CHAPTER 16

Software Quality Assurance Metrics

G. Gordon Schulmeyer

16.1 Introduction

What is the difference between a measure, a metric, and an indicator? Before delving into the details of software quality metrics, an understanding of these differences is in order. A *measure* (Figure 16.1) is to ascertain or appraise by comparing to a standard. A standard or unit of measurement encompasses: the extent, dimensions, capacity, and so on, of anything, especially as determined by a standard; an act or process of measuring; a result of measurement. Without a trend to follow or an expected value to compare against, a measure gives little or no information. It especially does not provide enough information to make meaningful decisions. A *metric* (Figure 16.2) is a quantitative measure of the degree to which a system, component, or process possesses a given attribute. It is a calculated or composite indicator based upon two or more measures. A metric is a comparison of two or more measures (in Figure 16.2 see body temperature over time) or defects per thousand source lines of code. An *indicator* (Figure 16.3) is a device or variable that can be set to a prescribed state based on the results of a process or the occurrence of a specified condition. An indicator generally compares a metric with a baseline or expected result. This allows the decision makers to make a quick comparison that can provide a perspective as to the "health" of a particular aspect of the project [1].

The purpose of software quality metrics is to assess throughout the development cycle whether the software quality requirements are being met. The use of metrics reduces subjectivity in the assessment of software quality by providing a quantitative basis for making decisions about software quality. The use of metrics, however, does not eliminate the need for human judgment in software evaluations. The use of software quality metrics within an organization or project is expected to have a beneficial effect by making software quality more visible.

One should note that the quality management models and metrics emerged from the practical needs of large-scale development projects and they draw on principles and knowledge in the field of quality engineering (traditionally being practiced in manufacturing and production operations). For software quality engineering to become mature, a systematic body of knowledge should encompass seamless links among the internal structure of design and implementation, the external behavior of the software system, and the logistics and management of the development project [2].

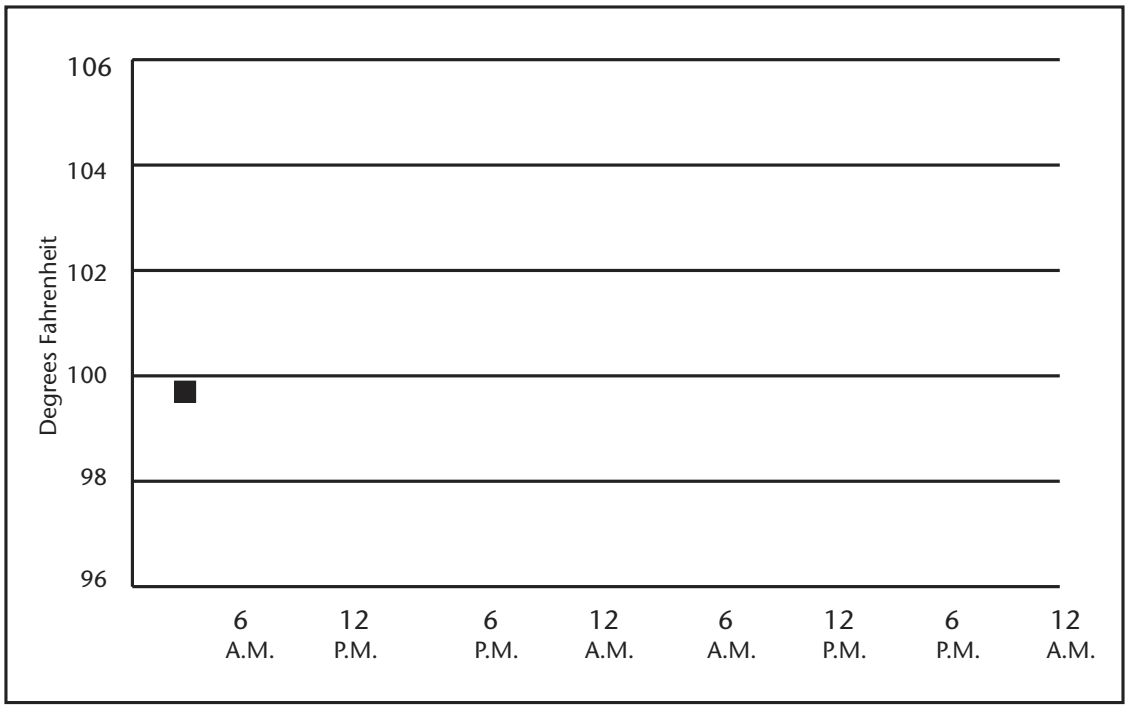


Figure 16.1 Body temperature (measure) sample. (*From:* [1]. © 1995 Bruce Ragland. Reprinted with permission.)

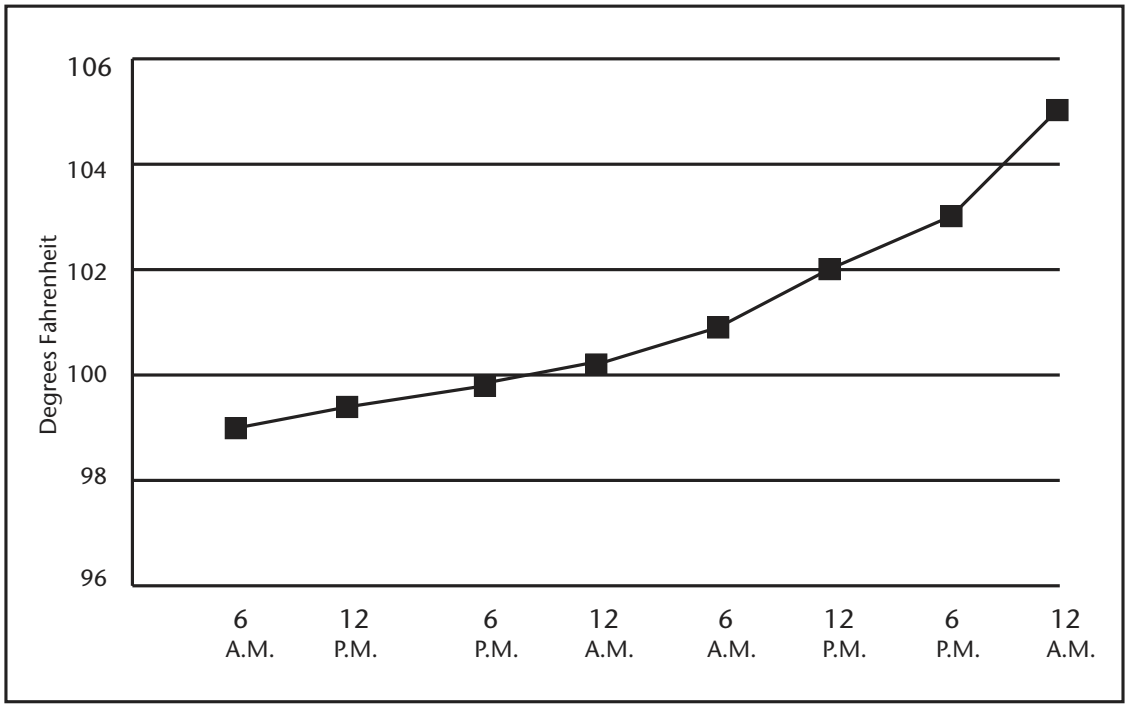


Figure 16.2 Body temperature (metric) sample. (*From:* [1]. © 1995 Bruce Ragland. Reprinted with permission.)

Software quality assurance metrics are intimately connected with software development metrics. So this chapter highlights software quality assurance metrics, but more generally addresses development metrics in order to provide a more complete picture of the interplay among these various metrics.

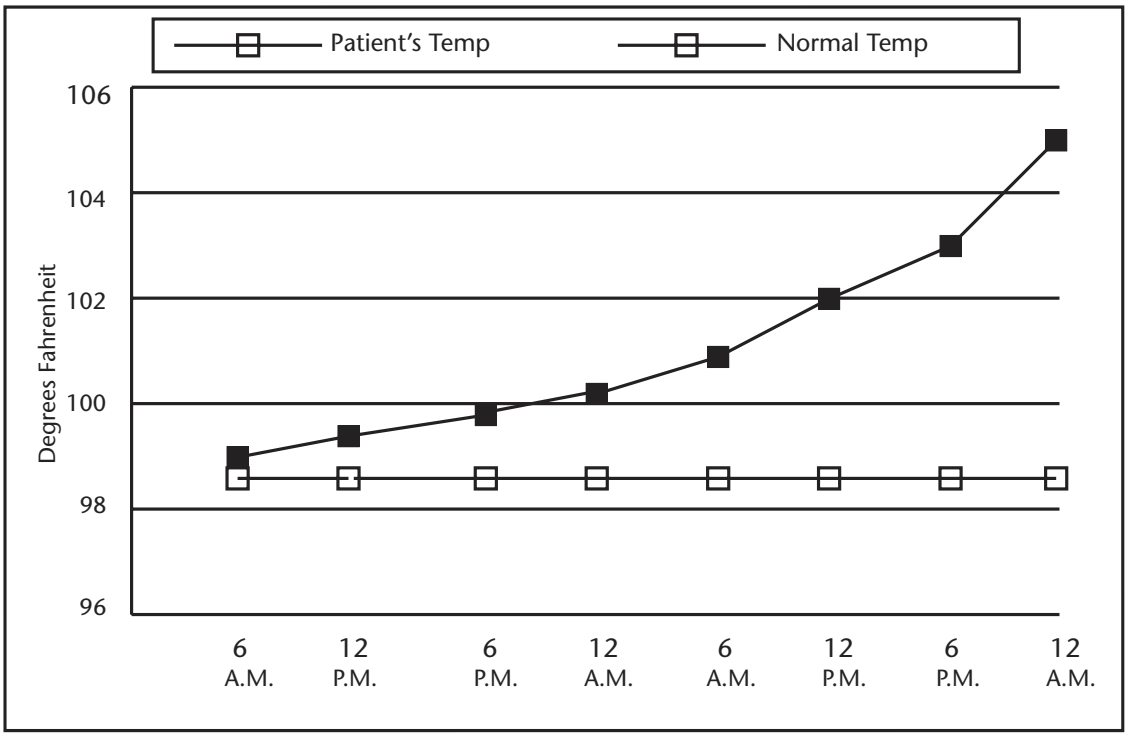


Figure 16.3 Body temperature compared with normal temperature (indicator) sample. (*From:* [1]. © 1995 Bruce Ragland. Reprinted with permission.)

16.2 Software Quality Indicators

Scientific Systems, Inc., under a contract to the Air Force Business Research Management Center, developed a set of software quality indicators (Table 16.1) to improve the management capabilities of personnel responsible for monitoring software development projects. The quality indicators address management concerns, take advantage of data that is already being collected, are independent of the software development methodology being used, are specific to phases in the development cycle, and provide information on the status of a project.

Some recommended quality indicators include:

1. *Progress:* Measures the amount of work accomplished by the developer in each phase. This measure flows through the development life cycle with a number of requirements defined and baselined, then the amount of preliminary and detailed designed completed, then the amount of code completed, and various levels of tests completed.
2. *Stability:* Assesses whether the products of each phase are sufficiently stable to allow the next phase to proceed. This measures the number of changes to requirements, design, and implementation.
3. *Process compliance:* Measures the developer's compliance with the development procedures approved at the beginning of the project. Captures the number of procedures identified for use on the project versus those complied with on the project.
4. *Quality evaluation effort:* Measures the percentage of the developer's effort that is being spent on internal quality evaluation activities. Percent of time

developers are required to deal with quality evaluations and related corrective actions.

5. *Test coverage:* Measures the amount of the software system covered by the developer's testing process. For module testing, this counts the number of basis paths executed/covered, and for system testing it measures the percentage of functions tested.
6. *Defect detection efficiency:* Measures how many of the defects detectable in a phase were actually discovered during that phase. Starts at 100% and is reduced as defects are uncovered at a later development phase.
7. *Defect removal rate:* Measures the number of defects detected and resolved over time. Number of opened and closed system problem reports (SPR) reported through the development phases.
8. *Defect age profile:* Measures the number of defects that have remained unresolved for a long period of time. By month reporting of SPRs remaining open greater than 1 month.
9. *Defect density:* Detects defect-prone components of the system. Provides measure of SPRs/Computer Software Component (CSC) to determine which is the most defect-prone CSC.
10. *Complexity:* Measures the complexity of the code. Collects basis path counts (cyclomatic complexity) of code modules to determine how complex each module is.

These quality indicators have certain characteristics. Quality measures must be oriented toward management goals. One need not have extensive familiarity with technical details of the project. Quality measures should reveal problems as they develop and suggest corrective actions that could be taken. Quality measures must be easy to use. They must not be excessively time consuming, nor depend heavily on extensive software training or experience. Measures that are clearly specified, easy to calculate, and straightforward to interpret are needed. Quality measures must be flexible [3].

16.3 Practical Software and Systems Measurement (PSM)

Practical Software and Systems Measurement (PSM) was developed to meet software and system technical and management challenges. It describes an information-driven measurement process that addresses the unique technical and business goals of an organization. The guidance in PSM represents the best practices used by measurement professionals within the software and system acquisition and engineering communities. PSM is sponsored by the Department of Defense and the U.S. Army. The goal of the PSM project is to provide project managers with the objective information needed to successfully meet cost, schedule, and technical objectives on programs. It is based on actual measurement experience on DoD, government, and industry programs. Measurement professionals from a wide variety of organizations participate in the project. PSM represents the best practices for measurement used within the software and system acquisition and engineering communities. PSM also supports information technology (IT) performance measurement requirements. (See

Table 16.1 Quality Indicators by Development Phase

	Software Requirements Analysis	*Preliminary Design*	*Detailed Design*	*Code and Unit Testing*	*CSC Integration and Testing*	*CSCI Testing*
Process Indicators						
Management Concern:						
Progress	Requirements volume	Top level design complete	Detailed design complete	Units completed	Tests accomplished	Tests accomplished
Stability	System requirements stability	Software requirements stability	Top level design stability	Detailed design stability	Software stability	Software stability
Compliance	Process compliance	<————	————	————	————	————>
Quality effort	Quality evaluation effort	<————	————	————	————	————>
Defect detection						
1. Test coverage				Percentage of paths executed	Percentage of paths executed	Percentage of functions executed
2. Defect detection efficiency		Defect detection efficiency	<————	————	————	————>
Product Indicators						
Completeness	System requirements stability	Software requirements traceability				
1. Defect removal rate	Open and closed problem reports	<————	————	————	————	————>
2. Age profile	Problem report age profile	<————	————	————	————	————>
3. Defect density	Defect density	<————	————	————	————	————>
Complexity	Requirements complexity	Design complexity	Design complexity	Code complexity		

Source: [4].

Chapter 14.) PSM treats measurement as a flexible process, not a predefined list of graphs or reports [5]. The PSM measurement process is defined by a set of nine best practices, called measurement principles [6]:

1. Information needs and objectives drive measurement requirements;
2. Measures based on technical and management processes;
3. Level of detail sufficient to identify and isolate risks and problems;
4. Independent analysis capability implemented;
5. Systematic analysis process to trace measures to decisions;
6. Measurement results in context of other project information;

7. Measurement integrated throughout life cycle;
8. Measurement process as a basis for objective communications;
9. Focus initially on project-level analysis.

An underlying concept of the PSM measurement process is that it should be flexible and able to be tailored based on the unique information needs and characteristics of each project or organization. Measurement must be iterative to support necessary changes that result from changing information needs and improvements in the measurement process itself. The PSM process, shown in Figure 16.4, describes four activities that are part of a successful measurement program:

1. *Plan measurement:* In this activity, measures are defined to provide insight into a project or organization's information needs. This includes identifying what the decision makers need to know, relating these information needs to those entities that can be measured, and then selecting and specifying prospective measures based on project and organizational processes. For example, a comparison of the number of defects written and the number closed addresses the question: "When will the system be ready for user acceptance test?"
2. *Perform measurement:* This activity involves collecting measurement data, performing measurement analysis, and presenting the results so that the information can be used to make decisions. Analysis can include estimation, feasibility analysis of plans, and performance analysis of actual data against plans. The performance analysis of a defect example includes evaluating the trends of written and closed defects, and calculating test readiness.
3. *Evaluate measurement:* In this activity, both the measurement process and the specific measures should be periodically evaluated and improved as necessary. For example, if the defect indicator does not provide enough information to adequately determine readiness for user acceptance testing, additional indicators may be added. The user may add an indicator of defect data by severity (generally all high-priority defects must be closed, but some number of low priority defects may be allowed).
4. *Establish and sustain commitment:* This activity involves establishing the resources, training, and tools to implement a measurement program effectively, and most importantly, ensuring that there is management commitment to use the information that is produced. In the defect example, if the measurement information is not used to develop plans for when user acceptance testing can begin, there is little need for collecting the data.

A measurement process that is flexible and tailored to project and organizational processes ensures that measurement is cost effective. Data should not be collected or reports distributed that are not needed or are not used. In addition, data collection and reporting should be automated whenever possible to provide an automatic by-product of normal project activity.

The process shown in Figure 16.4 provides a foundation for measurement for many disciplines, including software engineering, systems engineering, and process improvement measurement. An important thing to remember is that the same basic

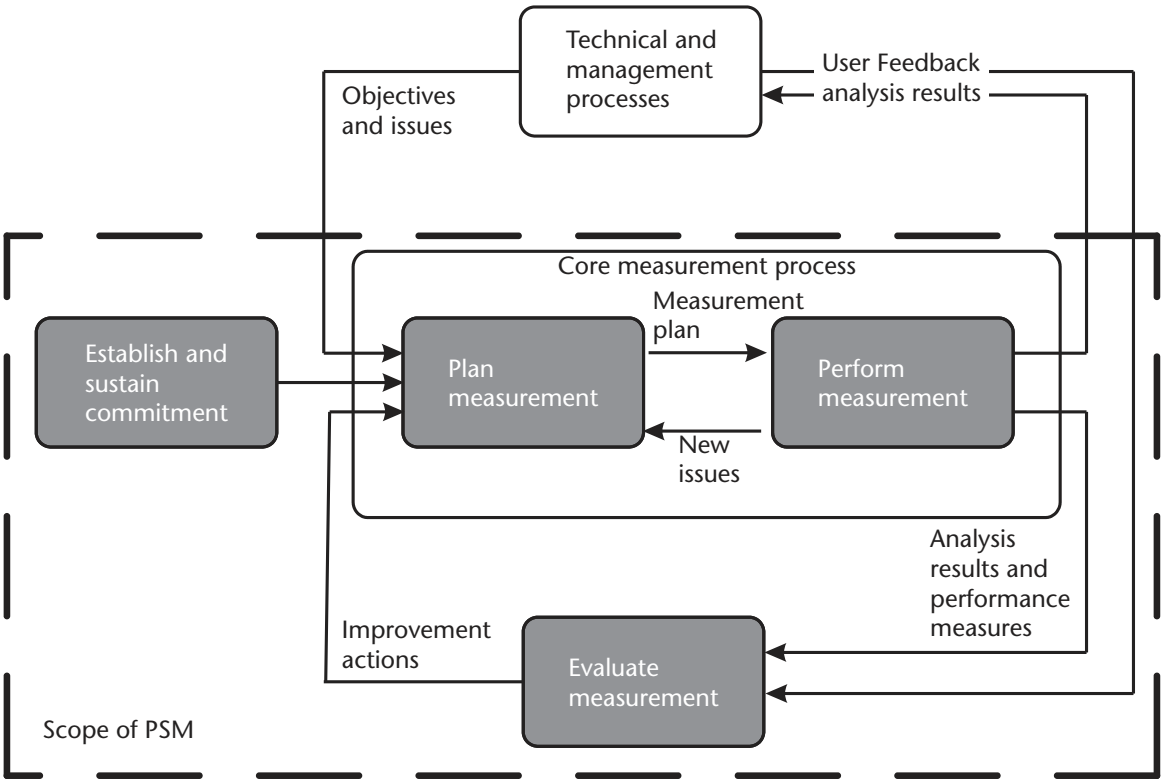


Figure 16.4 PSM measurement process. (*Source:* [7].)

measurement process can support a wide variety of distinct and changing information needs in each of these areas [8].

A highlight of the variety of suggested measures and how they are related is provided in Table 16.2. Drawing on the first example provided in the table, start with the *indicators* of "development milestone schedule" and "milestone progress." They fall under the *measure* of milestone dates. That is in the *measurement category* of "milestone performance." Lastly, it addresses the *common issue area* of "schedule and progress." And so Table 16.2 continues for the PSM suggested measures.

PSM Insight is a free PC-based software tool that automates the PSM process. PSM Insight includes tailoring, data entry, and analysis functions to help develop a project-specific software measurement database and analyses. While PSM Insight provides templates of commonly used issues and measures, it is also completely flexible for you to customize analysis to project-specific needs. Similar to PSM, PSM Insight is sponsored by the Department of Defense and the U.S. Army Software Metrics Office [9].

Inputs to PSM Insight may be manual or may be imported, particularly from Microsoft Excel. However, understanding the metric that one is going to input into PSM Insight is aided immensely by filling in the form shown in Figure 16.5 [10]. Completing the form with responses to proposed questions forces the project to think through aspects of the metric that will clarify why the metric is important for the project to collect and simplify using the PSM Insight tool.

PSM Insight offers a wide variety of outputs, including [11]:

- Trend (line) graphs;
- Snapshot (bar) graphs;

Table 16.2 Index of the Indicator Examples

Indicator	*Measure*	*Measurement Category*	*Common Issue Area*
Development milestone schedule Milestone progress–maintenance activities	Milestone dates	Milestone performance	Schedule and progress
Problem report status Problem report aging—open problem reports Problem report status—open by priority Problem report status—open priority 1 and 2 by configuration item Problem report status—Open priority 1 and 2 by type	Problem report status	Work unit progress	Schedule and progress
Design progress with replan Subsystem acceptance status	Component status	Work unit progress	Schedule and progress
Action item status	Action item status	Work unit progress	Schedule and progress
Incremental content	Incremental content—components	Incremental capability	Schedule and progress
Effort allocation with replan Effort allocation by development activity Staffing level	Effort	Personnel	Resources and cost
Staff experience	Staff experience	Personnel	Resources and cost
Cost and schedule variance	Earned value	Financial performance	Resources and cost
Planned cost profile Cost profile with actual costs	Cost	Financial performance	Resources and cost
Resource utilization—test facilities	Resource utilization	Environment and support resources	Resources and cost
Interface stability	Interfaces	Physical size and stability	Product size and stability
Software size by configuration item Software size—lines of code	Lines of code	Physical size and stability	Product size and stability
Electrical power budget	Physical dimensions	Physical size and stability	Product size and stability
Requirements stability Requirements stability by type of change	Requirements	Functional size and stability	Product size and stability
Multiple indicators for change requests Change requests by priority	Functional change workload	Functional size and stability	Product size and stability
Status of severity 1 defects Defect density Defect density in code inspections Defect classification defects configuration item A Defect density distribution	Defects	Functional correctness	Product quality
System failures and restorations Mean time to repair or fix Mean time to restore system, with threshold	Time to restore	Supportability—maintainability	Product quality

Table 16.2 (continued)

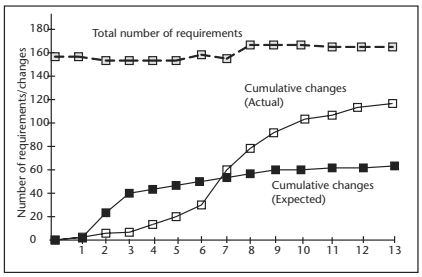
Indicator	*Measure*	*Measurement Category*	*Common Issue Area*
Software complexity—CI A Software complexity—CI A—units with complexity > 10	Cyclomatic complexity	Supportability—maintainability	Product quality
Response time—on-line functions Response time during test—on-line functions	Timing	Efficiency	Product quality
Interface compliance validation	Standards compliance	Portability	Product quality
Problem reports by type of problem data Operator error distribution by reason Device complexity distribution	Operator errors	Usability	Product quality
MTBF ranges based on historical data Reliability growth tracked with mean time to failure	Failures	Dependability—Reliability	Product quality
Reference model level—continuous type Reference model level—staged type	Reference model rating	Process compliance	Process performance
Process audit findings Audit findings by reason code	Process audit findings	Process compliance	Process performance
Software productivity—historical versus proposal Evaluating options using software productivity	Productivity	Process efficiency	Process performance
Requirements defects discovered after requirements phase	Defect containment	Process effectiveness	Process performance
Development effort by activity—compared to total rework effort Rework effort—by activity	Rework	Process effectiveness	Process performance
Critical technology requirements Technology fit—trends	Requirements coverage	Technology suitability	Technology effectiveness
Mean processing time Average cost per picture Estimated yearly maintenance cost	Technology impact	Impact	Technology effectiveness
Technical volatility—cumulative releases Technical volatility—emerging technology Technical volatility—established technology	Baseline changes	Technology volatility	Technology effectiveness
Customer satisfaction survey	Survey results	Customer feedback	Customer satisfaction
Composite performance award scores Performance award category scores	Performance rating	Customer feedback	Customer satisfaction
Total calls per month by priority Mean response time by priority	Requests for support	Customer support	Customer satisfaction

Indicator template
Indicator name/title: _____

Objective: Describe the objective or purpose of the indicator.

Questions: List the question(s) the indicator user is trying to answer. Examples: Is the project on schedule? Is the product ready to ship? Should we invest in moving more software organizations to CMMI® maturity level 3?

Visual display: Provide a graphical view of the indicator (for example).



Perspective: Describe the audience (for whom is this display intended) for the visual display.

Inputs

Data elements	**Definition**
List all the data elements in the production of the indicator.	Precisely define the data element used or point to where the definition can be found.
_____	_____
_____	_____

Data collection
How: Describe how the data will be collected.
When/how often: Describe when the data will be collected and how often.
By whom: Specify who will collect the data (an individual, office, etc.)
Forms: Reference any standard forms for data collection (if applicable) and provide information about where to obtain them.

Data reporting
Responsibility for Reporting: Indicate who has responsibility for reporting the data.
By/to whom: Indicate who will do the reporting and to whom the report is going to. This may be an individual or an organizational entity.
How often: Specify how often the data will be reported (daily, weekly, monthly, as required, etc.)

Data storage
Where: Indicate where the data is to be stored.
How: Indicate the storage media, procedures, and tools for configuration control.
Security: Specify how access to this data will be controlled.

Algorithm: Specify the algorithm or formula required to combine data elements to create input values for the indicator. It may be very simple, such as Input1/Input2, or it may be much more complex. It should also include how the data is plotted on the graph.

Assumption: Identify any assumptions about the organization, its processes, life cycle models, and so on that are important conditions for collecting and using this indicator.

Analysis: Specify what type of analysis can be done with the information.

Interpretation: Describe what different values of the indicator mean. Make it clear how the indicator answers the "Questions" section above. Provide any important cautions about how the data could be misinterpreted and measures to take to avoid misinterpretation.

Probing questions: List questions that delve into the possible reasons for the value of an indicator, whether performance is meeting expectations or whether appropriate action is being taken.

Evolution: Specify how the indicator can be improved over time, especially as more historical data accumulates, e.g., by comparison of projects using new processes, tools, environments with a baseline; using baseline data to establish control limits around some anticipated value based on project characteristics.

Feedback guidelines: A description of the procedure to use when recommending modification to the indicator template.

X-references: If the values of other defined indicators influence the appropriate interpretation of the current indicator, refer to them here.

Figure 16.5 Metrics Description Form. (*After:* [10].)

- Histogram (bar) graphs;
- Reliability models (SMERFS3 input data file);
- Tables.

One output type from PSM Insight can save valuable time by aggregating hundreds of values in any number of ways, as long as the data items are defined additive. Lines of code may be additive as displayed in Figure 16.6.

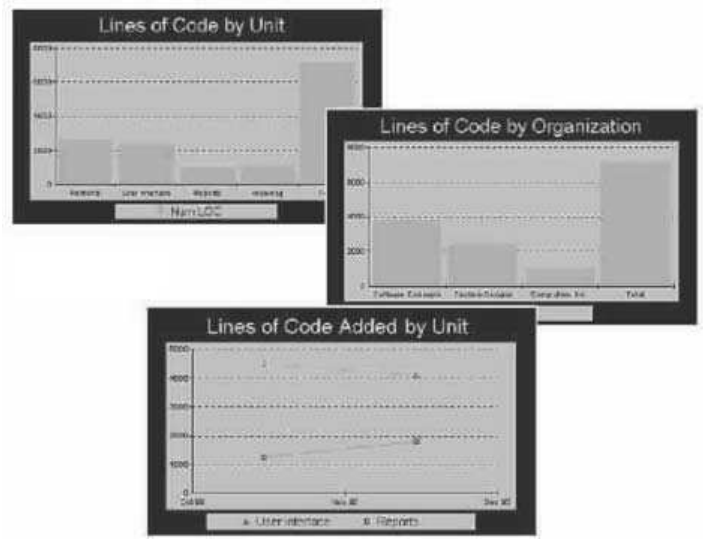


Figure 16.6 Additive lines of code displays from PSM insight. (*Source:* [12].)

Another "output" feature of the PSM Insight tool supports two types of analysis: feasibility analysis and performance analysis. Feasibility analysis determines if project plans are realistic and achievable. It is conducted during the initial planning phase and during all subsequent replanning periods. Performance analysis determines if development is meeting the plans, assumptions, and objectives of the project. It should be conducted regularly throughout the project life cycle, since issues can change at any time [13].

In summary, PSM Insight provides many benefits to the user, including [14]:

- Customization to project-specific needs;
- Templates of commonly-used issues, measures, and indicators from best practices;
- Insight into key software issues;
- Objective data for informed decision making;
- Identification of potential problems and solutions;
- Assistance in meeting cost and schedule objectives;
- Flexible data definitions and analysis tools;
- Presentation-quality graphs and reports;
- Support for risk management of software projects;
- Ease of use in tracking complex projects.

16.4 CMMI® Measurement and Analysis

The CMMI®-DEV, version 1.2 has a process area called Measurement and Analysis. "The purpose of Measurement and Analysis is to develop and sustain a measurement capability that is used to support management information needs" [15]. Within the model, Measurement and Analysis is described as a support process residing at Maturity Level 2. As a support process area, it provides services to other processes. "The Measurement and Analysis process area supports all process areas

by providing practices that guide projects and organizations in aligning measurement needs and objectives with a measurement approach that will provide objective results that can be used in making informed decisions, and taking appropriate corrective actions" [15]. It is consistent with the Goal-Question-Metric (GQM) approach (see Section 16.6.7 and Chapter 14) to identify what needs to be measured. Then, the job is to operationally define, collect, and analyze data, and report information back to the "calling" process.

The practices associated with the first goal, Align Measurement and Analysis Activities, include:

- Establish measurement objectives;
- Specify measures;
- Specify data collection and storage procedures;
- Specify analysis procedures.

As can be seen, these practices really establish the plan for measurement and analysis. They address: Why are we measuring? What are we going to measure? How are we going to measure? What will be done with the data once we have it? As with most, if not all, endeavors within organizations (and life), planning is crucial if we want to achieve our goals. The goal and associated practices within the process area explicitly recognize this need and its importance.

The practices associated with the second goal, Provide Measurement Results, include:

- Collect measurement data;
- Analyze measurement data;
- Store data and results;
- Communicate results.

The theme of these practices is to follow through with the plan; just do it. Note, however, that the goal is to get the results of performing measurement and analysis into the hands of those who will take action based on the results. The process area emphasizes the need that results must be communicated to those needing the information. It does no good to the organization to populate a "write-only" database [16].

In "Measurement within the CMMI®" Johnson and Kulpa relate that the Measurement and Analysis process area puts focus on measurement capability that is used to support management information needs [17]. They go on to note that many organizations have learned that measurements need to be [18]:

1. Aligned to the business objectives to provide benefit;
2. Used regularly in order to justify the effort and cost;
3. Well defined in order for people to understand and compare them;
4. Communicated in an unbiased manner.

There is a sample measurement set (Table 16.3) provided by Johnson and Kulpa that cuts across the process areas since it is for a generic practice: GP 2.8 "Monitor

Table 16.3 Sample Measures for GP 2.8

PAs	*Example Measures from GP 2.8 (Monitor and Control the Process)*
REQM	Requirement volatility (percentage of requirement changes)
RD	Cost, schedule, and effort expended for rework
	Defect density of requirement specifications
TS	Cost, schedule, and effort expended for rework
	Percentage of requirements addressed in design
	Size and complexity of product, product-component, interfaces, and documentation
	Defect density of technical solutions work products
PI	Product-component integration profile (e.g., assemblies planned and actual, and number of exceptions found)
	Integration evaluation problem report trends (e.g., number written and number closed)
	Integration evaluation report aging (i.e., how long each problem report has been open)
VAL	Number of activities planned versus actual
	Validation problem report trends
	Validation problem report aging
VER	Verification profile (e.g., number activities planned versus actual, and the defects found)
	Number of defect detected
	Verification problem report trends
	Verification problem report aging

Source: [20].

and control the measurement and analysis process against the plan for performing the process and take appropriate corrective action" [19].

16.5 CMMI® Higher Maturity Measurements

The CMMI® has a natural evolution of measurement that should occur as organizations strive to improve their processes across the levels. People struggle with the apparent paradigm shifts between the levels as they transition from Level 2 to 3, from Level 3 to 4, and from Level 4 to 5.

Measurement concepts are actually consistent and simply evolve through the levels [21]:

- Level 2: Primarily status measures. Planned versus actual size, effort, cost, and schedule; also includes number of changes, number of nonconformances in product and processes.
- Level 3: Adds measures for process improvement and quality measures including defect density and productivity.
- Level 4: Creation and usage of Process Performance Baseline and Process Performance Models. Looks like a drastic change, but Process Performance Baselines and Process Performance Models are based on historical data from lower levels.

- Level 5: Quantitative improvements based on baselines, using Process Performance Baselines to plan and demonstrate improvements.

Johnson and Kulpa tell us that Level 4 is focused on predicting the performance of the processes based on historical and project data and managing accordingly. Continuing with some important Level 4 concepts [22]:

- Event Level Measure: A measure taken at the completion of an event.
- Process Performance Baseline: Documents the historical results from a process. Used as a benchmark against actual project performance.
- Process Performance Model: Describes the relationship among attributes of a process and its work products. Process Performance Models are based on Process Performance Baselines and calibrated to the project. Process Performance Models are used to estimate or predict a critical project value that cannot be measured until later in the project's life (e.g., number of delivered defects or total effort).

To illustrate their description above, Table 16.4 contains example measures leading to Maturity Level 4 of the CMMI®.

Johnson and Kulpa then elaborate that Level 5 is focused on quantitative improvement based on quantitative understanding of the common causes of variation inherent in the processes. Continuing with some important Level 5 concepts [24]:

- Incremental Improvements: Stepwise improvement accomplished by making the current processes and tools a little better.
- Innovative Improvements: Major performance leaps accomplished by bringing in a significantly different process or technology.
- Target Improvements: Specific areas that have been identified as problematic, often by senior management (e.g., 20% decrease in complaints).

Table 16.4 Measures Leading to Maturity Level 4

Status Measures	*Event Level Measures*	*Process Performance Baselines*	*Process Performance Models*
Size Effort Cost Schedule	Hours per event—Productivity Requirement (defined) Requirement (designed) Object implemented Test executed Defects, Size, Hours per Event-Quality Design review Inspection Test executed Days late or early Task completed	Review Baseline Defects per page and per hour Productivity Baseline Hours per requirement by phase Effort Distribution Percentage of effort by phase	Effort (estimation and prediction) New development Maintenance Defect Insertion and Removal New development Maintenance ↓ Result in real project decisions

Source: [23].

- Common Causes of Variation: Variation caused by normal and expected interactions among components of the process.

To illustrate their description above, Table 16.5 contains example measures leading to Maturity Level 5 of the CMMI®.

Successful organizations focus on a small number of Process Performance Baselines and Process Performance Models that are used to make real decisions; for example, (1) Review Process Performance Baseline, Effort Distribution Process Performance Baseline, and Productivity Process Performance Baseline; and (2) Estimation/Prediction Process Performance Model for effort and duration and a Defect Insertion and Removal Model [26].

16.6 Practical Implementations

16.6.1 Hewlett Packard

Hewlett Packard (H-P) was an early leader in software quality metrics. Their improvement efforts have been reported at various conferences. The following is a list of those early H-P company-wide efforts:

- Management awareness training for every general manager and above;
- Developed Functionality, Usability, Reliability, Performance, and Supportability (FURPS) to describe software quality attributes, which was later revised to FURPS+ to include localization, predictability, and portability;
- Created the function of productivity manager in each research and development division;
- Formed a metrics council of interested engineers and managers and explored/collected many metrics from divisions, highlighted in the 1986 book, *Software Metrics: Establishing a Company-wide Program* [27]; H-P has for years kept very detailed records of software defect data after product release, and they have also had some divisions analyze the causes of defects found prior to release [28];

Table 16.5 Measures Leading to Maturity Level 5

Process Performance Baselines	*Process Performance Models*	*Quantitative Improvements*
Review Baseline Defects per page and per hour	Effort (estimation and prediction) New development Maintenance	Identify including Incremental, Innovative, and Targeted
Productivity Baseline Hours per requirement by phase	Defect Insertion and Removal New development Maintenance	Analyze expected effect on Process Performance Baselines Define and pilot improvements Measure improvements and recalculate process performance baselines
Effort Distribution Percentage of effort by phase		

Source: [25].

- Set up the Software Engineering Laboratory in corporate engineering for tools and methods;
- A high management level software 10X task force reaffirmed the magnitude of the issues, and the need for focus on software;
- The Break Even Time (BET) metric was introduced; this focuses on getting the right product to market in a timely manner with a goal of halving it in 5 years.

Company-wide measures at H-P are very few and very focused, and they serve as drivers for division efforts and programs, which have resulted in a set of best practices at the divisions [29]. A list of some of these measures from HP include [30]:

- *Defects found by customers:* Defects normalized by KLOC uncovered by the customer after a customer release of the system;
- *Percent of code tested:* Percent of code tested (count of LOC tested versus total LOC) by project;
- *Defect analysis by code module:* Defects per KLOC versus cyclomatic complexity of a module and post release defect density of the module;
- *Source of defects by category:* Provide percentage of defects by phase for identified categories of defects for that phase, as shown in Table 16.6;

Table 16.6 Source of Defects by Category

Phase	*Categories*
Requirements/ specifications	Requirements Specifications Functionality Hardware interface Software interface User interface Functional description
Design	Functionality Hardware interface Software interface User interface Functional description Procedural communications Data definition Module design Logic description Error checking Standards
Code	Logic Computation Data handling Module interface/implementation Standards
Test environment	Test software Test hardware Development tools Integration software

- *Inspection effectiveness:* Number of defects found by inspection in the same phase as created, and number of defects found by inspection in later phase than created versus phase inspection conducted in this later phase;
- *Defects released to a customer:* Number of defects uncovered by the customer for each external release to the customer.

16.6.2 Quantitative SQA

Basili and Rombach suggest a model for quantitative SQA that consists of three phases:

1. Define quality requirements in quantitative terms. Select the quality characteristics of interest, define priorities among and relations between those quality characteristics, define each characteristic by one or more direct measures, and define the quality requirements quantitatively by assigning an expected value.
2. Plan quality control through adequate actions to assure fulfillment of the defined quality requirements, control the proper execution of these actions, and evaluate the results.
3. Perform quality control, which consists of: (1) measurement, in which the methods and techniques specified during the planning phase are applied to gather actual values for all defined measures; and (2) evaluation, in which the direct measurements are compared to the quality requirements and indirect measurements are interpreted to explain or predict the values of direct measures. Evaluation also involves deciding if the requirements were met for each quality characteristic and for the entire set of project requirements.

The quantitative SQA model considers the importance of the process, not just the product. One reason quality and productivity are perceived as conflicting is that process quality is often neglected, at least until ISO. It is believed that productivity increases if a high-quality development process is employed.

The quantitative SQA model also accounts for the equal importance of analytic and constructive SQA activities. The term "assurance" (as opposed to analysis) indicates that the objective is both to determine if quality requirements are met (the analytic aspect) and, when they are not met, to suggest corrective action (the constructive aspect). The quantitative SQA model covers all phases of software development so that effective software quality corrective action may be suggested.

Finally, the model stresses the importance of separating responsibilities for development and SQA. It is not important *who* performs the measurement part of quality control as long as it is planned for and evaluated by development-independent personnel [31].

16.6.3 Pragmatic Quality Metrics

Walker Royce at TRW Space and Defense has calculated metrics on real-time Ada projects. Simplicity is achieved by keeping the number of statistics to be maintained

in a Software Change Order (SCO) database to five (type, estimate of damage in hours and SLOC, actual hours, and actual SLOC to resolve) along with the other required parameters of an SCO. (An SCO constitutes a direction to proceed when changing a configured software component.) Furthermore, metrics for configured source lines of code ($SLOC_C$) and total source lines of code ($SLOC_T$) need to be accurately maintained.

The metrics described here were easy to use by personnel familiar with the project context. Furthermore, they provide an objective basis for discussing current trends and future plans with outside authorities. Table 16.7 provides raw data definitions for source lines, errors, improvements, and rework. Table 16.8 shows the in-progress indicator definitions of rework ratio, backlog, and stability. Table 16.9 defines the end-product quality metrics of rework proportions, modularity, changeability, and maintainability, as well as some values determined from real projects [32].

16.6.4 Effectiveness Measure

Measuring the effectiveness of individual quality assurance procedures and of the entire program is an important component of quality control. Robert Dunn's simple effectiveness measure for individual activities is [36]:

$$E = \frac{N}{N + S}$$

where E = effectiveness of activity, N = number of faults (defects) found by activity, and S = number of faults (defects) found by subsequent activities.

This measure can be tuned by selecting only those faults (defects) present at the time of the activity and susceptible to detection by the activity [36]. Testing

Table 16.7 Raw Data Definitions

Statistic	*Definition*	*Insight*
Total source lines	$SLOC_T$ = Total Product SLOC	Total effort
Configured Source lines	$SLOC_C$ = Standalone Tested SLOC	Demonstrable progress
Errors	SCO_1 = No. of Open Type 1	Test effectiveness
	SCO_1 = No. of Closed Type 1	Test progress
	SCO_1 = No. of Type 1 SCOs	Reliability
Improvements	SCO_2 = No. of Open Type 2 SCOs	Value engineering
	SCO_2 = No. of Closed Type 2 SCOs	Design progress
	SCO_2 = No. of Type 2 SCOs	
Open rework	B_1 = Damaged SLOC Due to SCO_1	Fragility
	B_2 = Damaged SLOC Due to SCO_2	Schedule risk
Closed rework	F_1 = SLOC Repaired after SCO_1	Maturity
	F_2 = SLOC Repaired After SCO_2	Changeability
Total rework	$R_1 = F_1 + B_1$	Design quality
	$R_2 = F_2 + B_2$	Maintainability

Source: [33].

Table 16.8 In-Progress Indicator Definitions

Indicator	*Definition*	*Insight*
Rework ratio	$RR = \frac{R_1 + R_2}{SLOC_C}$	Future rework
Rework backlog	$BB = \frac{B_1 + B_2}{SLOC_C}$	Open rework
Rework stability	$SS = (R_1 + R_2) - (F_1 + F_2)$	Rework trends

Source: [34].

Table 16.9 End-Product Quality Metrics Definitions

Metric	*Definition*	*Insight*	*Value*
Rework	$R_E = \frac{Effort_{SCO} + Effort_{SCO}}{Effort_{Total}}$	Productivity rework	6.7%
Proportions	$R_S = \frac{(R_1 + R_2)_{Total}}{SLOC_{Total}}$	Project efficiency	13.5%
Modularity	$Q_{mod} = \frac{R_1 + R_2}{SCO_1 + SCO_2}$	Rework localization	54 SLOC/SCO
Changeability	$Q_C = \frac{Effort_{SCO} + Effort_{SCO}}{SCO_1 + SCO_2}$	Risk of modification	15.7 Hours/SCO
Maintainability	$Q_M = \frac{R_E}{R_S}$	Change productivity	0.49

Source: [35].

effectiveness (the percentage of all errors found in testing that were found in system testing) is one such important measure. The development manager needs to know how the testers are doing, as well as how the programmers are doing. Like error rate, testing effectiveness can be analyzed with a control chart. Establishing a chain of effectiveness measures spanning the life cycle also supports process improvement goals.

The percentage of effort spent in rework provides the best measure of the overall effectiveness of a quality assurance program. Estimates of rework in software development range from 30% to 50% (i.e., 30% to 50% of the total effort is spent on corrective problems). More errors mean more rework. More rework means lower productivity. An effective quality assurance program will decrease rework effort over time. Unfortunately, most software enterprises are very sensitive about measuring rework effort because it tends to be a measure of "failure" rather than of "success" [37].

16.6.5 Team Software Process (TSP®) and Personal Software Process (PSP®)

A principal TSP® and PSP® objective is to apply proven quality principles to the work of individual and teams of software engineers. The expectation is that this will give software work more of an engineering flavor and make it more manageable. (See Chapter 2.) A basic principle of quality management is that "If you don't demand quality work, you are not likely to get it." With TSP® and PSP® quality

management, engineers and development teams track their own defects, find their defect removal yields, and calculate cost of quality measures. Pareto defect analysis (see Chapter 6) is used to derive personal design and code review checklists, which the engineers update with defect data from each new project [38]. The following is a list of measures that are usually collected as part of the PSP®, as well as of the TSP® [39]:

- *Regression calculation—size:* Estimated versus actual new and changed lines of code;
- *Test defects versus appraisal to failure cost ratio—class:* Test defects per thousand lines of code versus appraisal to failure cost ratio;
- *Review yield:* Percent of defects that were in the product at review time found in the review [100 * (defects found)/(defects found and not found)];
- *Compile versus development test and postdevelopment defects:* Compile defects versus development test defects;
- *Size estimating error:* Size estimating error percent by projects;
- *Time estimating error:* Time estimating error percent by projects;
- *Compile time range:* Percent of total time by projects;
- *Defects found in compile:* Compilation defects per thousand lines of code by projects;
- *Test time range:* Percent of total time by projects;
- *Defects found in test:* Test defects per thousand lines of code by projects;
- *Productivity by project size:* Lines of code / hour versus project size (number of lines of code).

The value of PSP® and TSP® is better understood by reviewing the results of TSP® and PSP® measures which show the benefits gained from their use, as shown in Table 16.10.

16.6.6 Software Quality Fault Prediction

Software quality metrics dealing with fault predictions permit the evaluation of trends and the quantifiable analysis of quality, starting with system test. The measurements quantify:

Table 16.10 TSP® Measures Results

Measure	*Pre-TSP®*	*Post-TSP®*
Deviation from schedule	27% to 112% late	8% early to 5% late
System test duration	1 to 5 days/KLOC	0.1 to 1 day/KLOC
Acceptance test defects/KLOC	0.1 to 0.7 defect/KLOC	0.02 to 0.1 defect/KLOC
Postrelease defects/KLOC	0.1 to 1 defect/KLOC	0.0 to 0.1 defect/KLOC

Source: [40].

- The number of faults in generic software, normalized by software size;
- The responsiveness of development and customer support organizations in resolving customers' problems;
- The impact of software field fixes on customers.

Descriptions of these measurements follow [41]:

- *Cumulative fault density found internally:* Faults found internally depict the faults found by the development organization normalized by the total software size in the system test phase.
- *Cumulative fault density found by customers:* Faults found by customers depict the faults found by customers in the normal operation of released software, normalized by the total size of the released software.
- *Total serious faults found:* Provides the number of serious faults found and the status of those faults—open (uncorrected) or closed (corrected)—as of the report date.
- *Mean time to close serious faults:* Provides a measure of the responsiveness of the development and customer support organizations by showing the average time that the serious faults remain open.
- *Mean time still open for serious faults:* Provides a value for each month of the mean length of time that the serious faults, open at the end of the current month, have been open.
- *Total field fixes:* Provides a measure of the impact of software field fixes on customers.

A general model utilized by Bellcore is concerned with a number of customer releases of a certain software product and the underlying trend of software quality. In order to give a numerical illustration of the application of the methodology, consider a problem involving R = 3 releases of a software product. The number of lines of code in the three releases are:

- $L_1 = 160{,}000$;
- $L_2 = 150{,}000$;
- $L_3 = 155{,}000$.

Initially, the variances of predicted faults versus actual faults detected by the customer for the three releases based upon exponentially distributed fault discovery times were calculated as follows:

Release	Variance
# 1	0.04
# 2	0.03
# 3	0.02

But, the use of a formulation for tracking the quality of manufactured hardware which is used to predict the faults detected by customers for each release proved to be a better predictor for software released faults than the assumption of

exponentially distributed fault discovery times as shown by the smaller variances between predicted and actual fault detected by the customer, as follows [42]:

Release	Variance
# 1	0.003
# 2	0.012
# 3	0.093

16.6.7 Measuring Process Improvement Using Stoplight Charts

Using the Goal-Question-Metric (GQM) paradigm is a useful way to determine what the appropriate measures for process improvement status in an organization are. The goals are conceptual:

1. Business goals;
2. Initiative objectives;
3. Strategy or conceptual idea;

then the questions are operational:

1. How do we achieve goals?
2. What is the plan?
3. How do we execute or implement?

and finally the metrics are quantitative:

1. Data;
2. Measures and metrics;
3. Subjective and objective metrics.

Typical goals for a CMMI® program include: (1) better control on IT/software development budget; (2) high quality software delivery; (3) better control over project management; (4) obtain institutionalization of processes; and (5) achieve target maturity level. Of these five typical goals, the usual goal of interest for measuring process improvement is goal 5, achieve target maturity level. What follows are the suggested questions and associated metrics flowing from goal 5, achieve target maturity level [43]:

- Question 1: What is the CMMI® program plan for targeted maturity level?
- Measure 1: Track the percent of schedule variance predicated on the CMMI® program plan.
- Question 2: Is the performance of the process improvement initiative on track?
- Measure 2: Track actuals versus planned program staffing for the CMMI® process improvement initiative.
- Question 3: What is the status of implementation for all process areas within the CMMI®?
- Measure 3: Process area compliance score in the CMMI®.

A very convenient process area compliance score in the CMMI® measure is the "stoplight" chart. The stoplight chart is a roll-up measure of the status of each practice within a process area to a color code, hence the name stoplight chart. The measure is green if all practices in the process area are satisfactory, yellow if most of the practices are satisfactory, and red if many practices are unsatisfactory. That chart (Figure 16.7) provides a quick visibility into both projects and organization status of CMMI® implementation. It also provides some objective evidence for all process areas in the CMMI® for Generic Practice 2.8 (monitor and control "every" process area). Management can easily see the progress achieved in each process area by analyzing the prior reporting period data, as shown.

16.6.8 Six Sigma

An approach that aids in achieving higher maturity levels includes the use of Six Sigma in an organization. Initially, the focus of Six Sigma was to improve manufacturing processes. As it has matured and become more widely used, organizations have been applying this data-driven improvement initiative to the rest of their business life cycles and supply chains. Applications in service or transactional organizations are sometimes termed the "second wave" of Six Sigma implementation. Applications in engineering, including those in software and systems, are sometimes termed the "third wave" of Six Sigma implementation. The paradigm of statistical thinking is embodied in Six Sigma's methodologies, which are used as a basis for executing improvement projects. The framework of Define, Measure, Analyze, Improve, Control (DMAIC) currently prevails. DMAIC is used to improve and optimize existing processes and products. An example DMAIC roadmap is shown in Figure 16.8 [44].

From a process definition standpoint, there is natural synergy between the high maturity process areas and the tenets of Six Sigma's DMAIC framework. As such, the tactics of Six Sigma can be used to directly enrich the defined processes that address the high maturity process areas. For instance, the processes related to the Quantitative Process Management and Causal Analysis and Resolution process areas would reflect both the specific practices of those process areas and the roadmap steps, substeps, and tools of DMAIC [46].

16.6.9 Project Managers Control Panel

The Project Control Panel (Figure 16.9) is a concept and a tool that enables project managers to quickly and clearly monitor project status. Crucial metrics data is displayed on easy-to-read gauges that provide a means of predicting future project health and facilitating timely corrective actions, if required. Besides the typical project management metrics, the Control Panel highlights some specific quality-related metrics, as described.

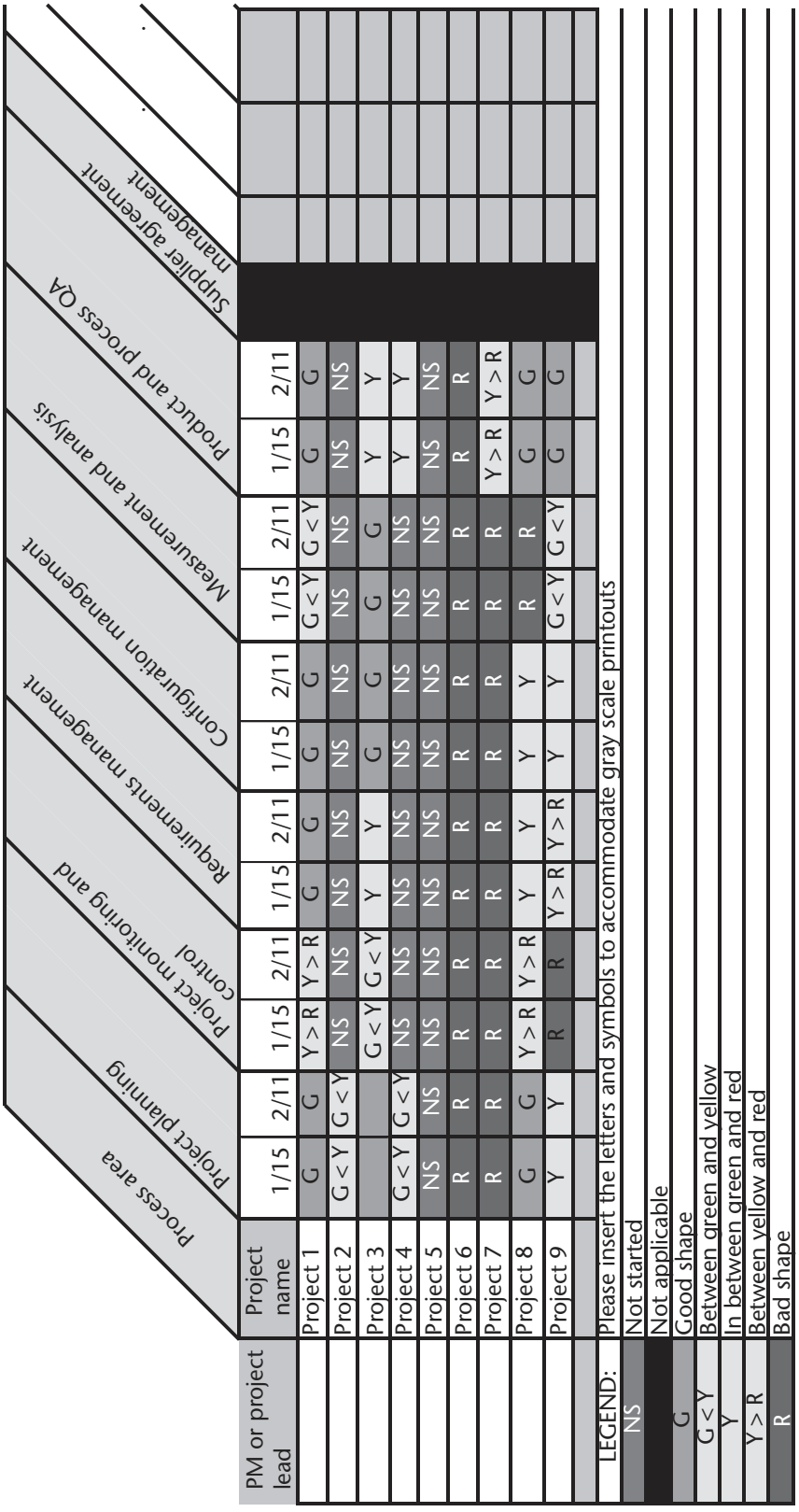
This Month graph shows the completion status of tasks during the current reporting period. A quality gate is a predefined completion criterion for a task. The criterion must be an objective yes/no indicator that shows a task has been completed. The indicators are as follows:

Organization status	Review date	Project planning	Project monitoring and control	Requirements management	Configuration management	Measurement and analysis	Product and process QA	Supplier agreement management			
	1/15/2006	NS	NS	NS	NS	NS	NS				
	2/1/2006	NS	NS	NS	NS	NS	NS				
	2/15/2006	G	G	G	G	G	G				
	3/1/2006	G	G	G	G < Y	G	Y				
	3/15/2006	G	G	Y	Y	G	G				
	4/1/2006	Y	G	Y	Y > R	G	G < Y				
	4/15/2006	G < Y	G	G	R	G	Y				
	5/1/2006										
	5/15/2006										
	6/1/2006										

LEGEND:	Please insert the letters and symbols to accommodate gray scale printouts
NS	Not started
	Not applicable
G	Good shape
G < Y	Between Green and Yellow
Y	In between Green and Red
Y > R	Between Yellow and Red
R	Bad shape

(a)

Figure 16.7 (a) Process area organization status report. (b) Process area projects status report.



(b)

Figure 16.7 (continued)

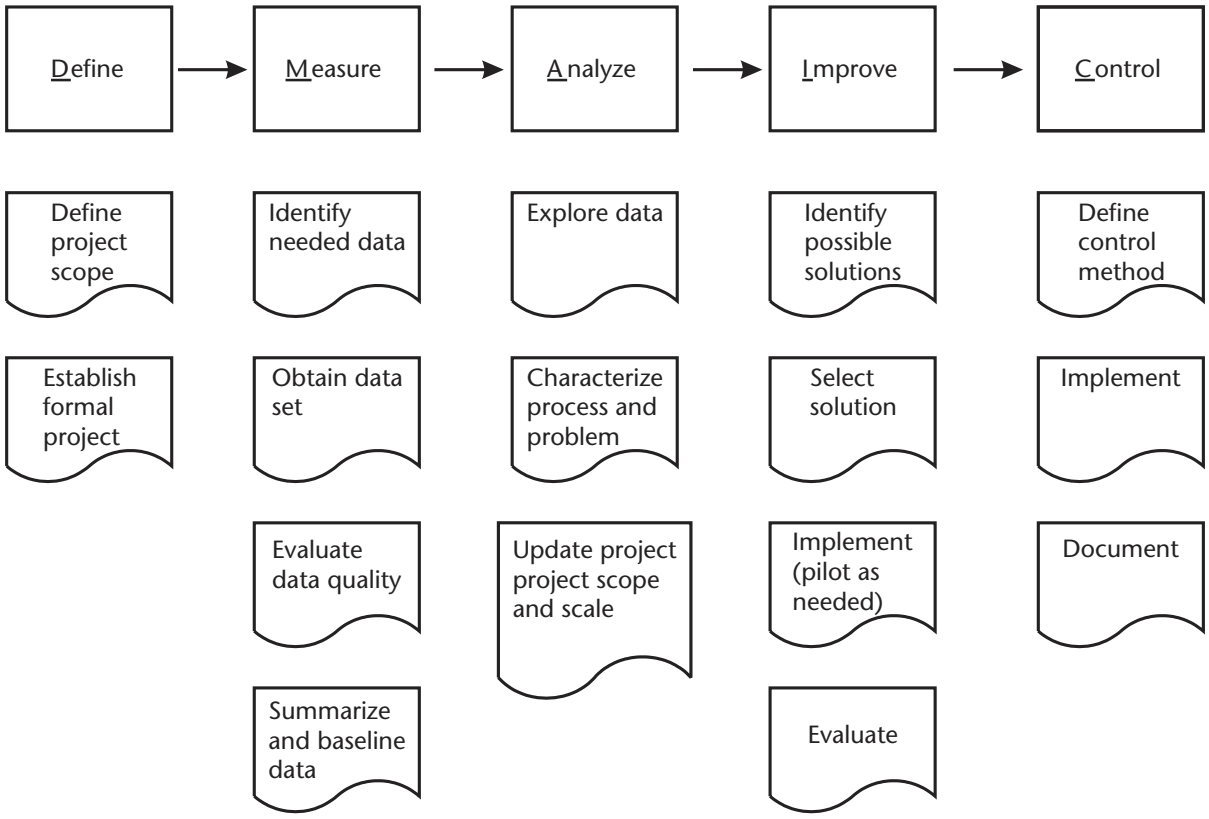


Figure 16.8 DMAIC road map. (*From:* [45]. © 2005 Software Engineering Institute. Reprinted with permission.)

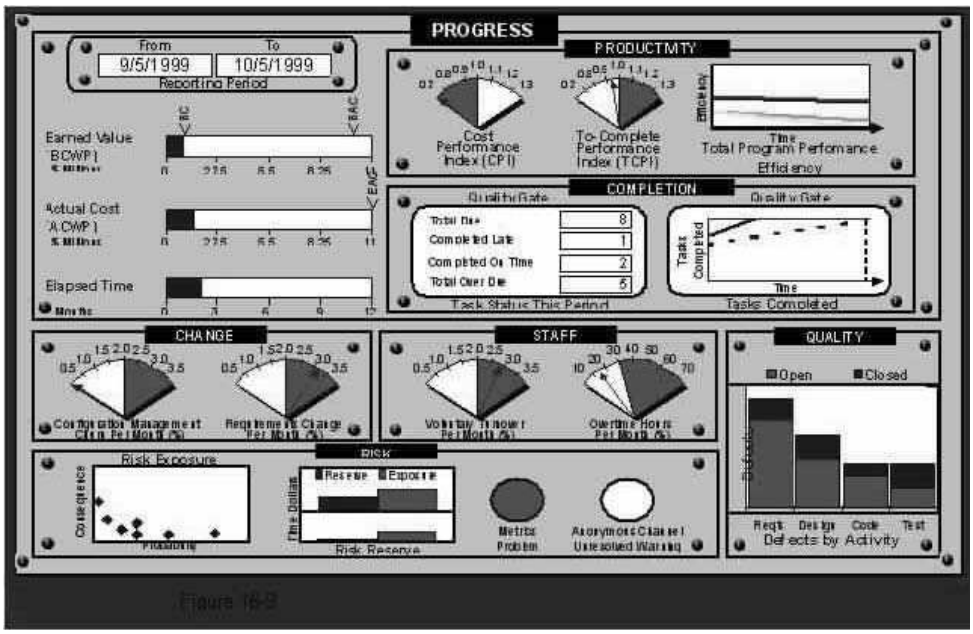


Figure 16.9 Project Manager's Control Panel. (*From:* [47]. © 2000 Integrated Computer Engineering, Inc. Reprinted with permission.)

- *Total Due* is the total number of tasks scheduled for completion during this reporting period plus any overdue tasks from previous periods. This indicates the total quantity of work required for the project to keep pace with the schedule.

- *Completed On Time* is the number of tasks originally scheduled for completion during this reporting period that were completed by their original scheduled due date. This number indicates how well the project is keeping up with scheduled work.
- *Completed Late* is the number of tasks completed late during this reporting period. This number includes those tasks scheduled for this period that were completed late, as well as any overdue tasks from previous periods that were completed in this period. The Completed Late number indicates how well the project is completing work, even if it is late according to the original schedule.
- *Total Overdue* is the total number of tasks for all previous reporting periods that are overdue by the end of the current reporting period. This is an indicator of the quantity of work needed to get the project back on schedule.

Note that the total number of tasks completed in this reporting period is the sum of Completed On Time and Completed Late. Total Overdue then is equal to Total Due minus Completed On Time and Completed Late.

The *Quality Gate Tasks Completed* graph shows the cumulative number of tasks completed by the end of each reporting period to date plotted with the cumulative number of tasks scheduled for completion.

Note that if the number of tasks completed falls below the number planned, then the horizontal distance on the time axis gives an idea of the current schedule slip to date.

Defects by Activity graph displays the number of detected defects open (i.e., yet to be fixed) and the number of defects closed in each phase of the project. Defects are problems that, if not removed, could cause a program to fail or to produce incorrect results. Defects are generally prioritized by severity level, with those labeled 1 being the most serious.

Note that the quality indicators on this chart help answer the question, "What is the quality of the product right now?" [48].

16.6.10 Predicting Software Quality

Basic models for predicting software quality from various contributors include:

- Akiyama: $D = 4.86 + 0.018\ L$.
- Gaffney: $D = 4.2 + 0.0015\ L^{3/4}$ where D is defects and L is lines of code (optimum module size 877 LOC).
- Compton and Withrow: $D = 0.069 + 0.00516\ L + 0.00000047\ L^2$ (optimum module size 83 Ada LOC).

The problems with models such as these are that:

- Defects are not solely caused by design complexity or size.
- Models ignore complexity of problem.
- If you do not test, then you do not find defects.

- Component people produce "better" designs.
- We cannot trust defect density figures.

The solution involves a need to better reflect "difficulties" of quality management. Synthesize partial quality models to (1) include elements from each approach, (2) explain existing empirical results, and (3) be consistent with good sense. There is a need to cope with uncertainty and subjectivity. These solutions are addressed by the Bayesian Belief Networks (BBNs). BBNs consist of three major components: (1) graphical models, (2) conditional probability tables which model prior probabilities and likelihoods, and (3) Bayes' theorem applied recursively to propagate data through the network. The graph topology models the cause-effect reasoning structures. A BBN is a graphical network that represents probabilistic relationships among variables. BBNs enable reasoning under uncertainty and combine the advantages of an intuitive visual representation with sound mathematical basis in Bayesian probability. With BBNs, it is possible to articulate expert beliefs about the dependencies between different variables and to propagate consistently the impact of evidence on the probabilities of uncertain outcomes, such as "future system reliability." BBNs allow an injection of scientific rigor when the probability distributions associated with individual nodes are simply "expert opinions." A BBN will derive all the implications of the beliefs that are input to it; some of these will be facts that can be checked against the project observations, or simply against the experience of the decision makers themselves. There are many advantages of using BBNs, the most important being the ability to represent and manipulate complex models that might never be implemented using conventional methods.

At a general level we can see how the use of BBNs and the defect density model provide a significant new approach to modeling software engineering processes and artifacts. The dynamic nature of this model provides a way of simulating different events and identifying optimum courses of action based on uncertain knowledge. These benefits are reinforced when we examine how the model explains known results, in particular is the "is bigger better?" dilemma. The new approach shows how we can build complex webs of interconnection between process, product, and resource factors in a way hitherto unachievable. It may also be seen how we can integrate uncertainty and subjective criteria into the model without sacrificing rigor and illustrate how decision making throughout the development process influences the quality achieved.

The benefits of this new approach are as follows [49]:

- It is more useful for project management than other analysis and classical statistics.
- It incorporates current research ideas and experience.
- It can be used to train managers and enable comparison of different decisions by simulation and what-if analyses.
- It integrates a form of cost and quality forecasting.

16.7 Conclusion

Traditionally, most of our business measurement processes have been financially based—produced by accountants, designed for accountants (or regulators), and not by or for managers. Measures come in the form of balance sheets, monthly profit and loss statements, and ROIs. They are often damage reports, telling nothing about what is being done do today or tomorrow. We can extrapolate from them, but we know how dangerous that is. Nor do most of our measurement processes tell us about those other objectives that a reengineer wants to constantly scrutinize, such as cycle time and quality, or if they do, reports come too late for us to take action. On the whole, today's measurement processes do not really help us manage [50].

Although the idea of measuring quality by surveying users is novel in software engineering, it is not at all unusual in reviewing other products. Reviews are by experts, who rely for their credibility primarily on the reputation of the organization, not on their personal qualifications. Such reviews are published for many kinds of products: audio equipment, cameras, automobiles, and PC software, to name a few.

We have argued that measuring quality is not just for quality assurance. We have suggested that it is wise to break free from narrow notions of what constitutes quality. From a user's perspective, we have indicated the importance of software multiple releases. We have asserted that subjective assessment of quality can be useful, and that objective measures should be used to support subjective assessment [51].

There are many software quality assurance metrics to choose from and many are being used rather successfully by various companies. Personal experience has shown that any metrics program is very time consuming to implement and difficult to define. I have chaired a software metrics working group for more than 2 years. The progress shown by the working group has been slow, and seemingly every software metric needs to be redone for a myriad of reasons.

While collecting and analyzing metrics remain difficult, that difficulty is diminishing as software tools make the task more manageable. Remember: You cannot control what you cannot measure [52].

Finally, listen to Andy Grove, former CEO of Intel: "What gets measured, gets done" [53].

References

[1] Ragland, B., "Measure, Metric, or Indicator: What's the Difference?" *CrossTalk*, Vol. 8, No. 3, March 1995, p. 29.

[2] Kan, S. H., *Metrics and Models in Software Quality Engineering*, Upper Saddle River, NJ: Pearson Education, Inc., 1995, p. 336.

[3] MacMillan, J., and J. R. Vosburgh, *Software Quality Indicators*, Scientific Systems, 500 West Cummings Park, Suite 3000, Woburn, MA 01801, (781)933-5355, 1986, pp. 1, 2, 7.

[4] MacMillan, J., and J. R. Vosburgh, *Software Quality Indicators*, Scientific Systems, Inc., 500 West Cummings Park, Suite 3000, Woburn, MA 01801, (781)933-5355, 1986, p. 25.

[5] About PSM, http://www.psmsc.com/AboutPSM.asp, December 2006.

[6] Jones, C., "Using PSM to Implement Measurement in a CMMI® Process Improvement Environment," *Software Technology Conference*, April 2003, p. 29.

[7] Jones, C., "Using PSM to Implement Measurement in a CMMI® Process Improvement Environment," *Software Technology Conference*, April 2003, p. 25.

[8] Jones, C., "Making Measurement Work," *CrossTalk*, Vol. 16, No. 1, January 2003, pp. 16, 17.

[9] PSM Insight, http://www.psmsc.com/PSMI.asp, December 2006.

[10] Goethert, W., and J. Siviy, "Applications of the Indicator Template for Measurement and Analysis," Technical Note CMU/SEI-2004-TN-024, Pittsburgh: Software Engineering Institute, © copyright 2004 Carnegie Mellon University, September 2004, pp. 17–19. Special permission to reproduce is granted by the Software Engineering Institute.

[11] Jones, C., *PSM Insight User's Manual*, February 2002, p. 60.

[12] Jones, C., *PSM Insight User's Manual*, February 2002, p. 73.

[13] Jones, C., *PSM Insight User's Manual*, February 2002, pp. 74, 75.

[14] Highlights and Benefits, http://www.psmsc.com/PSMIHighlights.htm, December 2006.

[15] *CMMI®—Development*, version 1.2 (CMMI®—DEV, v1.2), CMU/SEI-2006-TR-008, Pittsburgh: Software Engineering Institute, August 2006, p. 178. © 2006 Carnegie Mellon University. Special permission to use portions is gratned by the Software Engineering Institute.

[16] Zubrow, D., "The Measurement and Analysis Process Area in CMMI®," http://www.sei.cmu.edu/cmmi/publications/meas-anal-cmmi.pdf, December 2006, © 2006 Carnegie Mellon University. Special permission to use portions is granted by the Software Engineering Institute.

[17] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 5. Certain definitions were derived from the book *Interpreting the CMMI: A Process Improvement Approach,* M. Kulpa and K. Johnson, Boca Raton, FL: Auerbach Publications, 2003.

[18] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 4. Certain definitions were derived from the book *Interpreting the CMMI: A Process Improvement Approach,* M. Kulpa and K. Johnson, Boca Raton, FL: Auerbach Publications, 2003.

[19] *CMMI® – Development*, version 1.2 (CMMI®—DEV, v1.2), CMU/SEI-2006-TR-008, Pittsburgh: Software Engineering Institute, August 2006, p. 195.

[20] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 10.

[21] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, pp. 18–21.

[22] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 13.

[23] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 14.

[24] Johnson, K., and M. Kulpa, "Measurement within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 15.

[25] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 16.

[26] Johnson, K., and M. Kulpa, "Measurement Within the CMMI®," Software Engineering Process Group (SEPG) Conference, March 2004, p. 17.

[27] Grady, R. B., and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Upper Saddle River, NJ: Pearson Education, Inc., 1987.

[28] Grady, R. B., and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Upper Saddle River, NJ: Pearson Education, Inc., p. 22.

[29] Ward, T. M., "Software Measures and Goals at Hewlett Packard," *Juran Institute Conference Proceedings*, Atlanta, GA, 1989, pp. 8B-41–8B-42.

[30] Grady, R., "Practical Results from Measuring Software Quality," *Communications of the ACM*, Vol. 36. No. 11, November 1993, pp. 62–68, © 1993 ACM, Inc., included by permission.

[31] Basili, V. R., and H. D. Rombach, "Implementing Quantitative SQA: A Practical Model," *IEEE Software,* March 1990, p. 8, © 1990 IEEE.

[32] Royce, W., "Pragmatic Quality Metrics for Evolutionary Software Development Models," Private paper, May 1990, pp. 5, 17, 18, © 1990 ACM, Inc., included by permission.

[33] Royce, W., "Pragmatic Quality Metrics for Evolutionary Software Development Models," Private paper, May 1990, p. 8, © 1990 ACM, Inc., included by permission.

[34] Royce, W., "Pragmatic Quality Metrics for Evolutionary Software Development Models," Private paper, May 1990, p. 9.

[35] Royce, W., "Pragmatic Quality Metrics for Evolutionary Software Development Models," Private paper, May 1990, pp. 11, 14.

[36] Dunn, R. H., "The Quest for Software Reliability," in *Handbook of Software Quality Assurance*, G. Gordon Schulmeyer and J. I. McManus, (eds.), Upper Saddle River, NJ: Pearson Education, Inc., 1987, pp. 137–177.

[37] Card, D. N., and R. L. Glass, *Measuring Software Design Quality*, Upper Saddle River, NJ: Pearson Education, Inc., 1990, pp. 88, 89.

[38] Humphrey, W. S., "Making Software Manageable," *CrossTalk*, Vol. 9, No. 12, December 1996, pp. 3–6.

[39] Humphrey, W., "Personal Software Process Tutorial," *SEPG Conference Proceedings*, Software Engineering Institute, Pittsburgh, PA, March 1997, pp. 7–53.

[40] Goodenough, J., "Team Software Process Reliability Results," originally printed in the DoD DACS *Software Tech News*, Vol. 3, No. 4, June 2000, pp. 15–16. Requests for copies of the referenced newsletter may be submitted to the following address: DoD Data & Analysis Center for Software, Attn: Lon R. Dean, Editor, PO Box 1400, Rome, NY 13442-1400, (800) 214-7921; Fax (315) 334-4964, news-editor@dacs.dtic.mil.

[41] Inglis, J., "Standard Software Quality Metrics," *AT&T Technical Journal*, Vol. 65, Issue 2, March/April 1986, pp. 113–118, © 1986 Lucent Technologies, Inc., reprinted with permission of John Wiley & Sons, Inc.

[42] Weerahandi, S., and R. E. Hausman, "Software Quality Measurement Based on Fault-Detection Data," *IEEE Transactions on Software Engineering*, 1994, pp. 665–676, © 1994 IEEE.

[43] Prasad, R..T., "Measuring and Managing the CMMI® Journey Using GQM," *Software Engineering Process Group Conference,* March 2006, pp. 5, 6, 11.

[44] Siviy, J., M. L. Penn, and E. Harper, *Relationships Between CMMI® and Six Sigma*, Technical Note CMU/SEI-2005-TN-005, Pittsburgh, PA: Software Engineering Institute, December 2005, pp. 7, 8, © 2005 Carnegie Mellon University. Special permission to use portions is granted by the Software Engineering Institute.

[45] Siviy, J., M. L. Penn, and E. Harper, *Relationships Between CMMI® and Six Sigma*, Technical Note CMU/SEI-2005-TN-005, Pittsburgh, PA: Software Engineering Institute, December 2005, p. 9, © 2005 Carnegie Mellon University. Special permission to use portions is granted by the Software Engineering Institute.

[46] Siviy, J., M. L. Penn, and E. Harper, *Relationships Between CMMI® and Six Sigma*, Technical Note CMU/SEI-2005-TN-005, Pittsburgh, PA: Software Engineering Institute, December 2005, p. 11, © 2005 Carnegie Mellon University. Special permission to use portions is granted by the Software Engineering Institute.

[47] Integrated Computer Engineering, Inc., *Project Control Panel Users Guide* (Version 2.0 for Excel), http://www.iceincusa.com/supportlibrary.aspx?p=supportlibrary_controlpanel, December 2006.

[48] American Systems, *Project Control Panel Users Guide* (Version 2.0 for Excel), Copyright © 1996–2007 American Systems, All rights reserved, pp. 22, 23, 25, http://www.americansystems.com, keyword: Project Control Panel.

[49] Martin, N., and N. Fenton, "Predicting Software Quality Using Bayesian Belief Networks," *Software Engineering Workshop Proceedings*, December 1996, pp. 219, 223.

[50] Champy, J., *Reengineering Management: The Mandate for New Leadership*, New York: Harper Collins, 1995, p. 122, © 1995 James Champy.

[51] Gentleman, W. M., "If Software Quality Is a Perception, How Do We Measure It?" Ottawa: Institute for Information Technology, National Research Council of Canada, 1996, p. 9.

[52] Mills, H. D., and P. B. Dyson, "Using Metrics to Quantify Development," *IEEE Software*, March 1990, p. 16, © 1990 IEEE.

[53] Grove, A., *Only the Paranoid Survive*, New York: Random House, 1999.