Crude Function Points (CFP) – calculation form

Software System Component	Compexity level									Total CFP
	Simple			Average			Complex			
	Count	Weight factor	Points	Count	Weight factor	Points	Count	Weight factor	Points	
	A	B	C= A×B	D	E	F= D×E	G	H	I= G×H	J=C+F+I
User inputs	1	3	3	---	4	---	1	6	6	**9**
User outputs	---	4	---	2	5	10	1	7	7	**17**
User online queries	1	3	3	1	4	4	1	6	6	**13**
Logical files	1	7	7	---	10	---	1	15	15	**22**
External interfaces	---	5	---	---	7	---	2	10	20	**20**
Total CFP										**81**

Stage 2: Calculating the relative complexity adjustment factor (RCAF)

The evaluation of the complexity characteristics of *Attend-Master* and calculation of the relative complexity adjustment factor (RCAF) are presented in Table 21A.4.

Table 21A.4: Attend-Master RCAF – calculation form

No.	Affecting subjects	Grade
1	Requirement for reliable backup and recovery	0 1 2 3 4⑤
2	Requirement for data communication	⓪1 2 3 4 5
3	Extent of distributed processing	⓪1 2 3 4 5
4	Performance requirements	0 1 2 3 4⑤
5	Expected operational environment	⓪1 2 3 4 5
6	Extent of online data entries	0 1 2 3④5
7	Extent of multi-screen or multi-operation online data input	0 1②3 4 5
8	Extent of online updating of master files	0 1②3 4 5
9	Extent of complex inputs, outputs, online queries and files	0 1 2 3④5
10	Extent of complex data processing	0 1 2 3④5
11	Extent that currently developed code can be designed for reuse	0 1 2③4 5
12	Extent of conversion and installation included in the design	0 1②3 4 5
13	Extent of multiple installations in an organization and variety of customer organizations	0 1 2 3 4⑤
14	Extent of change and focus on ease of use	0 1 2 3 4⑤
	Total = RCAF	**41**

Stage 3: Competing the number of function points (FP)
After applying the stated formula, the calculation was performed as follows:
$FP = CFP \times [0.65 + 0.01 \times RCAF) = 81 \times (0.65 + 0.01 \times 41) = 85.86$

21A.4 Function point advantages and disadvantages

Main advantages:

- Estimates can be prepared at the pre-project stage and therefore can support the management in its project preparation efforts.
- As it is based on requirement specification documents (i.e., function points are not dependent on development tools or programming languages), the method's reliability is relatively high.

Main disadvantages:

- To some extent, FP results depend on the function point counting instruction manual used by the professionals who prepare the estimates.
- Estimates need to be based on detailed requirements specifications or software system specifications, which are not always available at the pre-project stage.
- The entire process requires an experienced function point team and devotion of substantial resources prior to computation of the FP.
- The many evaluations required result in subjective results.
- Most successful applications and research results are related to data processing systems. Other areas of software system require specialized adaptations. In other words, the function point method cannot be universally applied.

chapter 22

Costs of software quality

Chapter outline

More and more, management – whether of commercial companies or public organizations – is requiring economic evaluation of their quality assurance systems. Accordingly, it is becoming ever more likely for proposals for development of new quality assurance tools or investment in improved and expanded operation of existing systems to be examined through an "economic" microscope. Quality assurance units are thus being forced to demonstrate the potential profitability of any request they may make for the substantial funds required to finance additional system infrastructure or operating costs.

We would claim that *cost of software quality* – the economic assessment of software quality development and maintenance – is just another class of