Quality Report SQAS97-001

Guidelines for Software Measurement

April 1997

Software Quality Assurance Subcommittee
of the
Nuclear Weapons Complex Quality Managers

United States Department of Energy
Albuquerque Operations Office

Abstract

This document defines a core set of four software measures that can be used within any software organization to characterize a software project and to improve software project processes and products. The core measures are size, effort, progress to schedule, and defects. These measures support organization requirements for attaining the Software Engineering Institute's Capability Maturity Model Levels 2 and 3. This document also describes actions for establishing a software measurement program, integrating software measurement with software process improvement, and implementing core measures through use of a measurement plan and measurement case. These software measurement guidelines will help projects and organizations plan, manage, and improve the life cycle processes of their software systems.

Blank Page

ACKNOWLEDGMENT

The Software Quality Assurance Subcommittee of the Nuclear Weapons Complex Quality Managers initiated Work Item #10 to establish guidelines for software measurement. This document is a significant result of that work item. The working group and other major contributors to this document are listed below.

Brijesh Ajmani, HQ
Mike Blackledge, SA
Faye Brown, OR
Kathy Burris, LA
Kathleen Canal, DOE/HQ
Kathleen Centeno, DOE/HQ
John Cerutti, LA
Charlie Chow, LL
Alvin Cowen, PX
Ray Cullen, SR
Gary Echert, DOE/AL
Cathy Kuhn, KC
Mike Maier, DOE/NV
Carolyn Owens, LL
David Peercy, SA, Chair
Larry Rodin, PX
Ruth Ann Smith, HQ
Ellis Sykes, DOE/KC
Patty Trellue, SA, Editor
David Vinson, PX

The reference “Software Measurement for Semiconductor Manufacturing Equipment”, [SEMATECH], was especially useful in the preparation of Sections 2 and 3 of this guidelines document and requires special acknowledgment.

Blank Page

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION

The Software Quality Assurance Subcommittee (SQAS) is chartered to define methods for improving the quality of software developed within the Nuclear Weapons Complex (NWC). The core measures and implementation guidelines presented here provide a basis for data-driven software project management, quality engineering, and continuous process improvement.

1.1 Background -- The Need For Software Measurement

Within a given organization, software improvement typically involves the achievement of the following goals:

- delivering ever-improving software quality to customers/users while reducing cycle time, cost, and defects; and
- increasing the overall maturity, productivity, and effectiveness of the software engineering process (including both development and support).

The primary mechanism for accomplishing these goals is a structured and institutionalized program of continuous software process improvement based on software measurement. Developing such a program requires a clear understanding of software engineering capabilities and a baseline for measuring improvements.

Members of the SQAS have conducted a number of surveys and assessments within the NWC and other Department of Energy (DOE) sites (e.g., see [SQAS-SURVEY]). The following conclusions have been formed from the results of these efforts:

- most software developers do not know basic facts (such as cost, size, and defects) about their software development efforts;
- there is little understanding of the components of software development costs and profits and apparent uncertainty about what should be measured;
- little agreement exists on how to measure the software process or product quality -- there is little agreement on what characterizes good software quality;
- software project performance is not closely controlled within most projects and most organizations have difficulty accommodating change;
- software processes of most software development organizations are immature, with few (if any) management controls or measurements other than milestone progress; and,
- the arrival of the ship date is often used as the criterion for determining if software is ready for delivery.

The SQAS member surveys and assessments have found that no organization has a comprehensive, clearly defined software measurement programs. Those organizations that have some software measurement program elements tend to use different measurement and reporting mechanisms, and inconsistent measurement definitions. This makes effective comparisons across organizations impossible.

A measurement program supports management's need for answers to key questions about software-intensive projects:

- How large is the job?
- Do we have sufficient staff to meet our commitments?
- How are we doing with respect to our plans?
- Will we deliver on schedule?
- How good is our product?
- How much have we improved our capability to develop software?

To answer these questions, methods are needed to accurately measure software for size, effort, schedule progress, and the quality of the software processes and resulting products. Reliable measurement of these characteristics is crucial to managing project commitments and improving process maturity.

A measurement program provides data for the following important management functions:

- **project planning:** estimating costs, schedules, and defect rates;
- **project management:** tracking and controlling costs, schedules, progress, and quality; and,
- **process improvement:** providing baseline data, tracing root causes of problems and defects, identifying changes from baseline data, and measuring trends.

The measurement program described in this document is intended to support an organization and its specific projects. Each software project should initiate software measurement as an integral part of the project planning process and collect software measures to better quantify the project goals, successes, and failures. Each project's software measures are an integral part of the organization's software measurement program. An organization's software measurement program should be an integral part of a software process improvement initiative.

1.2 Purpose of These Guidelines

The information in this document will assist organizations and projects to establish and improve their software processes and products through measurement. A set of four core software measures is recommended. The core measures are size, effort, progress to schedule, and defects. These measures support requirements for an organization to attain the Software Engineering Institute's (SEI's) Capability Maturity Model (CMM) Levels 2 and 3. This document describes actions for establishing a software measurement program, integrating software measurement with software process improvement, and implementing core measures through use of a plan for measurement and a case of evidence describing the results from the measurement plan activities.

1.3 Scope of Application and Audience

The guidelines in this document apply to any organization that is responsible for the acquisition, development, support, or use of software. Business, scientific, and research software applications can benefit from these guidelines. The software manager, software engineer, software quality engineer, software process improvement group, and software customer have important software measurement roles. For any

specific software project there may be multiple managers, engineers, and customers involved in the measurement process. Some organizations will find it more efficient to have a specialist in software measurement on the staff.

Software measurement is a complex process that involves progressively detailed activities for identifying and applying software measures and for implementing a measurement program. Some aspects (e.g., core measures and implementation examples) of this document should be easily understood, even by personnel who are just beginning to assume responsibility for software. Other aspects (e.g., establishing a software measurement program) may be more easily understood by personnel who are experienced with software engineering methods and process improvement activities.

1.4 Overview of This Document

Section 2 describes guidelines for the recommended set of core measures.

Section 3 describes guidelines for implementing and using the core measures on projects.

Section 4 provides guidance on establishing a software measurement program. A structure for strategic, tactical, and engineering measures is described along with supporting models, methods, and example plan/case documentation outlines.

Section 5 is a summary of the main conclusions and recommendations in this document.

Appendix A includes definitions, acronyms, and abbreviations used in this document.

Appendix B includes a list of references and other important related documents.

Appendix C provides an example organization measurement plan standard with a thematic outline of a measurement plan.

Appendix D provides examples of the four core measures for various projects.

1.5 Owner and Updates

The owner of this document is the SQAS. Feedback from each of the NWC sites, the DOE, and any end-use customers is encouraged and will be addressed by the owner. As appropriate, work items will be established to update this document to reflect suggested changes.

Blank Page

2. RECOMMENDED CORE SOFTWARE MEASURES

This section presents recommendations for implementing a set of four core measures to support software projects. Methods for defining and reporting results are provided for each measure. These methods are supported with reasons for using the measures and recommendations are included for making the measures effective.The recommended core software measures are listed in Table 2-1 along with examples of measure units and the characteristics they address. Software development projects within any organization should use these measures whether acquiring, developing, using or maintaining software systems. Examples of specific core measure sets for a variety of projects are provided in Appendix D.

Table 2-1. Measures for Initial Implementation

Type of Measure	Examples of Measure Units	Characteristic Addressed
Size	Counts of physical source lines of code (SLOC) Function Points	Size, progress, reuse
Effort	Counts of staff hours expended Counts of staff hours to correct problems & defects	Effort, cost, resource utilization, rework
Progress to Schedule	Calendar dates (events/milestones)	Schedule, progress
Defects	Counts of software problems & defects	Quality, acceptability for delivery, improvement trends, customer satisfaction

Although other measurements also capture attributes of software resources, products and processes, the measures listed above are practical, produce meaningful information, and can be defined to promote consistent use. In that regard, the SEI has developed three reports that are useful in defining the methods that are to be used in collecting software measurements. In addition, these reports allow software project management to state what each basic measure includes and excludes. The reports are:

- Software Size Measurement: A Framework for Counting Source Statements [SEITR20].
- Software Effort and Schedule Measurement: A Framework for Counting Staff hours and Reporting Schedule Information [SEITR21].
- Software Quality Measurement: A Framework for Counting Problems and Defects [SEITR22].

Because experienced software managers are rarely satisfied with a single number, each measure requires collection of multiple data items to define the attribute mapping. For example, problems and defects usually are classified according to such attributes as process task activity, status, type, severity, and priority. Effort can be classified by labor class and type of process activity performed. Schedules are defined by process activity, dates and completion criteria. Size measures might be aggregated according

to programming language, development status, and production method. To be of value, both estimated and measured values must be collected at regular intervals (weekly or monthly).

Thus, what may first appear to be just a few measures is actually much more. It will be a significant accomplishment to implement a uniform collection and use of these measures across a single organization much less the many organizations within any given site. Neither the difficulty nor the value of this task should be underestimated.

2.1 Size

Recommendation
Adopt physical source lines of code (noncomment, nonblank source statements) as the measure of software size. Function points are an alternative recommended measure of software size.

Size of software product is dependent upon the attributes of product length, functionality, and complexity. Some of the more popular and effective measures of the length part of software size are physical source lines of code (SLOC) and logical source statement (instructions). Function points (or feature points) and counts of logical functions or computer software units (i.e., modules) might be the measure of the functionality part of software size. Many attributes contribute to complexity, such as the nineteen attributes that are part of the COnstructive COst MOdel (COCOMO) cost estimation algorithm [BOEHM].

As an initial core measure, organizations should adopt physical SLOC as the measure of software size. Even though this measure only represents the length part of size, it should be an initial core measure for the following reasons.

1. SLOC is easy to measure; measurements are made by counting end-of-statement markers, or simply the lines that are neither blank nor comments.
2. Counting methods strongly depend on the programming language used. One need only to specify how to recognize statement types not counted (e.g., comments, blank lines). Automated counters for physical SLOC measures are available.
3. Most historical data for constructing the cost models used for project estimating are based on measures of source code size.
4. Empirical evidence to date suggests that counting physical SLOC is generally as effective as any other existing size measure.

An explicit guide for defining the mapping of software product to the SLOC measure is provided in [SEITR20]. This document spells out rules that address all origins, stages of development, and forms of code production.

Size measurements can be used to track the status of code from each production process and to capture important trends. An example such as the relationships between planned and actual source code size as categorized by copied (reused), modified (partially reused), and planned (new code) is shown in Figure 2-1.

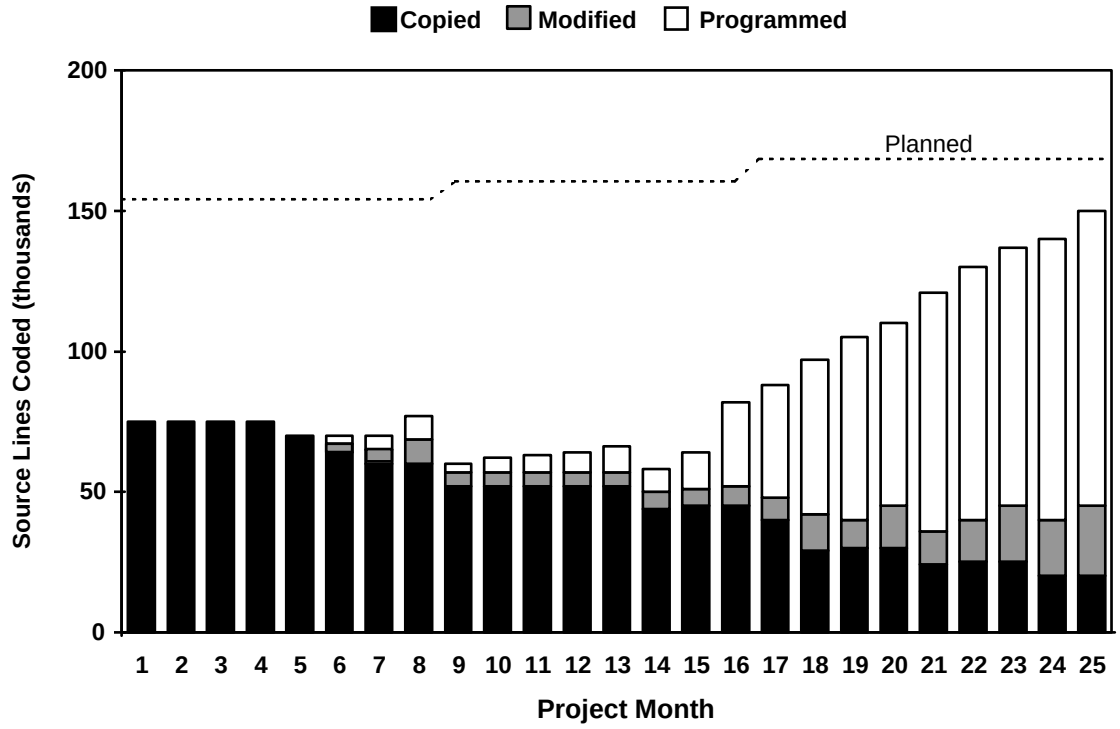

Figure 2-1. The Case of Disappearing Reuse

In this instance, it is apparent that a single measure of size would give a misleading picture of progress and cost. Similar graphs that plot the amount of code by development status also can be useful in relating progress to schedule. Examples using the size core measure are described in Section 3.

Function (feature) points provide a possible alternative or additional size measure, especially for information management software systems. A function point is defined as one end-user application function [DREGER]. Typical end-user application functions include:

- outputs: items processed by the computer for the end user, such as reports and displays;
- inputs: items sent by the user to the computer for processing and to add, change, or delete something;
- storage files: data stored for an application, as logically viewed by the user;
- interfaces: data stored by another application but used by the application of concern; and
- inquiries: simple output from direct inquiries into a master file or special data base using simple keys, requiring immediate system response, and performing no update functions.

The counting process for function points is somewhat more complicated than for SLOC. However, function points have the appealing advantage of relating directly to the complete set of attributes for the size measure: length, functionality, and

complexity properties of the software system. And, the specific function point attributes are typically known somewhat earlier in the software project than SLOC.

There are standards [IFPUG] for counting function points and detailed information [DREGER] on application of function points to support software project size, cost, and schedule estimations for both development and maintenance projects.

2.2 Effort

Recommendation
Adopt staff hours as the principal measure for effort.

Reliable measures for effort are prerequisites to dependable measures of software cost. By tracking the human resources assigned to individual tasks and activities, effort measures also provide the principal means for managing and controlling costs and schedules.

The measurement of effort and schedule, as addressed in [SEITR21], is recommended. This guide provides a useful definition checklist that enables managers to clearly state what is included and excluded in a staff hour. An alternate definition is the staff hour unit in the Institute of Electrical and Electronic Engineers' (IEEEs') draft Standard for Software Productivity Measurements [P1045].

Although other units for measuring and reporting effort data could be used, such as staff months and staff weeks, the use of staff hours is preferable for the following reasons:

1. No standard exists for the number of hours in a labor month. Practices vary widely across projects and reported values range from less than 150 to over 170 hours per labor month. Because of contractual requirements, individual organizations may define a labor month differently for different projects.
2. Staff months often do not provide the granularity needed for measuring and tracking individual activities and processes, particularly when the focus is on process improvement.
3. Measuring effort by staff weeks or staff days presents many of the same problems as staff months, as well as additional ones. For example, although the basic assumption is that a calendar week is five working days, the length of a standard working day varies across organizations. Weekend work, overtime, and holidays must also be addressed and defined if staff week measures are used.
4. Staff month, staff week, and staff day measures can be calculated from staff hours should these measures be needed for presentations or other summaries.

Information about staff hours is used to track resource utilization across tasks and products. Examples using the effort core measure are described in Section 3.

2.3 Progress to Schedule

Recommendation

Projects should adopt structured methods such as those in [SEITR21] for defining two important and related aspects of the schedules they report:

- Dates (both planned and actual) associated with project milestones, reviews, audits, and deliverables; and
- Exit or completion criteria associated with each date.

Schedule and progress are primary project management concerns. For many projects timely delivery is often as important as functionality or quality in determining the ultimate value of a software product. Moreover, project management can become especially complicated when delivery dates are determined by external constraints rather than by the inherent size and complexity of the software product. Overly ambitious, unrealistic schedules often result.

Because schedule is a key concern, it is important for managers to monitor adherence to intermediate milestone dates. Early schedule slips often foreshadow future problems. It is also important to have objective and timely measures of progress that accurately indicate status and that can be used to estimate completion dates for future milestones.

Cost estimators and cost model developers are also very interested in schedules. Project duration is a key parameter when developing or calibrating cost models. Model developers and estimators must understand what activities the duration includes and excludes. For example, if a project is described as having taken 3 1/2 years, it is reasonable to ask questions such as:

1. Exactly what milestones were included in that period?
2. What were the start/stop criteria that bounded the project?
3. Did the project include system requirements analysis and design, or just software activities?
4. Did the period include hardware-software integration and testing or just software integration?

The tracking of dates for milestones, reviews, audits, and deliverables provides a macro-level view of project schedule. In addition, tracking the progress of activities that culminate in reviews and deliverables can reveal potential problems. Tracking the ratio of tasks completed to tasks planned during each reporting period provides objective knowledge of where the project is at any given point and the trend over time. This "task completion ratio" provides a valuable understanding of dynamically changing tasks and schedules.

- **Potential completion criteria**

    1. internal review held;
    2. formal review with customer held;
    3. all high-priority action items closed;
    4. all action items closed;
    5. document baselined under configuration management;

6. product delivered to customer;
7. customer comments received;
8. changes incorporated; and,
9. customer sign-off obtained.

Project activities or "phases" are not sufficient. The variations associated with beginning and ending activities make it difficult to define start and end dates precisely. Most activities (e.g., requirements analysis, design, code) continue to some extent throughout the project. Where one project may consider requirements analysis finished with a software specification review, another may consider it to be ongoing throughout development. A second source of ambiguity stems from the fact that some activities start, stop, and restart, making it very difficult to establish meaningful dates.

Project milestones, reviews, audits, and deliverables should have clear-cut completion criteria linked to specific dates and should be reported separately by product, using the following practices:

- **Dates of milestones, reviews, audits, and deliverables**
  1. Require and report both planned and actual dates.
  2. Specify the dates to be reported. A good first set includes the baseline date for products developed as part of a given activity, the date of formal review, the date of delivery for interim products, and the date of formal sign-off.
  3. Allow some dates to apply to an entire build or system. In other cases, dates should be specified for each software component. For critical components, such as those developed by many NWC software projects, it may be appropriate to track dates for individual software units or modules.
  4. Require that planned and actual dates be updated at regular intervals. Keep a record of all planned schedules and any major events that have a significant impact on the planned schedules. Much can be learned by looking at the volatility of those plans over time and the extent to which they are based on supporting data (like the progress measures).

- **Progress measures**
  1. Specify the measures to be tracked.
  2. Require or produce a plan that shows the rate at which work will be accomplished. A plan should be developed for each major software component. Require that the planned and measured values be reported at regular intervals.
  3. Require objective completion criteria to make progress measures meaningful. Make sure that these criteria can be audited to be sure that progress is real.

2.4 Defects

Recommendation
Counts of software problems and defects and rework time should be used to help plan and track development and support of software systems.

Determining what a customer or user views as true software quality can be elusive. Whatever the criteria, it is clear that the number of problems and defects associated with a software product varies inversely with perceived quality. Counts of software problems and defects are among the few direct measures for software processes and products. These counts allow qualitative description of trends in detection and repair activities. They also allow the tracking of progress in identifying and fixing process and product imperfections. The counts and rework time should be used to help determine when products are ready for delivery to customers, identify the operational impact of software defects on the customer, and provide fundamental data for process and product improvement. In addition, problem and defect measures are the basis for quantifying other software quality attributes such as reliability, correctness, completeness, efficiency, and usability. IEEE *Standard for Software Quality Metrics Methodology* [P1061] addresses this subject.

Defect correction (rework) is a significant cost in most software development and maintenance environments. The number of problems and defects associated with a product contribute directly to this cost. Counting problems and defects can aid understanding of where and how they occur and provide insight into methods for early detection, prevention, and prediction. Counting problems and defects also can directly help track project progress, identify process inefficiencies, and forecast obstacles that will jeopardize schedule commitments.

The SEI report on software quality measurement [SEITR22] provides a structure for describing and defining measurable attributes for software problems and defects. It uses a checklist and supporting forms to organize the attributes so that methodical and straightforward descriptions of software problem and defect measurements can be made. These forms can be used to define or specify a wide variety of problem and defect counts, including those found by static processes (e.g., design reviews or code inspections) or by dynamic processes (e.g., testing or customer operational use).

Opportunities abound for using defect and problem reporting to advantage across many applications including the following:

Past projects. For projects in the post-release stage, a basic measurement is defect logging (software trouble/problem reports), classification, prioritization, resolution, and customer notification. This is a customer service function and is critical to increasing customer satisfaction.

Ongoing projects. For projects currently in development and already measuring problems and defects, the data collected should be used to estimate project completion and potential failure rates during operational use. This may reveal that the

measurements are less than clear and precise in their meaning or that they fall short of what is needed to control the development or maintenance activity.

New projects. For projects establishing or expanding a measurement system, an initial task is to define the measurements that will be used to determine and assess progress, process stability, and attainment of quality requirements or goals.

Serving the needs of many. Software problems and defect measurements apply directly to estimating, planning, and tracking various parts of the software development process. Users are likely to have different purposes for using and reporting this data. The data can serve as a starting point for developing a repository of problem and defect data that can be used as a basis for comparing past experience to new projects, showing the degree of improvement or deterioration, justifying equipment or tool investment, and tracing product reliability and responsiveness to customers.

3. IMPLEMENTING AND USING THE CORE MEASURES

This section outlines some ways to implement and use the recommended core measures identified in Section 2. The recommended core measures can be used to provide early warnings of potential problems, generate reliable projections, or suggest and evaluate process improvements. This section includes discussion and examples of the following:

1. Establishing project feasibility: using size/schedule/cost measures prior to the start of a project.
2. Evaluating existing project plans and data: using size/effort/schedule/defect data to derive patterns that characterize high-risk projects.
3. Tracking project progress: identifying trend data that reflect likely problems and projecting the trend data into the future.
4. Improving the software process: using measurement results to streamline maintenance and evaluate the impact of design and code inspections.
5. Calibrating cost models: using project data to assist an organization in calibrating commercially available cost models.

When implementing a measurement program, the information being reported should be formally described. Clear descriptions for measurement results can minimize misunderstandings, and inappropriate decisions can be avoided. Such descriptions provide a basis for standardizing measurement definitions across projects and provide guidelines to support such consistency. In addition, these descriptions support training of project personnel.

An appropriate time to conduct data collection activities is at each activity stage entry and exit, with a focus on the early stages and defect removal. This provides for an "in-process" prioritization to using measurement data to improve a project's or organization's processes and products.

3.1 Establishing Project Feasibility

If a project is to deliver the required functionality on schedule, within budget, and with acceptable quality,the project shouldbegin with realistic estimates.

Software cost models provide an objective basis for determining the feasibility of the planned functionality/effort/schedule combination. Functionality is represented by estimates of size combined with descriptions of complexity and hardware constraints. On many projects, the schedule is determined by outside pressures (for example, when the hardware will be ready or when the marketing department promised the new release). The budget also may be determined by outside constraints. Software cost models allow estimators to combine these basic project dimensions to determine whether the project is feasible. If not, then early corrective actions can be taken relative to plans, schedules, and delivered product. Such actions can involve any or all of the following:

- reduction of planned functionality;
- expansion of the time planned to carry out the project; and,

- increasing the budget so as to allow additional resources and/or an improvement in the development environment.

Several cost models allow estimators to enter a specific schedule as a constraint and to observe its effect on total effort. Some also allow users to enter effort as a constraint and observe the effect on schedule.

> *A highly compressed schedule leads to substantially increased effort and cost.*

Typical models accept historical data that describe projects in terms of size (e.g., SLOC) and complexity attributes and provide a estimated profile of staff hours over project time and total project duration.

Managers like to avoid impossible projects and control risky projects. In both cases, examinations of tradeoffs using software cost models provide a basis for understanding the extent of cost and schedule risk.

3.2 Evaluating Plans

Much can be learned, often before any software has been developed, by examining project plans. The following examples show how potential problems sometimes can be identified from staffing and schedule plans and from successive size estimates.

3.2.1 Size

Size is often underestimated early in a project. A great deal of useful information can be obtained from using historical data from similar projects and periodically updating size estimates. As more is understood about the product, the estimates are likely to change. Size growth will impact cost and schedule in ways that should be identified and dealt with as early as possible.

One such case is shown in Figure 3-1. In this figure, counts of reused and new code have been extracted from each of a series of development plans. This project was planned so that there would be substantial reuse of existing code. From Figure 3-1, note that the code growth appears to be nearing 10 - 20% for each new forecast, with costs likely to rise similarly. It also indicates that the situation is actually much worse: by the final plan, all of the planned reuse has disappeared and the forecast for new code development is up by approximately 60%. If such information is not reflected in current schedules and cost estimates, serious questions should be asked.

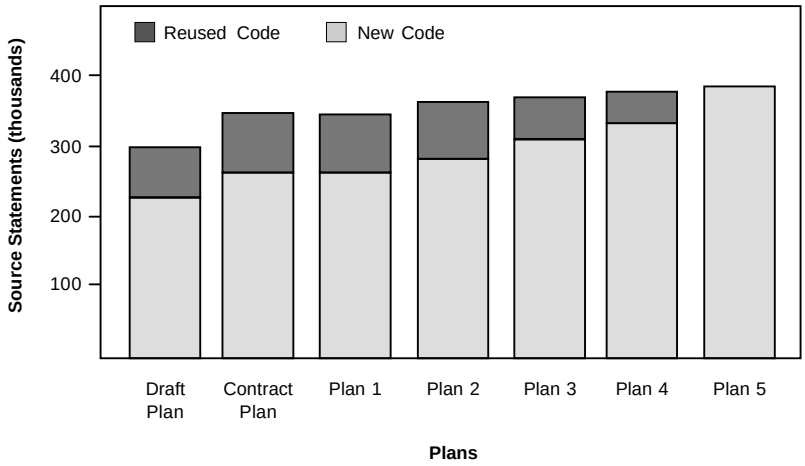

Figure 3-1. Exposing Potential Cost Growth from Disappearing Code Reuse

The importance of keeping planned and actual data is illustrated in Figure 3-2 and inspires some probing questions. Since the project has made only minor changes in the planned completion date despite falling significantly below the original profile over the last nine months, there is reason to examine the code production rate that the current plan implies. Upon doing this, it is easily calculated that the current plan requires an average rate of 12,000 statements per month for months 12 through 20 to reach the planned completion date. This is highly suspect, since the demonstrated capability has yet to reach an average rate of even 2,500 statements per month, considerably less than the 7,600 statements per month that were required in the original plan. It is appropriate to question how the rate of code production can be quadrupled under the current plan. If the project relies on improvements of this magnitude to meet the original completion date, then there should be more emphasis on measuring and tracking the quality of the evolving product.

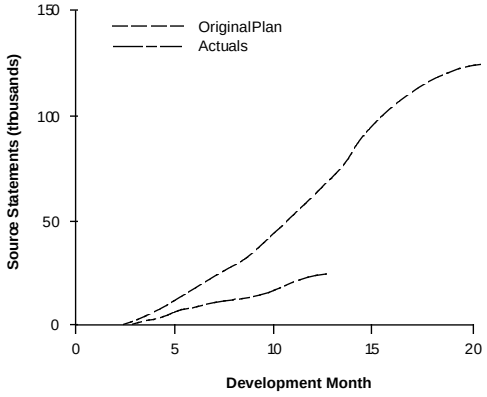

Figure 3-2. Deviations from Original Plan Indicate Problems

3.2.2 Effort

Effort profiles, such as the example in Figure 3-3, can provide early indication of project problems. Project managers should beware of steep ramp-up curves and of

throwing extra people at troubled projects. They are subject to 'Brooks Law' as stated in [BROOKS]. That law says that "Adding manpower to a late project can make it later."

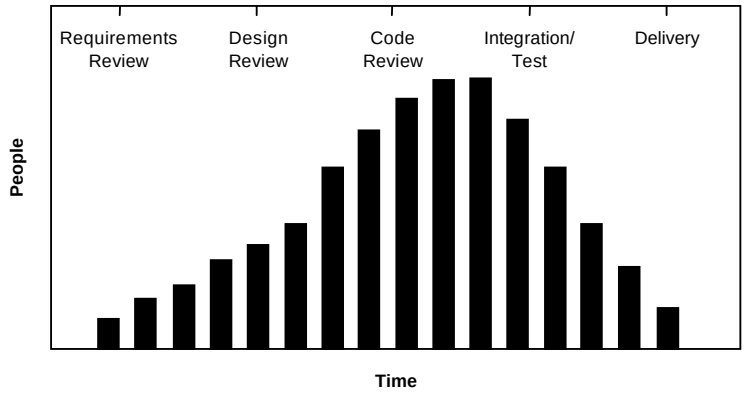

Figure 3-3. Staffing Profile

3.2.3 Progress to Schedule

Attempts to compress schedules typically lead to increased risk. As previously described, fairly severe limits exist as to how quickly any project can ramp up in terms of staffing. A common manifestation of a compressed schedule is the type of plan, illustrated for computer software Module 1 in Figure 3-4, in which fundamentally sequential activities run in parallel. Contrast this with the plan for Module 2. Schedules of the latter type are much more desirable.

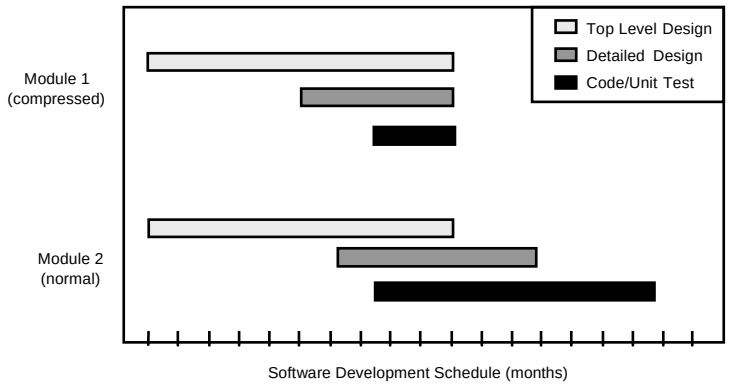

Figure 3-4. Comparison of Compressed and Normal Schedules

Another obvious symptom of a project in trouble is a series of continually slipping milestones such as that as shown in Figure 3-5. In each new plan, the scheduled delivery date slips, resulting in a continually moving delivery date. In this case, new plans were made every two to three months, with each new delivery date slipping by about the same amount of time. It is likely that when a schedule has slipped, then the original estimates for the remainder of the schedule are also too optimistic.

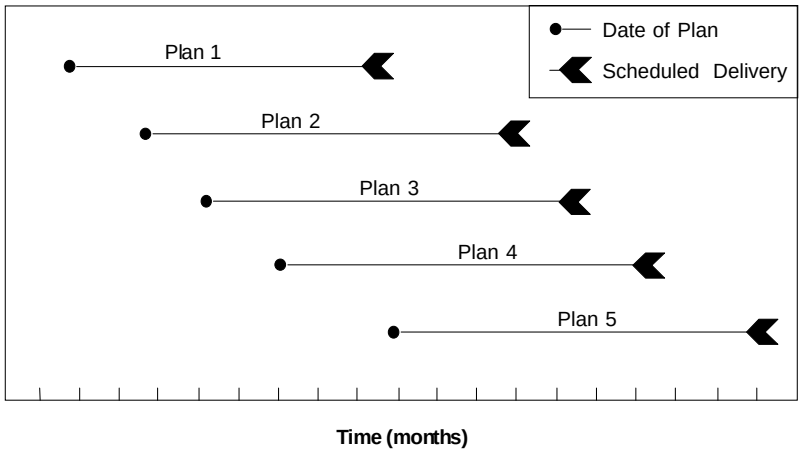

Figure 3-5. Continually Slipping Milestones

3.2.4 Defects

When intermediate milestones keep slipping without corresponding adjustments to the delivery date, the result is that the amount of time allocated to software integration and test gets compressed. When this happens defect detection rates reach sustained high levels with resulting backlogs of open problem reports.

The detection pattern shown in Figure 3-6 is typical for projects in trouble. It shows manpower and detected defects peak during integration and test. These high levels should not be interpreted as phenomena to be tolerated when shortening schedules. Rather, they indicate that considerably more time than anticipated will be required to complete the software. The choices are realistically adjusting the delivery date or delivering a product with a high number of residual defects.

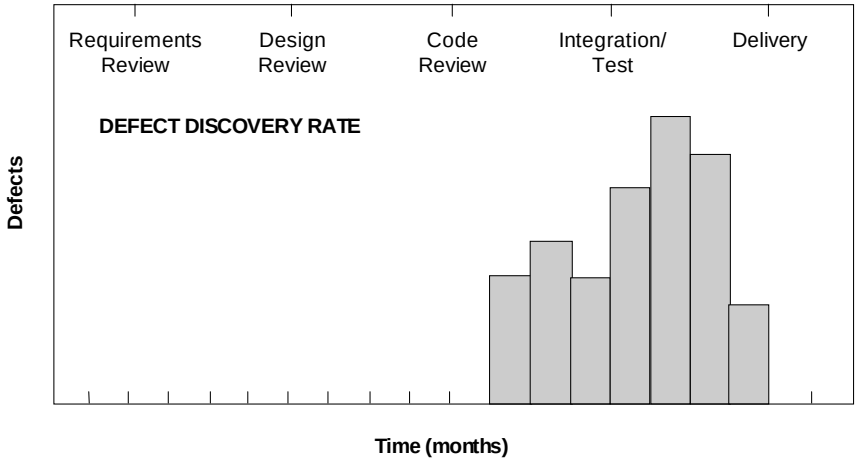

Figure 3-6. Effects of Slipping Intermediate Milestones

In cases where integration and test steps are compressed, objective progress measures can play a key role in providing a basis for defensible schedule projections. The use

of size estimates and projected defect rates and their comparisons with periodic measurement results also provide a basis for this type of analysis.

3.3 Tracking Progress

In managing a project, the following questions are fundamental:

- How much has been done?
- How much is left to do?
- When will it be completed?

These questions can be answered for any activity with outputs or products that can be expressed in quantifiable units. For requirements analysis, this could be the number of requirements to be allocated to each software component. For preliminary design, it could be the number of external interfaces to be specified. For integration and test, it could be the number of test procedures to be executed. To implement, estimate the total number of units to be completed and then, at regular intervals (weekly or monthly), track the actual number completed. Extrapolation of the resulting curve will give an objective basis for projecting the completion date for that activity. A simple linear extrapolation is often accurate.

An example of this type of analysis for code production is shown in Figure 3-7. In this case, the total number of SLOC was estimated at 120,000. The actual SLOC completed was plotted over a five-month period. An extrapolation made at that point yielded a very accurate projection of when coding would be complete.

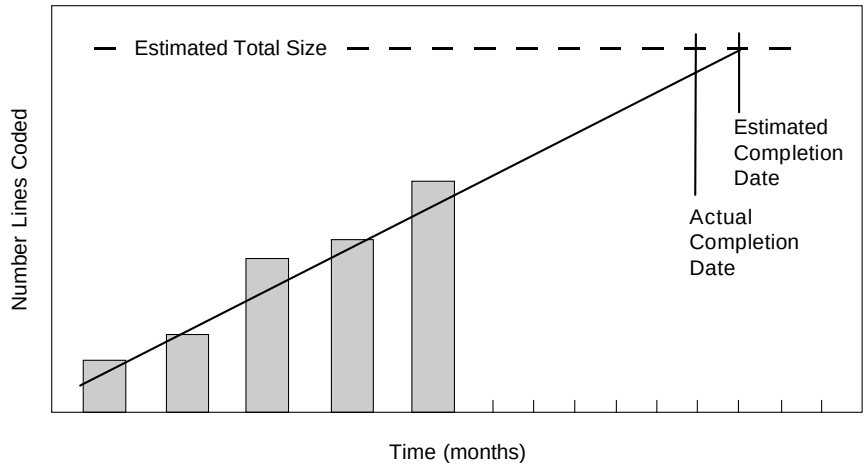

Figure 3-7. Extrapolating Measurements to Forecast a Completion Date

This type of analysis is valid only if objective criteria exists for counting units as complete. In this example, the criteria were that each line of code had completed unit test and had been entered under configuration control. At that point, it was processed by an automated code counter.

The same analysis can be performed for individual components. If some components are lagging more people can be assigned to work on them or the approach can be

changed to increase the code production rate for the affected components. But, be careful of Brooks Law!

Another measure that provides valuable information for projecting completion dates is the rate of defect discovery, especially during integration and test. Ideally, a steady decline in defect discoveries will be seen as the scheduled delivery date approaches. An example of a project that delivered on schedule is shown in Figure 3-8. Contrast this with the pattern shown previously in Figure 3-6, where the number of defects detected peaked near the end of integration and test.

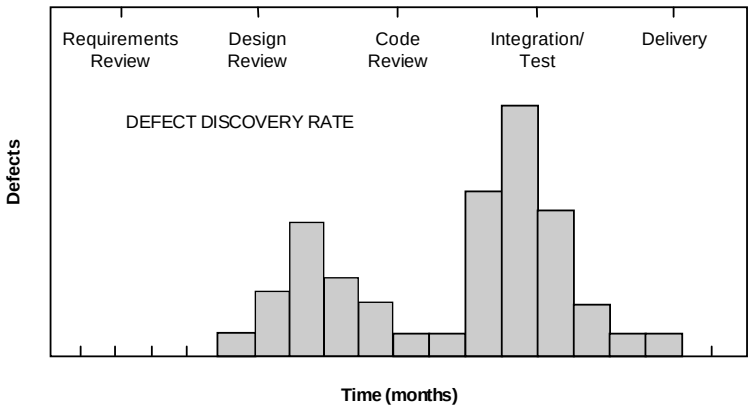

Figure 3-8. Effects of Normal Schedules

3.4 Improving the Process

The recommended core measures provide a basis for evaluating the impact of changes made to software development and maintenance processes. Two such examples are described in this section.

3.4.1 Evaluating the Impact of Design and Code Inspections

Problem reports can be used to evaluate the impact of implementing design and code inspections. A time history of problem reports is illustrated in Figure 3-9 as reported over time for a project that implemented inspections to find defects early in the process. The defects found peaks during design and coding activities and drops off quickly (as desired) during integration and testing. Compare the pattern in Figure 3-9 with Figures 3-6 and 3-8.

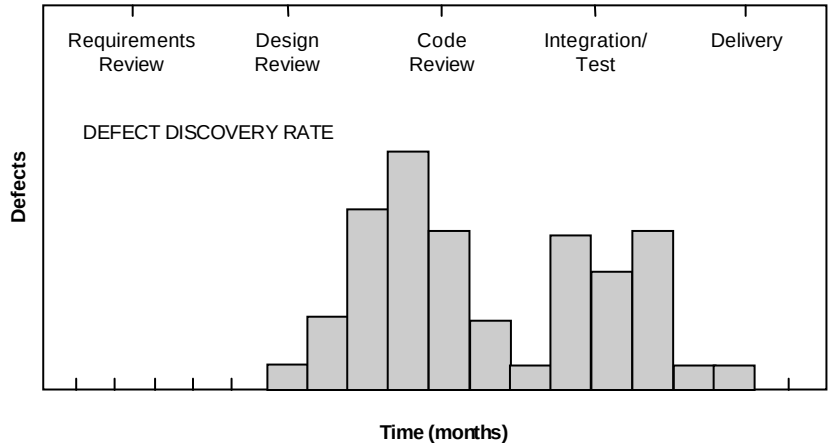

Figure 3-9. Effects of Early Defect Detection

3.4.2 Improving Maintenance

In this example, measures were used to bring much needed visibility to software maintenance. The organization implementing these measures was maintaining fielded versions of their software worldwide. When a problem came in from the field, the organization tracked which system components were at fault and the amount of time spent isolating and fixing the problem. This typically involved a workaround for the customer and a revision to the software for the next release. With information about where the problems occurred and the effort to fix them, the organization was able to identify their most error-prone components and know exactly how much was being spent on maintenance. The organization could make informed decisions about the costs and benefits of redesigning these components.

The defect reduction trend is illustrated in Figure 3-10 over a five-year period of a project in which customer software problem reports are logged and defects are identified and removed. Post-release (software maintenance) defect tracking is one of the keys to increased customer satisfaction.

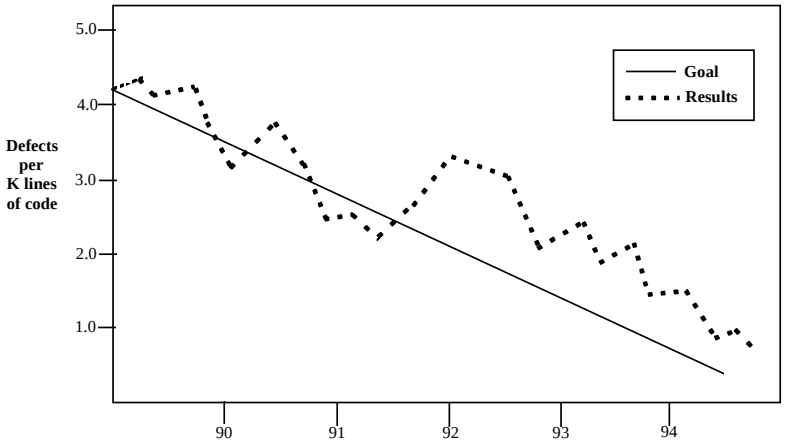

Figure 3-10. Declining Defect Density

3.5 Calibrating Cost Models

Core software measures and data from existing projects can also be used in calibrating cost models. As noted earlier, several commercially available cost models allow estimators to use historical size, effort, and duration data to calibrate underlying estimation algorithms. Others use the data to derive tailored values for parameters that characterize resources, productivity, product difficulties, or organizational capabilities in more complex ways. In either case, once these values are determined from completed projects, the resulting baselines can be used as references to help make future estimates consistent with demonstrated past performance. To be valid, calibration of cost models requires full knowledge of the definitions used when collecting and reporting effort, schedule, and size measurements. Calibration is greatly assisted when consistent definitions and measurement rules are used across projects and organizations.

Blank Page

4. ESTABLISHING A SOFTWARE MEASUREMENT PROGRAM

This section provides an overview of software measurement and an infrastructure for establishing a software measurement program. It is recommended [GRADY87] to start small and build on success. It is also recommended [SEMATECH] to combine a software measurement program with a software process improvement initiative so the measurement program is sustainable. As far as possible, establish automated mechanisms for measurement data collection and analysis. Automated methods should be a support resource of the measurement process rather than a definition of the process. Regularly collect the core measurements and additional measurements specific to the local goals in the organization. Plan and schedule the resources that will be required to collect and analyze the measurement data within the organization's overall software process improvement efforts and the specific organization's projects. Evolve the measurement program according to the organization's goals and objectives. Provide a mechanism for projects and the organization's software process improvement group to consolidate software project measurements.

The four steps identified in Figure 4-1 illustrate a comprehensive process for establishing a software measurement program. An organization may decide to implement a subset of these activities. Organizations should tailor their use of the activities as necessary to meet organization and project goals and objectives. Each of these four major activities is described in the following subsections.

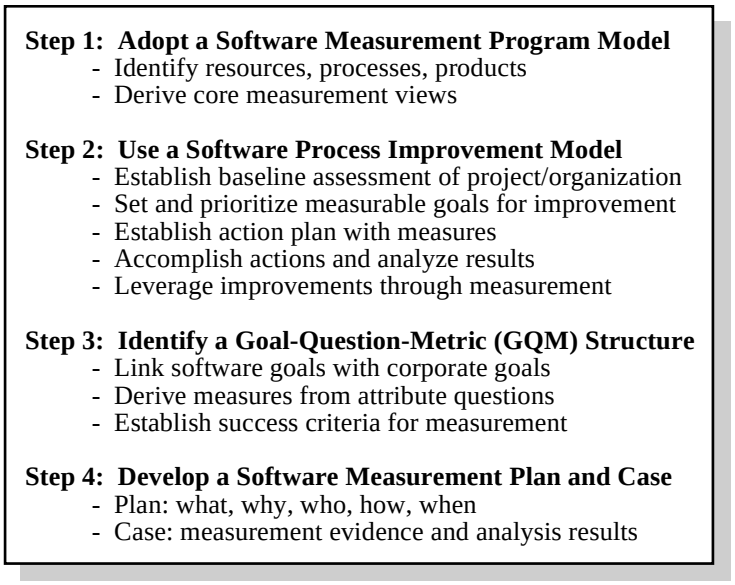

Figure 4-1. Steps to Establishing a Software Measurement Program

4.1 Adopt a Software Measurement Program Model

An organization or a project must understand what to measure, who is interested in the results, and why. To assist this understanding, it is recommended that a software measurement program model be adopted such as illustrated in Figure 4-2.

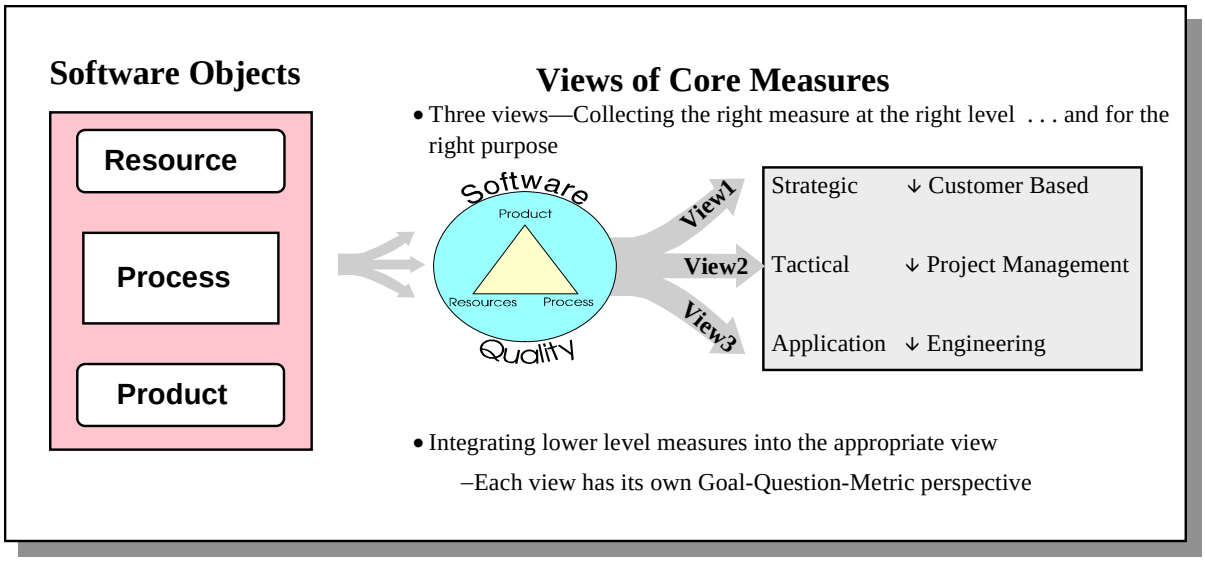

Figure 4-2. Software Measurement Program Model

The measurement program model provides a simple framework for specifically identifying what software attributes are of potential interest to measure, who the various customers of measurement results might be, and why such measurement results are of interest to those customers. The measurement program model includes the general software objects of measurement interest such as resources, processes, and products. The measurement customers include the end-use customer, software organization and project management, and software application personnel. These customers need software measures for different reasons. Their viewpoints drive the eventual measurement selection priorities and must be integrated and consistent to be most effective.

To establish a successful measurement program (e.g., one that is used for organization and/or project decision making and lasts more than two years), it is necessary to have a basic understanding of measurement. The following subsections provide an introduction to attributes of resources, processes, and products that might be useful to measure and some software measurement terminology that relates to the Software Measurement Program Model. Additional information can be found in [FENTON], [NASA], and [PSM].

4.1.1 Resources, Products, Processes

Software objects such as resources, products, and processes have attributes that characterize software projects and are therefore of interest to measure. A software measure is an objective assignment of a number (or symbol) to a software object to characterize a specific attribute [FENTON].

Resources are inputs to processes. Such inputs specifically include personnel, materials, tools, and methods. Resources for some processes are products of other processes. An attribute of great interest that is relevant to all of these types of resources is *cost*. Cost is dependent on the number of resources and the market price

of each resource. For personnel, the cost is dependent upon the effort expended during the process and the market price value of each person assigned to the process.

Processes are any software related activities such as requirements analysis, design activity, testing, formal inspections, and project management. Processes normally have *time and effort* as attributes of interest, as well as the number of incidents of a specified type arising during the process. Certain incidents may be considered to be *defects in the process* and may result in *defects or faults in products*.

Products are any artifacts, deliverables, or documents that are produced by software processes. Products include specifications, design documentation, source code, test results, and unit development folders. Products normally have *size and inherent defects* as attributes of interest.

4.1.2 Direct and Indirect Software Measurement

Direct measurement of a software attribute does not depend on the measurement of any other attribute. Measures that involve counting, such as number of SLOC and number of staff hours expended on a process, are examples of a direct measure.

Indirect or derived measurement involves more than one attribute. Rates are typically indirect measures because they involve the computation of a ratio of two other measures. For example, software failure rate is computed by dividing the count of the failures observed during execution by the execution time of the software. Productivity is also an indirect measure since it depends on the amount of product produced divided by the amount of effort or time expended.

Two other very important aspects of the measurement assignment are preservation of attribute properties and mapping uniqueness. The mapping should preserve natural attribute properties (e.g., such as order and interval size). If another assignment mapping of the attribute is identified, there should be a unique relationship between the first mapping and the second mapping. It is very difficult to ensure that measures satisfy these preservation and uniqueness properties. This document will not consider these issues in any detail. See [FENTON] for a more complete discussion.

4.1.3 Views of Core Measures

The three views (strategic, tactical, application) of the core measures illustrated in Figure 4-2 identify important attributes from the viewpoints of the customer, project management, or applications engineers, respectively. It is extremely important for the measurement program to be consistent across the three views of core measures. There must be agreement and consistency on what measures mean, what measures are important, and how measures across the three views relate to and support each other.

Strategic View: This view is concerned with measurement for the long term needs of the organization and its customers. Important measures include product cost (effort), time to market (schedule), and the trade-offs among such quality measures as functionality, reliability, usability, and product support. It may be critical to an organization to establish new customers and solidify old customers through new

product capabilities -- with limited reliability and usability, but with a well-planned support program. Time to market is usually a critical measure, and may become one of upper management's most important measures.

Tactical View: This view is concerned with short and long term needs of each individual project's management goals. The project measures that support the Tactical View should be able to be aggregated to show a relationship to the organization's strategic goals. If not, then individual projects will appear to be "out of sync" with the organization. The primary measures of interest to project management are schedule progress and labor cost.

Application View: This view is concerned with the immediate resource, process and product engineering needs of the project. Resources (e.g. personnel and support equipment) are of some interest in this view, but the engineer is primarily interested in the process activities to produce a high quality product. The engineering definitions of process and product quality should be consistent with project management or upper level organization management understanding. Product size, complexity, reliability, and inherent defect measures are important to the engineers because they indicate achievement of functional and performance requirements.

4.2 Use a Software Process Improvement Model

In order for a software measurement program to be successful, the measurement activities should be conducted within the environment of continuous software process improvement. Without such an environment measures will not be seen as value-added and the measurement program will not be sustainable. Two models are important to a software process improvement initiative and the integration of software measurement, as illustrated in Figure 4-3. The IDEAL model [SEIIDEAL] provides an organization with an approach to continuous improvement. The Capability Maturity Model [SEICMM] can be used to establish a measurement baseline [SQAS-SPA].

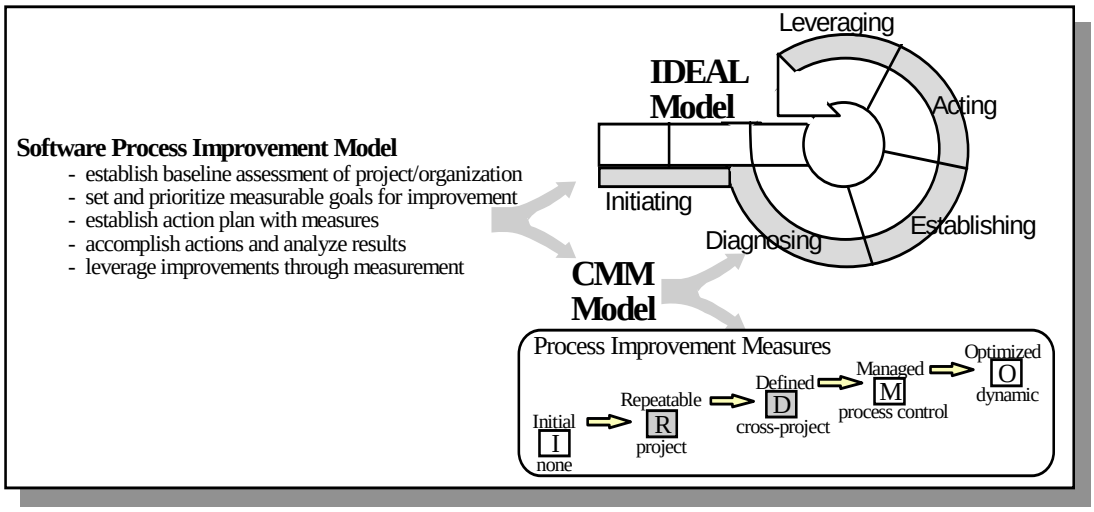

Figure 4-3. Software Process Improvement Models

4.2.1 SEI IDEAL Model

The IDEAL model [SEIIDEAL] provides a framework for conducting process improvement activities at the organization level and the project level. The IDEAL model is similar to the Plan/Do/Check/Act model identified in [DEMING].

Organization Software Measurement. During the Initiate stage, the organization goals and measures for the improvement are defined along with success criteria. The Diagnose stage includes baselining the organization's current process capability (e.g., using the SEI CMM during a Software Process Assessment) in accordance with the measures inherent in the assessment process. The Establish stage provides focus on identifying specific improvements that will be accomplished by action teams and the measures for those improvements. Prioritized improvement actions are determined and action teams are formed to develop specific plans that address the high priority improvements. The Act stage includes implementation of the action team plan including collection of measurements to determine if the improvement has been (or can be) accomplished. The Leverage stage includes documenting the results of the improvement effort and leveraging the improvement across all applicable organization projects.

Project Software Measurement. During the Initiate stage, the project goals and measures for success are defined along with success criteria. A project Software Measurement Plan should be developed or included as part of the software project management information (e.g., referenced as an appendix to a Software Development Plan). The Diagnose stage includes documenting and analyzing the project's measures as a Measurement Case during the project life cycle in accordance with the measures in the Measurement Plan. The Establish stage provides focus on identifying specific project or organization improvements that might be accomplished. Prioritized improvement actions are determined and assigned to project or organization level, as appropriate. For more mature organizations, project teams can accomplish the improvements during the project. For less mature organizations, the identified improvements will serve as a lessons learned for future projects. Action teams are formed (by the project or organization) and a plan developed to address the high priority improvements. The Act and Leverage stages for the project are limited to making mid-course project corrections based on the measurement information. Such measurement data and the actions taken are recorded in the Measurement Case. The project's Measurement Case then becomes the complete documentation of the project management and engineering measures, any changes to project direction based on measurement analysis, and lessons learned for future projects.

4.2.2 SEI CMM

The SEI CMM serves as a guide for determining what to measure first and how to plan an increasingly comprehensive improvement program. The measures suggested for different levels of the CMM are illustrated in Table 4-1. The set of core measures described in this document primarily address Level 1, 2 and 3 issues.

Table 4-1. Relationship of Software Measures to Process Maturity

Maturity Level	Measurement Focus	Applicable Core Measures
1	Establish baselines for planning and estimating project resources and tasks	Effort, Schedule Progress (Pilot or selected projects)
2	Track and control project resources and tasks	Effort, Schedule Progress (Project by project basis)
3	Define and quantify products and processes within and across projects	Products: Size, Defects Processes: Effort, Schedule (Compare above across projects)
4	Define, quantify, and control subprocesses and elements	Set upper and lower statistical control boundaries for core measures. Use estimated vs actual comparisons for projects and compare across projects.
5	Dynamically optimize at the project level and improve across projects	Use statistical control results dynamically within the project to adjust processes and products for improved success.

Level 1 measures provide baselines for comparison as an organization seeks to start improving. Measurement occurs at a project level without good organization control, or perhaps on a pilot project with better controls.

Level 2 measures focus on project planning and tracking. Applicable core measures are the staff effort and schedule progress. Size and defect data are necessary to understand measurement needs for level 3 and level 4 and to provide a data base for future evaluations. Individual projects can use the measurement data to set process entry and exit criteria.

Level 3 measures become increasingly directed toward measuring and comparing the intermediate and final products produced across multiple projects. The measurement data for all core measures are collected for each project and compared to organization project standards.

Level 4 measures capture characteristics of the development process to allow control of the individual activities of the process. This is usually done through techniques such as statistical process control where upper and lower bounds are set for all core measures (and any useful derived measures). Actual measure deviation from the estimated values is tracked to determine whether the attributes being measured are within the statistically allowed control bounds. A decision process is put into place to react to projects that do not meet the statistical control boundaries. Process improvements can be identified based on the decision process.

Level 5 processes are mature enough and managed carefully enough that the statistical control process measurements from level 4 provide immediate feedback to individual projects based on an integrated decisions across multiple projects.

Decisions concerning dynamically changing processes across multiple projects can then be optimized while the projects are being conducted.

4.3 Identify a Goal-Question-Metric (GQM) Structure

One of the organization's or project's most difficult tasks is to decide what to measure. The key is to relate any measurement to organization and project goals. One method for doing this is to use Basili's Goal-Question-Metric (GQM) paradigm described in [BASILI] and illustrated in Figure 4-4 with a partial example related to software reliability.

This method links software goals to corporate goals and derives the specific software measures that provide evidence of whether the goals are met. Since such measures are linked directly to organization goals, it is much easier to show the value of the measurement activity and establish success criteria for measurement.

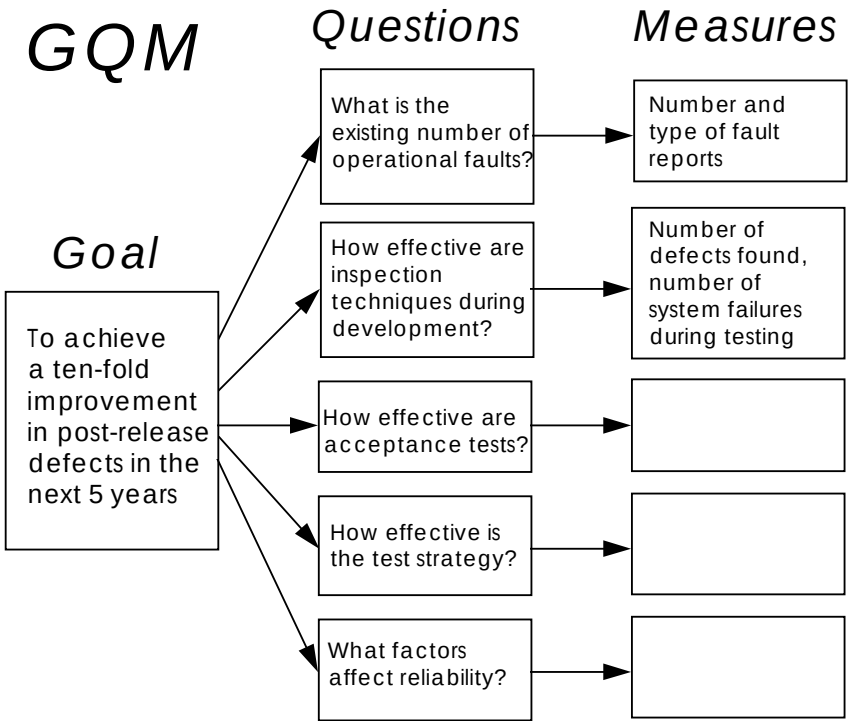

Figure 4-4. Basili's Goal-Question-Metric (GQM) Paradigm

The GQM method to software measurement uses a top down approach with the following steps:

1. Determine the goals of the organization and/or project in terms of what is wanted, who wants it, why it is wanted , and when it is wanted.
2. Refine the goals into a set of questions that require quantifiable answers.
3. Refine the questions into a set of measurable attributes (measures for data collection) that attempt to answer the question.

4. Develop models relating each goal to its associated set of measurable attributes.

Some attributes of software development, such as productivity, are dependent on many factors that are specific to a particular environment. The GQM method does not rely on any standard measures and the method can cope with any environment.

This activity may be conducted concurrently with any other software measurement activities and may be used to iteratively refine the software measurement program model, core measurement views, and process improvement efforts.

4.4 Develop a Software Measurement Plan and Case

The software measurement program activities provide organization and project-specific planning information and a variety of measurement data and analysis results. These plans, data, and results should be documented through use of a software measurement plan and software measurement case as illustrated in Figure 4-5.

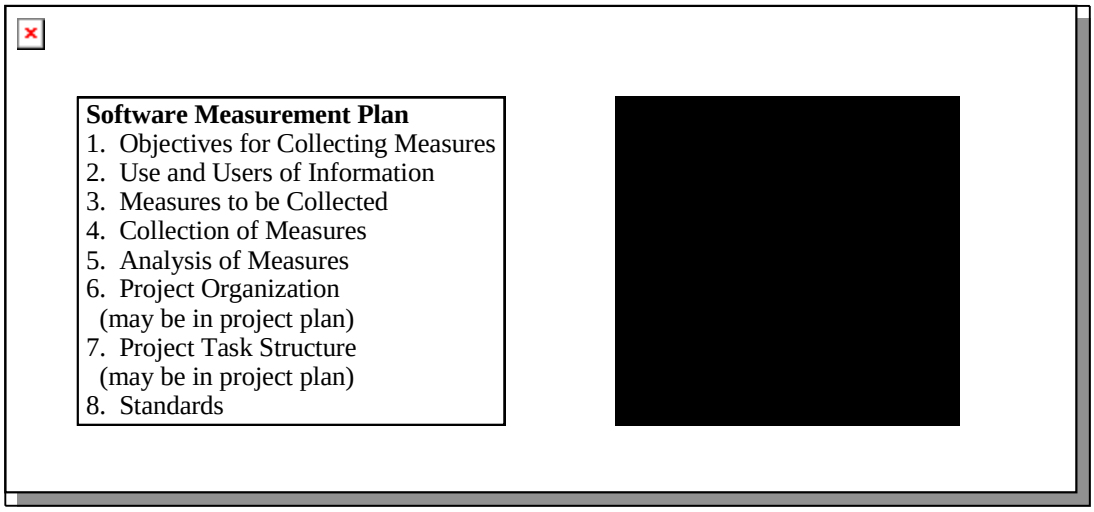

Figure 4-5. Software Measurement Plan and Case

A software measurement plan defines:

- what measurement data are to be collected;
- how the data are to be analyzed to provide the desired measures; and
- the representation forms that will describe the measurement results.

Such a plan also provides information as to who is responsible for the measurement activities and when the measurement activities are to be conducted. A software measurement plan should be developed at an organization level to direct all measurement activity and at a project level to direct specific project activity. In most cases a project's software measurement plan can be a simple tailoring of the organizational plan. The organization's software measurement plan can be a separate document or might be an integrated part of the organization's Software Management Plan or Software Quality Plan. An example of an organization measurement plan standard and plan outline is provided in Appendix C.

A software measurement plan at either the organization or project level should relate goals to specific measures of the resource, process, and product attributes that are to be measured. The GQM method can be used to identify such measures. Improvement in accordance with the SEI CMM key process areas should be an integrated part of the derivation. The identified measures may be a core measure or derived from one or more core measures. The integration of core measures with process improvement and GQM is illustrated in Figure 4-6.

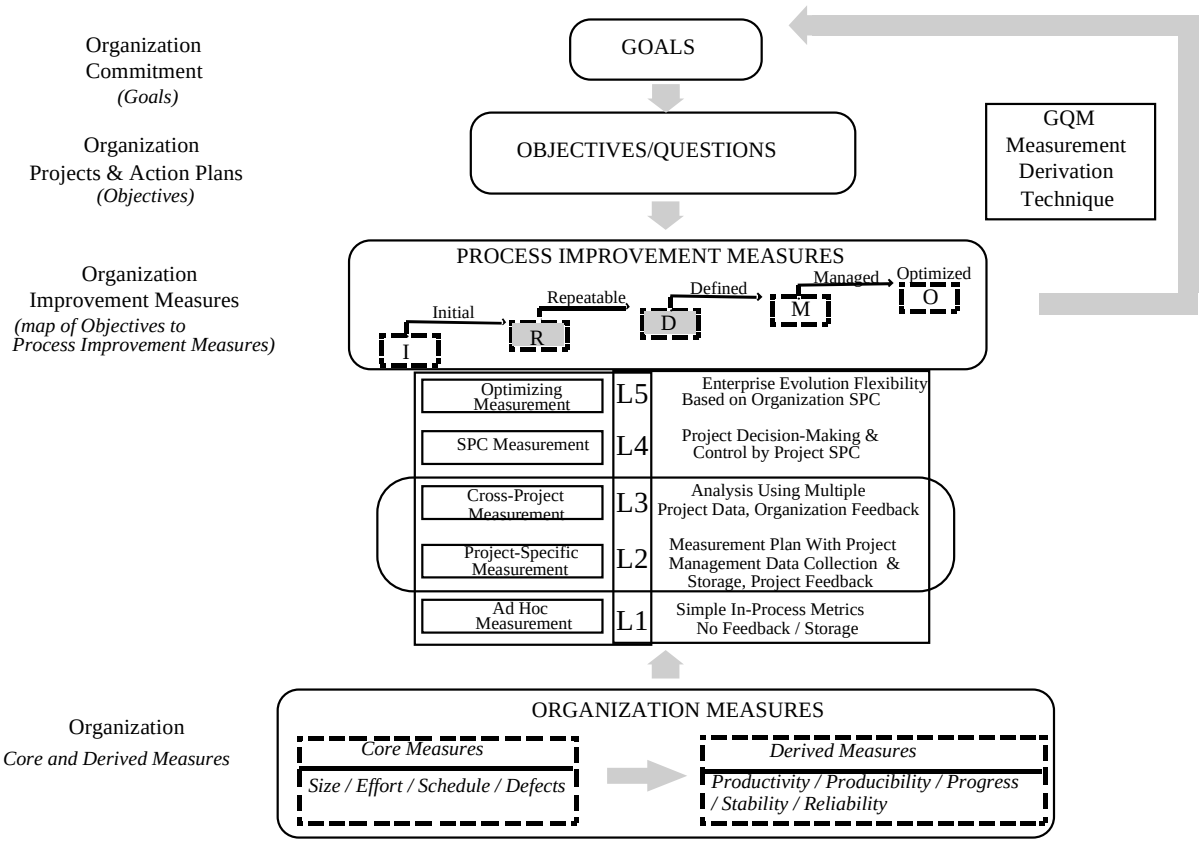

Figure 4-6. Integration of Software Measurement and Process Improvement

The following activities are key to developing a software measurement plan:

1. **Establish Program Commitment.** Define why the program is needed, obtain management approval, and identify ownership.
2. **Determine Goals and Expected Results.** Use software process assessment results to set the improvement context.
3. **Select Project Measurements.** Apply the GQM method to derive project measures.
4. **Develop Measurement Plan.** Document the measures to be collected, data collection, analysis and presentation methods, and relationship to an overall improvement program.

The software measurement case documents the actual data, analysis results, lessons learned, and presentations of information identified in an associated software measurement plan. The following activities are key to developing a Software Measurement Case:

1. **Implement Measurement Plan.** Collect and analyze data, provide project feedback, and modify project/program as necessary.
2. **Analyze Measurement Results.** Store project measurement results, analyze results against historical project results.
3. **Provide Measurement Feedback.** Report results of analysis as project lessons learned, update measurement and process improvement programs, and repeat the process of developing/updating a Measurement Plan and Case.

5. SUMMARY OF RECOMMENDATIONS

Specific software measurement actions on individual projects and within organizations will depend on existing software capability and initiatives. The following recommendations summarize the guidelines in this document.

5.1 Software Measurement Program

Adopt a measurement model appropriate to the organization. Identify core measures of product, process, and resource attributes as a baseline model. Integrate measurement as a part of a process improvement program. Baseline current process and measurement practices using a model such as the SEI CMM. Initiate process improvement activities following a model such as the SEI IDEAL. Use the Goal-Question-Metric approach to link organization goals to software measures. Use the CMM and the core measures to link the software measures to process improvement. Develop organization and project measurement plans and document measurement evidence as standard activities. Use the measurement evidence to influence organization and project decision-making.

5.2 Core Measures

Define and collect the four core measures of size, effort, progress to schedule, and defects for all projects.

Size. Some of the more popular and effective measures of software size are physical source lines of code (noncomment, nonblank source statements); logical source statement (instructions); function points (or feature points); and counts of logical functions or computer software units (i.e., modules). Size measurements can be used to track the status of code from each production process and to capture important trends. It is recommended that projects adopt physical source lines of code or function points as the principal measure for size.

Effort. Reliable measures for effort are prerequisites to dependable measures of software cost. By tracking human resources assigned to individual tasks and activities, effort measures provide the principal means for managing and controlling costs and schedules. It is recommended that projects adopt staff hours as the principal measure for effort.

Progress to Schedule. Schedule and progress are primary project management concerns. Accordingly, it is important for managers to monitor adherence to intermediate milestone dates. Early schedule slips often foreshadow future problems. It is also important to have objective and timely measures of progress that accurately indicate status and that can be used to project completion dates for future milestones.

At a minimum, the following information should be planned for and tracked:

- Major milestone completion progress -- estimates and actuals: requirements, design, implementation, test, delivery;

- Intermediate milestone completion progress -- estimates and actuals: modules coded, modules integrated;
- Estimated size progress -- estimates and actuals by date completed; and,
- Exit or completion criteria associated with each milestone date.

Defects. The number of problems and defects associated with a software product varies inversely with perceived quality. Counts of software problems and defects are among the few direct measures for software processes and products. These counts allow qualitative description of trends in detection and repair activities. They also allow the tracking of progress in identifying and fixing process and product imperfections. In addition, problem and defect measures are the basis for quantifying other software quality attributes such as reliability, correctness, completeness, efficiency, and usability.

5.3 Automated Methods

In order to make the software measurement program as efficient as possible, it is recommended to establish automated mechanisms for measurement data collection and analysis. Automated methods should be a support resource of the measurement process rather than a definition of the process. Regularly collect the core measurements and additional measurements specific to the local goals in your organization.

APPENDIX A: DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

A.1 Definitions and Terminology

Generic software terminology is defined in [IEEE610] and [SQAS-GLOSSARY]. Some terminology that is specific to this document is derived from [FENTON]. Customers of this document include:

Software Manager: The software manager is responsible for ensuring the project is on schedule, is within budget, and has the requisite quality. The software manager is concerned with in-process measures that can provide early warning information. The software manager is responsible for the measurement planning information and for ensuring the measurement evidence is collected, analyzed, used in project decisions, and documented for lessons learned.

Software Engineer: The software engineer is responsible for collecting process and product measurement data, analyzing the data to determine the impact on quality, and ensuring the measurement data and analysis results are properly communicated to the software quality engineer and software manager. The software engineer uses software measurement data to determine whether process entry and exit criteria have been achieved.

Software Quality Engineer: The software quality engineer is responsible for ensuring quality is engineered into the engineering processes and resulting software products as acquired, developed, supported, or used by an organization or project. The software quality engineer may be part of a software organization or part of an independent quality assurance group, and should provide expertise in establishing a software measurement program. Measurement data provide in-process assurance and qualification evidence that requisite software quality has been attained.

Software Process Improvement Group: The software process improvement group is responsible for improving the software processes and resulting software products at an organization level. Improvement requires a measurement baseline and a continuous measurement program applied to organization projects to determine progress toward improvement goals.

Software Customer: The software customer is any organization that receives a software product from a supplier for the purpose of qualification review, acceptance review, or operational use. The customer may be a Nuclear Weapons Complex organization or site, a Department of Energy organization or site, or an organization such as the Department of Defense. The customer is primarily interested in the cost of acquiring the software, schedule time to achieve an operational software capability, software product defects that may be in the delivered software product, and on-going product support. The customer is responsible for the feedback of defects found during operational use.

A.2 Acronyms and Abbreviations

CMM	Capability Maturity Model
COCOMO	Con\structive COst MOdel
DOE	Department of Energy
GQM	Goal-Question-Metric
IDEAL	Identify, Diagnose, Establish, Act, Leverage
IEEE	Institute of Electrical and Electronics Engineers
NWC	Nuclear Weapons Complex
SEI	Software Engineering Institute
SLOC	Source Lines of Code
SQAS	Software Quality Assurance Subcommittee
WBS	Work Breakdown Structure

APPENDIX B: REFERENCES AND BIBLIOGRAPHY

[BASILI] Basili, V. and Weiss, D.M., "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, vol. SE-10, No. 6, November 1984, pp. 728-738.

[BOEHM] Boehm, B. W., *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[BROOKS] Brooks, F. P., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading, MA, 1975.

[DEMING] Deming, W. E., *Out of the Crisis*, MIT Press, Cambridge, MA, 1986.

[DODCORE] Carleton, A.D., Park, R.E., Goethert, W.B., Florac, W.A., Bailey, E.K., Pfleeger, S.L., *Software Measurement for DoD Systems: Recommendations for Initial Core Measures* (CMU/SEI-92-TR-19), Software Engineering Institute, Pittsburgh, PA, 1992.

[DOESEM] *Software Engineering Methodology*, U.S. Department of Energy, March 1996.

[DREGER] Dreger, J. Brian, *Function Point Analysis*, Prentice Hall Advanced Reference Series, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[FENTON] Fenton, N. E., *Software Metrics A Rigorous Approach*, Chapman & Hall, London, England, 1991.

[GRADY87] Grady, R. B. and Caswell, D.L., *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[GRADY92] Grady, R. B., *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[IEEEstds] IEEE Software Engineering Standards Collection: 1994 Edition, Wiley-Interscience, New York, NY, 1994.

[IEEE610] IEEE Std-610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology (ANSI)," IEEE Software Engineering Standards Collection: Spring 1991 Edition, Wiley-Interscience, New York, NY, 1991.

[IEEE982] IEEE-Std-982.1-1988, "Standard Dictionary of Measures to Produce Reliable Software," Institute of Electrical and Electronic Engineers, Inc., Washington, DC, 1988.

[IFPUG] International Function Point Users Group Standard, *Function Point Counting Practices Manual*, Release 4.0, 1994.

[ISO9000-3] ISO 9000-3:1991, "Quality Management and Quality Assurance Standards - Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software," International Standards Organization (ISO), 1991.

[KAN] Kan, S. H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley Publishing Company, Reading MA, 1995.

[KITSON] Kitson, D. and Masters, S., *An Analysis of SEI Software Process Assessment Results* (CMU/SEI-92-TR-24), Software Engineering Institute, Pittsburgh, PA, July 1992.

[MUSA] Musa, J. D., Ianinno, A. and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, McGraw Hill, New York, NY, 1987.

[NASA] NASA-GB-001-94, *Software Measurement Guidebook*, National Aeronautics and Space Administration, August 1995.

[PSM] *Practical Software Measurement*, Joint Logistics Commanders joint Group on Systems Engineering, Version 2.1, March 1996.

[P1045] IEEE-P1045/D5.0, “Standard for Software Productivity Metrics (draft),” Institute of Electrical and Electronic Engineers, Inc., Washington, DC, 1992.

[P1061] IEEE-P1061, “Standard for Software Quality Metrics Methodology,” Institute of Electrical and Electronic Engineers, Inc., New York, NY, 1990.

[SEICMM] Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V., *Capability Maturity Model for Software, Version 1.1* (CMU/SEI-93-TR-024), Software Engineering Institute, Pittsburgh, PA, 1993.

[SEIIDEAL] McFeeley, B., *IDEAL: A User’s Guide for Software Process Improvement* (CMU/SEI-96-HB-001), Software Engineering Institute, Pittsburgh, PA, February 1996.

[SEITR20] Park, R.E., *Software Size Measurement: A Framework for Counting Source Statements* (CMU/SEI-92-TR-20), Software Engineering Institute, Pittsburgh, PA, July 1992.

[SEITR21] Goethert et al., *Software Effort & Schedule Measurement: A Framework for Counting Staff hours and Reporting Schedule Information* (CMU/SEI-92-TR-21), Software Engineering Institute, Pittsburgh, PA, July 1992.

[SEITR22] Florac, W.A., *Software Quality Measurement: A Framework for Counting Problems and Defects* (CMU/SEI-92-TR-22), Software Engineering Institute, Pittsburgh, PA, 1992.

[SELLERS] Henderson-Sellers, B., *Object-Oriented Metrics: Measures of Complexity,* Prentice Hall, Upper Saddle River, NJ, 1996.

[SEMATECH] *Software Measurement for Semiconductor Manufacturing Equipment*, Technology Transfer #95012684A-TR, SEMATECH, March 31, 1995.

[SQAS-GLOSS] SQAS-90-001, "NWC Glossary of Preferred Software Engineering Terminology," Software Quality Assurance Subcommittee, October 1990.

[SQAS-PRACT] SQAS93-003, “Preferred Practices for Software Quality within the Nuclear Weapons Complex,” June 1993.

[SQAS-SPA] SQAS-95-001, "Planning for a Software Process Assessment," Software Quality Assurance Subcommittee, May 1995.

[SQAS-SURVEY] SQAS90-002, "Software Within the NWC: 1989 Software Engineering Survey," Software Quality Assurance Subcommittee, October 1990.

Blank Page

APPENDIX C: EXAMPLE MEASUREMENT PLAN STANDARD

Abstract

This document contains an example of a standard defining the contents and structure of a Software Measurement Plan for each project of an organization. The term Measurement Plan will be used throughout.

Table of Contents

C.1 Introduction

This standard provides guidance on the production of a Measurement Plan for individual software projects.

C.1.1 Scope

This standard is mandatory for all projects. Assistance in applying it to existing projects will be given by the Organization Measures Coordinator.

C.2 Policy

It is policy to collect measures to assist in the improvement of:

- the accuracy of cost estimates;
- project productivity;
- product quality; and,
- project monitoring and control.

In particular, each project will be responsible for identifying and planning all activities associated with the collection of these measures. The project is responsible for the definition of the project's objectives for collecting measures, analyzing the measures to provide the required presentation results, and documenting the approach in an internally approved Measurement Plan. The project is also responsible for capturing the actual measurement information and analysis results. The form of this actual measurement information could be appended to the Measurement Plan or put in a separate document called a Measurement Case.

C.3 Responsibility and Authorities

The Project Leader/Manager shall be responsible for the production of the project Measurement Plan at the start of the project. Advice and assistance from the Organization Measures Coordinator shall be sought when needed.

The Measurement Plan shall be approved by the Project Leader/Manager (if not the author), Product Manager, Organization Measures Coordinator and Project Quality Manager.

C.4 General Information

C.4.1 Overview Of Project Measures Activities

The collection and use of measures must be defined and planned into a project during the start up phase. The haphazard collection of measures is more likely to result in the collection of a large amount of inconsistent data that will provide little useful information to the project management team, or for future projects.

The following activities shall be carried out at the start of the project:

- Define the project's objectives for collecting measures.
- Identify the users of the measures derived information, as well as any particular requirements they may have.
- Identify the measures to meet these objectives or provide the information. Most, if not all, of these should be defined at the Organization level.

- Define the project task structure, e.g., Work Breakdown Structure (WBS).
- Define when each measure is to be collected, in terms of the project task structure.
- Define how each measure is to be collected, in terms of pre-printed forms/ tools, who will collect it and where/ how it will be stored.
- Define how the data will be analyzed to provide the required information, including the specification of any necessary algorithms, and the frequency with which this will be done.
- Define the organization, including the information flow, within the project required to support the measures collection and analyses activities.
- Identify the standards and procedures to be used.
- Define which measures will be supplied to the Organization.

C.4.2 Purpose Of The Measurement Plan

The project's Measurement Plan is produced as one of the start up documents to record the project's objectives for measures collection and how it intends to carry out the program. The plan also:

- Ensures that activities pertinent to the collection of project measures are considered early in the project and are resolved in a clear and consistent manner.
- Ensures that project staff are aware of the measures activities and provides an easy reference to them.

The Measurement Plan complements the project's Quality and Project Plans, highlighting matters specifically relating to measures. The Measurement Plan information can be incorporated into the Quality and/or Project Plans. Information and instructions shall not be duplicated in these plans.

C.4.3 Format

Section 5 defines a format for the Measurement Plan in terms of a set of headings that are to be used, and the information required to be given under each heading. The front pages shall be the minimum requirements for a standard configurable document.

C.4.4 Document Control

The Measurement Plan shall be controlled as a configurable document.

C.4.5 Filing

The Measurement Plan shall be held in the project filing system.

C.4.6 Updating

The Measurement Plan may require updating during the course of the project. Updates shall follow any changes in requirements for collecting measures or any change to the project which results in change to the project WBS. The Project Leader/Manager shall be responsible for such updates or revisions.

C.5 Contents Of Measurement Plan

This section details what is to be included in the project's Measurement Plan. Wherever possible, the Measurement Plan should point to existing Organization standards, etc.,

rather than duplicating the information. The information required in the Plan is detailed below under appropriate headings.

For small projects, the amount of information supplied under each topic may amount to only a paragraph or so and may not justify the production of the Measurement Plan as a separate document. Instead, the information may form a separate chapter in the Quality plan, with the topic headings forming the sections/ paragraphs in that chapter. On larger projects a separate document will be produced, with each topic heading becoming a section in its own right.

Thematic Outline for a Measurement Plan

Section 1. Objectives For Collecting Measures

The project's objectives for collecting measures shall be described here. These will also include the relevant Organization objectives. Where the author of the Measurement Plan is not the Project Leader/Manager, Project Management agreement to these objectives will be demonstrated by the fact that the Project Manager is a signatory to the Plan.

Section 2. Use and Users Of Information

Provide information that includes:

- Who will be the users of the information to be derived from the measures.
- Why the information is needed.
- Required frequency of the information.

Section 3. Measures To Be Collected

This section describes the measures to be collected by the project. As far as possible the measures to be collected should be a derivative of the Core Measures. If Organization standards are not followed, justification for the deviation should be provided. Project specific measures shall be defined in full here in terms of the project tasks.

A Goal-Question-Metric (GQM) approach should be used to identify the measures from the stated project objectives. The results of the GQM approach should also be documented.

Section 4. Collection of Measures

Provide information that includes:

- Who will collect each measure.
- The level within the project task against which each measure is to be collected.
- When each measure is to be collected in terms of initial estimate, re-estimates and actual measurement.
- How the measures are to be collected, with reference to proformas, tools and procedures as appropriate.
- Validation to be carried out, including details of the project specific techniques if necessary, and by whom.
- How and where the measures are to be stored - including details of electronic database/ spreadsheet/ filing cabinet as appropriate, how the data is amalgamated and when it is archived, who is responsible for setting up the storage process, who is responsible for inserting the data into the database.
- When, how and which data is provided to the Organization Measures database.

Thematic Outline for a Measurement Plan

(continued)

Section 5. Analysis Of Measures

Provide information that includes:

- How the data is to be analyzed, giving details of project specific techniques if necessary, any tools required and how frequently it is to be carried out.
- The information to be provided by the analysis.
- Who will carry out the analysis.
- Details of project specific reports, frequency of generation, how they are generated and by whom.

Section 6. Project Organization

Describe the organization within the project that is required to support the measurement activities. Identify roles and the associated tasks and responsibilities. These roles may be combined with other roles within the project to form complete jobs for individual people.

The information flow between these roles and the rest of the project should also be described.

Section 7. Project Task Structure

Describe or reference the project's the project task structure. It should be noted that the project's measurement activities should be included in the project task structure.

Section 8. Standards

Describe the measurement standards and procedures to be used by the project must be given, indicating which are Organization standards and which are project specific. These standards will have been referenced throughout the plan, as necessary. If it is intended not to follow any of the Organization standards in full, this must be clearly indicated in the relevant section of the Measurement Plan, and a note made in this section.

Blank Page

APPENDIX D: EXAMPLE PROJECT CORE MEASURES

This appendix provides examples, summarized in Table D-1, that illustrate the use of the recommended core measures (with some minor variations) for a variety of software projects.

Table D-1. Core Measures for Example Projects

Core Measures	Project A: Large Embedded Development	Project B: Commercial Purchase	Project C: Information System Development	Project D: Simulation Analysis Code Support	Project E: Graphical User Interface Small Development
Size	SLOC (reused & new)	Disk Space (utilized)	Function Points (reused & new)	SLOC (total, new & modified for each release)	Function Points (reused & new)
Effort	Staff Hours (development)	Staff Hours (installation & updates)	Staff Hours (development)	Staff Hours (total, change request for each release)	Staff Hours (development)
Progress to Schedule	Total Months (estimated & actual) Task Months (estimated & actual) Task Completion Ratio per reporting period	Installation Time (estimated & actual for initial release and updates)	Total Months (estimated & actual) Task Months (estimated & actual) Task Completion Ratio per reporting period	Total Months (estimated & actual for each release) Task Months (estimated & actual for each release) Task Completion Ratio per reporting period	Total Months (estimated & actual) Task Months (estimated & actual) Task Completion Ratio per reporting period
Defects	Inspection Defects (major & minor) Test Failures (major & minor) Operational Problem Reports (all)	Operational Failures (all) Operational Problem Reports (all)	Inspection Defects (major & minor) Test Failures (major & minor) Operational Problem Reports (all)	Inspection Defects (major & minor) Test Failures (major & minor total and in modified code) Operational Problem Reports (all and for modified code)	Test Failures (major & minor) Operational Problem Reports (all)

Blank Page