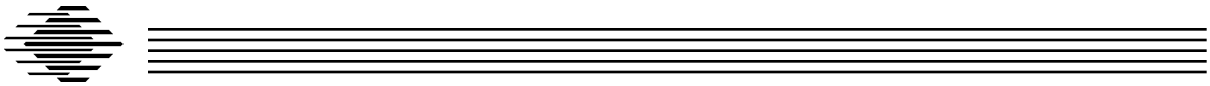
Goal-Driven Software Measurement —A Guidebook

Robert E. Park
Wolfhart B. Goethert
William A. Florac
August 1996

HANDBOOK
CMU/SEI-96-HB-002

Handbook
CMU/SEI-96-HB-002
August 1996

Goal-Driven Software Measurement —A Guidebook



Robert E. Park

Wolfhart B. Goethert

William A. Florac

Software Engineering Measurement and Analysis

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Suite C201, Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

…What measure is there of the relations of pleasure to pain other than excess and defect, which means that they become greater and smaller, and more and fewer, and differ in degree? For if anyone says: "Yes, Socrates, but immediate pleasure differs widely from future pleasure and pain"—to that I should reply: And do they differ in anything but pleasure and pain? There can be no other measure of them. And do you, like a skillful weigher, put into the balance the pleasures and the pains, and their nearness and distance, and weigh them, and then say which outweighs the other? If you weigh pleasures against pleasures, you of course take the more and greater; or if you weigh pains against pains, you take the fewer and less; or if pleasures against pains, then you choose that course of action in which the painful is exceeded by the pleasant, whether the distant by the near or the near by the distant; and you avoid that course of action in which the pleasant is exceeded by the painful. Would you admit, my friends, that this is true?…

— Plato[1]

---

[1] From *Protagoras*, in *The Dialogues of Plato*, translated by Benjamin Jowett, 4th ed., Vol. 1, pp. 183-184. Oxford: Clarendon Press, 1953.

Table of Contents

List of Figures

Preface

Software engineering has been defined as "the disciplined application of engineering, scientific, and mathematical principles, methods, and tools to the production of quality software" [Humphrey 89]. Its domain includes activities such as planning, estimating, modeling, designing, implementing, testing, maintaining, and managing. One's prospects for success in executing and improving these activities rise significantly when decisions can be based on factual, quantitative information—knowledge that can be obtained only by observing and measuring the products, processes, and resources involved. But one of the dangers in enterprises as complex as software development and support is that there are potentially so many things to measure that we are easily overwhelmed by the opportunities. For measurement to be cost effective, it must be designed and targeted to support the business goals of the organization.

The process that we describe in this guidebook will help you find and define software measures that directly support your organization's business goals. By ensuring traceability to well-identified goals, the activities that result will be better able to stay focused on their intended objectives.

We have chosen a tutorial style for describing the goal-driven process because it provides a good way for guiding people through the steps that we advocate. The risk here is that our classroom tone may suggest that the steps in the process are simply student exercises. They are not. Rather, they (and the techniques that we illustrate) are elements in a very practical process that can help you organize the efforts of your own process improvement teams—especially as they plan and initiate measurement activities.

We encourage you to use this handbook as a process guide. Your measurement planning teams should treat the exercises in Chapter 4 and Appendix A as assigned tasks and perform them in the order presented, with iterations where needed. If they do this, they (and you) will end up with clearly focused and well-defined measures that can be implemented and applied consistently by everyone in your software organization.

Acknowledgments

The following people have contributed to this work, either as reviewers of earlier drafts or as members of the team that generated the ideas and charts we use in the SEI course: Engineering an Effective Software Measurement Program. We thank them for helping us make this guidebook (and our course materials) as readable and useful as possible.

Steven Burke
Computer Sciences Corporation

Anita Carleton
Software Engineering Institute

Suzanne Couturiaux
Software Engineering Institute

Gary Ford
Software Engineering Institute

Will Hayes
Software Engineering Institute

Jim Herbsleb
Software Engineering Institute

Andy Huber
Data General

Nancy Mead
Software Engineering Institute

Jim Rozum
Software Engineering Institute

Bob Stoddard
Texas Instruments

David Zubrow
Software Engineering Institute

Michael Zuccher
Software Engineering Institute

Goal-Driven Software Measurement—A Guidebook

1 Introduction

> *The business of pinning numbers on things—which is what we mean by measurement—has become a pandemic activity in modern science and human affairs. The attitude seems to be: if it exists, measure it. Impelled by this spirit, we have taken the measure of many things formerly considered to lie beyond the bounds of quantification. In the process we have scandalized the conservatives, created occasional chaos, and stirred a ferment that holds rich promise for the better ordering of knowledge.*
>
> *— S. S. Stevens, 1959*

1.1 Purpose

This guidebook shows you how to identify and define software measures to support your own organization's business goals. The process that we illustrate produces measures that provide insights into the management issues that are most important to you. These measures are traceable back to your business goals, so that your data-collection activities are better able to stay focused on their intended objectives.

We call this process goal-driven measurement. In goal-driven measurement, the primary question is not "What metrics should I use?", but "What do I want to know or learn?" [Rombach 89]. Because the answers depend on your goals, no fixed set of measures is universally appropriate. So instead of attempting to develop generic, all-purpose lists of questionably useful measures, we have prepared this guidebook to describe an adaptable process that teams and individuals can use to identify and define measures that provide insights into their own management issues.

Our intended audiences are program managers, project managers, process managers, process improvement teams, and measurement teams. If you are among the people who manage, measure, or improve software activities, the methods in Chapter 4 and the exercises in Appendix A can help you identify what you should be measuring and understanding in order to make your own organizations and processes successful.

1.2 Outline

This chapter explains the purpose of the guidebook and identifies the intended audience.

Chapter 2 reviews some of the foundations of software measurement. It introduces important terms and concepts that are used in Chapters 3 and 4.

Chapter 3 gives an overview of the goal-driven measurement process. It introduces the concept of mental models and illustrates the roles that mental models play in providing insights and focus for the process steps that follow.

Chapter 4 contains the heart of the guidebook. The materials in this chapter are presented as a sequence of tutorials, each supported by examples, exercises, and worksheets. Many of the materials were developed originally for a three-day course which we teach at the Software Engineering Institute and for sponsoring organizations [SEI 96]. The order of presentation in this guidebook follows that of our classroom delivery.

Chapter 5 briefly summarizes some important recommendations that we have collected from organizations that have implemented software measurement activities.

Chapter 6 closes the loop by relating the goal-driven measurement process back to elements that are inherent in the basic structure of any business management process.

Appendices A and B contain instructions, worksheets, and forms that you can reproduce and use as you and your teams plan and execute your own measurement activities.

2 Foundations

2.1 Why Measure?

> *Apparently—all other things being equal—it is better to measure than not to measure.*
>
> *— C. West Churchman, 1959*

> *The only sustainable competitive advantage you can achieve is to learn faster than your competitors.*
>
> *— David Kreutzer, 1995*

> *Ignorance is a voluntary misfortune.*
>
> *— Nicholas Ling*

There are four reasons for measuring software processes, products, and resources:

- to characterize
- to evaluate
- to predict
- to improve

We *characterize* to gain understanding of processes, products, resources, and environments, and to establish baselines for comparisons with future assessments.

We *evaluate* to determine status with respect to plans. Measures are the sensors that let us know when our projects and processes are drifting off track, so that we can bring them back under control. We also evaluate to assess achievement of quality goals and to assess the impacts of technology and process improvements on products and processes.

We *predict* so that we can plan. Measuring for prediction involves gaining understandings of relationships among processes and products and building models of these relationships, so that the values we observe for some attributes can be used to predict others. We do this because we want to establish achievable goals for cost, schedule, and quality—so that appropriate resources can be applied. Predictive measures are also the basis for extrapolating trends, so estimates for cost, time, and quality can be updated based on current evidence. Projections and estimates based on historical data also help us analyze risks and make design/cost tradeoffs.

We measure *to improve* when we gather quantitative information to help us identify roadblocks, root causes, inefficiencies, and other opportunities for improving product quality and process performance. Measures also help us plan and track improvement efforts. Measures of current performance give us baselines to compare against, so that we can judge whether or not our improvement actions are working as intended and what the side effects may be. Good measures also help us communicate goals and convey reasons for improving. This helps engage and focus the support of those who work within our processes to make them successful.

2.2 Measurement Elements

> *Quantities are measurements of qualities.*
>
> *— Paul Kirchner*

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterize the attributes by clearly defined rules [Fenton 95]. Thus, measurement requires

- entities (objects of interest)
- attributes (characteristics of entities)
- rules (and scales) for assigning values to the attributes

In general, the class or amount of an attribute is what we measure.

This means that, before we can measure, we must clearly identify the entities and attributes we will address and the rules we will use to assign values to the attributes.

Entities and Attributes

There are several kinds of *entities* that we may wish to describe. Examples include

products	artifacts	organizations
processes	activities	environments
resources	agents	constraints

Entities can also be sets or collections of other entities. For example, a software process may contain many subprocesses and flowpaths, each producing, transforming, or transmitting products and by-products. The individual products, by-products, subprocesses, flowpaths, and data elements within these entities are themselves entities that organizations may want to characterize in consistent, well-understood ways. Similarly, a historical database (an entity) may contain many measurement results (other entities), together with their associated descriptions and definitions.

Attributes are characteristics or properties of entities. Just as a person (entity) can be described by characteristics such as height, color of eyes, sex, IQ, age, and years of experience (attributes), so can software entities be described by attributes such as size, cost, elapsed time, effort expended, response time, transaction rates, number of defects found, and operational reliability. The art of measurement lies in deciding which attributes to use to give useful pictures of the entities we deal with.

Some authors have proposed taxonomies for classifying software entities. Fenton, for example, says that an entity is either a product, a process, or a resource [Fenton 91]. Armitage et al, on the other hand, use a classification scheme based on artifacts, activities, and agents [Armitage 94]. Both schemes have advantages and disadvantages. Neither seems to deal naturally with low-level entities such as defects. (Is a defect a product, a process, or a resource?…An artifact, activity, or agent?) And both taxonomies seem awkward when addressing environmental elements.

Fortunately, we do not have to resolve the ambiguities and clashes here. In goal-driven measurement, we do not need to ensure that entities are assigned to proper classes. So it matters little which structure is best—or even correct. We use Fenton's and Armitage's taxonomies interchangeably, primarily as prompters, to help us identify elements and attributes that we can study, manage, or improve. In practice, we find that using more than one taxonomy often helps give additional insights.

Figures 2-1 through 2-3 show some examples of entities that software organizations produce, use, or manage. Each entity is accompanied by attributes that characterize the entity and measures that could be used to quantify the attributes. These lists could easily be expanded, but we suggest waiting until we have introduced a framework for identifying the specific business and measurement goals that are important to your own organization. Otherwise it is easy to become overwhelmed by the opportunities.

Resource Entities	**Attributes**	**Possible Measures**
assigned staff	team size	number of people assigned
	experience	years of domain experience
		years of programming experience
CASE tools	type	name of type
	is_used?	yes/no (a binary classification)
time	start date, due date	calendar dates
	elapsed time	days

Figure 2-1: Examples of Resource Measures

Product Entities	Attributes	Possible Measures
system	size	number of modules
		number of bubbles in a data-flow diagram
		number of function points
		number of physical source lines of code
		number of memory bytes or words required (or allocated)
	defect density	defects per KSLOC
		defects per function point
module	length	physical source lines of code
		logical source statements
	percent reused	ratio of unchanged physical lines to total physical lines, comments and blanks excluded
unit	number of linearly independent flowpaths	McCabe complexity
document	length	number of pages
line of code	statement type	type names
	how produced	name of production method
	programming language	language name
defect	type	type names
	origin	name of activity where introduced
	severity	an ordered set of severity classes
	effort to fix	staff-hours
	age (of open defects)	elapsed time (days) since receipt of defect report

Figure 2-2: Examples of Product Measures

Process Entities	Attributes	Possible Measures
development process	elapsed time	calendar days working days
	milestones	calendar dates
	development effort	staff-hours, days, or months
	phase containment	percent of total defects found in phase where introduced
	process compliance	percent of tasks complying with standard procedures or directives
	performance	number of tests passed divided by number of tests executed
detailed designing	elapsed time	calendar days working days
	design quality	defect density: number of design defects found in down-stream activities divided by a measure of product size, such as function points or physical source lines of code.
testing	volume	number of tests scheduled
	progress	number of tests executed number of tests passed
maintenance	cost	dollars per year staff-hours per change request
change request backlog	size	number of change requests awaiting service estimated effort (staff-hours) for pending requests

Figure 2-3: Examples of Process Measures

Measurement Scales

The discussion on scales that follows is somewhat more detailed and theoretically oriented than the guidelines and examples in the rest of this guidebook. It can be skipped at first reading. Although it is important to know how scales can affect (and sometimes limit) the things we can legitimately do with measurement results, you need not let this topic sidetrack you now. Be sure to come back and read it later, though.

Scales provide values and units for describing attributes. For example, a person's height may be 68 inches, his weight may be 163 pounds, his eyes may be brown, and his disposition may be aggressive. Similarly, a software project may produce 39,000 lines of code, have a planned completion date of 30 August, use 11,243 staff-hours of effort, and have an application type classified as real-time command and control. Each of these observations has been quantified (or labeled) with a value from a (presumably) well-defined scale.

As we shall see, scales for assigning values to attributes need not always be quantitative, nor are subjectively determined values necessarily undesirable. But wherever measurement occurs, and whatever its form, it always requires well-defined scales for capturing and recording measured results.

Measurement scales are derived from the rules that we use for assigning values to attributes. Different rules lead to different scales. Figure 2-4 shows a system for classifying measurement scales that is based on the transformations that can be made to a scale without changing the structure of the scale [Stevens 51, Stevens 59, Fenton 91].

Transformations that do not change the structure of a scale are called admissible transformations.[1] Admissible transformations limit the ways we can validly use measurement results. For example, statements and inferences based on data from measurement scales are meaningful if and only if their truth (or falsity) remains unchanged under all admissible transformations of all the scales involved. Thus the admissible transformations associated with each scale have important implications about what we can and cannot validly do when we use data to compute the values (statistics) that we use in charts, analyses, and management reports.

Figure 2-4 lists five types of scales—nominal, ordinal, interval, ratio, and absolute [Roberts 79]. The ordering is from the least restrictive admissible transformations to the most restrictive. The basic empirical operations associated with the scales then range inversely from the most restrictive to the least restrictive. These empirical operations are cumulative in the sense that measurements made with one of these scales may also be used as inputs to any of the empirical operations associated with any scale that precedes it in the list.

Other scales such as log-interval and difference scales are possible, but they have less practical use [Stevens 59, Krantz 71, Roberts 79, Luce 90].

---

[1] Transformations are mappings such as $y = ax$, $y = ax + b$, or (more generally) $y = f(x)$.

Scale Type	Admissible Transformations	Basic Empirical Operations	Examples
nominal	Any one-to-one transformation	determination of equality	labels or classifications such as programming language names (Ada, C, C++, Fortran, JOVIAL, CMS-2, Pascal) job functions (engineer, manager, programmer, QA person, customer support person) activities (analyzing, designing, coding, testing) customer IDs problem types numbering of football players
ordinal	$y_2 \geq y_1$ iff $x_2 \geq x_1$ (strictly monotone increasing transformation)	the above, plus determination of greater or less	rankings or orderings such as hardness of minerals intelligence scores (raw scores) severity and priority assignments CMM maturity levels subjective evaluations made with Likert scales or low-medium-high ratings street numbers
interval	$y = ax + b,\ a > 0$ (positive linear transformation)	the above, plus determination of the equality of intervals or differences	clock time calendar date temperature in degrees Fahrenheit or Celsius intelligence scores (“standard scores”)
ratio	$y = ax,\ a > 0$ (similarity transformation)	the above, plus determination of the equality of ratios	time intervals cost, effort (staff-hours), length, weight, & height temperature in degrees Kelvin
absolute	$y = x$ (identity)	the above, plus determination of equality with values obtained from other scales of the same type	counting probability

Figure 2-4: Measurement Scales

The following paragraphs describe the five scales and point out some of the limitations associated with using each scale.

Nominal: A nominal scale provides a name or label as the value for an attribute. The order of values on the scale has no significance. Familiar examples include the color of a person's hair (red, brown, black, blonde, etc.), numbers for football players (nominal values limited to one player per number), and identifying attributes such as part numbers, job codes, defect classes, language names, and statement types (nominal values where several entities can share a common label). Any one-to-one mapping is an admissible transformation.

Nominal measures are often used to classify entities so that they can be sorted prior to counting the number of occurrences or aggregating measured values. For example, we may want to know the number of *executable* source statements that we have in a software program. The process of assigning a statement to a class such as *executable*, as opposed to *data declaration* or *comment*, is a measurement made according to a nominal scale. Similarly, we may want to know the number of *direct* labor hours that were expended by our *quality assurance* people. Here the labels *direct* and *quality assurance* are values on nominal scales.

In these examples, the terms *executable*, *data declaration*, *comment*, *direct*, and *quality assurance* are (nominal) labels that we use to describe attributes like source statement types, labor classes, and personnel classes. The subsequent counting of occurrences (or summing of values) to obtain totals leads us beyond nominal measures to the use of either absolute or ratio scales. Ratio and absolute scales will be discussed shortly in the paragraphs that follow.

When analyzing nominal measures, we are usually limited to nonparametric or distribution-free statistics such as modes and frequency counts or the use of contingency coefficients and chi-square tests. Many computations that make sense for higher order scales serve only to produce results that have little meaning. For example, although we *could* compute the average number worn by players at a football game, the result has little practical significance. Similarly, average defect classes, average hair colors, and the standard deviations of part numbers or job codes are unlikely to have useful interpretations.

Ordinal: An ordinal scale permits measured results to be placed in ascending (or descending) order. However, distances between locations on the scale have no meaning. The Capability Maturity Model[sm] for Software (CMM[sm]), for instance, provides a 1-to-5 (integer-valued) ordinal scale for summarizing the results of process capability assessments and evaluations [Paulk 93a, Paulk 93b]. But there is no concept or distance associated with this scale, so we have no idea how far a CMM Level 4 rating is from Level 3. Nor is there

[sm] CMM and Capability Maturity Model are service marks of Carnegie Mellon University.

any implication that a Level 2 rating is twice as high as Level 1. Similar observations hold true for other ordinal measures such as defect severities and change-request priorities.

Ordinal scales remain ordinal scales when transformed by any monotonically increasing function (e.g., $y = \ln x$, $z = a + 5x^2$, etc.). The function does not have to be continuous, nor must it pass through the origin. Rank-order statistics, plus all statistics applicable to nominal scales, can be used with ordinal measures. But computing averages, such as, "The average CMM levels for these organizations is 2.3" and "The average severity level for design defects is 3.16," is inconsistent with the use of an ordinal scale. These kinds of computations can lead to misinterpretations and invalid conclusions.

As with nominal scales, valid analyses of ordinal data will usually require nonparametric or distribution-free methods. But the range of possibilities with ordinal data is somewhat wider. For example, because the scale is ordered, run tests and sign tests are now possible.

Interval: An interval scale adds the concept of distance. If the temperature reached 40ºF today and 20ºF yesterday, we can say that the difference is 20ºF—but we cannot say that it was twice as warm today as it was yesterday. These limitations exist because the Fahrenheit scale has no concept of an origin (true zero-value). Interval scales permit us to add and subtract values, and we can make statements such as, "The average daily peak temperature was 30ºF." Interval scales remain interval scales when multiplied by positive constants or translated laterally. Any transformation of the form $y = a + bx$, $(b > 0)$, is admissible. The permissible statistics are all the statistics that apply for ordinal scales, plus arithmetic means.

Clock times, calendar dates, and normalized intelligence scores are examples of frequently used measures from interval scales.

Ratio: A ratio scale adds an origin (a meaningful, nonarbitrary zero value). With a true origin, division and multiplication become meaningful, and all the mathematical operations that we customarily use for real numbers are legitimate. Ratio scales remain ratio scales when multiplied by any positive constant. Permissible statistics include ratios and percentages, as well as all statistics that apply to interval scales.

Examples of familiar measures that use ratio scales include personnel attributes such as height, weight, and age. Examples more directly related to software include development cost, integration cost, time between failures, and schedule length.

Absolute: Absolute scales are special cases of ratio scales in which the only admissible multiplier is 1. They often arise (in a discrete form) out of attributes that are simple counts of frequencies of outcomes measured on nominal or ordinal scales. For example, counting the number of occurrences in each of several nominal classes (e.g., the number of design defects, code defects, etc.) is an instance of the use of an absolute (counting) scale for the attribute "number of occurrences found." Likewise, the number of executable source statements and the number of programmers assigned are familiar software examples.

Counts are measures on absolute scales in the sense that once the counting rules are defined, there is one and only one way to count. Any count that is multiplied by a constant other than 1.0 or that has its origin shifted is no longer a count. Moreover, if N "things" are counted, the interpretation of N as a number of "things" remains the same for all counting scales, regardless of the "things" counted or the reasons for counting.

Permissible Statistics

As pointed out in the preceding paragraphs, the types of computations that are appropriate depend on the kinds of scales that we use. Figure 2-5, which is adapted from [Stevens 59], relates four of the scales to some of the statistical measures that may be appropriately used with each scale. As in Figure 2-4, this table is cumulative. A statistic applicable to any given scale type will be applicable to all scale types that follow it in the list. In general, the more restrictive the admissible transformations, the more unrestricted the statistics. Thus, nearly all statistics are applicable to measurements made on ratio scales, but only a very limited group of statistics may be applied to measurements made on nominal scales. The basic rule is this: Having measured a set of items by assigning values in accordance with a set of rules, we are free to change the assignments by any group of transformations that preserves the empirical information contained in the scale [Stevens 59].

Scale Type	Measures of Location	Measures of Dispersion	Measures of Association or Correlation	Significance Tests
nominal	mode	information (H)	information transmitted (T) contingency correlation	chi-square
ordinal	median	percentiles	rank-order correlation	sign test run test
interval	arithmetic mean	standard deviation average deviation	product-moment correlation	t test F test
ratio	geometric mean, harmonic mean	percent variation	correlation ratio	

Figure 2-5: Examples of Statistical Uses Appropriate to Measurements Made on Different Classes of Scales

The classifications in Figures 2-4 and 2-5 are based on ideas reported by S. S. Stevens in 1946 [Stevens 46, Stevens 51, Stevens 59]. The discussions in [Stevens 51], pp. 23–30, are perhaps the most instructive we have read. It is worth your while to have at least one person on your measurement team review these materials. Familiarity with the issues that Stevens discusses can help avoid pitfalls caused by overreliance on "intuitive common sense."

Zuse and Fenton also give brief descriptions of the scale types and relate them to formal measurement structures [Zuse 91, Fenton 91]. Krantz et al. and Roberts present even more penetrating discussions in their books on the foundations of measurement [Krantz 71, Roberts 79].

Stevens's scheme for classifying measurement scales has drawn criticism from several well-known statisticians [Velleman 93]. The major point that the statisticians make is that, when analyzing data, one should never use scale types to (prematurely) select or restrict statistical methods. The reason is that the nature of a scale will often change, depending on the questions that we ask and on additional information that may become available.[2] In short, scale types are not fundamental attributes of data. Rather, they derive from both how the data are obtained and what we conclude from the data. So, don't let your knowledge of scale types overly dominate the types of analyses you consider. Instead, use your knowledge of measurement scales to test the consistency of your reasoning and the validity of the conclusions you reach.

Despite the cautions that statisticians raise, they do agree that an understanding of measurement scales and the associated admissible transformations can help ferret out nonsense. In particular, conclusions reached will ultimately require that the data belong to one or another type of measurement scale. Once we have arrived at a conclusion, we should always check whether the measurement scales that are required for that conclusion to hold are consistent with the data that were used and the way they were collected.

---

[2] Velleman and Wilkinson [Velleman 93] provide several interesting examples. In one, a raffle was held at a conference based on tickets allotted at the door. A winning number, 126, was selected and announced. One participant compared it to her ticket to see if she had won, thus interpreting "126" correctly as a nominal value. Another, assuming that the tickets were issued sequentially (which they were), compared his ticket (number 56) to the winning number and realized that he had arrived too soon, thus interpreting the values ordinally. With data about the rate and regularity of arrivals, he might have tried to estimate how long he should have delayed his arrival so as to reduce the 70-ticket difference between his ticket number and the winner's, thus treating the numbers as interval scale values. The first person then looked around the room and observed, "It doesn't look like there are 126 people here." Here she was assuming that tickets were issued consecutively, beginning with number "1," and was interpreting the numbers as ratio scale values.

Scales Can Change as Knowledge Matures

Measurement can progress from lower to higher scales as societies, organizations, and practices mature. Stevens provides an illuminating example [Stevens 59]:

> *We can imagine, for example, that certain Eskimos might speak of temperature only as freezing or not freezing and, thereby, place it on a nominal scale. Others might try to express degrees of warmer and colder, perhaps in terms of some series of natural events, and thereby achieve an ordinal scale. As we all know, temperature became an interval scale with the development of thermometry, and, after thermodynamics had used the expansion ratio of gases to extrapolate to zero, it became a ratio scale."*
>
> *— S. S. Stevens, 1956*

There is an important lesson here for software engineers and managers—do not expect that everything you will want to measure can be expressed with ratio scales today. Software engineering is a young discipline. Just as with Stevens's Eskimos, it may take us time to evolve to where our measurement practices become comparable with those of other disciplines. Be willing to start with nominal and ordinal scales, just to get measurement started. But be mindful of the limitations of the computations and interpretations that you can make with the kinds of data you collect, and look for opportunities to evolve your measurement practices toward scales that provide greater information.

2.3 Objective and Subjective Measures

There is a tendency in some circles to say that all measurements must be objective. We emphatically disagree. Insisting on objectivity misses the point that objective and subjective measurements often address fundamentally different needs. Moreover, the real issues are not objectivity versus subjectivity, but consistency, repeatability, and the minimization of measurement errors and noise.

While it is admirable to strive for measurements that are as objective as possible, you should not hesitate to use subjective measurements when the information they provide helps you characterize, evaluate, predict, or improve your processes or products. Cost estimators, for example, have been doing this for years—especially when they use subjectively determined factors as inputs to parametric cost models like SLIM, SEER-SEM, PRICE S, or COCOMO. In fact, the inputs to these models—objective and subjective together—are often excellent vehicles for summarizing the contextual information that we must have to correctly interpret the results of software measurement activities.

When you do use subjective measurements, though, you should always strive to introduce processes that continually improve the consistency with which the measurements get made and recorded. Models, tools, training, tracking, trend analyses, and feedback are useful techniques that can help you achieve this goal.

3 A Process Model for Identifying and Defining Software Measures

> *Now, things do not, in general, run around with their measures stamped on them like the capacity of a freight-car: it requires a certain amount of investigation to discover what their measures are.*
>
> *— Norbert Wiener, 1920*

This chapter provides an overview and roadmap for the goal-driven measurement process that we describe in Chapter 4. The emphasis throughout goal-driven measurement is on gathering information that helps you achieve your business goals—and on maintaining traceability from measures back to business goals, so that measurement efforts do not wander astray.

The structure of the goal-driven measurement process draws extensively on ideas and experience reported by Victor Basili and Dieter Rombach [Basili 88, Basili 89, Rombach 89]. The process dynamics and several of the illustrations that we use have their origins in measurement process guidelines developed by the ami ESPRIT project [ami 92, Pulford 96]. The goal-driven measurement process also incorporates experience that we have gained at the Software Engineering Institute in designing and using checklist-based frameworks for defining software measures and communicating measurement results [Florac 92, Goethert 92, Park 92, SEI 96].

3.1 Overview: The Precepts and the Process Steps

The goal-driven measurement process is based on 3 precepts, and it consists of 10 steps. The three precepts are

- Measurement goals are derived from business goals.
- Evolving mental models provide context and focus.
- GQ(I)M[1] translates informal goals into executable measurement structures.

The 10 steps are

1. Identify your business goals.

[1] GQ(I)M is an acronym for goal-question-(indicator)-measure. The "I" in parentheses distinguishes this from the closely related GQM methodology introduced and described by Basili and Rombach [Basili 88, Basili 89, Rombach 89]. Our use of GQ(I)M is described in Sections 4.5–4.8.

2. Identify what you want to know or learn.
3. Identify your subgoals.
4. Identify the entities and attributes related to your subgoals.
5. Formalize your measurement goals.
6. Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
7. Identify the data elements that you will collect to construct the indicators that help answer your questions.
8. Define the measures to be used, and make these definitions operational.
9. Identify the actions that you will take to implement the measures.
10. Prepare a plan for implementing the measures.

Figure 3-1 shows a graphical view of the process model that guides our steps (the step numbers are circled). We will use excerpts from this model in Sections 1 through 10 of Chapter 4 to highlight our progress. We will also illustrate the importance of the precepts as we describe the process steps.

But before describing the process steps, it is useful to reflect briefly on the roles that mental models play in guiding our perceptions of the business issues that we manage when we develop and support software systems. These roles and the elements associated with mental models are discussed in the sections that follow. They will be illustrated in more detail when we walk through the process steps.

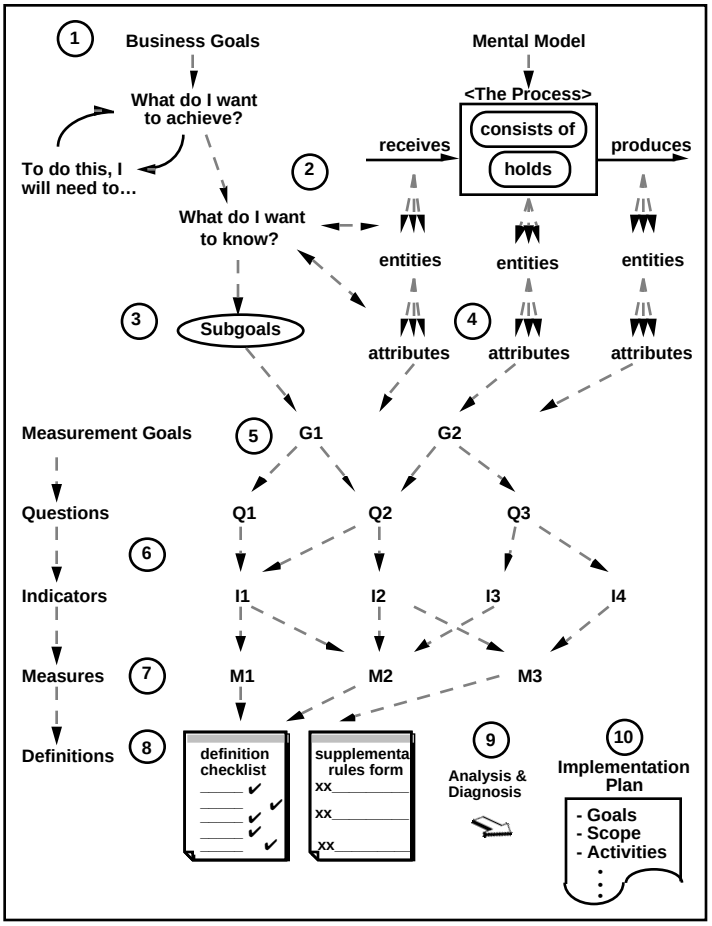


Figure 3-1: A Process Model for Selecting Software Measures

3.2 The Role of Mental Models

The driving forces in goal-driven measurement are the business goals of your organization and the information that you would like to have about your products, processes, and resources to help you meet these goals.

The primary mechanisms for translating these goals into issues, questions, and measures are the mental models that you have for the processes you use. These mental models gain substance and evolve as you begin to make them explicit. They are the engines that generate the insights that guide you to useful measures and actions.

Although mental models are abstractions, they are far from artificialities. We all use mental models every day to provide contexts and frameworks for translating observations into conclusions and actions. These personal mental models are seldom stated in explicit ways—but they always exist, if only in our minds. We derive these models from our personal experiences, and they exist even if we don't think consciously about them. They provide the contexts for interpreting and acting on the data we see in everyday life.

Although mental models are powerful tools, their full power will never be realized while they exist solely in your mind. Mental models must be made explicit if they are to be effective vehicles for framing concerns and sharing knowledge with others. They must also be made explicit if they are to become productive bases for team-oriented process-improvement activities.

Peter Senge and Charlotte Roberts make the above points very effectively in *The Fifth Discipline Fieldbook* [Senge 94]. They also provide examples of techniques that can be used in team settings to elicit and formulate explicit representations of mental models so that they can become productive bases for discussion and action.

Figure 3-1 on the preceding page is our top-level diagram for the mental model that we use for the goal-driven measurement process. This model has been invaluable in identifying issues that need to be addressed and in guiding us toward a consistent methodology.

As you begin applying the steps of the goal-driven process to identify and define your own software measures, you should construct flowgraphs or other diagrams of the processes that you manage or work within. When you do this, you will find several things happening:

- Process entities such as inputs, outputs, tasks, and flowpaths will become immediately visible.
- You will be led easily and naturally to identifying attributes of process and product entities about which information will be useful.
- You will find that you have vehicles (process models) that enable you to communicate your goals, concerns, questions, and interpretations to others.

- By explicitly stating your mental models as process models, you will help ensure that all who work on the processes have the same picture of the issues that need to be addressed.
- By stating your models explicitly, others can now help you evolve them and make them more exact. This will improve your ability to identify key issues and subgoals and to formulate questions that give insights into factors that affect achievement of the subgoals.
- The explicit process models will provide contexts for you and others to interpret and act on measurement results.

In short, flowgraphs are tools for visualizing processes. They use the power of pictures to communicate complex relationships with clarity in small amounts of time. Drawing a flowgraph is often a vital first step in continuous process improvement because it helps define the object of one's efforts. Whether the project involves one person or many, proper visualization of the process is essential to having people work on the right thing. This is especially true with administrative systems, where the flowgraph may be the only way to make the process visible [Wheeler 92].

If your organization has defined processes that it uses (and reuses) for its software activities, these make excellent starting points for formulating and evolving mental models to guide you through the steps in the goal-driven measurement process.

3.3 The Elements of Mental Models

Mental models of processes summarize relationships that exist among the elements associated with the processes. Although many kinds of models are possible, simple flowgraphs are a good place to start. They will help you initiate productive discussions that explore the needs and options associated with your business and technical goals.

Figure 3-2 illustrates a flowgraph for a generic process model. Figure 3-3 shows how expanding the central box helps identify internal entities and their attributes. Often the power of flowgraphs is not realized until expanded versions such as Figure 3-3 are created.

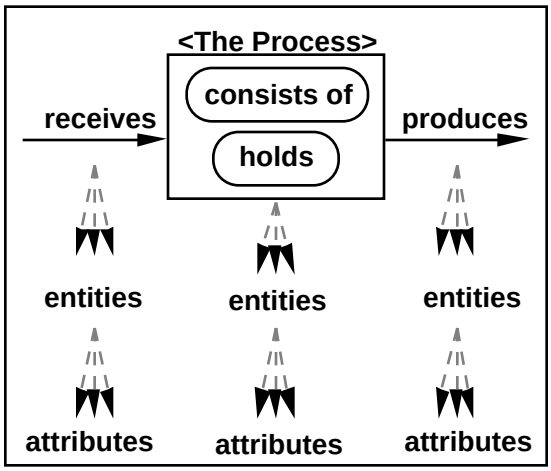


Figure 3-2: A Generic Process Model

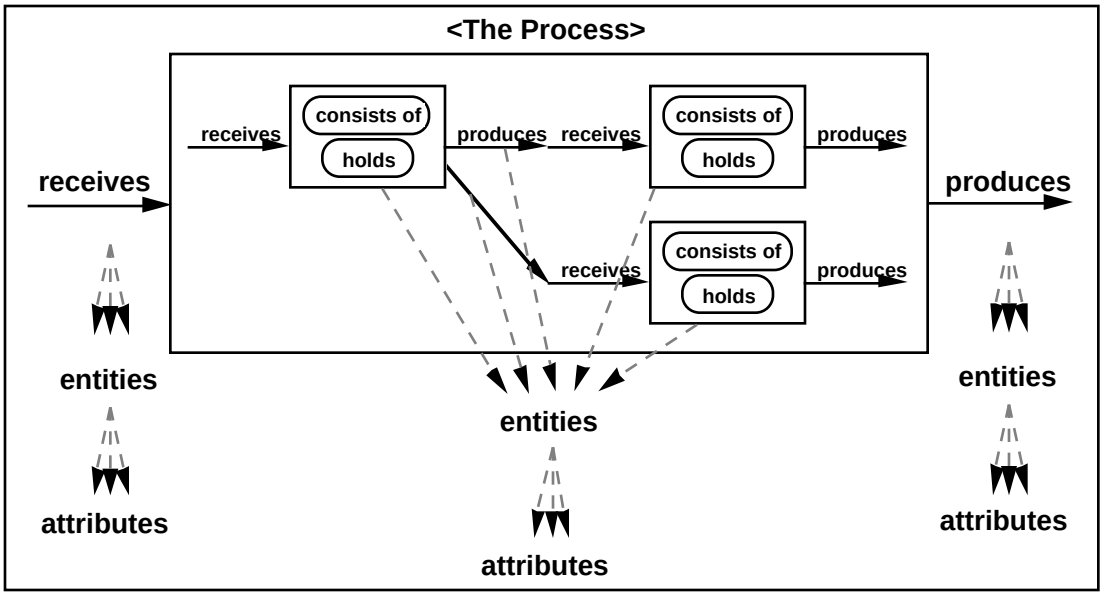


Figure 3-3: An Expanded Process Model

When we examine flowgraphs of processes, we find that they deal with four types of elements:

- things they receive (inputs and resources)—these are used or consumed
- things they produce (outputs)—these include products, by-products, and effects
- things they consist of (activities, flowpaths, and agents)—the structure of the process
- things they hold or retain (internal artifacts, such as inventory and work in process)

Each "thing" or element in a process is an entity, and each entity has attributes that relate either directly or indirectly to business goals. Elements with significant effects or that yield important insights are the ones that your mental process models should emphasize.

This taxonomy of inputs, outputs, activities, and internal artifacts leads naturally to useful mental models, as well as to checklists and other tools that help us identify the entities and attributes that we manage and improve to achieve our goals. For example, Figure 3-4 lists several entities that you may want to identify and consider as candidates for measurement in your processes. We suggest that you refer to this table when you begin to create flowgraphs for your mental models. It may call things to your attention that you might otherwise overlook.

To help keep you alert to opportunities and to show how general the input-process-output concepts of flowgraphs like Figures 3-2 and 3-3 are, we have included some nonsoftware entities in the lists.

A Process:

Receives	**Consists of**	**Holds**	**Produces**

resources
  people
  facilities
  tools
  money
consumables
  time
  fuel
  energy
  effort
  raw materials
guidelines and directions
  policies
  procedures
  goals
  constraints
  rules
  laws
  regulations
  training
  instructions
products and by-products from other processes
	processes
	  tasks
	  steps
	  activities
	  subprocesses
	  transformations
	  reviews
	  inspections
	controllers
	  flow controllers
	  sensors
	  signal processors
	  gates
	  throttles
	  valves
	flowpaths
	  product paths
	  resource paths
	  data paths
	  control paths
	buffers and dampers
	  queues
	  stacks
	  bins
	  reservoirs
	  accumulators
	  conveyors
		inventory
		materials
		work in process
		tools
		data
		knowledge
		experience
			products
			  requirements
			  specifications
			  plans
			  architectures
			  preliminary designs
			  detailed designs
			  reviewed designs
			  code
			  reviewed code
			  integrated code
			  tested code
			  test cases
			  test results
			  tested components
			  change requests
			  documentation
			  defects
			  defect reports
			  data
			  acquired materials
			by-products
			  knowledge
			  experience
			  skills
			  process improvements
			  waste & scrap
			  heat, light, & noise
			  pollution
			  data
			  good will
			  satisfied customers

Figure 3-4: Potentially Measurable Elements (Entities) in a Software Process Model

3.4 The Importance of Environmental Factors

> *No count or measurement has any meaning apart from its context.*
>
> *— David Wheeler, 1995*

You may also have occasion to measure or characterize entities that lie outside your process models. For example, it is important to keep in mind that every process operates in an environment that contributes to or detracts from its prospects for success. The kinds of products you have historically produced, the nature of your physical facilities, and the types of customers and markets you deal with can all significantly influence the performance of your process. Quantitative information about these environmental factors will help you and others interpret data that are gathered from the entities that you examine. This contextual information also helps you identify barriers to success and opportunities for improvement—two activities that are inherent in achieving almost all business goals.

It is important, too, to remember that processes have customers, both internal and external, who you may want to characterize. This is especially so when you have goals related to marketing or satisfying customer needs.

4 Applying the Goal-Driven Process

> *Without the right information, you're just another person with an opinion.*
>
> *— Tracy O'Rourke, CEO of Allen-Bradley*

The goal-driven process begins with identifying business goals and breaking them down into manageable subgoals. It ends with a plan for implementing well-defined measures and indicators that support the goals. Along the way, it maintains traceability back to the goals, so that those who collect and process measurement data do not lose sight of the objectives.

This chapter contains the heart of the guidebook. The materials in Sections 4.1 through 4.10 are presented as a sequence of tutorials, supported by examples. Each section ends with an exercise. You should treat the exercises as tasks to be performed by your measurement planning team(s), with iteration where needed. If you do this, you should end up with clearly defined measures that can be implemented and applied consistently by everyone in your organization, in ways that directly support your business goals. Figure 4-1 provides a roadmap for the process steps that we will cover.

Section	Process Step
4.1	Identify your business goals.
4.2	Identify what you want to know or learn.
4.3	Identify your subgoals.
4.4	Identify the entities and attributes related to your subgoals.
4.5	Formalize your measurement goals.
4.6	Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
4.7	Identify the data elements that you will collect to construct the indicators that help answer your questions.
4.8	Define the measures to be used, and make these definitions operational.
4.9	Identify the actions that you will take to implement the measures.
4.10	Prepare a plan for implementing the measures.

Figure 4-1: Roadmap for the Goal-Driven Measurement Process

Appendix A contains detailed instructions for the exercises that are at the end of each section, together with forms and templates that measurement planning teams can use to

help structure their tasks. We suggest that you reproduce these instructions and use them as handouts to guide your teams' activities. You may also reproduce and adapt the checklists and forms in Appendix B to help create operational definitions for your measures. Guidelines and examples for using the checklists can be found in [Park 92], [Goethert 92], and [Florac 92].

4.1 Step 1: Identify Your Business Goals

> *In truth, a good case could be made that if your knowledge is meagre and unsatisfactory, the last thing in the world you should do is make measurements. The chance is negligible that you will measure the right things accidentally.*
>
> *— George Miller, a psychologist*

Business Goals

The first step in identifying and defining software measures is to identify the business goals that drive your organization's efforts (Figure 4-2). As Lynch and Cross point out in their book *Measure Up!,* business goals are often a function of where you sit [Lynch 91]. The focus in departments and work centers differs from the focus in the business units they report to. Nevertheless, the goals in hierarchical organizations are related, and they all come down to competing on the basis of productivity, flexibility, and customer satisfaction. Goals framed in terms of these concerns by business units lead, in departments and work centers, to more specialized goals that address issues of quality, delivery, cycle time, and waste.

Since the goal-driven measurement process in this guidebook is designed to help teams reason from general business goals to specific measures and indicators, it can be initiated at any organizational level where goals can reasonably be identified. Entering at a high level with goals such as "Reduce cycle time" or "Improve customer satisfaction" has the advantage of ensuring traceability of the resulting measures back to the primary business goals at that level. The disadvantage is that iteration may be needed with Steps 2 and 3 (and possibly Step 4) in order to push the implications of these goals down to levels where specific measures can be effectively formulated. Entering the goal-driven process at lower levels avoids some of this work, but increases the likelihood of starting with goals whose traceability to higher level objectives may be less than apparent. This could affect the support you receive from higher level managers.

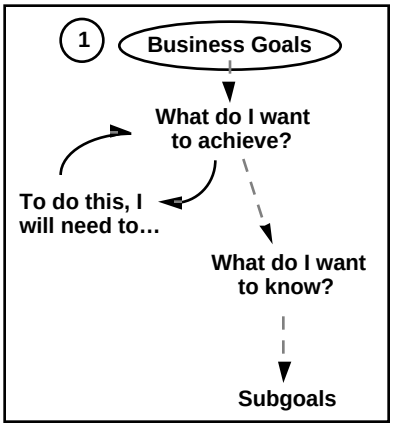


Figure 4-2: The First Target—Identify Your Business Goals

Regardless of the level, the important thing is to get started. The goal-driven process is sufficiently flexible to allow you to enter it at any level where valid goals can reasonably be identified. Perhaps the best advice we can give you is to begin with the goals of the person

whose sponsorship will be most important to implementing and sustaining the resulting measurement efforts. This person could be a senior executive with broad responsibilities. Or, if you reside within a project team, it could be your software project manager. Identifying, defining, and implementing measures does not have to be an organization-wide program. It can be initiated at any level where quantitative information about products, processes, or resources would improve your abilities to plan, control, and improve your processes.

When you do begin the process, Step 1 (identifying business goals) is usually best done in team settings, with managers participating. This helps ensure that you start with the right goals, and that important things are not overlooked. Findings and action items from software process assessments and software risk assessments make good starting points for identifying goal-driven measures, as do outputs from strategic planning exercises. Structured brainstorming and the Nominal Group Technique (NGT) can be helpful methods for identifying and prioritizing your goals [Scholtes 90]. In addition, interviewing business managers can be an excellent way to gain insights into the way business goals are perceived and translated into lower level goals and objectives.

The product of Step 1 is a prioritized set of business goals. It is wise to have these goals reviewed by upper-level managers before proceeding further, to ensure that your priorities are appropriate and that nothing important has been overlooked or misinterpreted.

Exercise 1: Identifying Business Goals

Use structured brainstorming or the Nominal Group Technique to generate a list of the business goals that drive your own software processes. Merge similar goals and sort the results into a prioritized list. Figure 4-3 illustrates this step. Reproducible instructions for this exercise are presented in Appendix A.

Figure 4-3: Identifying Business Goals

4.2 Step 2: Identify What You Want to Know or Learn

A prudent question is one-half of wisdom.

— Francis Bacon

Setting Out on the Path from Goals to Measures

With your business goals identified, the next step is to begin identifying what you would like to know in order to understand, assess, predict, or improve the activities related to achieving your goals. By asking questions such as

"What activities do I manage or execute?"

and

"What do I want to achieve or improve?"

and by completing statements such as

"To do this, I will need to…",

you can begin identifying the quantitative information that you would like to have. You should repeat these questions several times as you break top-level goals down into specific things that you want to accomplish and issues that you will need to address. The ellipse in Figure 4-4 highlights this part of the process and shows how it draws upon your mental model of the processes you manage.

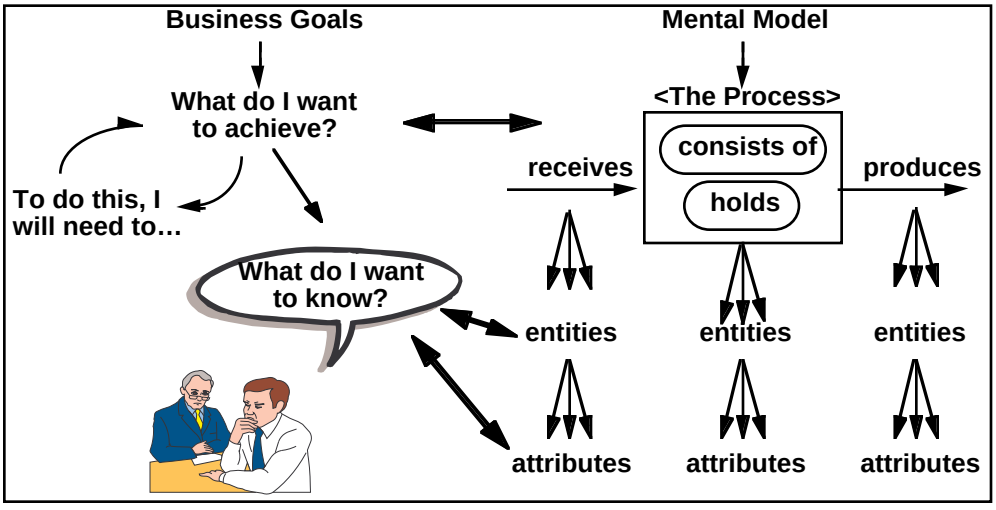


Figure 4-4: The Second Target—What Do You Want to Know?

Scenario

We use several examples in the materials that follow to show you how entities, attributes, and evolving mental models help guide and focus the goal-driven process. To give substance to these examples, we offer the scenario shown in Figure 4-5.

Scenario

- Your organization has just completed a software process assessment.
- One of the findings was that your projects are delivering software functions and documentation on time, but customers are not as satisfied as you would like them to be.
- The related action item is: *Improve customer satisfaction*.
- Your process improvement team has identified this action item as one of its primary goals.
- You are a project manager, and you need a set of measures that will help your project make progress toward this goal.

Figure 4-5: Scenario

So, your first goal—and the one we illustrate in Sections 4.2 through 4.3—will be to *improve customer satisfaction*. You (and others in your organization) may have other goals that are just as important, but we will focus on this one now. The goal-driven measurement process can (and should) be applied later to your other goals as well.

The Entity-Question List—A Template for Framing Questions

The tool we recommend to help identify and frame questions is called an entity-question list. Our example of the use of this tool is patterned after a similar illustration in the ami handbook [ami 92]. The sequence of tasks is as follows:

1. Start with one of the top-level goals that your team identified in Step 1.
2. Identify the persons or groups whose concerns your team will address. (This may be you or the organization you lead.) This defines your perspective and the roles that you and the team will assume in Tasks 3 through 6 here and in the remaining steps of the goal-driven measurement process.
3. Create rough sketches (mental models) of the relevant processes that you, in your role, manage or affect. As you do this, be guided by what you want to achieve and the issues you will have to address to achieve it.

4. List the important things (entities) in your processes that you, in your role, manage or influence. Make sure that you address each of the four kinds of process entities below:

   – inputs and resources
   – products and by-products
   – internal artifacts such as inventory and work in process
   – activities and flowpaths

   You may also want to list some of the environmental entities outside your processes that affect your work.

5. For each entity, list questions that, if answered, would help you, in your role, plan and manage progress toward your goals. For example:

   – How big is it?
   – How much is there?
   – How many components?
   – How fast is it?
   – How long does it take?
   – How much does it cost?

6. Then step back and look at your process as a whole to see if you have missed anything. By asking questions such as

   – Is the process stable?
   – How is it performing now?
   – What limits our capability?
   – What determines quality?
   – What determines success?
   – What things can we control?
   – What do our customers want?
   – What limits our performance?
   – What could go wrong?
   – What might signal early warnings?
   – How big is our backlog?
   – Where is backlog occurring?

   and most importantly

   – How will we know?

   you may discover additional entities whose properties may be worth measuring.

7. Repeat Tasks 1–6 for your other goals.

When listing questions related to entities (Task 5 above), do not spend excessive time trying to frame the perfect question. Precise framing of questions will come later—you are still early in the goal-driven process. Your questions at this point can be either very general or quite specific. The important thing is to get your concerns on the table so that you can use them to help elaborate your mental model, and so that they can set the stage for identifying and articulating subgoals in Step 3 of the goal-driven process (Figure 3-1).

Note that Task 7 of Step 2 does not have to be performed immediately after Tasks 5 and 6. Usually it is best to continue on through Step 7 of the goal-driven measurement process, to the point where you have clearly identified the data elements that address one or two of your business goals, before branching off on additional goals. This helps keep the goal-driven process manageable. Ultimately, though, you will want to give priority to measures that

address more than one goal. This is especially true when you begin to reduce the large set of potential measures to the ones that are most valuable to you. There are no universal guidelines for when to address additional goals—it all depends on the situation. You and your team will have to judge this for yourselves.

Examples

Figures 4-6 through 4-9 on the following pages list some of the entities and questions that might concern the project manager in our example scenario when he or she is trying to improve customer satisfaction. Your lists for your own organizations or projects will likely be more encompassing.

Entities managed by a project manager	Questions related to customer satisfaction
Inputs and resources	
people	Are our people qualified to produce the results the customer wants?
	Is personnel turnover hampering product quality?
subcontractors	Are the practices of our subcontractors consistent with those of the activities they support?
computers	Is the target system meeting its performance requirements?
	Is the target system reliable?
customer change requests	Do customer change requests contain the information that we must have to produce timely and effective changes?

Figure 4-6: Entity-Question List (Part 1): Inputs and Resources

Entities managed by a project manager	Questions related to customer satisfaction
Internal artifacts	
customer change requests (work in process)	How large is our backlog of customer change requests?
	Where are the backlogs occurring?

Figure 4-7: Entity-Question List (Part 2): Internal Artifacts

Entities managed by a project manager	Questions related to customer satisfaction
Activities & flowpaths	
development	Is development progress visible to the customer?
testing	Are our testing procedures adequate for the operational use of the system?
	Does the customer accept the testing procedure and test results?
fixing	Is the response time for fixing bugs compatible with customer constraints?
	Is change control adhered to?
	Are high-priority changes getting implemented in a timely fashion?
	Are status and progress visible to the customer?

Figure 4-8: Entity-Question List (Part 3): Activities and Flowpaths

Entities managed by a project manager	Questions related to customer satisfaction
Products and by-products	
documents	Are the documents we produce readable?
	Is it possible to trace system features from one document to the next?
	Are documents concise and complete?
	Is the terminology correct?
source code & compiled products	Is the source code consistent with the documents?
	Is the source code error free?
	Does source code follow programming standards?
	Is the system response time adequate?
	Is the man-machine interface satisfactory?
plans	Are plans consistent with customer constraints?
	Are they kept up to date?
	Are plans and changes communicated to the customer?
budget	Are budgets consistent with plans?
	Are budgets consistent with customer constraints?

Figure 4-9: Entity-Question List (Part 4): Products and By-products

As you generate entity-question lists, keep in mind that your purpose is simply to identify things that you want to know or understand. You do not have to classify them correctly. It is much more important to produce a useful list of entities and questions than to worry about getting elements assigned to their correct slots.

Exercise 2: Identifying What You Want to Know or Learn

Select one of the goals you identified in Exercise 1. With your team, perform Tasks 1 through 6 of the sequence that we just outlined to fill in the template shown in Figure 4-10. Repeat the process, as appropriate, for your remaining goals. Appendix A contains instructions and full-sized templates to support you in this exercise.

(Remember that the scenario we have used is only an illustration. In this exercise and those that follow, you should focus not on our scenario, but on your own business goals and on using your goals to identify and define the measurements you will make to help you achieve the goals.)

Entities of Interest	Questions Related to Business Goal(s)
Products and by-products	
Inputs and resources	
Internal artifacts (work in process, backlogs, inventory, etc.)	
Activities and flowpaths	

Figure 4-10: A Template for Generating Entity-Question Lists

4.3 Step 3: Identify Your Subgoals

> *Measurements are important, but what is measured is more important.*
>
> *— Francis S. Patrick*

Grouping Related Questions Helps Identify Subgoals

The third step in the goal-driven process is to translate your top-level goals into subgoals that relate specifically to activities that you manage or perform. You can use the entity-question lists from Exercise 2 to help you do this.

The ellipse in Figure 4-11 shows where we are in the process. The questions that were prompted by the evolving mental model of your operational process(es) point, either implicitly or explicitly, to entities and attributes associated with achieving your goals. You now need to identify the questions that you have about the entities, then group them and identify the issues they address. You will then be able to translate the results into manageable subgoals. (If you are familiar with team-based problem-solving processes, you will recognize this as a convergent step that organizes and structures the results of the divergent process we used to generate the entity-question lists.)

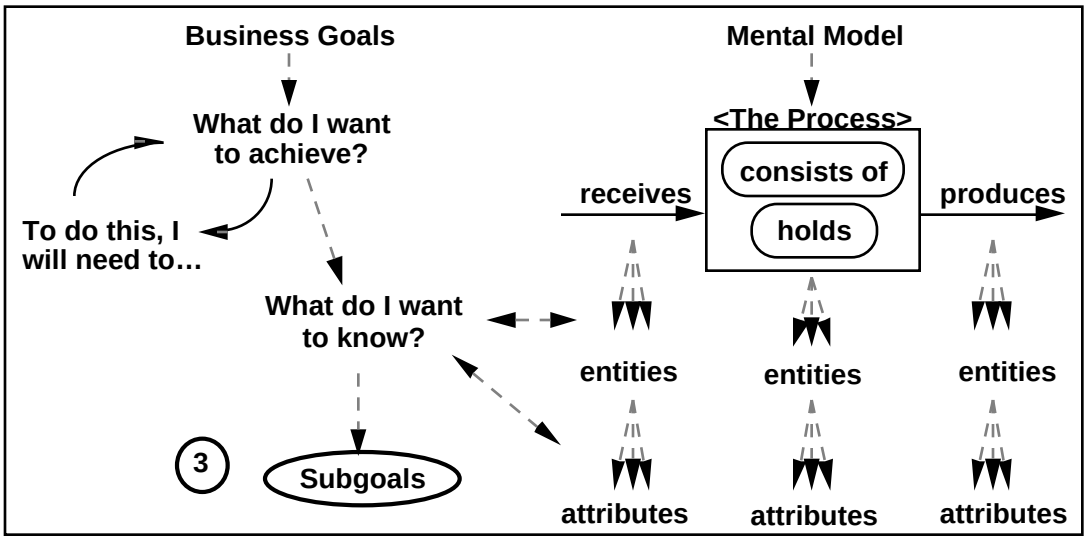


Figure 4-11: The Third Target—Clearly Identified Subgoals

Examples

Figure 4-12 shows some results for our example scenario. Here, the groupings are easily identified. For example, the first four questions listed for products and by-products address documentation. This issue is grouped already, since we generated the questions by focusing on the document entities we produce.

Entities managed by a project manager	Questions related to customer satisfaction
Products and byproducts	
documents	Are the documents we produce readable?
	Is it possible to trace system features from one document to the next?
#1	Are documents concise and complete?
	Is the terminology correct?
source code & compiled products	Is the source code consistent with the documents?
	Is the source code error free?
#2	Does source code follow programming standards?
	Is the system response time adequate?
	Is the man-machine interface satisfactory?
plans	Are plans consistent with customer constraints?
	Are they kept up to date?
	Are plans and changes communicated to the customer?
budget	Are budgets consistent with plans?
	Are budgets consistent with customer constraints?

Figure 4-12: Identifying Related Questions (Part 1)

As we scan down the entity-question list, we see that the next five questions relate to the quality and performance of the software product. This concern continues in Figures 4-13 and 4-14. Since this seems to be a consistent theme, we elect to collect these questions as group #2.

Entities managed by a project manager	Questions related to customer satisfaction
Inputs and resources	
people	Are our people qualified to produce the results the customer wants?
	Is personnel turnover hampering product quality?
subcontractors	Are the practices of our subcontractors consistent with those of the activities they support?
computers **#2**	Is the target system meeting its performance requirements?
	Is the target system reliable?
customer change requests	Do customer change requests contain the information that we must have to produce timely and effective changes?

Figure 4-13: Identifying Related Questions (Part 2)

Entities managed by a project manager	Questions related to customer satisfaction
Activities & flowpaths	
development	Is development progress visible to the customer?
testing **#2**	Are our testing procedures adequate for the operational use of the system?
	Does the customer accept the testing procedure and test results?
fixing	Is the response time for fixing bugs compatible with customer constraints?
#2	Is change control adhered to?
	Are high-priority changes getting implemented in a timely fashion?
	Are status and progress visible to the customer?

Figure 4-14: Identifying Related Questions (Part 3)

As we continue identifying principal themes or issues, we collect the questions related to each issue and transfer them to a new list, sorted by issue. Figure 4-15 shows the results for our scenario. Keep in mind that the groupings will always be somewhat arbitrary, since they are based on the team's perceptions of central themes. Moreover, it is perfectly appropriate for a question to be repeated in more than one grouping.

Questions related to customer satisfaction	
Grouping #1 **(documents)**	Are the documents we produce readable? Is it possible to trace system features from one document to the next? Are documents concise and complete? Is the terminology correct?
Grouping #2 **(software product)**	Is the source code consistent with the documents? Is the source code error free? Does source code follow programming standards? Is the system response time adequate? Is the man-machine interface satisfactory? Is the target system meeting its performance requirements? Is the target system reliable? Is change control adhered to? Are our testing procedures adequate for the operational use of the system?
Grouping #3 **(project management)**	Are plans consistent with customer constraints? Are they kept up to date? Are budgets consistent with plans? Are budgets consistent with customer constraints?
Grouping #4 **(change management)**	Do customer change requests contain the information that we must have to produce timely and effective changes? How large is our backlog of customer change requests? Is the response time for fixing bugs compatible with customer constraints? Is change control adhered to? Are high-priority changes getting implemented in a timely fashion?

Figure 4-15: Summary of Groupings

Questions related to customer satisfaction	
Grouping #5 (communications)	Are plans and changes communicated to the customer? Is development progress visible to the customer? Does the customer accept the testing procedure and test results? Are status and progress of change requests visible to the customer?
Other	Are our people qualified to produce the results the customer wants? Is personnel turnover hampering product quality? Are the practices of our subcontractors consistent with those of the activities they support?

Figure 4-15: Summary of Groupings (Part 2)

The groupings of issues and questions then translate naturally into candidate subgoals, which can be prioritized. The results for our scenario are illustrated in Figure 4-16. In the real world, you would want to validate these subgoals with your customer(s) to ensure that you are addressing their true concerns. This would also help you assign priorities to your subgoals.

Derived Subgoals	
Subgoal #1	**Improve readability and traceability of documents.**
Subgoal #2	**Improve reliability and performance of released code.**
Subgoal #3	**Improve monitoring of plans and budgets.**
Subgoal #4	**Improve performance of the change management process.**
Subgoal #5	**Improve communications with the customer.**

Figure 4-16: Derived Subgoals—A Project Manager's Perspective of the Goal "Improve Customer Satisfaction"

The issues in the training scenario we have been using have mapped nicely, one-to-one, into subgoals. This will not always happen in real life. In your work, you may find several issues mapping into a single subgoal, or single issues mapping into several subgoals.

There is nothing wrong with this—formal decomposition of business goals is not what we are after. The point is simply to arrive at rational subgoals that

- can be addressed by managerial or technical actions
- point us toward focused measurement goals

If your list of groupings is long and you perceive it leading to many measures, your teams may want to assign priorities to the issues and sort them into rough priority order. This will help them focus on first things first in the steps that follow.

Exercise 3: Identifying Subgoals

Review your entity-question lists from Exercise 2 and identify related questions. Group related questions and identify the issues they address. (Issues are the central themes that caused you to view the questions as related.)

Hint: If you write each question on a separate piece of paper (or Post-It, 3x5 card, etc.), your team will be able to rearrange the questions and experiment with alternative groupings.

Use the issues and their associated questions to formulate and state manageable subgoals that are derived from the goals and questions you identified in Exercise 2.

A template for this exercise is illustrated in Figure 4-17. Appendix A contains reproducible worksheets that you can use to record your groupings and subgoals.

Groupings	Questions Related to Business Goal(s)
Grouping #1 ( )	
Grouping #2 ( )	
Grouping #3 ( )	
Grouping #4 ( )	

	Derived Subgoals
Subgoal 1	
Subgoal 2	
Subgoal 3	
Subgoal 4	

Figure 4-17: A Template for Mapping from Questions to Subgoals

4.4 Step 4: Identify the Entities and Attributes

> *Measurement presupposes something to be measured, and, unless we know what that something is, no measurement can have any significance.*
>
> *— Peter Caws, 1959*

Using Subgoals, Issues, and Questions to Identify Specific Entities and Attributes

You now have a list of manageable subgoals, together with lists of related issues and questions. The next step is to use the questions to refine your mental model(s) and the entities and attributes associated with them (the ellipse in Figure 4-18). This will set the stage for formulating the well-stated measurement goals that you will use subsequently in Step 6 to start the GQ(I)M process. It may also lead to refining your questions, issues, and subgoals. After all, since your mental models are evolving, there is no reason to suspect that you have gotten everything exactly right the first time.

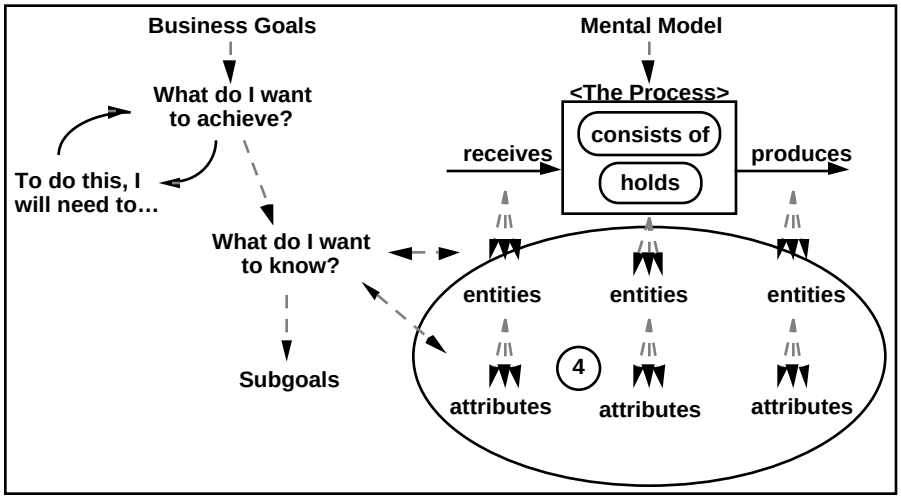


Figure 4-18: The Fourth Target—Refined Entities and Attributes

You should begin this step by making a preliminary sketch of your mental model(s) for the process(es) you manage or execute. Then list questions that you believe to be important to answer. These questions are usually ones associated with your highest priority subgoals. This sketching and listing is an iterative process—your sketches will suggest questions, and your questions will lead you to refine your sketches. Similar iterations are likely to occur in later steps as well, as you continue to flesh out your mental model(s).

Once you have a list of questions, you should examine each question and identify the entities that are implicit in it. Then list the pertinent attributes associated with each entity.

Pertinent attributes are those which, if quantified, help you answer your question or establish a context for interpreting the answers. Pertinent attributes are usually cited in the question, either explicitly or implicitly. Identifying the entities may also cause you to think of other questions and attributes. Your list of entities and the attributes for each entity are the principal outputs of this step. The attributes will become candidates for the things you will measure.

Figures 4-19 through 4-22 illustrate this process. Figure 4-19 lists the questions that our project manager had with respect to grouping #4 (change management). Figures 4-20 through 4-22 then focus on individual questions from that list. For each question, the figures identify the entity cited. Then they list attributes that, if measured, would help shed light on the issue that motivated the question.

When listing attributes, it is useful to keep in mind the distinction between attributes (the characteristics of an entity) and measures (the scales and rules used to assign values to attributes). Many teams tend to get overly specific when they build their initial lists of attributes. Others sometimes argue unproductively about the differences between attributes and measures. While the distinctions can be important, they are not important here. The point now is simply to identify key characteristics of the entity and its environment that will help shed light on the question. Building lists of specific measures too soon tends to overly constrain your views and keep you from recognizing other measurement options and opportunities.

To avoid focusing on detailed measures too soon, you should consciously scrub each list, looking for entries that seem to imply unique measures. When you find one, do as we have done in Figure 4-20: identify the broader characteristic of interest (e.g., size) and then list your prospective measures as examples of how to measure that characteristic. You may find that this helps you identify other useful measures and attributes.

Grouping #4 **(change management)**	Do customer change requests contain the information that we must have to produce timely and effective changes? How large is our backlog of customer change requests? Is the response time for fixing bugs compatible with customer constraints? Is change control adhered to? Are high-priority changes getting implemented in a timely fashion?
Derived Subgoal	Improve performance of the change management process.

Figure 4-19: A Project Manager's Questions Related to Change Management

Subgoal 4: Improve Performance of the Change Management Process

Question 1:

- Do change requests received from customers contain the information that we must have to produce timely and effective changes?

Entity

- the set of change requests received from customers

Attributes

- size (e.g., the number of change requests received)
- adequacy (e.g., the percentage of change requests with all required fields filled in correctly)
- defect distribution (e.g., for each required field, the percentage of requests
  - with that field omitted
  - with that field filled in incorrectly)

Figure 4-20: Entity and Attributes Associated with Question #1

Subgoal 4: Improve Performance of the Change Management Process

Question 2:

- How large is our backlog of customer change requests?

Entity

- the backlog of customer change requests

Attributes

- size of backlog (backlog = change requests received but still in the pipeline)
- size of queues/stacks/bins awaiting action at each stage of the change management process
- total effort (estimated) required to clear the backlog

Figure 4-21: Entity and Attributes Associated with Question #2

Subgoal 4: Improve Performance of the Change Management Process

Question 3:

- Is the response time for fixing bugs compatible with customer constraints?

Entity

- the change management process

Attributes

- the customer's expectation for cycle time
- the frequency distribution of time from receipt of a change request until it is implemented
- the frequency distribution of time from receipt of a change request until it is installed at the customer's site
- average amount of time that requests spend at each step of the change process

Figure 4-22: Entity and Attributes Associated with Question #3

Exercise 4: Identifying Entities and Attributes

Review the groupings your team identified in Exercise 3. List the entities and attributes associated with each question. Your results will become the basis for formalizing the measurement goals in Step 5. A template for recording your results is illustrated in Figure 4-23. A full-sized, reproducible worksheet is presented in Appendix A.

Question

-

Entity

-

Attributes

-
-
-
-
-
-

Figure 4-23: A Template for Recording Entities and Attributes

4.5 Step 5: Formalize Your Measurement Goals

> *…if one is to make a formal measurement, one must accept responsibility for making some effort to define one's purpose.*
>
> *— Paul Kirchner*

Up to this point, you have been focusing on identifying business goals and things that affect your achievement of those goals. You are now ready for the fifth step—translating your issues and concerns into clearly stated measurement goals. You will be guided here by the subgoals you identified in Step 3 (Section 4.3) and the refinements you made to your mental model(s) in Step 4 (Section 4.4). The ellipse in Figure 4-24 shows where we are in the goal-driven measurement process.

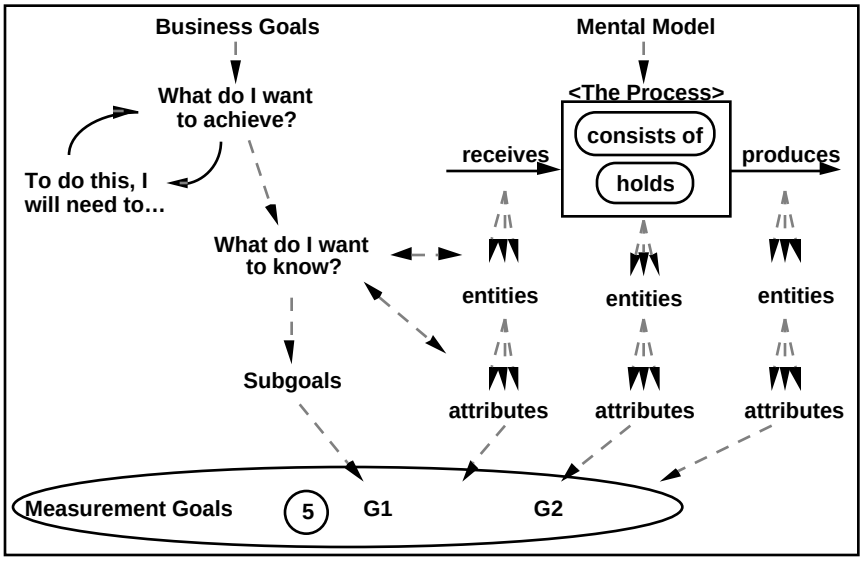


Figure 4-24: The Fifth Target—Measurement Goals

The purpose of Steps 1 through 4 has been to get to a point where the goal-question-metric (GQM) paradigm of Basili and Rombach [Basili 88, Basili 89, Rombach 89] can be applied effectively. As we shall see in Section 4.6, supplementing the paradigm by adding an "indicator" step between the Q and M of GQM is usually helpful.[1] But first, you should establish a solid foundation for GQ(I)M by identifying your measurement goals and preparing structured statements for them. The structured aspect is important, because it will help you ensure that key points are not overlooked when you set about defining and collecting measures.

---

[1] By "indicator," we mean a picture or display of the kind one would like to have to help answer the question. Our experience is that sketches of such pictures and displays help significantly in identifying and defining appropriate measures.

Measurement Goals Can Be Active or Passive

When defining structured measurement goals, it helps to keep in mind that there are two kinds of goals: active and passive. Awareness of this and of the distinctions between active and passive goals may alert you to important opportunities.

Active measurement goals are directed toward controlling processes or causing changes to products, processes, resources, or environments. These are the kinds of goals that are commonly found in project management and process improvement activities.

Passive measurement goals, on the other hand, are meant to enable learning or understanding. They are fundamental to improving our understanding to the point where well-chosen, active measurement goals can be formulated. Passive goals are often accomplished by characterizing objects of interest according to some productivity or quality model. Figure 4-25 shows some examples of active and passive goals.

Active Goals	Passive Goals
Meet the scheduled completion date	Understand the current development process
Reduce variability	Identify root causes
Improve product reliability	Assess product maintainability
Improve the productivity of the process	Identify capabilities and trends, so that we can better predict future performance
Improve time-to-market	Understand relationships among attributes, so that we can develop models for predicting and estimating
Reduce employee turnover	

Figure 4-25: Examples of Active and Passive Goals

In Chapter 2 we listed four reasons for measuring software processes, products, and resources: to characterize, to evaluate, to predict, and to improve. As Figure 4-25 suggests, active goals are usually associated with evaluating and improving, while passive goals are more often linked to characterizing and predicting.

Many organizations become so focused on active goals that they give short shrift to the value of information associated with passive goals. You, of course, will not fall into this trap.

What Do Formal Goals Look Like?

Well-structure measurement goals have four components:

- an object of interest (an entity)
- a purpose

- a perspective
- a description of the environment and constraints

Figure 4-26 is a template for stating structured measurement goals. Figures 4-27 through 4-30 are supporting templates for constructing the principal elements of Figure 4-26. We have adapted these templates from ideas and materials that have been developed by Victor Basili and Dieter Rombach [Basili 88, Basili 89, Rombach 89].

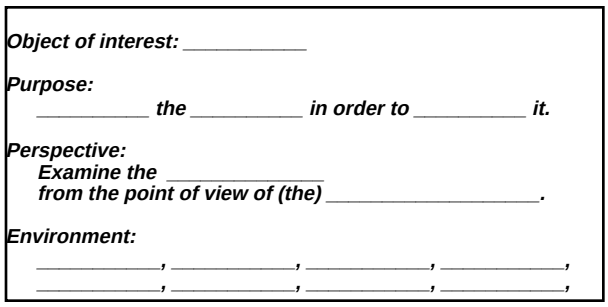


Figure 4-26: A Template for Stating Measurement Goals

The following paragraphs describe the elements of a measurement goal and the supporting templates. Full-sized templates and a worksheet for stating measurement goals are presented in reproducible form in Exercise 5 of Appendix A.

Object

The object of interest may be a product, process, resource, agent, artifact, activity, metric, or environment. It may also be a set or collection of other entities or objects. In short, any "thing," real or abstract, that you want to describe or know more about is a potential object for measurement.

Thus, an object is an entity. But it is not just any entity. Rather, it is the specific entity that you want to describe with measured values.

Sometimes we have situations where one person's attribute becomes another's object for measurement. For example, at a macro-level many people perceive things like quality and complexity to be attributes of software and software processes. But since there are (and can be) no single measures for these kinds of abstract concepts, the only logical approach is to treat the concepts as entities which we characterize (measure, describe) in terms of more concrete attributes or dimensions.

In all cases, though, it is important to have (and keep) a clear picture of what we are looking at. By explicitly identifying the object that we seek to describe, we are able to keep a common picture in the minds of all team members. Figure 4-27 is a simple template for stating the object of interest.

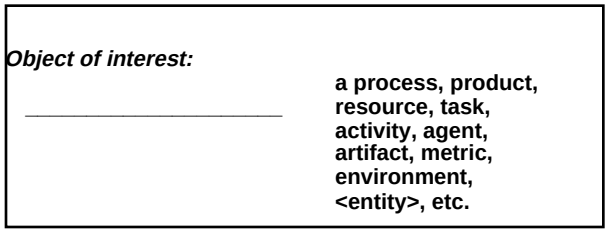


Figure 4-27: A Template for Stating the Object of Interest

Purpose

The purpose of a measurement activity may be to understand, predict, plan, control, compare, assess, or improve some productivity or quality aspect of the object. Examples of aspects that can be assessed, understood, predicted, planned, improved, or controlled include

- cost
- size
- reliability
- test coverage
- responsiveness
- peer review effectiveness
- process compliance
- time to market
- quality
- customer satisfaction

Since an aspect can involve several attributes, it is usually best to defer identifying specific attributes until we have formulated not only our purpose, but also the other parts of our measurement goal.

The purpose of any measurement activity should be stated explicitly. Figure 4-28 is a structured template for doing this. Examples of completed purpose statements will be presented shortly, in Figures 4-31 through 4-33.

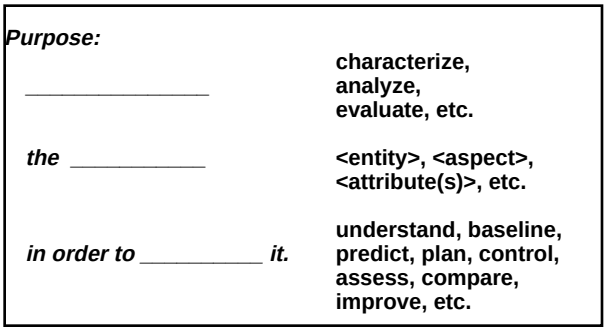


Figure 4-28: A Template for Defining the Purpose of a Measurement Activity

Perspective

The perspective identifies who is interested in the measurement results. It identifies the principal viewpoint that is to guide the measurement activity, such as that of the developer, maintainer, manager, or customer. The perspective is stated to clarify the purpose of the measurement activity.

Example: The goal of improving productivity may take on entirely different meanings, depending on who you are and where you sit. For instance, if you are a software engineer, improving productivity may mean increasing the amount of code produced per staff-hour. If you are a project manager, your view of improving productivity may mean bringing your project in on schedule. And if you hold a corporate position, you may take improving productivity to mean increasing revenues or returns on investment [Rombach 89].

When those who define measures and collect data understand the perspectives of the people who use the data they collect, they will be much more likely to construct and apply measures in ways that give real value and avoid misinterpretations.

Figure 4-29 is a structured template for stating a measurement perspective.

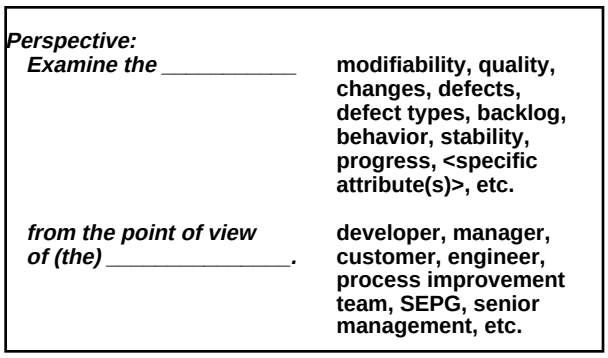


Figure 4-29: A Template for Defining the Measurement Perspective

Environment

A description of the environment provides a context for interpreting measurement results. When the context is not made explicit, it may not be understood by all who use the reported data. The chances for misuse are then high, and erroneous conclusions can easily be reached.

The environment includes everything that affects or is affected by the object to be measured. In particular, it includes all significant constraints on the object of measurement (time, resources, unusual performance criteria, etc.), as well as constraints on the scope or time span of the measurement process itself.

For example, if you use data about project size, cost, and time to look at trends in productivity improvement, it helps to know the kind of application the data come from, whether the product was developed under contract or not, and if the project was accelerated to meet externally specified schedules.

Similarly, if the purpose is to analyze an organization's problem-reporting process in order to improve its effectiveness, the environment may include

- the maintenance process
- the work facilities and supporting tools
- the organizational structure
- the maintained products
- the time span and names of projects examined

Descriptions of environments are, by their very nature, open ended. You will avoid floundering, and the information you provide will be most useful, if you focus on two aspects:

1. What makes this product, process, or setting *similar to* others that people may be familiar with. This provides the context for comparisons.

2. What makes this product, process, or setting *different from* others that people may be familiar with. This provides the context for exploring, understanding, and accounting for differences.

Figure 4-30 illustrates the kinds of issues that should be addressed when describing the measurement environment.

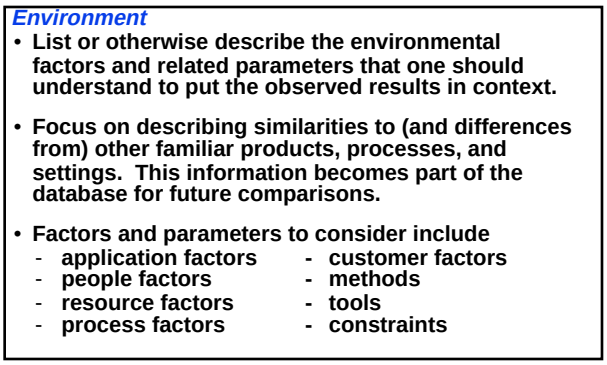


Figure 4-30: A Template for Characterizing the Environment in Which Measurements Will Be Made

Examples of Formalized Measurement Goals

Figures 4-31 through 4-33 show examples of formally stated measurement goals. The environment section in each has been abbreviated to keep the examples concise. In your work, you will usually find it useful to provide more complete characterizations of your environments than our figures illustrate.

(Note: At this point we leave our "customer satisfaction" scenario, so that you can see how goal-driven methods apply in other scenarios as well.)

Object of interest
- ***The peer review process at plant XYZ***

Purpose
- ***Evaluate the peer review process in order to identify opportunities for improving its effectiveness.***

Perspective
- ***Examine the controllable factors, costs, and results from the point of view of a process improvement team.***

Environment
- ***New development. Military avionics. CMM Level 2 (working on Level 3). 8000 people in plant. 2000 software developers. Customer is the DoD. Constraints: Examine projects completing unit testing 1 Jan 93–30 Jun 95. Exclude reused modules.***

Figure 4-31: A Formally Stated Measurement Goal (Example 1)

Object of interest
- ***The peer review process at plant XYZ***

Purpose
- ***Characterize the peer review process in order to predict its impact on future projects.***

Perspective
- ***Examine the effectiveness of peer reviews from the point of view of managers, planners, and cost estimators of new projects.***

Environment
- ***New development. Military avionics. CMM Level 2 (working on Level 3). 8000 people in plant. 2000 software developers. Customer is the DoD. Constraints: Examine projects completing unit testing 1 Jan 93–30 Jun 95.***

Figure 4-32: A Formally Stated Measurement Goal (Example 2)

Object of interest
- ***The software development process at plant XYZ***

Purpose
- ***Evaluate the extent to which the software organization is using peer reviews prior to unit testing in order to assess the organization's compliance with policies and directives.***

Perspective
- ***Examine the coverage and consistency of peer reviews from the perspective of a software process assessment team.***

Environment
- ***New development. Military avionics. CMM Level 2 (working on Level 3). 8000 people in plant. 2000 software developers. Customer is the DoD.***

Figure 4-33: A Formally Stated Measurement Goal (Example 3)

Maintaining Traceability

As you develop structured measurement goals, it is wise to maintain traceability back to the subgoals and business goals that motivated each measurement goal. Then, if questions arise later about the wording or intent of a measurement goal, people will be able to look back to the origins of the goal and make implementation decisions that are consistent with your business objectives.

This concept of traceability is illustrated in Figure 4-34. Ideally (in theory) we would like to have full traceability back to our business goals. But full traceability may be difficult, since the process that we used to move from high-level goals to concrete and enactable subgoals used informal groupings of sometimes loosely related questions. Moreover, when we posed the questions, we made no attempt then to make them precise. That comes later, in Steps 6 and 7. We simply used the early questions to help us interpret the top-level goals and restate them as manageable subgoals. In some cases, questioning paths will be retraceable, but in others they may not. In practice, the picture will often be more like the one in Figure 4-35, where measurement goals are traceable (easily) only to business subgoals.

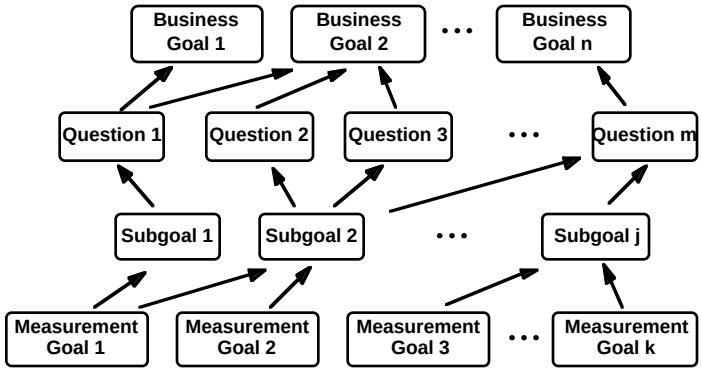


Figure 4-34: Maintaining Traceability to Business Goals—Theory

Whether you can draw useful, traceable paths from your subgoals all the way back to your top-level business goals, only you can determine. If you can, and if this provides context for the efforts (and decisions) that are yet to come, we encourage you to record the paths. As a minimum, though, you should explicitly record the paths from your measurement goals back to your business subgoals, as illustrated in Figure 4-35. In simple situations, graphs like Figure 4-35 may suffice. In other cases, especially if you have many measures or subgoals, matrix representations or alternative structures may be more appropriate. You may find it possible to borrow some mapping techniques or tools from the people in your organization who have experience in requirements tracing or quality function deployment (QFD).

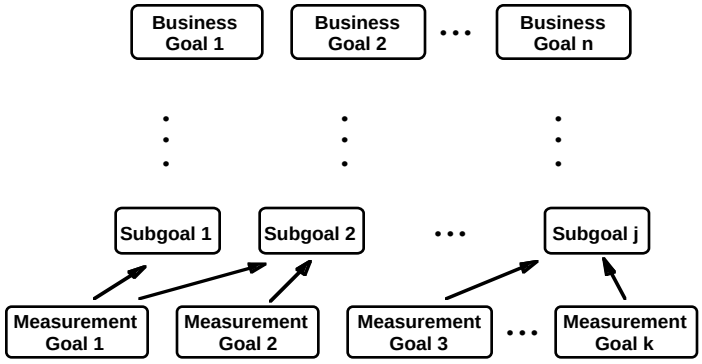


Figure 4-35: Maintaining Traceability to Business Goals—Practice

Exercise 5: Formalizing Measurement Goals

The objective of this exercise is to translate your subgoals, entities, attributes, and questions into formal measurement goals. Each measurement goal will identify an object of interest and the purpose, perspective, and environment that will guide the specific questions to be addressed. These measurement goals will provide the ground rules for your subsequent measurement activities. Figure 4-36 illustrates these objectives. The tasks are

1. Review the subgoals, questions, entities, and attributes you identified in Exercises 3 and 4.
2. Identify the activities that you propose to undertake to get the information you need.
3. Express your goals for these activities as structured statements that identify the object, purpose, perspective, environment, and constraints associated with each measurement activity.
4. Identify and record the business subgoal(s) that each measurement goal addresses.

Templates for constructing structured measurement goals and a worksheet for recording the results are presented in Appendix A.

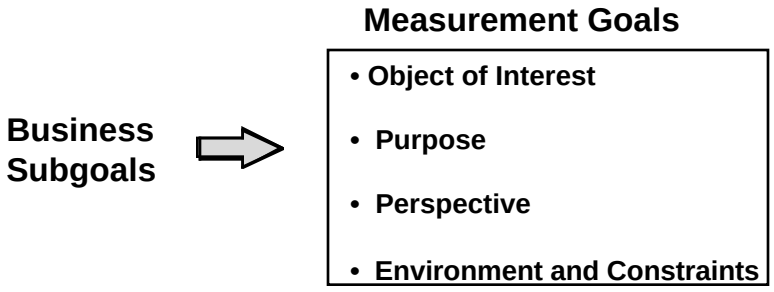


Figure 4-36: Mapping from Subgoals to Structured Statements of Measurement Goals

4.6 Step 6: Identify Quantifiable Questions and Indicators

The GQM Paradigm

The structured measurement goals constructed in Step 5 provide a solid foundation for the goal-question-metric (GQM) steps that come next. In fact, you have already completed the first step of GQM—stating your measurement goals. You are now ready to pose *quantifiable* questions and to formulate indicators that support the questions. This is Step 6.

GQM is useful because it facilitates identifying not only the precise measures required, but also the reasons why the data are being collected. The "why?" is important because it defines how the data should be interpreted, and it provides a basis for reusing measurement plans and procedures in future projects and activities [Rombach 89].

Terminology and the Way It Shapes Our Use of GQM

But first, a few words about terminology. When we talk about goals in the GQM paradigm, we mean measurement goals, not business goals. In fact, the whole purpose of Steps 1 through 4 has been to get to the point where we can formulate clearly stated measurement goals that support our business objectives. In our experience, GQM is unlikely to be productive if applied at too high a level. Trying to jump directly from high-level business goals to software measures is too big a leap. Business goals must first be decomposed and refined to a point where meaningful entities, purposes, perspectives, and environments can be identified. When goals are stated at too high a level, GQM easily leads to vaguely stated questions. Vague questions shed little light on the measures that are needed to understand the fundamental attributes of the processes and products that people manage.

Our second point about terminology is that, from now on (except in quotations), we will never use the word "metric." To us, the M in GQM stands for Measure, not Metric. The problem with "metric" is not that no one knows what it means, but that everyone thinks it means something different. Measurement, on the other hand, has a generally accepted definition. For example:

> *Measurement: the assignment of numerals to objects or events according to [a] rule [Stevens 59].*
>
> *Measurement: the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterize the attributes by clearly defined rules [Fenton 91, Fenton 95].*

This view of measurement as a mapping from the real world to a numeric or symbolic system has been a subject of study for years, and much foundational knowledge has been accumulated. See, for example, our discussion in Chapter 2 on measurement scales and

their relationships to permissible statistics. You can also find whole reference works on the subject of measurement—[Krantz 71], [Roberts 79], and [Ghiselli 81] for example. Some of these (e.g., [Ghiselli 81]) provide excellent discussions of important issues such as measurement reliability and validity—subjects that are important, but beyond the scope of this guidebook. To the best of our knowledge, there are no equivalent foundational works on the subject of "metrics." Since our goal is to see a practice of software measurement that rests on firms foundations, it seems wise to begin where the foundations are strongest. There is no reason that we know of why software measurement should be approached differently from measurement in any other domain.

Our third point is that we use the term "indicator" to mean a display of one or more measurement results that is designed to communicate or explain the significance of those results to a reader. This lets us distinguish clearly between the concept of communication and that of measure, something that users of the term "metrics" often do not do. Our use of the word "indicator" is consistent with the definitions that appear in [Baumert 92] and [PSM 96].

So, why all this fuss over terminology? As we pointed out in Section 4.5, we have found it helpful to insert an "Indicator" step in the GQM paradigm, making it GQ(I)M. We do this because seeing how measurement data will be displayed helps point to and clarify exactly what we must measure. This puts us in a better position to construct operational specifications for the data we wish collected.

GQ(I)M—Proceeding from Measurement Goals to Questions and Indicators

The ellipses in Figure 4-37 show where we are in the goal-driven measurement process.

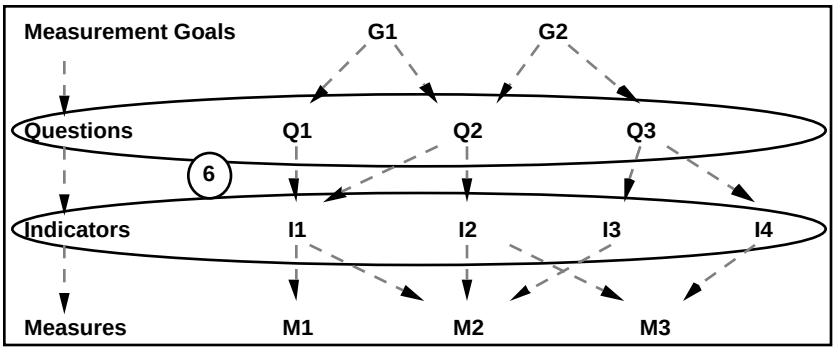


Figure 4-37: The Sixth Target—Quantifiable Questions and Indicators

When identifying questions and defining indicators, it is important to keep in mind the goal(s) you are addressing and how the measurement results will be used. In rapidly changing environments, precise numbers and elaborate statistical analyses are often less valuable than simpler answers to insightful, well-directed questions. For example, knowing that the

most costly failures are usually detected or prevented by techniques X and Y may be almost as useful as knowing the exact percentages or confidence intervals, yet far easier to determine [Rombach 89].

Step 6 is best illustrated by an example. Suppose you have the measurement goal shown in Figure 4-38.

Object of interest
- ***The software development process at plant XYZ***

Purpose
- ***Analyze the software defects introduced during development in order to identify opportunities for reducing costs and improving product quality.***

Perspective
- ***Examine defect insertion, detection, and repair from the point of view of a process improvement team.***

Environment
- ***Telephone trunking and switching. CMM Level 2 (working on Level 3). 3000 people in plant. 800 software developers. Coding is done in C and assembly. Constraints: Examine only projects completing unit testing since 1 Jan 93.***

Figure 4-38: A Measurement Goal

Some questions you might ask are

- Where are defects being found?
- Where are the defects being introduced?
- Are the right defects being fixed first?
- Where do we stand with respect to our subgoal of improving phase containment?

Some of the charts (indicators) that we have seen used to address questions like these are illustrated in Figures 4-39 through 4-43.

Examples of Indicators

Figure 4-39 gives a picture of where problems of different types are being found. Among other things, it shows that the largest set of design problems is being found during system testing. Requirements problems, on the other hand, seem to be getting discovered relatively earlier, closer to where they are introduced. Moving the detection of design problems forward may be an issue that you will want to address. Figure 4-39 also shows that nearly 10% of the total problems are not being found until after systems have been fielded, and that this includes a surprisingly large portion of the total documentation errors. These may also be symptoms that are worth attention.

Figure 4-40 shows that useful indicators come in many forms. Not all are graphs. This table shows one way of summarizing the observed performance of an organization in addressing its action items from peer reviews. It displays the current backlog as a joint distribution of severity and age. The chart suggests that high-severity items are getting taken care of first, but that the process may be a bit slower than desired. Additional information—such as product size, number of peer reviews, development status, and number of action items

completed—would be useful to help put the total number of backlogged action items in perspective.

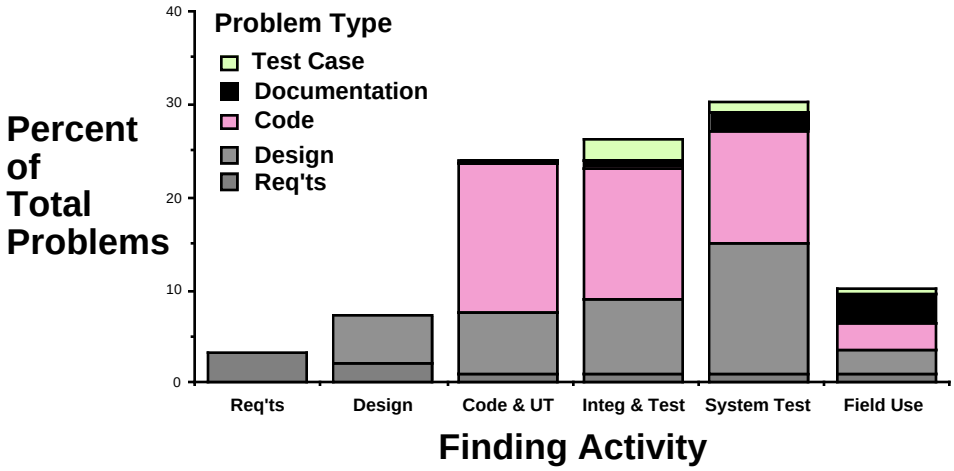


Figure 4-39: Problem Type vs. Finding Activity

Severity Levels	Number of Peer Review Action Items That Have Been Open x Days				Totals
	x ≤ 30	30 < x ≤ 60	60 < x ≤ 90	x > 90	
Severity 1	2	1			3
Severity 2	3	1	1		5
Severity 3	3	2	1	1	7
Severity 4	4	3	3	2	12
Severity 5	8	6	3	3	20
Totals	20	13	8	6	47

Figure 4-40: Age of Peer Reviews

Figure 4-41 shows one organization's perception of current relations that exist between where software faults are injected and where they are found [Carleton 94]. It illustrates clearly how having a mental model of a process helps identify measurable attributes, and how mental models can be used to set and communicate goals. Organizations operating at Level 3 of the CMM have defined processes that can often be used as a basis for mental models and graphical communications.

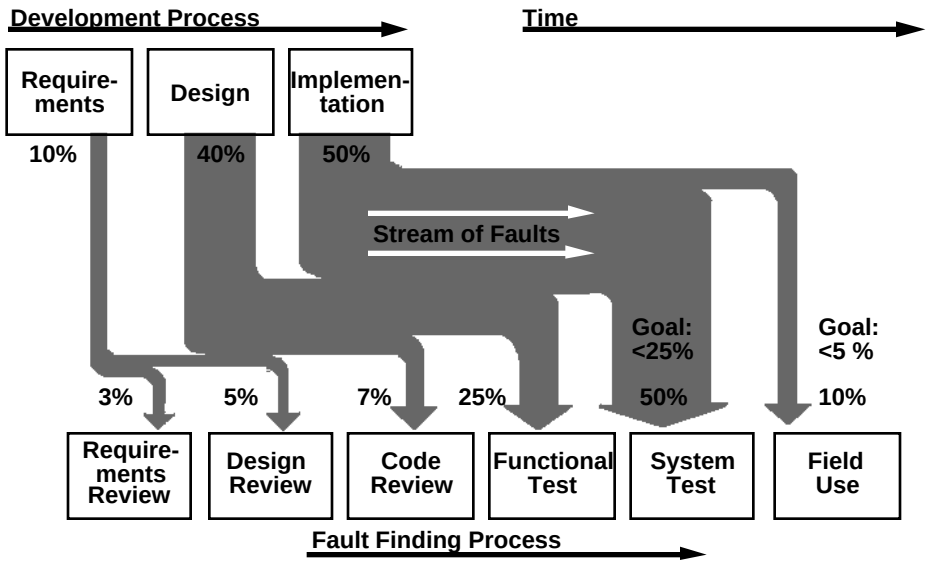
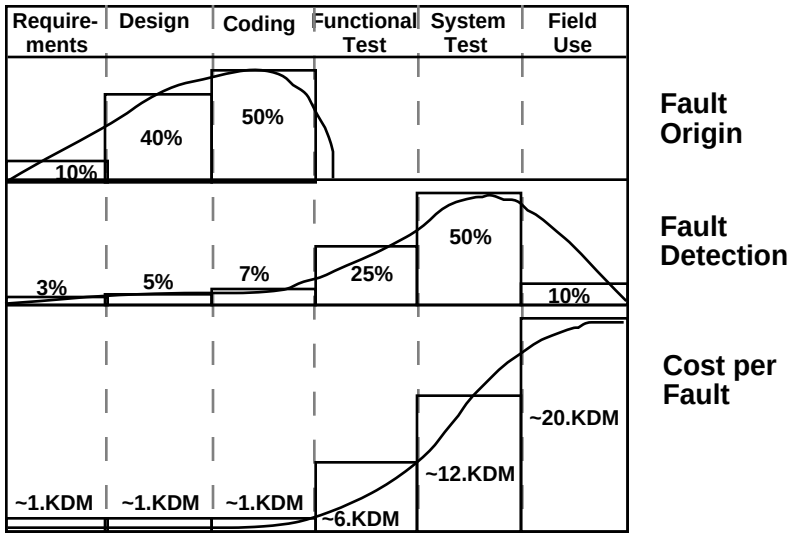


Figure 4-41: Process Models Help Us Construct Indicators—Fault Stream Analysis

Figure 4-42 sheds additional light on the fault stream analysis depicted in Figure 4-41. It translates the insertion and detection distributions into trajectories and adds information about the costs for eliminating faults. The illustration comes from a German company, so the costs are shown in thousands of Deutschmarks (KDM). Cost distributions like this, even if approximate, help us look at the economic consequences of current conditions and proposed actions. This often permits more informed decision making than is possible with simple Pareto charts.



KDM=kilo-deutsch marks

Figure 4-42: Faults as a Cost Driver

Figure 4-43 shows the same organization's perception of the forward shift in fault detection distributions that they expect to realize as they move up the maturity scale of the SEI's Capability Maturity Model [Paulk 93a]. As in Figure 4-42, the problem-insertion distribution is displayed at the top and the cost of fixing a defect is at the bottom. Rows 1–5 then depict the anticipated shifting of detection frequencies as the organization improves its maturity. These perceptions (and the economic analyses they enable) can be useful in motivating process improvement activities. They can also be used as references for interpreting status and trends in the organization's measured (actual) detection distribution.

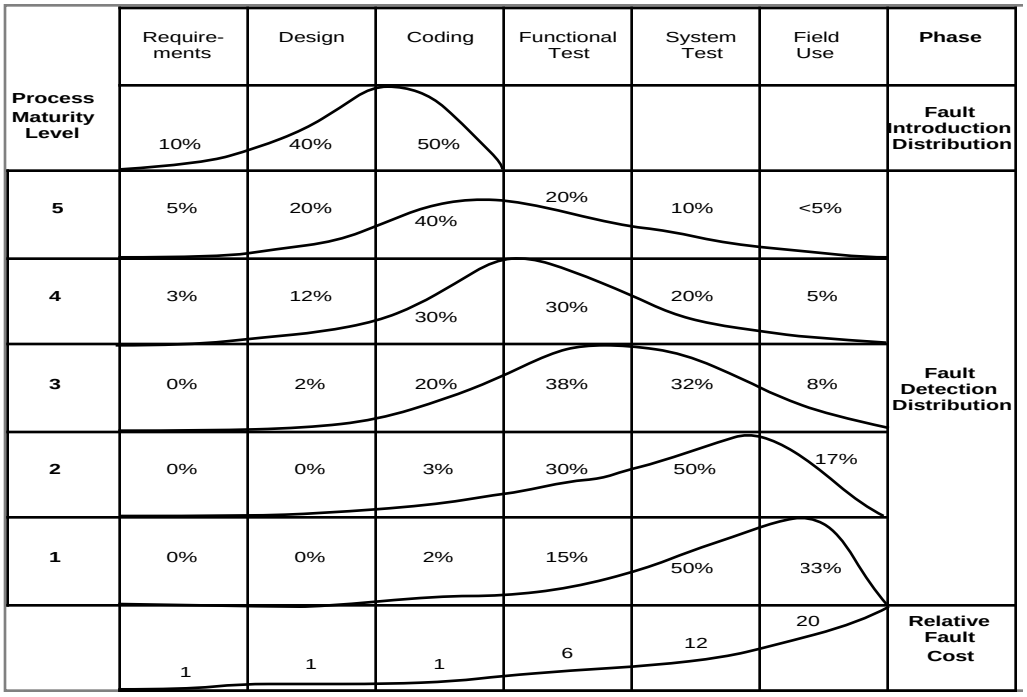


Figure 4-43: Shift in Fault Distributions as Process Maturity Increases

Validating Your Questions and Indicators

As you can see, constructing useful indicators is a highly creative process. Unfortunately, it is often easy to construct nifty indicators that mislead both the creators and their audiences. Therefore, before you leave Step 6, you should review your proposed indicators to ensure that the pictures they present validly address the questions you have asked. This is best illustrated by an example.

Suppose that you propose to collect data to examine the effects of analyst and programmer experience on development effort. You are likely to be expecting that effort (staff-hours) will decrease as the experience of the people assigned increases, as in Figure 4-44(a).

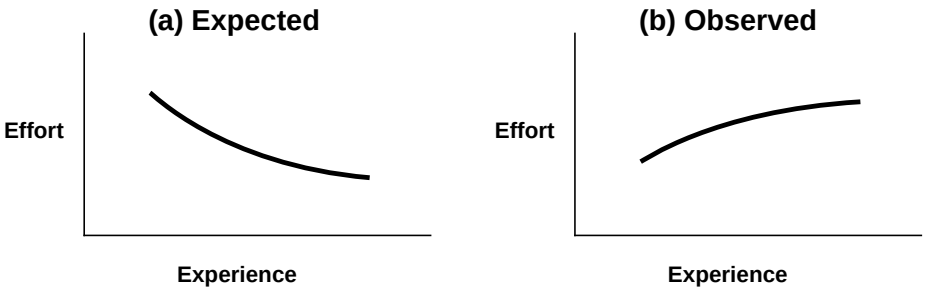


Figure 4-44: Effort Versus Experience—(a) Expected and (b) Observed

Now look ahead a bit. Suppose that after you collect your data, you find that it shows effort increasing with increasing experience, as in Figure 4-44(b)! Will you be willing to accept and act on this observation? We suspect that your answer is "No!" What, then, is wrong? What is missing?

The most likely explanation is that other things have not remained the same, and that concomitant changes in factors you have not asked about have clouded the issue. For example, if projects with higher team experience are larger in size or more technologically demanding or inflicted with overly aggressive schedules, then you might well see measured effort decrease as experience goes down. This could happen, for instance, if it is your practice to assign your less experienced people to the simplest projects.

The solution: When examining relationships between personnel experience and effort, ask also about other factors that might vary with experience. For example, "Do our team assignment practices vary with project difficulty (i.e., the need for experience)? If so, you will probably want to ask additional questions that enable you to characterize project difficulty (size, application type, reuse, schedule acceleration, etc.). Answers to these questions will better enable you to validly interpret the indicators you plot.

The moral: Envisioning *unexpected* results is an excellent way to help you refine your questions and indicators.

Exercise 6: Identifying Quantifiable Questions and Indicators

Select one of your measurement goals. Identify quantifiable questions related to this goal that you would like answered. Prepare sketches for displays (indicators) that will help you address your questions and communicate the results of your analyses to others, as shown in Figure 4-45. Prioritize the indicators and identify the ones that will be most useful to you. Repeat these steps for your other measurement goals.

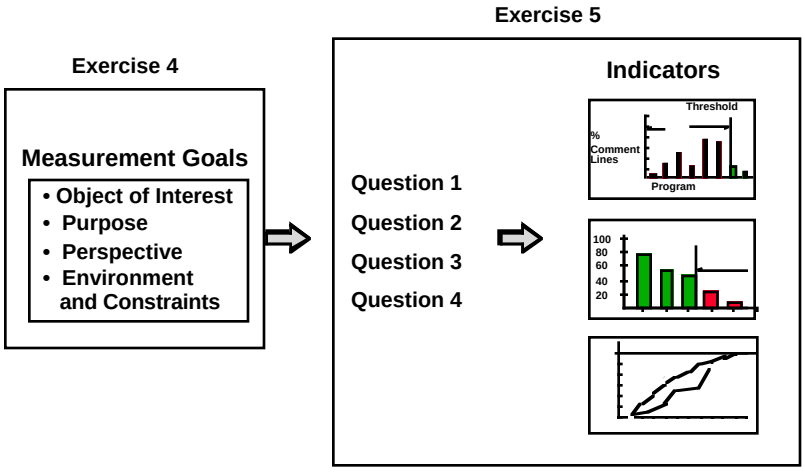


Figure 4-45: Moving from Measurement Goals to Quantifiable Questions and Indicators

A worksheet for recording your results is in the materials for Exercise 6 in Appendix A.

4.7 Step 7: Identify the Data Elements

There is measure in all things.

— Horace (Quintus Horatius Flaccus)

Developing Focused Measures (Data Elements)

With pictures of what you want to plot and display in hand, we now turn to identifying the data elements that you will have to collect to create the displays. The path you have followed to this point ensures that the data you will be collecting have clearly defined management purposes. They are not data for data's sake alone. The ellipse in Figure 4-46 shows where we are in the goal-driven measurement process.

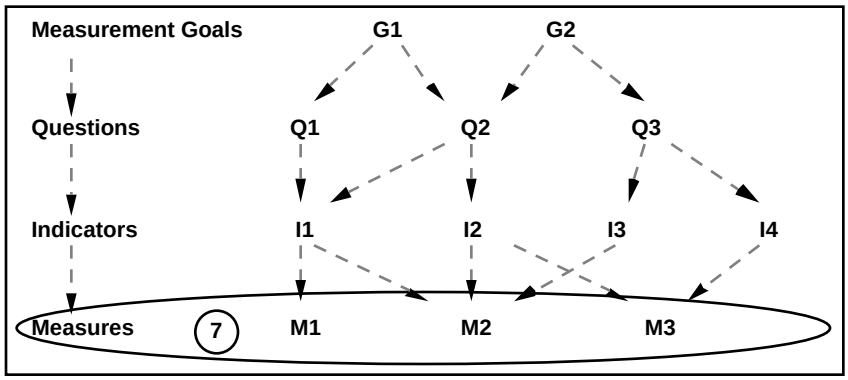


Figure 4-46: The Seventh Target—Data Elements and Measures

In this step and the next, you have two things to do:

1. Identify the data elements.
2. Define how the measures will be collected.

We deal with the first task here. Task 2 is the subject of Section 4-8.

Task 1 is highlighted by the bold ellipse in Figure 4-47. To complete this task, you simply make a list of all the data elements that you will need to collect to construct your indicators. As you do this, it will help in the prioritizing to come if you map these data elements back to the indicators they serve. Figure 4-48 illustrates this idea. Among other things, you will be interested in seeing which measures are able to serve multiple needs.

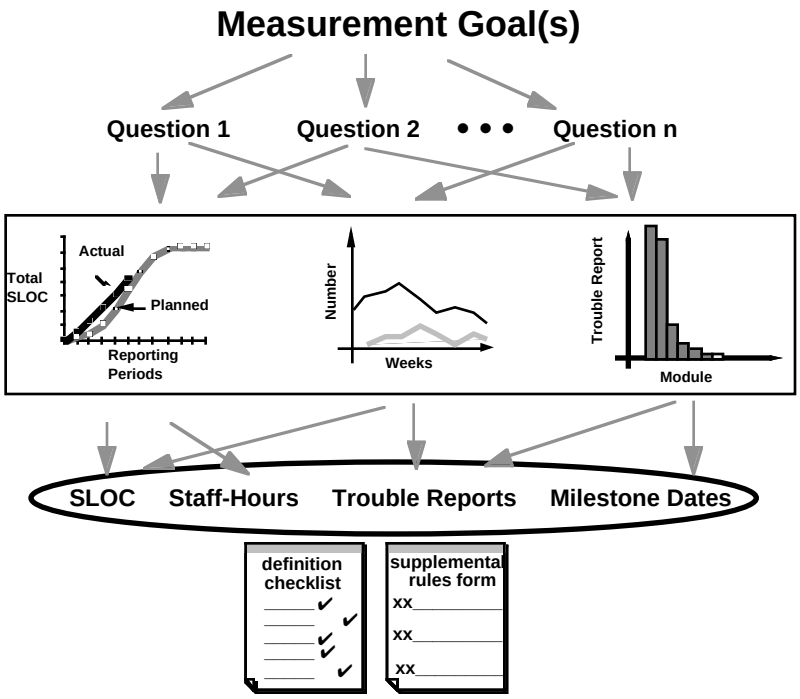


Figure 4-47: Using Indicators to Identify Data Elements

	Ind-1	Ind-2	Ind-3	Ind-4
Source lines of code	✔		✔	
Development staff-hours		✔		✔
Defects found in testing	✔	✔		
Milestone completion dates for…			✔	
Number of work units completed				✔

Measures that serve multiple indicators may have greater value.

Figure 4-48: Taking Inventory—Mapping Measures to the Indicators They Serve

Exercise 7: Identifying the Data Elements to Be Collected

Review your results (the questions and indicators) from Exercise 6. Identify the data elements that you will have to collect to construct your indicators. List these data elements and map them back to your indicators, as illustrated in Figures 4-48 and 4-49.

Instructions for this exercise and a worksheet for recording your results are presented in Appendix A.

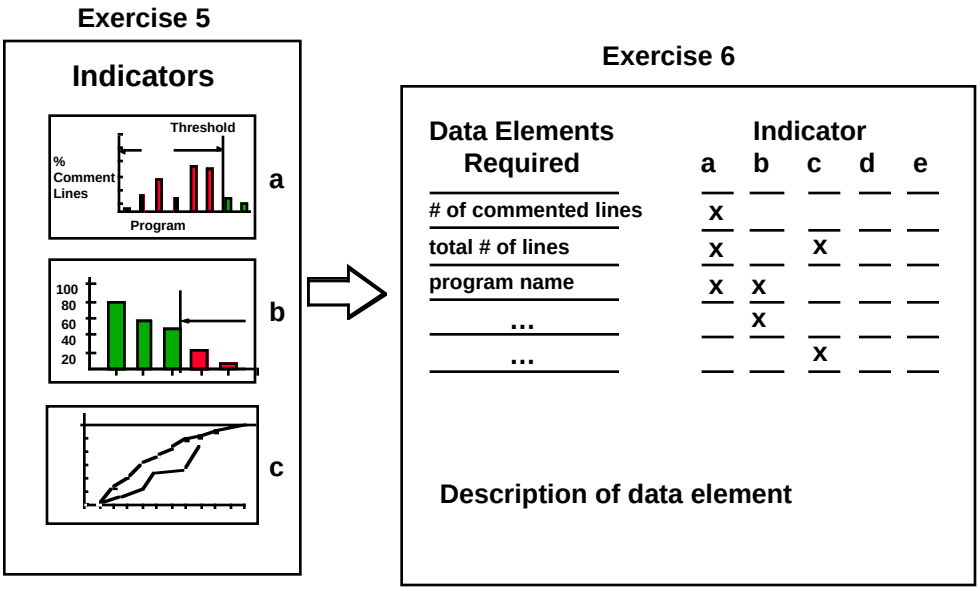


Figure 4-49: Identifying Data Elements and Mapping Them to Needs

4.8 Step 8: Define Your Measures

It is a common human weakness to imagine that because a metric is intended to measure something, it actually does!

— source unknown

In the opinion of many people in industry, there is nothing more important for the transaction of business than use of operational definitions. It could also be said that no requirement of industry is so much neglected.

— W. Edwards Deming, 1986

Data without definitions are indistinguishable from numbers.

– source unknown

Now that you have identified your measures, you must define them. Names for measures alone do not suffice. You must be able to tell others exactly how each measure is obtained, so that they can interpret the values correctly. The bold ellipse in Figure 4-50 shows schematically where we are in the goal-driven measurement process.

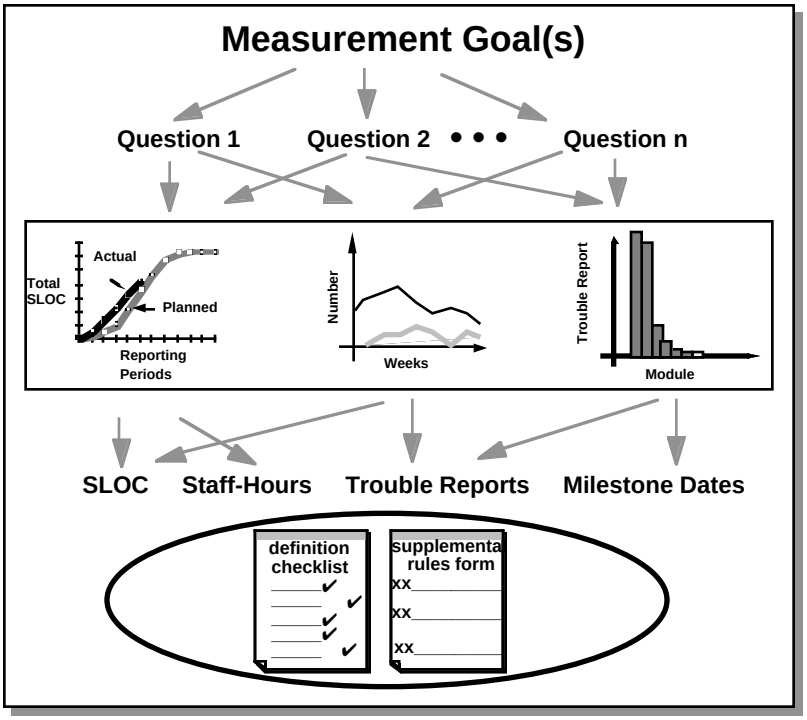


Figure 4-50: The Eighth Target—Defined Measures

The Role of Structured Frameworks

Measurement of software products and processes is not new. Some organizations have been measuring for years. As a minimum, we have all dealt with development schedules. Many organizations have also recorded effort expenditures, perhaps weekly, if for no other reason than to ensure that employees get paid. Some organizations use this data in conjunction with measures of software artifacts to track and control progress, especially if developing products under contract. Some of these organizations have structured estimating processes that use empirical models to help them translate records from past projects into bids, proposals, and plans for future work.

But despite all this measurement activity, few in the software industry would call measurement a success story. This is especially true when we attempt to use data that were collected or reported by someone else. Some reasons for our lack of success are

- *Different users of measurement data have different needs*. Data collected for one purpose may not be suitable for another, because the rules used for collecting the data are inconsistent with the ways others want to use the data.
- *Different organizations have different established practices*. In many cases these practices have sound reasons behind them and should not be changed. Moreover, it may be difficult and often impractical to change the way an organization collects data, just to satisfy an external need.
- *Unambiguous communication of measurement results is inherently difficult*. Even if someone understands perfectly well how their data are collected, it is not easy for them to communicate adequate descriptions of their operational rules to others. These rules may be complex, and they may never have been stated explicitly.
- *Structured methods for communicating measurement results seldom exist*. What you think you hear is often not what they meant to say. This, in a way, restates the ambiguity point just made, but frames it so as to suggest a potential solution.

Our proposal, then, is to use structured frameworks to help us define, implement, and communicate operational definitions for software measures. The primary issue is not whether a definition for a measure is correct, but that everyone understand—completely—what the measured values represent. Only then can we expect people to collect values consistently, and only then can others interpret the results correctly and apply them to reach valid conclusions.

Communicating clear and unambiguous definitions is not easy. Having structured methods for identifying all the rules that are used to make and record measurements can be very helpful in ensuring that important items of information do not go unmentioned. When designing structured methods for defining measures, you should keep in mind that things that do not matter to one user are often important to another. This means that measurement definitions—and frameworks for recording the definitions—usually become larger and more

encompassing than the definitions most organizations have traditionally used. This is all the more reason to have a structured approach. Definition deals with details, and structured methods help ensure that all details get addressed and recorded.

What Makes a Definition Operational?

> *An operational definition puts communicable meaning into a concept.…Without operational definition, a specification is meaningless.*
>
> *— W. Edwards Deming, 1986*

> *Recall that measurement implies the existence of rules for the assignment of numbers, labels, vectors, etc. Where the rules are unknown or nonexistent we do not consider this to be measurement…*
>
> *— Martin Shepperd and Darrel Ince, 1993*

Operational definitions tell users *how* data are collected. As Deming said often in his lectures, "If you change the method, you change the result!" When users of data do not know how the data were collected, they easily make invalid assumptions. This leads to incorrect interpretations, improper analyses, and erroneous decisions.

Operational definitions must satisfy two important criteria:

- *Communication:* Will others know what has been measured, how it was measured, and what has been included and excluded?
- *Repeatability:* Could others, armed with the definition, repeat the measurements and get essentially the same results?

These criteria are closely related. In fact, if you cannot communicate *exactly* what was done to collect a set of data, you are in no position to tell someone else how to do it. Far too many organizations propound measurement definitions without first determining what users of the data need to know about the measured values in order to use them intelligently. It is no surprise, then, that measurements are often collected inconsistently and at odds with users' needs. When it comes to implementation, rules such as "Count all noncomment, nonblank source statements" are open to far too many interpretations to provide repeatable results.

Stephen Kan [Kan 95] provides an illuminating example: Suppose you are interested in measuring the height of school children, say between the ages of 3 and 12. If you define height simply as standing height, measured in inches, you will get one set of results. You will get quite another if you define height to be standing height (exclusive of piled up hair and

hats), measured in inches, with shoes off, on a recently calibrated school scale, by a trained nurse, between 8 and 9 o'clock in the morning. Which set of height measurements would you expect to be more reliable, more consistent across schools, and more fit for informed interpretation?

Unfortunately, the software world abounds with nonoperational pseudodefinitions that are even more loosely stated than the definition for height in Kan's example. For instance, "Our measure for software size is the number of noncomment, nonblank, executable source statements" and "Our measure for cost is the total direct labor hours from the start of the project" are two examples of definitions that are so ill-specified that unrepeatable results and misinterpretations of reported values are almost guaranteed. We will illustrate some checklist-based techniques for making definitions like these more operational shortly.

Communication Precedes Repeatability

Although communicating measurement definitions in clear, unambiguous terms requires effort, there is good news as well. When someone can describe exactly what has been collected, it is easy to turn the process around and say, "Please do that again." Moreover, you can give the description to someone else and say, "Please use this as your definition, but with these changes." In short, when we can communicate clearly what we *have* measured, we have little trouble creating repeatable rules for collecting future data.

The moral: One should not attempt to tell others how to measure until they have clear structures for describing the data that they use today.

Examples of Operational Definitions

> *Before we can assign numbers to our observations, we must understand the process by which we obtained them in the first place.*
>
> *— Gerald Weinberg, 1993*

> *The only communicable meaning of a word, prescription, instruction, specification, measure, attribute, regulation, law, system, edict is the record of what happens on application of a specified operation or test.*
>
> *— W. Edwards Deming, 1986*

> *…if there is no criterion for determining whether a given numeral should or should not be assigned, it is not measurement.*
>
> *— S. S. Stevens, 1959*

The three SEI reports that were cited earlier provide frameworks and examples for constructing operational definitions for some frequently used software measures [Park 92, Goethert 92, Florac 92]. These frameworks are based on checklists, supplemented by forms that summarize operational information not amenable to checklist treatment. The central theme in the checklists lies in stating exactly *what* is included in—and excluded from—reported results. The supplemental forms describe *how* the inclusions and exclusions were (or are to be) accomplished. These operational practices should be part of an operational definition, since they affect the way measured results should be interpreted.

Although the first (and most important) use of definition checklists and supplemental forms is to let users of data know exactly how data were obtained, the same framework can be used to specify how future measurements are to be made. The latter "let me tell you what to do" approach is the one we usually see in software organizations, but without visible structures for ensuring that the measurement instructions will be interpreted and executed consistently by all who collect the data.

The references we cited [Park 92, Goethert 92, Florac 92] show how checklists can be used to construct and communicate operational definitions for software measures. Examples 1 through 6 on the pages that follow use excerpts from these checklists to illustrate the kinds of issues that must be identified, resolved, and communicated if measurement definitions are to be consistently employed and correctly interpreted. You should ensure that your organization goes to equivalent levels of detail when defining each software measure it uses.

Example 1: Counts of Source Statements

> *There's no sense being precise about something when you don't even know what you're talking about.*
>
> *— John von Neumann*

Figure 4-51 shows the first two pages of a checklist for one of the software size definitions illustrated in [Park 92]. This checklist makes explicit what many people mean when they define their size measure to be noncomment, nonblank, source statements (NCNBSS). As you can see, the definition is quite detailed. But without this level of detail, not everyone will have the same understanding. It then becomes easy either to measure incorrectly or to misinterpret measured results.

The full checklist in [Park 92] contains additional pages that spell out the specific rules used with different programming languages—for example, do counts include **null** statements, **begin** and **end** labels, curly braces on lines by themselves, **with** and **use** clauses, **continue** statements, or keywords such as **interface**, **implementation**, **forward**, **procedure division**, and **end declaratives**. Knowledge of these practices is needed for consistent counting and for correctly interpreting the data that are collected.

At first glance, the source statement counting checklist may seem excessively detailed. But the issues that it makes visible are exactly those that must be dealt with by the people who build or operate measurement tools like source code counters. If your definitions do not get down to details like these, you will essentially be saying, "I don't care how you do it—you make the decisions!" Tool builders and users will then do something that seems reasonable to them, but you (and others) will never know what that is, and inconsistency and ambiguity will abound. You may end up with numbers, but they will not be measures.

Definition Checklist for Source Statement Counts

Definition name: ***Physical Source Lines of Code (basic definition)*** Date: ***8/7/92***

Originator: ***SEI***

Measurement unit:		**Physical source lines** ✔					
		Logical source statements					
Statement type	**Definition** ✔		**Data array**			**Includes**	**Excludes**
	When a line or statement contains more than one type, classify it as the type with the highest precedence.						
1 Executable			**Order of precedence ->**		1	✔	
2 Nonexecutable							
3 Declarations					2	✔	
4 Compiler directives					3	✔	
5 Comments							
6 On their own lines					4		✔
7 On lines with source code					5		✔
8 Banners and nonblank spacers					6		✔
9 Blank (empty) comments					7		✔
10 Blank lines					8		✔
11							
12							
How produced	**Definition** ✔		**Data array**			**Includes**	**Excludes**
1 Programmed						✔	
2 Generated with source code generators						✔	
3 Converted with automated translators						✔	
4 Copied or reused without change						✔	
5 Modified						✔	
6 Removed							✔
7							
8							
Origin	**Definition** ✔		**Data array**			**Includes**	**Excludes**
1 New work: no prior existence						✔	
2 Prior work: taken or adapted from							
3 A previous version, build, or release						✔	
4 Commercial, off-the-shelf software (COTS), other than libraries						✔	
5 Government furnished software (GFS), other than reuse libraries						✔	
6 Another product						✔	
7 A vendor-supplied language support library (unmodified)							✔
8 A vendor-supplied operating system or utility (unmodified)							✔
9 A local or modified language support library or operating system						✔	
10 Other commercial library						✔	
11 A reuse library (software designed for reuse)						✔	
12 Other software component or library						✔	
13							
14							
Usage	**Definition** ✔		**Data array**			**Includes**	**Excludes**
1 In or as part of the primary product						✔	
2 External to or in support of the primary product							✔
3							

Figure 4-51: A Checklist-Based Definition for Source Lines of Code

Definition name: ***Physical Source Lines of Code (basic definition)***

	Includes	Excludes
Delivery — Definition ☑ — Data array ☐		
1 Delivered		
2 Delivered as source	✔	
3 Delivered in compiled or executable form, but not as source	✔	
4 Not delivered		
5 Under configuration control		✔
6 Not under configuration control		✔
7		
Functionality — Definition ☑ — Data array ☐	Includes	Excludes
1 Operative	✔	
2 Inoperative (dead, bypassed, unused, unreferenced, or unaccessed)		
3 Functional (intentional dead code, reactivated for special purposes)	✔	
4 Nonfunctional (unintentionally present)		✔
5		
6		
Replications — Definition ☑ — Data array ☐	Includes	Excludes
1 Master source statements (originals)	✔	
2 Physical replicates of master statements, stored in the master code	✔	
3 Copies inserted, instantiated, or expanded when compiling or linking		✔
4 Postproduction replicates—as in distributed, redundant, or reparameterized systems		✔
5		
Development status — Definition ☑ — Data array ☐	Includes	Excludes
Each statement has one and only one status, usually that of its parent unit.		
1 Estimated or planned		✔
2 Designed		✔
3 Coded		✔
4 Unit tests completed		✔
5 Integrated into components		✔
6 Test readiness review completed		✔
7 Software (CSCI) tests completed		✔
8 System tests completed	✔	
9		
10		
11		
Language — Definition ☐ — Data array ☑	Includes	Excludes
List each source language on a separate line.		
1 ***Separate totals for each language***	✔	
2 Job control languages		
3		
4 Assembly languages		
5		
6 Third generation languages		
7		
8 Fourth generation languages		
9		
10 Microcode		
11		

Figure 4-51: A Checklist-Based Definition for Source Lines of Code (Page 2)

Example 2: Defining and Using Arrayed Data—A Project Tracking Example

When measuring software attributes, we often need more than just single numbers. Instead, we almost always want to identify and understand relations within and among sets of entities or attributes. Figure 4-52 is an interesting example, based on size measures. It shows what one organization observed as it tracked its progress on a development project. The job was started with the expectation that nearly half the product could be constructed from existing components. As time passed, they found that many of the components they had planned to reuse failed to meet the requirements of the product. This led to extensive, unplanned modifications and to additional development effort and cost. Early warnings like those made visible by Figure 4-52 were used to alert managers that action was needed.

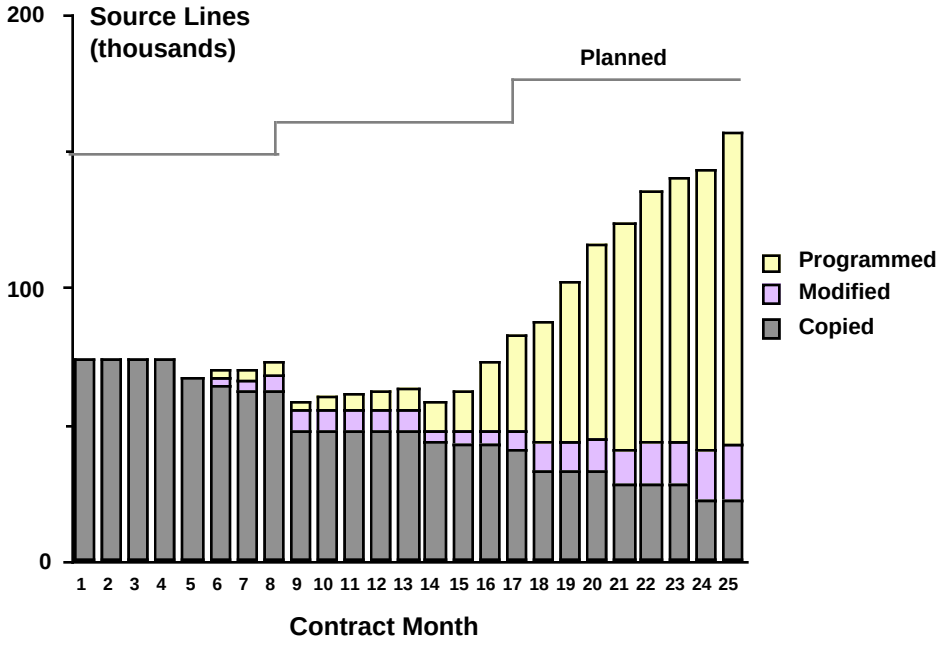
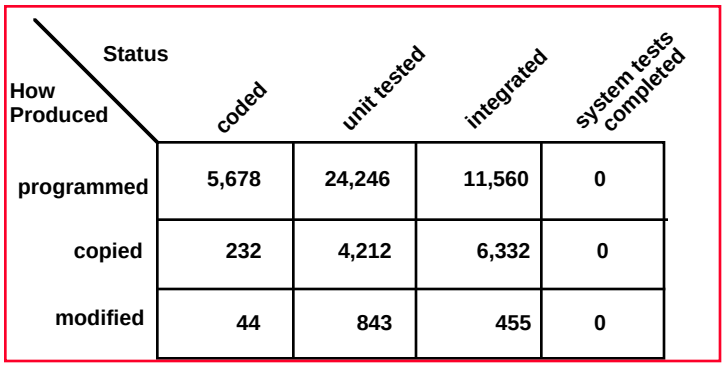


Figure 4-52: The Case of Disappearing Reuse

Charts like Figure 4-52 cannot be created with single measures alone. Instead, we need arrays of data, collected periodically. A 3-by-4 array of the kinds of information we often want is illustrated in Figure 4-53. Because arrays are often used to relate data within (or among) software measures, effective frameworks for defining software measures should provide means for defining the elements of these arrays.



How Produced \ Status	coded	unit tested	integrated	system tests completed
programmed	5,678	24,246	11,560	0
copied	232	4,212	6,332	0
modified	44	843	455	0

Figure 4-53: A Summary of Arrayed Data for Tracking Development Progress

Figure 4-54 shows one way to do this. Here two sections of the SEI size checklist have been used to specify exactly which data elements will be collected to construct the table illustrated in Figure 4-53. The checks in the "data array" boxes say that the counts for three "how produced" classes are to be arrayed against counts for "development status." ([Florac 92] illustrates an alternative way to specify arrays.) Figure 4-54 shows only a portion of the checklist. We must still use the sections that describe the rules for the remaining attributes, since including or excluding values for those attributes could affect measured results.

How produced — Definition — Data array ✔	Includes	Excludes
1 Programmed	✔	
2 Generated with source code generators		✔
3 Converted with automated translators		✔
4 Copied or reused without change	✔	
5 Modified	✔	
6 Removed		✔
7		
8		

Development status — Definition — Data array ✔	Includes	Excludes
Each statement has one and only one status, usually that of its parent unit.		
1 Estimated or planned		✔
2 Designed		✔
3 Coded	✔	
4 Unit tests completed	✔	
5 Integrated into components	✔	
6 Test readiness review completed		✔
7 Software (CSCI) tests completed		✔
8 System tests completed	✔	
9		
10		
11		

Figure 4-54: Using a Definition Checklist to Specify the Collection of Arrayed Data

Example 3: Effort Measures

Figure 4-55 on the next three pages shows how a checklist similar to the size checklist can be used to define measures of development effort. The information provided by the completed checklist is typical of the degree of understanding that cost estimators should have as a basis for planning and estimating future projects. Note that the checks in the rightmost column ask for not just one measured value, but for breakouts of total effort into regular time and overtime work, effort applied for each major functional component, effort for integrating the results into builds (or releases), and effort for system-level development activities. The important thing that the checklist does for each breakout is to make explicit, via the other attributes, exactly what is included in and excluded from each breakout.

The format of the checklist in Figure 4-55 does not have explicit facilities like the data array boxes in the size checklist for defining arrays of measured values. Instead, it has a third column (Report totals) that was designed to be used to designate one-dimensional arrays within individual attributes. Our experience at the time that the checklist was created suggested that multidimensional arrays of effort data were seldom used or needed. If you do want multidimensional arrays, you can do one of two things. Either

(a) add a fourth column, as in the quality checklist in Example 5, or

(b) use symbols (a, b, c, etc.) in the "Report totals" column to define structures where one attribute's values will be arrayed against those of another attribute.

Staff-Hour Definition Checklist

Definition Name: **Total System Staff-Hours For Development**

Date: **7/28/92**

Originator:

Page: 1 of 3

	Totals include	Totals exclude	Report totals
Type of Labor			
Direct	✔		
Indirect		✔	
Hour Information			
Regular time			✔
Salaried	✔		
Hourly	✔		
Overtime			✔
Salaried			
Compensated (paid)	✔		
Uncompensated (unpaid)	✔		
Hourly			
Compensated (paid)	✔		
Uncompensated (unpaid)	✔		
Employment Class			
Reporting organization			
Full time	✔		
Part time	✔		
Contract			
Temporary employees	✔		
Subcontractor working on task with reporting organization	✔		
Subcontractor working on subcontracted task	✔		
Consultants	✔		
Labor Class			
Software management			
Level 1	✔		
Level 2	✔		
Level 3		✔	
Higher		✔	
Technical analysts & designers			
System engineer	✔		
Software engineer/analyst	✔		
Programmer	✔		
Test personnel			
CSCI-to-CSCI integration	✔		
IV&V	✔		
Test & evaluation group (HW-SW)	✔		
Software quality assurance	✔		
Software configuration management	✔		
Program librarian	✔		
Database administrator	✔		
Documentation/publications	✔		
Training personnel	✔		
Support staff	✔		

Figure 4-55: A Checklist-Based Definition for Measuring Effort Expended

Definition Name: **Total System Staff-Hours For Development** Page: 2 of 3

	Totals include	Totals exclude	Report totals
Activity			
Development			
Primary development activity	✔		
Development support activities			
Concept demo/prototypes	✔		
Tools development, acquisition, installation, & support	✔		
Non-delivered software & test drivers	✔		
Maintenance			
Repair		✔	
Enhancements/major updates		✔	
Product-Level Functions			
CSCI-Level Functions (Major Functional Element)			✔
Software requirements analysis	✔		
Design			
Preliminary design	✔		
Detailed design	✔		
Code & development testing			
Code & unit testing	✔		
Function (CSC) integration and testing	✔		
CSCI integration & testing	✔		
IV&V	✔		
Management	✔		
Software quality assurance	✔		
Configuration management	✔		
Documentation	✔		
Rework			
Software requirements	✔		
Software implementation			
Re-design	✔		
Re-coding	✔		
Re-testing	✔		
Documentation	✔		
Build-Level Functions (Customer Release)			✔
(Software effort only)			
CSCI-to-CSCI integration & checkout	✔		
Hardware/software integration and test	✔		
Management	✔		
Software quality assurance	✔		
Configuration management	✔		
Documentation	✔		
IV&V			

Figure 4-55: A Checklist-Based Definition for Measuring Effort Expended (Page 2)

Definition Name: **Total System Staff-Hours For Development**	Page: 3 of 3		
Product-Level Functions continued	**Totals include**	**Totals exclude**	**Report totals**
System-Level Functions			✔
(Software effort only)			
System requirements & design			
System requirements analysis	✔		
System design	✔		
Software requirements analysis	✔		
Integration, test, & evaluation			
System integration & testing	✔		
Testing & evaluation	✔		
Production and deployment		✔	
Management	✔		
Software quality assurance	✔		
Configuration management	✔		
Data	✔		
Training			
Training of development employees	✔		
Customer training		✔	
Support	✔		

Figure 4-55: A Checklist-Based Definition for Measuring Effort Expended (Page 3)

Example 4: Milestones and Schedules

Figure 4-56 shows how a checklist can be used to define the milestones for which calendar dates are to be reported. The codes in the rightmost column of the checklist show the criteria that are used for determining when each milestone will be considered met. These codes are explained at the bottom of the checklist pages.

Schedule Checklist
Part A: Date Information

Date: _____
Originator: _____
Page 1 of 3

Project will record planned dates: Yes ✔ No _____
If Yes, reporting frequency: Weekly _____ Monthly ✔ Other: _____
Project will record actual dates: Yes ✔ No _____
If Yes, reporting frequency: Weekly _____ Monthly ✔ Other: _____

Number of builds _____

Milestones, Reviews, and Audits	Include	Exclude	Repeat each build	Relevant dates reported*
System-Level				
System requirements review		✔		
System design review		✔		
CSCI-Level				
Software specification review	✔			**2,3,6**
Preliminary design review	✔			**2,3,6**
Critical design review	✔		✔	**2,3,6**
Code complete	✔		✔	**1**
Unit test complete	✔		✔	**6**
CSC integration and test complete	✔		✔	**5**
Test readiness review	✔		✔	**3**
CSCI functional & physical configuration audits	✔		✔	**3**
System-Level				
Preliminary qualification test	✔			**3**
Formal qualification test	✔			**3**
Delivery & installation		✔		
Other system-level: Delivery to prime contractor	✔			**3**

*Key to indicate "relevant dates reported" for reviews and audits
1 - Internal review complete
2 - Formal review with customer complete
3 - Sign-off by customer
4 - All high-priority action items closed
5 - All action items closed
6 - Product of activity/phase placed under configuration management
7 - Inspection of product signed off by QA
8 - QA sign-off
9 - Management sign-off
10 - _____
11 - _____

Figure 4-56: A Checklist-Based Definition for Defining Schedule Milestones

Part A: Date Information (cont.)

Deliverable Products	Include	Exclude	Repeat each build	Relevant dates reported*
System-Level				
Preliminary system specification		✔		
System/segment specification		✔		
System/segment design document		✔		
Preliminary interface requirements spec.		✔		
Interface requirements specification		✔		
Preliminary interface design document	✔			**3**
Interface design document	✔			**1,3,5,6**
Software development plan	✔			**3,5,6**
Software test plan	✔			**3,5,6**
Software product specification(s)		✔		
Software user's manual		✔		
Software programmer's manual		✔		
Firmware support manual		✔		
Computer resources integrated support doc.		✔		
Computer system operator's manual	✔			**1,6**
CSCI-Level				
Preliminary software requirements spec(s)	✔			**3**
Software requirements specification(s)	✔			**1,3,5,6**
Software preliminary design document(s)	✔			**1,3,5,6**
Software (detailed) design document(s)	✔		✔	**1,3,5,6**
Software test description(s) (cases)	✔		✔	**1,3,5,6**
Software test description(s) (procedures)	✔		✔	**1,3,5,6**
Software test report(s)	✔		✔	**3,7**
Source code	✔		✔	**1,2,3,6,7**
Software development files		✔		
Version description document(s)		✔		

*Key to indicate "relevant dates reported" for deliverable products

1 - Product under configuration control
2 - Internal delivery
3 - Delivery to customer
4 - Customer comments received
5 - Changes incorporated
6 - Sign-off by customer
7 - IV&V sign-off
8 -

Figure 4-56: A Checklist-Based Definition for Defining Schedule Milestones (Page 2)

There is one important milestone issue that these examples do not address: the criteria that are used to determine when the project starts. If you use or adapt forms like Figure 4-56 for schedule checklists in your organization, you will want to add a means for defining and recording the entry criteria for the beginning of projects. Then, when project costs are recorded for all work done from the start to the end of a project, you will know exactly what work is encompassed.

Example 5: Counts of Problems and Defects

Figure 4-57 on this page and the next shows a structured method for defining the data to be extracted from a problem- or defect-tracking system. The *Include* and *Exclude* columns address issues that collectors of data must pin down to ensure that the counts they get are what they want. They also make explicit the measurement practices that users of problem counts need to know about to use the information correctly. The checklist gives an orderly way for addressing the issues and communicating the measurement rules that are used.

Problem Status	**Include**	**Exclude**	**Value Count**	**Array Count**
Open	✔		✔	
Recognized				✔
Evaluated				✔
Resolved				✔
Closed	✔		✔	
Problem Type	**Include**	**Exclude**	**Value Count**	**Array Count**
Software defect				
Requirements defect	✔		✔	
Design defect	✔		✔	
Code defect	✔		✔	
Operational document defect	✔		✔	
Test case defect		✔		
Other work product defect		✔		
Other problems				
Hardware problem		✔		
Operating system problem		✔		
User mistake		✔		
Operations mistake		✔		
New requirement/enhancement		✔		
Undetermined				
Not repeatable/Cause unknown		✔		
Value not identified		✔		
Uniqueness	**Include**	**Exclude**	**Value Count**	**Array Count**
Original	✔			
Duplicate		✔	✔	
Value not identifed		✔		
Criticality	**Include**	**Exclude**	**Value Count**	**Array Count**
1st level (most critical)	✔			✔
2nd level	✔			✔
3rd level	✔			✔
4th level	✔			✔
5th level	✔			✔
Value not identified	✔			
Urgency	**Include**	**Exclude**	**Value Count**	**Array Count**
1st (most urgent)	✔			
2nd	✔			
3rd	✔			
4th	✔			
Value not identified	✔			

Figure 4-57: A Checklist-Based Definition for Counting Defects

Finding Activity	Include	Exclude	Value Count	Array Count
Synthesis of				
Design		✔		
Code		✔		
Test procedure		✔		
User publications		✔		
Inspections of				
Requirements		✔		
Preliminary design		✔		
Detailed design		✔		
Code		✔		
Operational documentation		✔		
Test procedures		✔		
Formal reviews of				
Plans		✔		
Requirements		✔		
Preliminary design		✔		
Critical design		✔		
Test readiness		✔		
Formal qualification		✔		
Testing				
Planning		✔		
Module (CSU)		✔		
Component (CSC)		✔		
Configuration item (CSCI)		✔		
Integrate and test		✔		
Independent verif. and valid.		✔		
System	✔			
Test and evaluate		✔		
Acceptance		✔		
Customer support				
Production/deployment		✔		
Installation		✔		
Operation		✔		
Undetermined				
Value not identified		✔		
Finding Mode	**Include**	**Exclude**	**Value Count**	**Array Count**
Static (non-operational)	✔			
Dynamic (operational)	✔			
Value not identified	✔			

Figure 4-57: A Checklist-Based Definition for Counting Defects (Page 2)

Formats for checklists like those in Figure 4-57 should be tailored to the problem-tracking process that is used within your organization. These processes and the terms they employ vary from organization to organization. You should make sure that the checklists you use to define problem and defect counting fit your needs. This is true for other measures as well.

Example 6: Defining Your Terms—What Does "Open" Mean When Used to Describe Problems or Defects?

When constructing rules for counting problems and defects, you should be sure to define your terms. This is especially true when terms or the meanings of terms can vary from time to time or from organization to organization. Once again, checklists can provide useful structures. Figure 4-59 shows an example.

To understand Figure 4-59, a sketch of the scenario may help. This scenario is shown in Figure 4-58 (yet another example of a mental model). In this scenario, problem reports are classified as recognized (i.e., open) when certain minimum criteria are met. Subsequently, when additional criteria are satisfied, they are reclassified as evaluated. Then they go into a problem resolution process, which ends with each problem report assigned to one of three resolved states—resolved (nondefect-oriented), resolved (defect-oriented), or resolved (duplicate). The problem report is then closed.

The checklist in Figure 4-59 lists the states and identifies the exit criteria that problems or defects must satisfy to pass to the next state [SEI 96, Florac 92].

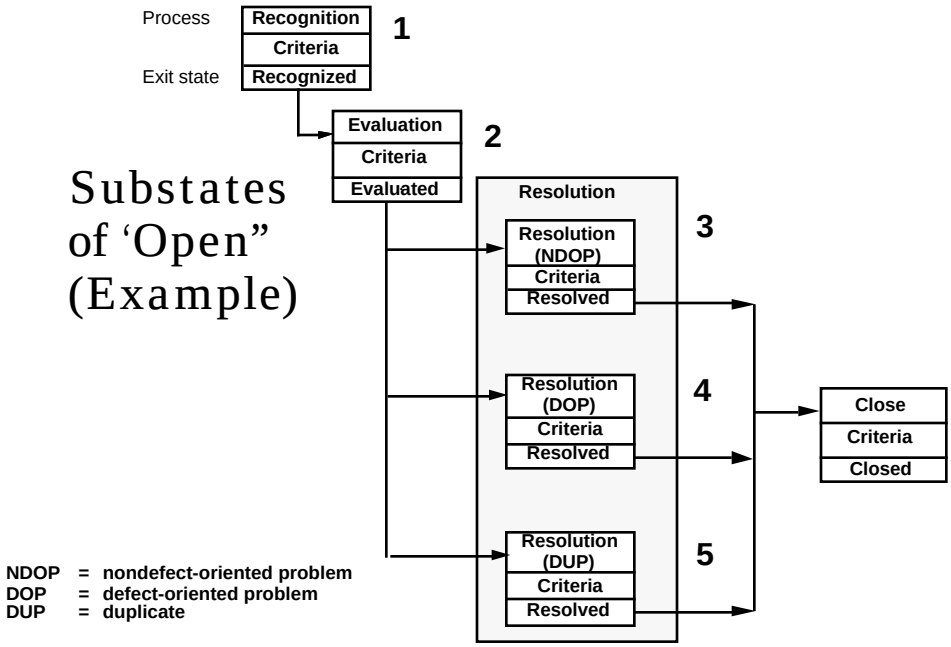


Figure 4-58: A Simple Process Model for Defect Tracking

I. Open and close criteria

II. Identifying substates

III. Defining substate criteria

IV. Identifying transition rules

Problem Status Definition Rules

Product ID: Example — **Status Definition ID: Customer probs**

Finding Activity ID: Customer Support — **Definition Date: 06/30/92**

Section I

When is a problem considered to be Open? A problem is considered to be Open when all the attributes checked below have a valid value:		When is a problem considered to be Closed? A problem is considered to be Closed when all the attributes checked below have a valid value:	
✔	Software Product Name or ID		Date Evaluation Completed
✔	Date/Time of Receipt		Evaluation Completed By
	Date/Time of Problem Occurence		Date Resolution Completed
✔	Originator ID		Resolution Completed By
	Environment ID		Projected Availability
	Problem Description (text)		Released/Shipped
	Finding Activity		Applied
	Finding Mode	✔	Approved By
	Criticality	✔	Accepted By

Section II

What Substates are used for Open?

#	Name	#	Name
1	***Recognized***	6	
2	***Evaluated***	7	
3	***Resolved-NDOP (nondefect-oriented)***	8	
4	***Resolved-DOP (defect-oriented problem)***	9	
5	***Resolved -DUP (duplicate)***	10	

Problem Status Definition Form-2

Product ID: Example — **Status Definition ID: Customer probs**

Finding Activity ID: Customer Support — **Definition Date: 06/30/92**

Section III

What attributes are unconditionally required to have values as criteria for each substate?

Attribute #	Name	Substate Number 1	2	3	4	5	6	7	8	9	10
1	Problem ID		✔	✔	✔	✔					
2	Software Product Name or ID	✔	✔	✔	✔	✔					
3	Date/Time of Receipt	✔	✔	✔	✔	✔					
4	Date/Time of Problem Occurence		✔	✔	✔	✔					
5	Originator ID	✔	✔	✔	✔	✔					
6	Environment ID		✔	✔	✔	✔					
7	Problem Description (text)		✔	✔	✔	✔					
8	Finding Activity		✔	✔	✔	✔					
9	Finding Mode		✔	✔	✔	✔					
10	Criticality		✔	✔	✔	✔					
11	Problem Type		✔	✔	✔	✔					
12	Uniqueness		✔	✔	✔	✔					
13	Urgency		✔	✔	✔	✔					
14	Date Evaluation Completed		✔	✔	✔	✔					
15	Evaluation Completed By			✔	✔	✔					
16	Date Resolution Completed			✔	✔	✔					
17	Resolution Completed By			✔	✔	✔					
18	ID of Original Problem					✔					
19	Changes Made To				✔						
20	Related Changes				✔						
21	Defect Found In				✔						
22	Defects Caused By				✔						
23	Projected Availability				✔						
24	Released/Shipped										
25	Applied										
26	Approved By:			✔	✔	✔					
27	Accepted By:			✔	✔	✔					

Section IV

List the substates with conditional attribute values		Substates affected	
Substate #	Conditional Attribute/Value	Substate #	Attribute Numbers
2	***Problem Type = not a software defect***	*3*	*11*
2	***Problem Type = software defect, Uniqueness = Original***	*4*	*11, 12*
2	***Problem Type = software defect, Uniqueness = duplicate***	*5*	*11,12*

Figure 4-59: A Checklist-Based Definition for Defect States

Creating Your Own Definition Frameworks

There are many measures for which checklists and descriptive forms do not yet exist. When your teams propose measures that have no current checklists, you should challenge them to develop similar (or equivalent) vehicles for communicating the rules and procedures that they want used to capture and record their data. Checklists are useful, especially when inclusion and exclusion decisions affect results.

Whatever frameworks you choose, your structured methods must tell people who collect data exactly what is to be included in (and excluded from) the values they report to you. Where it makes a difference—and it usually does—they must also describe how the measurements will be carried out. An appropriate definition framework ensures that any variation in the method for measuring that could affect either the values themselves or the way they should be interpreted gets described.

When constructing measurement definition checklists and supporting forms, you will find that the surest way to ensure full coverage and achieve consensus is to focus not on telling people what they should do, but on identifying what you and others need to know to use the data correctly. Not only will this minimize controversy and confrontation, but once you have a structure that communicates all relevant information about a measurement's result, it is easy to use that structure to tell others how to collect the data you want.

Dealing with Complexity

Although definition checklists sometimes seem longer than you would like them to be, they are actually quite efficient vehicles for identifying and communicating many of the details that affect the values obtained when measurements are made. Effective checklists seem long at times because they are making visible many of the issues, assumptions, and decisions that have historically gone unrecorded. When details such as rules for inclusion and exclusion are not made explicit, users of data are forced to guess. Often they guess incorrectly, assume things that are not true, and end up misusing or misinterpreting the reported values. When checklists are too brief, they may not be identifying all the important issues.

An important side benefit of definition checklists (and their supporting forms) is that they can be used as mechanisms for identifying and resolving the differing needs of different users. Checklists are very productive guides in group settings for helping teams step through issues one at a time—identifying individual needs, negotiating consensus, and recording decisions as they go.

Exercise 8: Defining Software Measures

Choose one of your indicators for definition. If suitable checklists and forms are available for defining the data elements in this indicator, use them to create explicit definitions (specifications) for the measurements that must be made to meet the requirements for the indicator. Repeat these steps until you have created clearly defined rules for all of the data elements that you will use to build the indicator. Figure 4-60 illustrates this process.

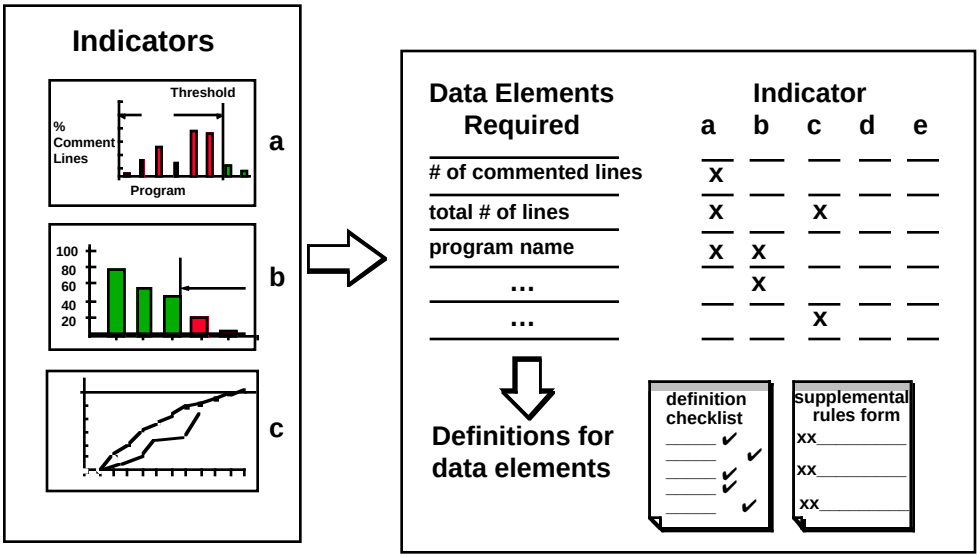


Figure 4-60: Constructing Operational Definitions

Repeat the exercise for your other indicators. If checklists and supporting forms do not exist, create them (or construct alternative structured formats) to ensure that you have workable vehicles for communicating operational definitions for all the data elements you use.

Checklists and supporting forms for defining some frequently used software measures have been illustrated in this chapter. Blank, reproducible copies are presented with the materials for this exercise in Appendix A. More detailed advice, together with examples and guidelines for using these checklists, are available in [Florac 92], [Goethert 92], and [Park 92].

4.9 Step 9: Identify the Actions Needed to Implement Your Measures

What has puzzled us before seems less mysterious, and the crooked paths look straighter as we approach the end.

— Jean Paul Richter

Translating Measurement Definitions into Action Plans

You now have indicators that address your questions and operational definitions for the measures you want to collect. Moreover, the GQ(I)M process that you followed has given you traceability from your measures back to the questions and measurement goals that motivated them (Figure 4-61). Since you have previously traced the measurement goals back to business subgoals (Figure 4-35 and Exercise 5), you have a clear means for ensuring that the measures your organization implements and uses stay focused on business needs.

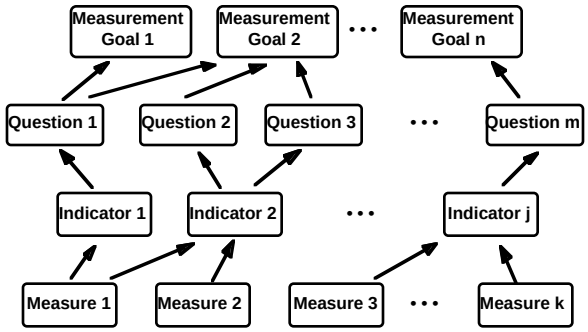


Figure 4-61: GQM Provides Traceability Back to Measurement Goals

Your ninth step is to assemble information about the current status and use of your measures, so that you can prepare an effective plan for implementing the measures you have defined (Figure 4-62). The three words that should guide you are *analysis, diagnosis,* and *action*.

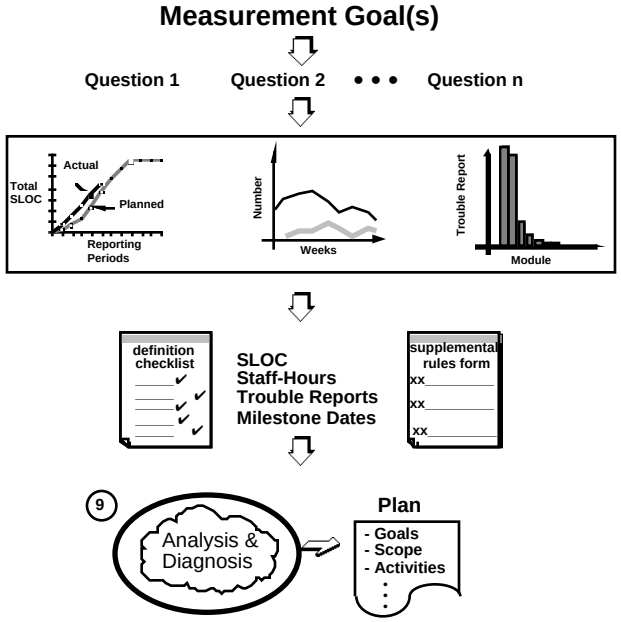


Figure 4-62: The Ninth Target—The Facts Needed to Prepare an Effective Action Plan

Analysis

Analysis means probing for facts that help you understand where you are starting from. This involves identifying the measures that your organization is using now and understanding how it is collecting them. This information

determines your starting point for implementing the goal-driven measures you have defined. If your organization is like most, you will not be starting from scratch. Some measurement activities will already be in place. It usually makes sense to build on things that are currently in use, strengthening them in the process, and refocusing them were necessary. In measurement, just as in life, evolution meets less resistance than revolution.

When analyzing your existing measures and measurement practices, you should ask questions like these:

- What data elements are required for my goal-driven measures?
- Which data elements are collected now?
- How are they collected?
- What are the processes that provide the data?
- How are the data elements stored and reported?

As you sort through these issues, you may find it helpful to use tabular displays to summarize what you learn. Figure 4-63 illustrates one possible summary. Here the sources for planned and actual values for one organization's current management measures are identified and listed opposite the issue they address.

What software processes are sources for data?

	Planned	Actual
Size	✔	**Configuration Management**
Effort	✔	**Labor Tracking**
Quality	✔	**Problem Tracking**
Schedule	✔	**Configuration Management**

✔ = estimates from project management

Figure 4-63: Taking Inventory

Although displays like Figure 4-63 are helpful, you will find them even more useful if you list explicit measures, not just broad measurement classes. For example, there are many possible measures for size, and the organization could be using more than one. By naming explicit measures instead of simply what they are attempting to measure, you help focus attention on finding all possible sources and on matching the sources to the data used. For example, it could be that while the configuration management group does report measurements for size, these are not the values that are used by estimators and managers for project estimating and tracking.

When you dig for data, be persistent. There are often more potential data sources than are initially apparent. Again, mental models of your processes help. For example, several sources often exist for data about defects and problems. Figure 4-64 is typical of many organizations. Here people who build products (product synthesis) write problem reports, teams that inspect products (as in peer reviews) prepare inspection reports, participants in formal milestone reviews produce action items, test groups produce test reports, and

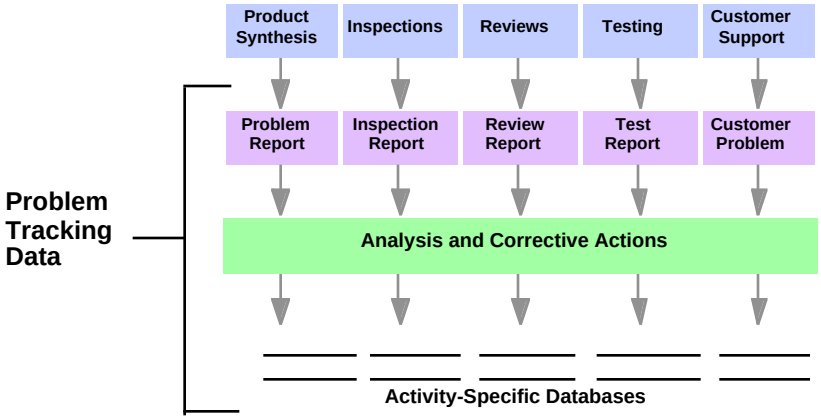


Figure 4-64: Sources for Problem-Tracking Data

customer support groups document customer problems. All of these reports are followed by analysis and corrective action, and the results and status are usually recorded somewhere, often in a database. Since there may be more than one database, you will want to find all of them to understand what they can give you. So dig, and be thorough. You may be surprised at what you find.

Diagnosis

Diagnosis means evaluating the data elements that your organization is collecting now, determining how well they meet the needs of your goal-driven measures, and proposing appropriate actions for

- using the data
- adapting the data to your needs
- adapting your needs to the data
- obtaining what is missing

Where analysis is fact finding, diagnosis is evaluative and judgmental. When diagnosing, you are identifying alternatives and setting the stage for finding solutions. You are asking questions such as

- What existing measures and processes can be used to satisfy our data requirements?
- What elements of our measurement definitions or practices must be changed or modified?
- What new or additional processes are needed?

Figure 4-65 shows a useful way to relate the data that you need to the data that your organization now uses. Here you begin by summarizing your results from Exercises 6, 7, and 8. Then you assess the suitability and availability of existing data and identify current and potential sources for this information and for the other data elements you need. Figure 4-65 also illustrates a coding scheme for classifying degrees of availability. You may use this scheme or devise another that serves your needs better.

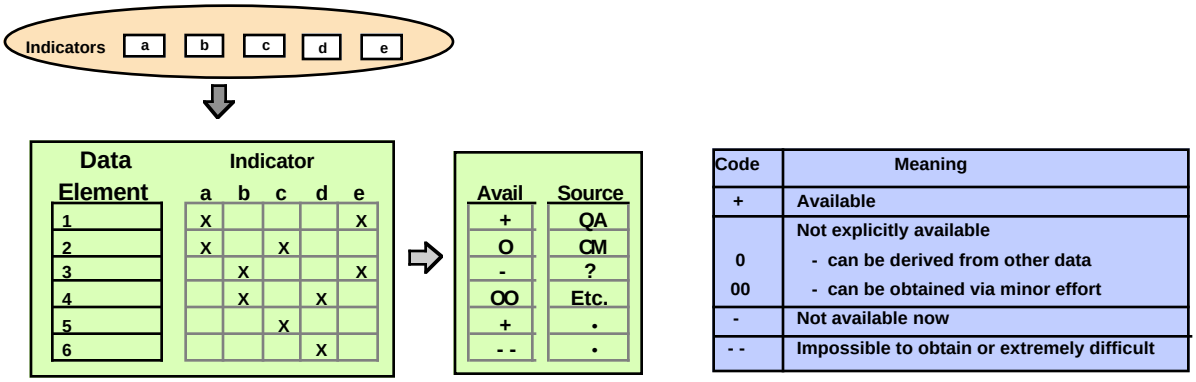


Figure 4-65: Evaluating Abilities of Existing Data to Satisfy Needs

Action

Action means translating the results of your analyses and diagnoses into implementable steps. It is concerned with finding solutions and with making the solutions happen. It includes identifying tasks and assigning responsibilities and resources.

Action starts with identifying the elements that you will build on or address in your measurement plan. Some things you will want to do before writing the plan are

- Identify the sources of data within your existing software process(es).
- Define the methods that will be used to collect and report the data.
- Identify (and specify) the tools that will be required to support collecting, reporting, and storing the data.
- Determine your requirements for points in time and frequencies of measurement.
- Document your data collection procedures in detail.
  - Identify responsible persons and organizations.
  - Determine where, how, and when to collect and report.
  - Create sketches for the data collection records you will use.
- Determine who will use the data.
- Define how the data will be analyzed and reported.
- Prepare a data definition and collection process guide.

You should also analyze your data storage and access requirements. This includes identifying or determining

- your historical retention needs
- who will collect, store, maintain, and access the data
- the organizational levels to be served (serving more than one organizational level often translates into a need for more than one database.)
- the granularity of the data
- the procedures to be used for dynamically editing and verifying data as it is entered into the database
- the number of people with access to the data
- the need for recording the definitions associated with the data, so that users can tie the data to the descriptive information that is needed to use the data correctly

In addition, you should pay close attention to issues of data privacy, wherever they may be encountered. This is especially important for data that could be used (or perceived to be used) to evaluate the performance of individuals or teams. Much anecdotal evidence exists to suggest that the surest way to make measurement fail is to have people suspect that the measures might be used against them.

An Action Item Checklist

When you are preparing to write your measurement plan, it helps to have a checklist to ensure that nothing gets overlooked. We offer the one in Figure 4-66.

This checklist can easily be transformed into a display for summarizing your status with respect to defining the measurement process you intend to implement. This is illustrated in Figure 4-67. Here codes are used to show the status of each task. Actions to complete these tasks are things that you will want to address as you prepare your measurement implementation plan.

Action Item Checklist

- Define the data elements (Exercise 8).
- Define the frequencies of collection and the points in the process where measurements will be made.
- Define the timelines required for moving measurement results from the points of collection to databases or users.
- Create forms and procedures for collecting and recording the data.
- Define how the data are to be stored and how the data will be accessed. Identify who is responsible for designing the database and for entering, retaining, and overseeing the data.
- Determine who will collect and access the data. Assign responsibilities for these actions.
- Define how the data will be analyzed and reported.
- Identify the supporting tools that must be developed or acquired to help you automate and administer the process.
- Prepare a process guide for collecting the data.

Figure 4-66: Action Item Checklist

	Data element						
Planning tasks	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data elements defined	**Y**	**N**	**60%**	**Not Doc'd**	**Y?**		
Data collection frequencies and points in the software process defined	**50%**	**N**	**60%**	**Not Doc'd**			
Timelines defined for getting measurement results to databases and users	**N**	**N**	**30%**	**Not Doc'd**			
Data collection forms defined	**N**	**N**	**N**	**N**			
Data collection procedures defined	**N**	**N**					
Data storage, database design, and data retention responsibilities defined	**N**	**N**					
Who will collect and who will access the data identified	**N**	**N**					
Analysis processes defined	**N**	**N**					
Reporting processes defined	**N**	**N**					
Supporting tools identified and made available	**N**	**N**					
Process guide for data definition and collection prepared	**Y**						

Figure 4-67: Action Planning Status

Exercise 9: Analysis, Diagnosis, Action

Review the indicators and data elements that you identified and defined in the preceding exercises. Analyze the extent to which your organization meets the measurement needs for these data elements now. Identify (diagnose) what else your organization must do to meet these needs. Prepare summaries of the results and of the status of your analyses, diagnoses, and planning activities. Figures 4-65 and 4-67 can be used as templates for your summaries. Figure 4-68 illustrates the process flow for this exercise.

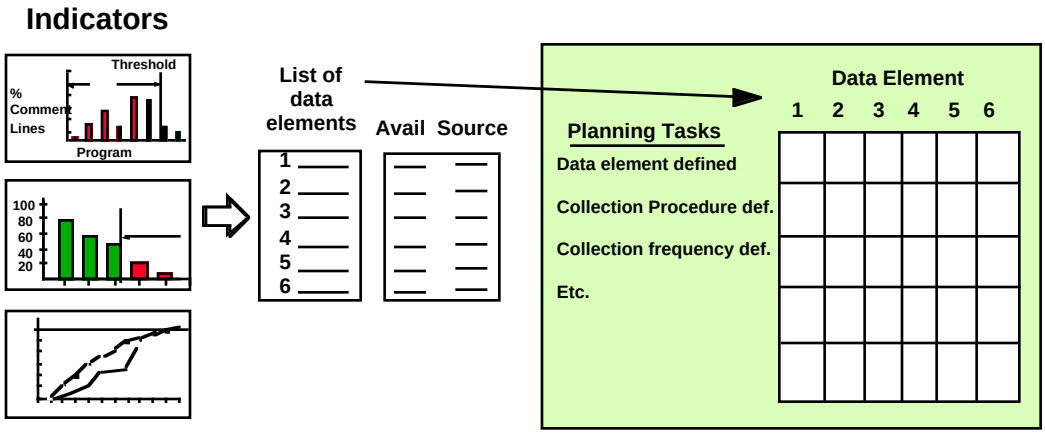


Figure 4-68: Identifying Implementation Tasks and Assessing Their Status

4.10 Step 10: Prepare a Plan

Once you know what you have to start with (analysis), how well your present measures meet your business needs (diagnosis), and the actions that you will to take to meet the remaining needs (action), you are ready to prepare a plan for implementing the actions you have identified. Your 10th and final step in this guidebook is to write your plan. The ellipse in Figure 4-69 shows where we are in the goal-driven process.

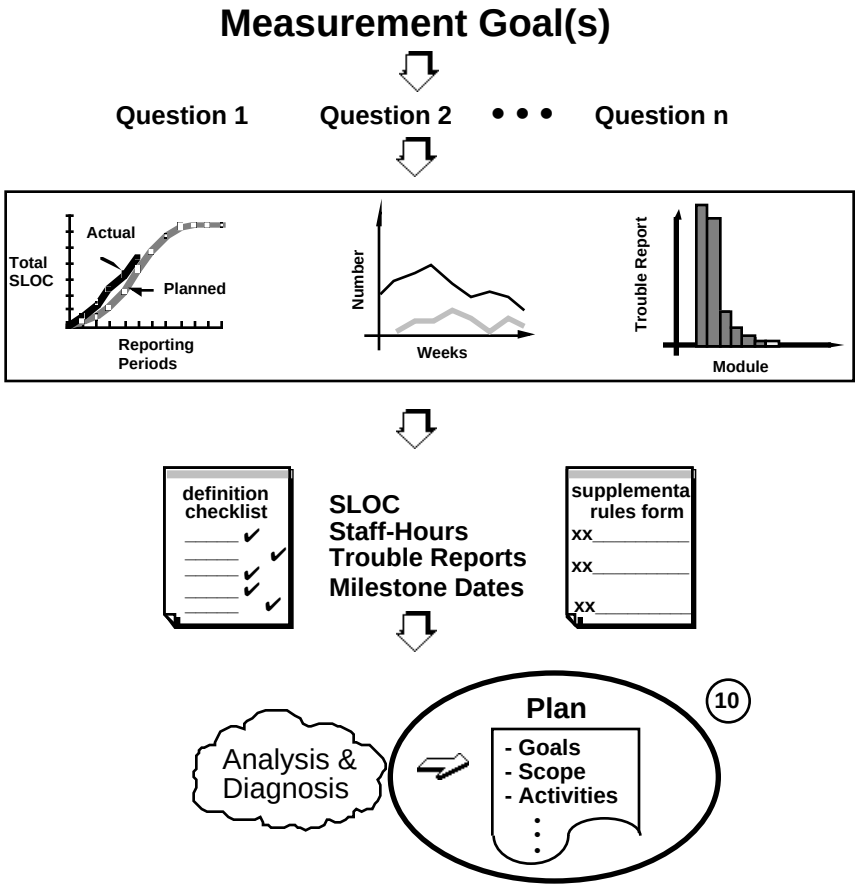


Figure 4-69: The Tenth Target—A Plan for Implementing the Measures

A Measurement Planning Template

On the next three pages, we offer a template to help you identify and structure the key issues that your plan should address. You (or your organization) may have other formats that you prefer. If so, by all means use any reasonable alternative that works well for you in your environment. But whatever the structure of your plan, be sure to address the issues that appear in our template. Each has been derived from accumulated experience in implementing action items in real-world, people-centered environments.

Measurement Implementation Plan (a Template)

1. Objective

List the principal objective(s) of this measurement implementation effort. Identify the measures to be implemented, explain why they are important to your organization, and summarize the expected outcomes.

2. Description

Outline the origins of the plan, describe the goals and scope of the activities encompassed, and explain how the measures and efforts in the plan relate to other efforts and activities. The subsections that provide this information are described below.

Background

Give a brief history of the events that have led to or motivated this plan. Describe the origins of the plan, the work that has been done to date, who participated, and (optionally) the process that was used. Relate the planned actions to other existing or concurrent measurement activities within your organization and (if appropriate) in those of your customers or suppliers.

Goals

List and explain the goals that motivate and guide the activities under this plan. This section identifies three kinds of goals: (a) business goals, (b) measurement goals, and (c) the goals of this plan.

- The *business goals* frame the importance of the program and the level of support to be provided by senior executives.
- The *measurement goals* are more detailed and more specific. They guide the methods that will be used for collecting, storing, and using measured results. Each measurement goal should be identified and related to one or more of the business goals.
- The *goals for this plan* are more operationally oriented. They specify the outcomes that are sought. What do you want to achieve? How will you know when you are done? Often the most effective way to express these goals is in terms of exit criteria or criteria for success.

Scope

Relate the measures that this plan implements to the measurement goals they serve and describe their range of application. Do the measures apply to new projects only? To development projects? To procurement actions? To maintenance projects? To contractors and subcontractors? To large or small programs? To only certain divisions or departments?…etc.? Who are the major stakeholders? Who will be

affected by the measurement practices, processes, and methods? Who will use the results? Identify the time span over which this plan is to be effective.

Relationship to Other Software Process Improvement Efforts

Describe how the measurement efforts in this plan relate to other process improvement activities at your organization. Explain how the efforts relate to any goals or actions your organization may have established with respect to the CMM, the Baldrige Award, or ISO 9000 certification.

Relationship to Other Functional Activities

Describe how the measurement efforts in this plan relate to (and interface with) other functional groups and activities at your organization, such as cost estimating, time and effort reporting, cost accounting, procurement, technical writing, and quality assurance.

3. Implementation

Describe the actions that are to be taken to implement the measures identified in Section 2. For example, will you use pilot projects? Will you use focused subsets of the measures, perhaps locally, before broad organization-wide implementation? Put together a comprehensive strategy that addresses all aspects of implementation, including the tools and training needed to introduce, use, and sustain effective measurement. Address data-storage issues and the steps for incorporating these measures and measurement practices into your organization's policies, procedures, practices, and training curricula. Describe how you will use the measured results and how you will obtain feedback to continuously improve the measurement processes. Describe your plans for identifying problem areas and successes, and for publishing success stories and lessons learned. The subsections that provide this information are described below.

Activities, Products, and Tasks

Describe how the effort is to be accomplished. Partition the effort into manageable activities, products, and tasks that can be used as a basis for planning, reporting, management, and control. For each activity, product, or task, state the objective and identify the principal subtasks. Identify all sequences and dependencies that affect either the schedule or assignment of resources. Where possible, identify the entry and exit conditions that will determine start and completion of the task.

Schedule

Describe when each of the activities, products, or tasks is to be accomplished. Use Gantt charts, PERT charts, or alternative displays where appropriate to describe sequences and dependencies. Translate key actions, events, and deliverables into milestones so that performance can be tracked against plans.

Resources

Describe the resources that are being allocated to this effort. Address personnel, money, facilities, teaming plans, computer resources, etc.

Responsibilities

Name the individuals or groups that will be responsible for overseeing, planning, implementing, managing, approving, and funding this effort. Assign responsibility and authority for acquiring tools, for training, and for implementing and operating databases.

Measurement and Monitoring

Describe how the progress of implementing these measures will be measured, analyzed, and reported. Identify replanning points and describe how significant schedule deviations or changes and revised funding needs will be handled.

Assumptions

Identify the key assumptions upon which this plan is based. Key assumptions are ones which, if not satisfied, pose risks for successful implementation.

Risk management

Describe how you will identify, assess, track, and do contingency planning for the risk factors associated with the measurement implementation efforts covered by this plan. Describe the actions that will be taken to monitor the assumptions, and provide mechanisms for reacting if assumptions are not met. Also, identify all places where planned schedules and resources differ from estimates and describe the actions that are being taken to make the planned outcomes achievable.

4. Sustained Operation

Describe the actions that will be taken to sustain and use the measures implemented in Section 3. Assign resources and responsibilities and make provisions for continuing evolution. Describe the practices that will be used to evaluate and monitor the effectiveness of the measures and to assess their business value and their effects on organizational performance. Alternatively, if appropriate, provide direction and resources for preparing an operational plan for sustaining the collection, use, retention, evolution, and evaluation of these measures.

Exercise 10: Writing the Plan

With the results from Exercises 1 through 9 in hand, you have most of the information that you need to begin planning actions to implement the measures you have defined. Your final assignment is to write an action plan. You will, of course, follow through by getting approval for your plan and by implementing and tracking the measurement and management actions that are called for in the plan.

The template that was presented on the previous pages provides a generic outline for the major topics that your plan should address. You may use this template as a guide for your plan, or you may select an alternative structure that better meets your needs.

5 Following Through

This chapter summarizes significant recommendations that others have reported or that we have observed from working with organizations that have implemented various aspects of software measurement.

5.1 Measurement Principles for Functional Managers

- Set clear goals.
- Get your staff to assist in defining your measures.
- Provide active management oversight—ask for and use the data.
- Understand the data that your people report to you.
- Never use measurement data to reward or punish the people who make the measurements, and ensure that they understand that you and everyone else will obey this rule.
- Establish practices that protect anonymity. Provisions for protecting anonymity build trust and foster the collection of reliable data.
- Support your people when their reports are backed by data useful to the organization.
- Do not emphasize one measure or indicator to the exclusion of others.

5.2 Measurement Principles for Project Managers

- Know the strategic focus of your organization and emphasize measures that support the strategy.
- Gain agreement with your teams on the measures that you will track, and define the measures in your project plan.
- Provide regular feedback to your teams about the data they have collected.
- Do not measure individuals.

5.3 Measurement Principles for Project Teams

- Do your best to report accurate, timely data.
- Help management focus project data on improving your software processes.
- Do not use software measures to brag about how good you are, or you will encourage others to use other data to show the opposite.

5.4 General Principles

- Software measurement must not be a strategy unto itself.
- Integrate software measurement with your overall strategy for software process improvement. You should have (or develop) such a strategy in conjunction with your software measurement plan.
- Start small with common goals and issues.
- Design a consistent measurement process that
  - is linked to organizational goals and objectives
  - includes rigorous definitions
  - continuously evolves
- Test the measures and processes you design before implementing them broadly.
- Measure and monitor the effectiveness of your software measures and measurement activities.

6 Putting It All in Context

Paul Kirchner, at the December 1956 meetings of the American Association for the Advancement of Science, listed seven elements that he believed to be inherent in the basic structure of a business measurement process [Kirchner 59]. These elements, phrased here as steps, are as follows:

1. Determine the objective of the business entity—the purpose which is to be served in a particular situation.
2. Determine the types of factors which might serve to attain the objective.
3. Select the key aspects of the factors—the aspects which are to be measured.
4. Choose
   (a) a measuring method
   (b) a measuring unit
5. Apply the measuring unit to the object to be measured—the central action of measurement.
6. Analyze the measurement—relating it to other measurements (other in time or in kind).
7. Evaluate the effectiveness of the measurement by determining the extent to which it assisted in attaining the objective.

Although we were not aware of Kirchner's views at the time we developed the materials in Chapter 4, we are heartened to see how closely the goal-driven measurement process mirrors his observations. In fact, Kirchner's elements, when phrased as actions, do an admirable job of putting the steps of this guidebook into the larger contexts of software measurement architecture [McAndrews 93] and implementation [SEI 96]. Figure 5-1, which is taken from these references, shows a top-level view of the process model that has been guiding us. As you can see, both we and McAndrews have been faithful to Kirchner's calling.

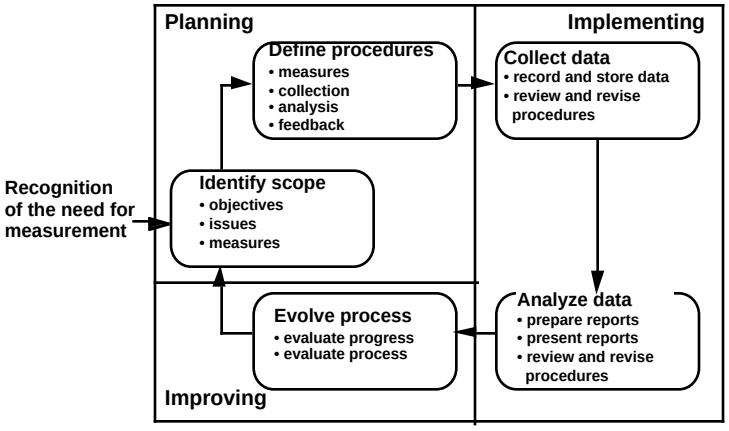


Figure 5-1: A Measurement Process Architecture

References

[ami 92] The ami Handbook: *A Quantitative Approach to Software Management.* London, England: The ami Consortium, South Bank Polytechnic, 1992.

[Armitage 94] Armitage, James W. & Kellner, Marc I. "A Conceptual Schema for Process Definitions and Models," 53-165. *Proceedings of the 3rd International Conference on the Software Process* . Reston, Va., Oct. 10-11, 1994. IEEE Computer Society Press, 1994.

[Basili 88] Basili, Victor R. & Rombach, H. Dieter. "The TAME Project: Towards Improvement-Oriented Software Environments." *IEEE Transactions on Software Engineering, Vol. 14, No. 6* (June 1988): 758-773.

[Basili 89] Basili, Victor R. "Using Measurement for Quality Control and Process Improvement." *Second Annual SEPG Workshop,* Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., June 21-22, 1989.

[Baumert 92] Baumert, John H. *Software Measures and the Capability Maturity Model* (CMU/SEI-92-TR-25, ADA 257 238). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1992.

[Carleton 94] Carleton, Anita; Rozum, James; & Paulish, Daniel. *Engineering a Software Measurement Process for Your Organization.* Tutorial notes, SEPG National Meeting, April 1994.

[Caws 59] Caws, Peter. "Definition and Measurement in Physics," 3–17. *Measurement: Definitions and Theories*, C. West Churchman & Philburn Ratoosh, ed. New York, N.Y.: John Wiley & Sons, Inc., 1959.

[Churchman 59] Churchman, C. West. "Why Measure?," 83–94. *Measurement: Definitions and Theories*, C. West Churchman & Philburn Ratoosh, ed. New York, N.Y.: John Wiley & Sons, Inc., 1959.

[Deming 86] Deming, W. Edwards. *Out of the Crisis*. Cambridge, Mass.: Massachusetts Institute of Technology, Center for Advanced Engineering, 1986.

[Fenton 91] Fenton, Norman E. *Software Metrics: A Rigorous Approach*. London: Chapman & Hall, 1991.

[Fenton 95] Fenton, Norman E. & Whitty, Robin. "Introduction," 1-19. *Software Quality Assurance and Measurement, A Worldwide Perspective,* Norman Fenton, Robin Whitty, and Yoshinori Iizuka, ed. London: International Thomson Computer Press, 1995.

[Florac 92] Florac, William A. et al. *Software Quality Measurement: A Framework for Counting Problems and Defects* (CMU/SEI-92-TR-22, ADA 258 556). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, September 1992.

[Ghiselli 81] Ghiselli, Edwin E.; Campbell, John P.; & Zedeck, Sheldon. *Measurement Theory for the Behavioral Sciences*. San Francisco, Calif.: W. H. Freeman and Company, 1981.

[Goethert 92] Goethert, Wolfhart B. et al. *Software Effort Measurement: A Framework for Counting Staff-Hours* (CMU/SEI-92-TR-21, ADA 258 279). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, September 1992.

[Humphrey 89] Humphrey, Watts S. *Managing the Software Process*. Reading, Mass.: Addison-Wesley, 1989.

[Kan 95] Kan, Stephen H. *Metrics and Models in Software Quality Engineering*. Reading, Mass.: Addison-Wesley, 1995.

[Kirchner 59] Kirchner, Paul. "Measurements and Management Decisions," 64–82. *Measurement: Definitions and Theories*, C. West Churchman & Philburn Ratoosh, ed. New York, N.Y.: John Wiley & Sons, Inc., 1959.

[Krantz 71] Krantz, David H.; Luce, R. Duncan; Suppes, Patrick; & Tversky, Amos. *Foundations of Measurement, Vol.1*. New York, N.Y.: Academic Press, 1971.

[Luce 90] Luce, R. Duncan; Krantz, David H.; Suppes, Patrick; & Tversky, Amos. *Foundations of Measurement, Vol.3*. San Diego, Calif.: Academic Press, 1990.

[Lynch 91] Lynch, Richard L. & Cross, Kelvin F. *Measure Up!*. Cambridge, Mass.: Blackwell Publishers, 1991.

[McAndrews 93] McAndrews, Donald R. *Establishing a Software Measurement Process* (CMU/SEI-93-TR-16, ADA 267 896). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, July 1993.

[Park 92] Park, Robert E. et al. *Software Size Measurement: A Framework for Counting Source Statements* (CMU/SEI-92-TR-20, ADA 258 304). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, September 1992.

[Paulk 93a] Paulk, Mark C. et al. *Capability Maturity Model for Software, Version 1.1* (CMU/SEI-93-TR-24, ADA 263 403). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, February 1993.

[Paulk 93b] Paulk, Mark C. et al. *Key Practices of the Capability Maturity Model, Version 1.1* (CMU/SEI-93-TR-25, ADA 263 432). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, February 1993.

[PSM 96] *Practical Software Measurement: A Guide to Objective Program Insight*. Washington D.C.: Joint Logistics Commanders, Joint Group on Systems Engineering, March 1996.

[Pulford 96] Pulford, Kevin; Kuntzmann-Combelles, Annie; & Shirlaw, Stephen. *A Quantitative Approach to Software Management: The ami handbook*. Worthingham, England: Addison-Wesley, 1996.

[Roberts 79] Roberts, Fred S. *Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences*. Reading, Mass.: Addison-Wesley, 1979.

[Rombach 89] Rombach, H. Dieter & Ulery, Bradford T. "Improving Software Maintenance Through Measurement." *Proceedings of the IEEE, Vol. 77,* No. 4 (April 1989): 581-595.

[Scholtes 90] Scholtes, Peter R. *The Team Handbook*. Madison, Wisconsin: Joiner Associates, Inc., 1990.

[SEI 96] *Engineering an Effective Measurement Program: Course Notes*. Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, June 1996.

[Senge 94] Senge, Peter M. *The Fifth Discipline Fieldbook.* New York, N.Y.: Doubleday, 1994.

[Shepperd 93] Shepperd, Martin & Ince, Darrel. *Derivation and Validation of Software Metrics.* Oxford: Clarendon Press, 1993.

[Stevens 46] Stevens, S. S. "On the Theory of Scales of Measurement." *Science, Vol. 103,* No. 2684 (1946): 677–680.

[Stevens 51] Stevens, S. S. "Mathematics, Measurement, and Psychophysics," 1–49. *Handbook of Experimental Psychology*, S. S. Stevens, ed. New York, N.Y.: John Wiley & Sons, Inc., 1951.

[Stevens 59] Stevens, S. S. "Measurement, Psychophysics, and Utility," 18–63. *Measurement: Definitions and Theories*, C. West Churchman & Philburn Ratoosh, ed. New York, N.Y.: John Wiley & Sons, Inc., 1959.

[Velleman 93] Velleman, Paul W. & Wilkinson, Leland. "Nominal, Ordinal, and Ratio Typologies Are Misleading." *The American Statistician, Vol. 47*, No. 1 (February 1993): 65–72.

[Weinberg 93] Weinberg, Gerald M. *Quality Software Management, Vol. 2: First-Order Measurement*. New York, N. Y.: Dorset House Publishing, 1993.

[Wheeler 92] Wheeler, Donald J. & Chambers, David S. *Understanding Statistical Process Control.* Knoxville, Tenn.: SPC Press, 1992.

[Wiener 20] Wiener, Norbert. "A New Theory of Measurement: A Study in the Logic of Mathematics." *Proc. London Math. Soc., Ser. 2,* 19 (1920): 181–205.

[Zuse 91] Zuse, Horst. *Software Complexity: Measures and Methods*. Berlin: Walter de Gruyter, 1991.

Appendix A: Exercises and Worksheets

This appendix contains instructions and supporting materials for the exercises in Chapter 4.

The pages in this appendix are designed to be reproduced and used as worksheets to guide the activities of your measurement definition teams. You may make as many copies of each page as you wish.

Exercise 1: Identifying Business Goals

The objective of this exercise is to identify the principal business goals that relate to issues that concern you. These will form the starting point for identifying and defining traceable measures in the exercises that follow.

Directions (team exercise)

1. Generate ideas for completing the following statement:

   "One of our principal business goals is…"

2. Group your responses with those of your teammates.
3. Merge similar goals and sort them into a rough priority order.
4. Prepare a chart or table to summarize your results.

Exercise 2: Identifying What You Want to Know or Learn

The objective of this exercise is to identify things that you would like to know to help you understand, assess, predict, control, improve, or motivate elements of your organization with respect to achieving your business goals.

Directions

1. Select one or more of your business goals.
2. Identify the persons or groups whose concerns your team will address. (This may be you or the organization you lead.) This defines your perspective and the roles that you and the team will assume in Tasks 3 through 6 here and in the remaining steps of the goal-driven measurement process.
3. Create rough sketches (mental models) of the relevant processes that you, in your role, manage or affect. As you do this, be guided by what you want to achieve and the issues you will have to address to achieve it.
4. List the important things (entities) in your processes that you, in your role, manage or influence. Make sure that you address each of the four kinds of process entities below:

   – inputs and resources
   – products and by-products
   – internal artifacts such as inventory and work in process
   – activities and flowpaths

   You may also want to list some of the environmental entities outside your processes that affect your work.
5. For each entity, list questions that, if answered, would help you, in your role, plan and manage progress toward your goals. For example:

   – How big is it?
   – How much is there?
   – How many components?
   – How fast is it?
   – How long does it take?
   – How much does it cost?
6. Then step back and look at your process as a whole to see if you have missed anything. By asking questions such as

   – Is the process stable?
   – How is it performing now?
   – What limits our capability?
   – What determines quality?
   – What determines success?
   – What things can we control?
   – What do our customers want?
   – What limits our performance?
   – What could go wrong?
   – What might signal early warnings?
   – How big is our backlog?
   – Where is backlog occurring?

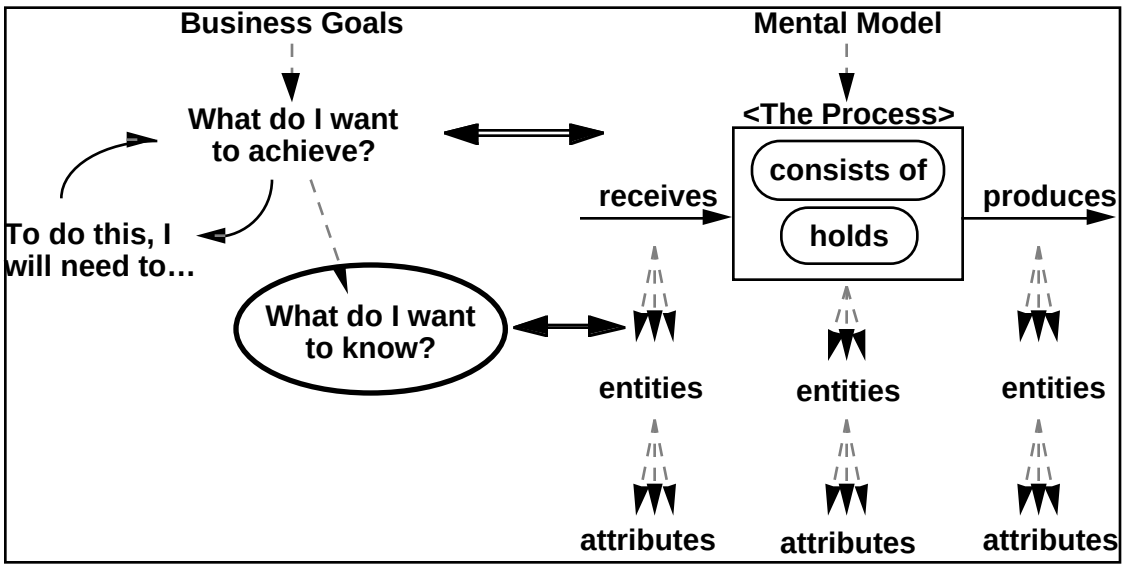
and most importantly

– How will we know?

you may discover additional entities whose properties may be worth measuring.

7. Prepare entity-question lists to summarize your results.

A worksheet to support these tasks is provided on the following pages.



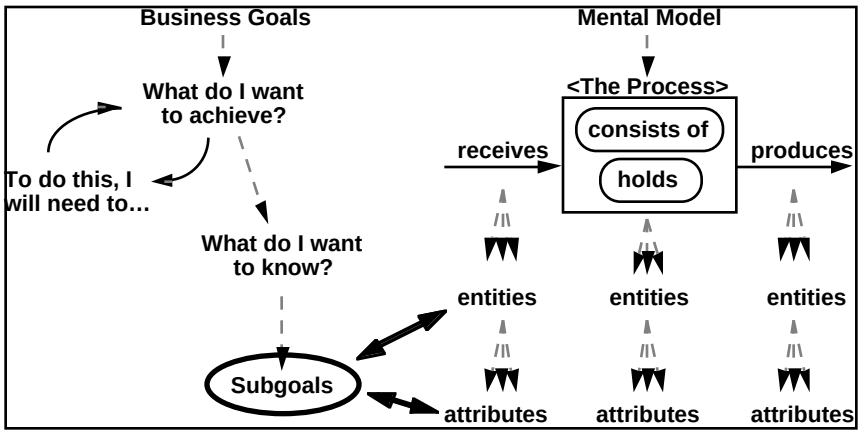
Worksheet—A Template for Recording Entities and Questions

Entities of Interest	Questions Related to Business Goal(s)
Products and by-products	
Inputs and resources	

Entities of Interest	Questions Related to Business Goal(s)
Internal artifacts (work in process, backlogs, inventory, etc.)	
Activities and flowpaths	

Exercise 3: Identifying Subgoals

The objective of this exercise is to identify and formulate subgoals that directly support the business goals and questions you identified in Exercise 2.



Directions

1. Group the questions you identified in Exercise 2 into related topics. Use copies of Worksheet 1 (or an equivalent template) to list your results.
2. Use the related topics to formulate a set of subgoals that support your higher level business goals.
3. Prepare tables or charts to summarize your results. Worksheet 2 is an example template for this summary.

Groupings	Questions Related to Business Goal(s)
Grouping #1 ( )	
Grouping #2 ( )	
Grouping #3 ( )	
Grouping #4 ( )	

⇨

	Derived Subgoals
Subgoal 1	
Subgoal 2	
Subgoal 3	
Subgoal 4	

Worksheet 1: Template for Grouping Related Questions

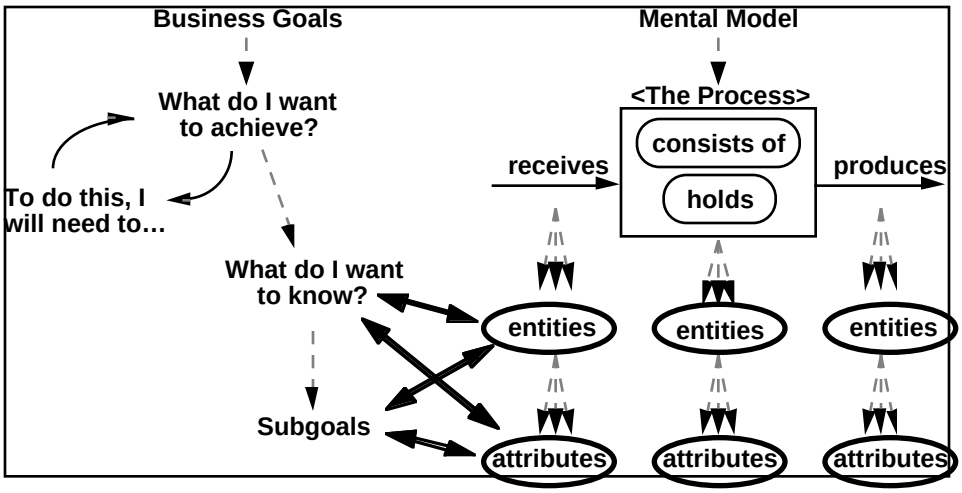
Groupings (Issues)	Questions Related to Business Goal(s)
Grouping #____ **( )**	
Grouping #____ **( )**	

Worksheet 2: Template for Derived Subgoals

	Derived Subgoals
Subgoal #1	
Subgoal #2	
Subgoal #3	
Subgoal #4	
Subgoal #5	
Subgoal #6	
Subgoal #7	
Subgoal #8	
Subgoal #9	
Subgoal #10	

Exercise 4: Identifying Entities and Attributes

The objective of this exercise is to identify the specific entities and attributes associated with your principal subgoals. The results will form the basis for formalizing your measurement goals (Exercise 5).



Directions

1. Select one of the subgoals your team identified in Exercise 3.
2. Review the related issue(s) and questions.
3. For each question, identify the entity (or entities) that the question seeks information about.
4. For each entity, list attributes that, if quantified, would help you answer the question. A worksheet for summarizing your results is on the next page.

Question
-

Entity
-

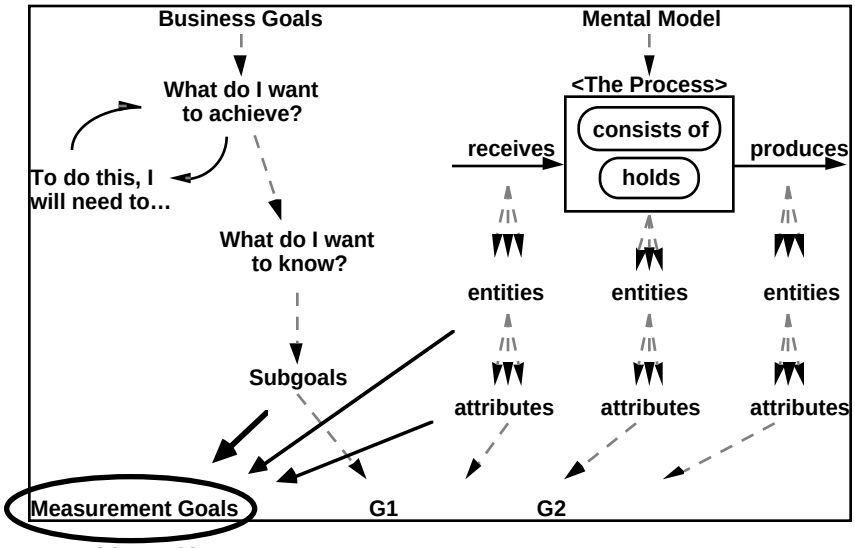
Attributes
-
-
-
-
-
-

Worksheet for Entities and Attributes

Goal, Issue, or Grouping:	
Question	
Entity	
Attributes	

Exercise 5: Formalizing Measurement Goals

The objective of this exercise is to translate your subgoals, entities, attributes, and questions into formal measurement goals. Each measurement goal will identify an object of interest and the purpose, perspective, and environment that will guide the specific questions to be addressed. These measurement goals will provide the ground rules for your subsequent measurement activities.



- **Object of interest**
- **Purpose**
- **Perspective**
- **Environment and constraints**

Directions

1. Review the subgoals, questions, entities, and attributes you identified in Exercises 3 and 4.
2. Identify the activities that you propose to undertake to get the information you need.
3. Express your goals for these activities as structured statements that identify the object, purpose, perspective, environment, and constraints associated with each measurement activity.
4. Identify and record the business subgoal(s) that each measurement goal addresses.

Templates and forms to support these tasks are reproduced on the following pages.

Templates

Object of interest: _____

Purpose:
_____ ***the*** _____ ***in order to*** _____ ***it.***

Perspective:
Examine the _____
from the point of view of (the) _____***.***

Environment:
_____, _____, _____, _____,
_____, _____, _____, _____,

Object of interest:

_____	**a process, product, resource, task, activity, agent, artifact, metric, environment, <entity>, etc.**

Purpose:

_____	**characterize, analyze, evaluate, etc.**
the _____	**<entity>, <aspect>, <attribute(s)>, etc.**
in order to _____ ***it.***	**understand, baseline, predict, plan, control, assess, compare, improve, etc.**

Perspective:

Examine the _____	**modifiability, quality, changes, defects, defect types, backlog, behavior, stability, progress, <specific attribute(s)>, etc.**
from the point of view of (the) _____.	**developer, manager, customer, engineer, process improvement team, SEPG, senior management, etc.**

Environment

- **List or otherwise describe the environmental factors and related parameters which one should understand to put the observed results in context.**
- **Focus on describing similarities to (and differences from) other familiar products, processes, and settings. This information becomes part of the database for future comparisons.**
- **Factors and parameters to consider include**
  - **application factors**
  - **people factors**
  - **resource factors**
  - **process factors**
  - **customer factors**
  - **methods**
  - **tools**
  - **constraints**

Worksheet for Measurement Goals

Subgoal(s) _____

_____

_____

Measurement Goal # _____ :

Object of Interest:

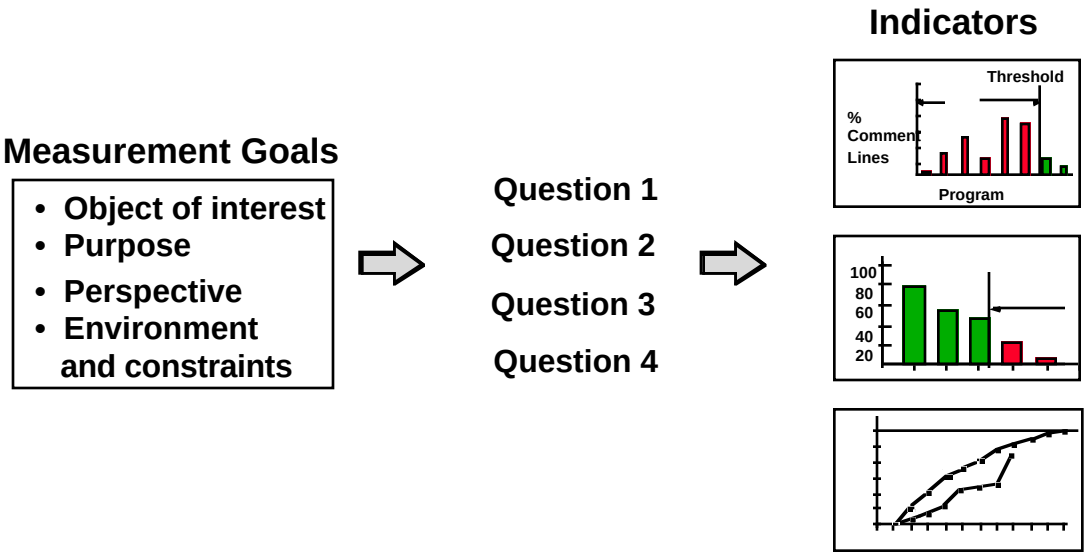
Purpose:

Perspective:

Environment and Constraints:

Exercise 6: Identifying Quantifiable Questions and Indicators

The objective of this exercise is to translate your measurement goals into indicators that enable you to examine quantifiable questions and communicate the results to others.



Directions

1. Select one of your measurement goals.
2. Identify quantifiable questions related to this goal that you would like answered. Worksheet 1 on the next page is a template that you can use for listing your questions.
3. Prepare sketches for displays (indicators) that will help you address your questions and communicate the results of your analyses to others. The pages following Worksheet 1 give some examples of indicators that others have used.
4. Identify the indicators that you believe will be most useful.
5. Repeat Steps 1–4 for your other measurement goals.
6. Prioritize your indicators, to help you focus on the most important measures first.

Worksheet 1— Measurement Questions

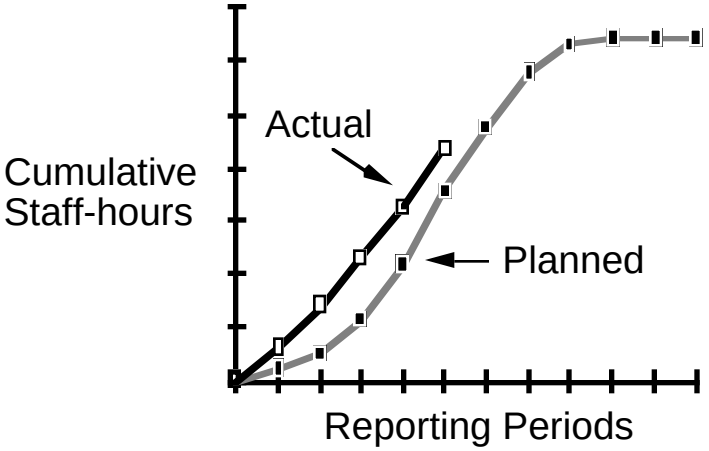
Measurement Goal # _____ :

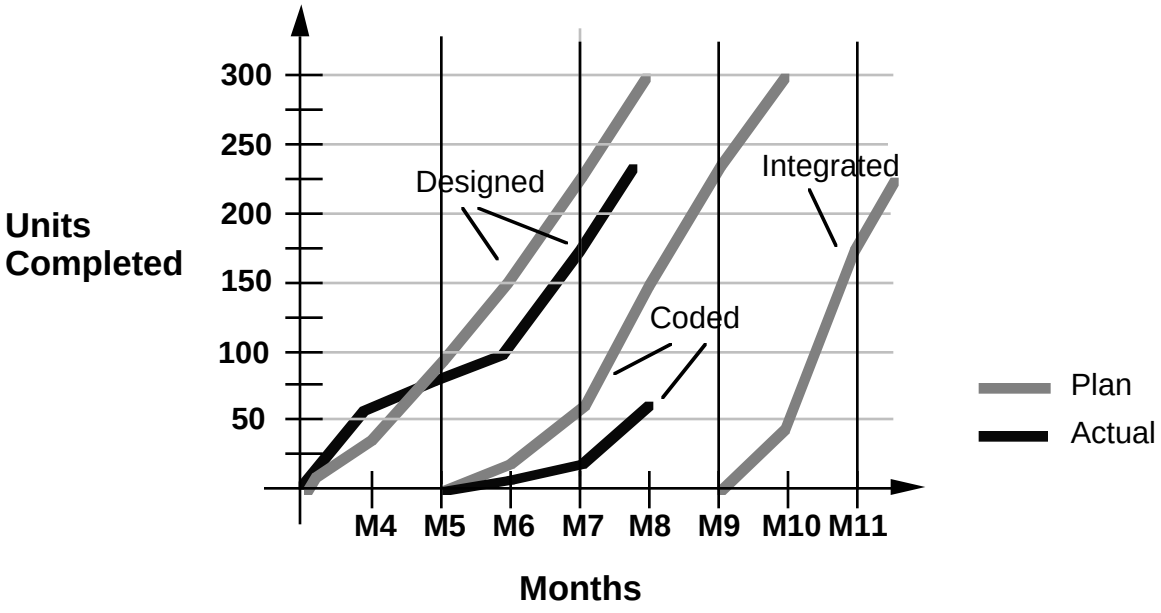
Questions: Relative to this measurement goal, questions that we would like answered are:

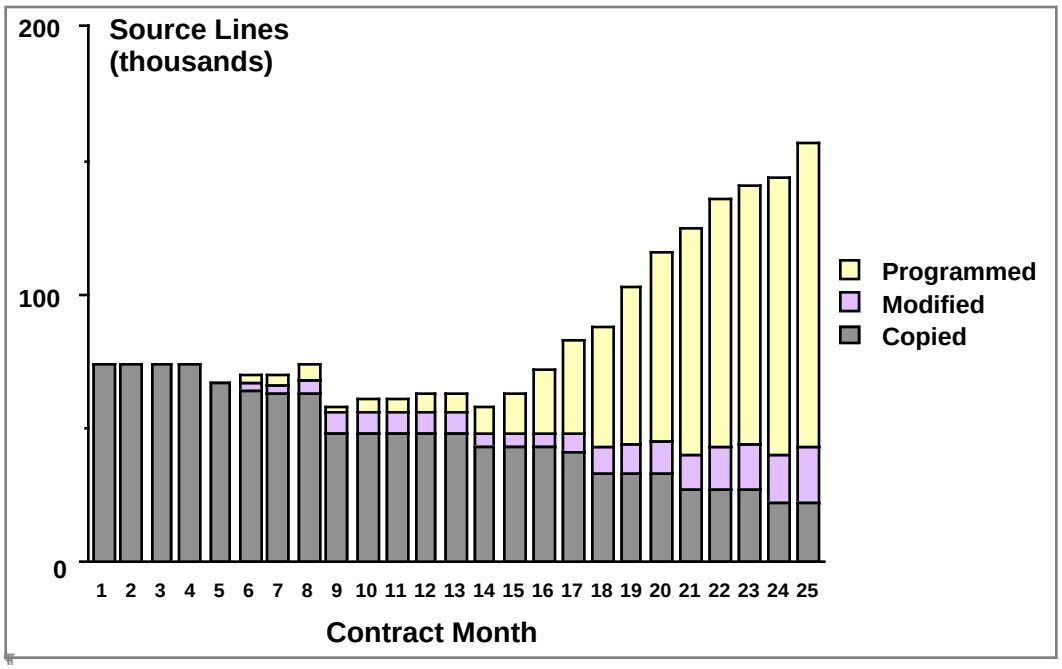
1. _____
   _____
   _____
2. _____
   _____
   _____
3. _____
   _____
   _____
4. _____
   _____
   _____
5. _____
   _____
   _____
6. _____
   _____
   _____
7. _____
   _____
   _____
8. _____
   _____
   _____
9. _____
   _____
   _____
10. _____
    _____
    _____

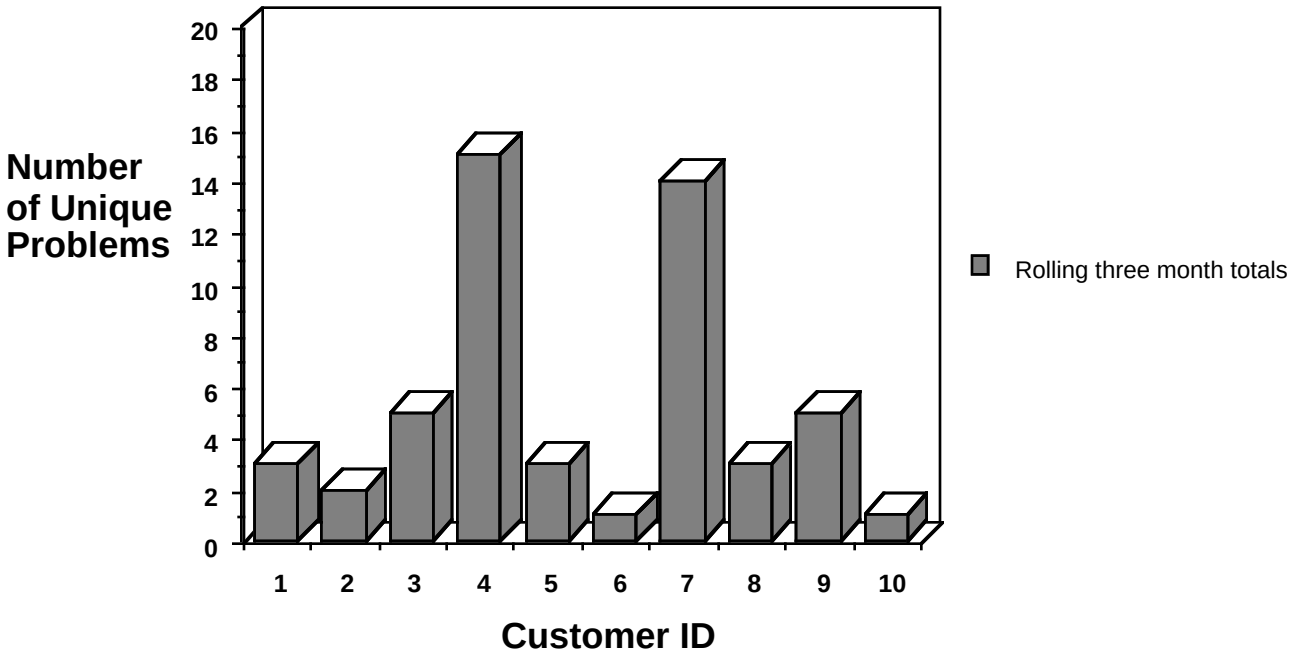
Indicators—To Prompt Your Thinking

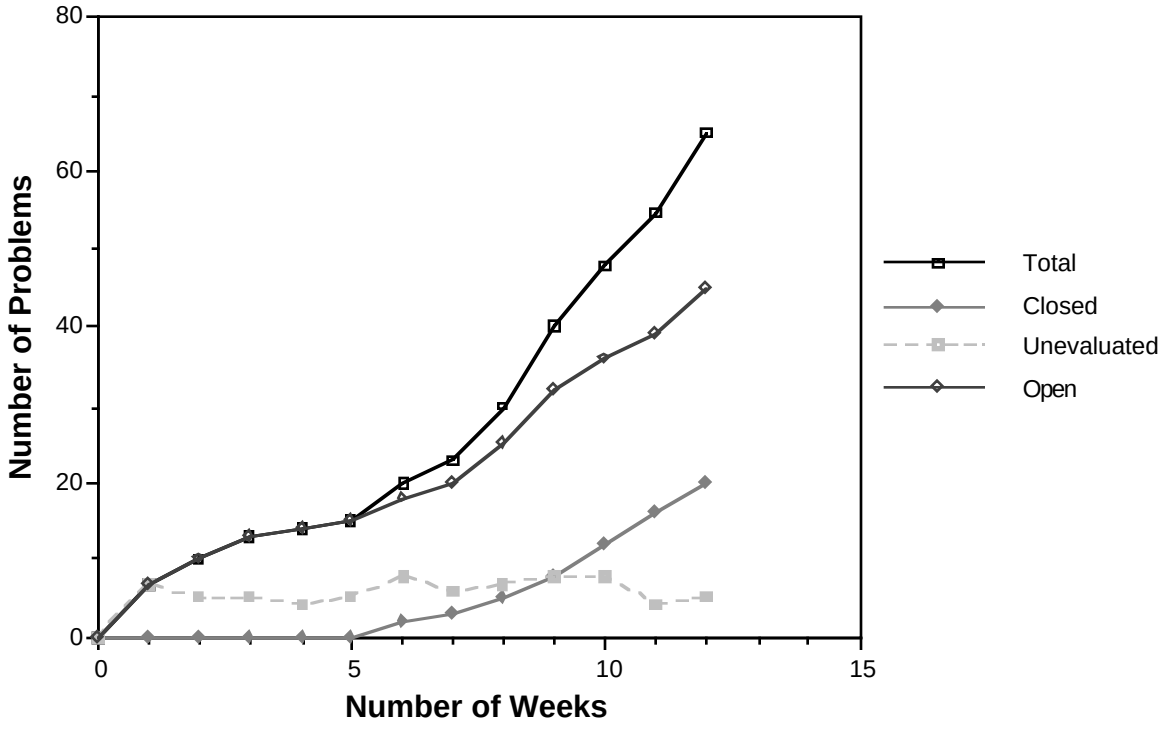
Examples of indicators that others have used are shown here and on the pages that follow.

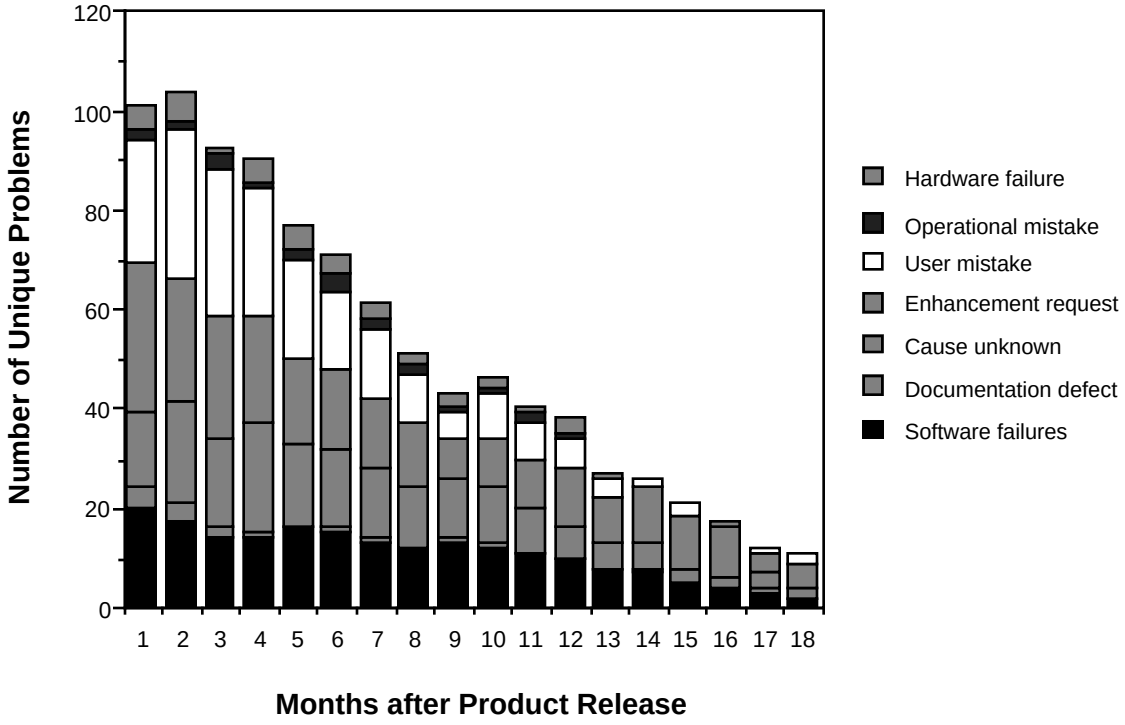


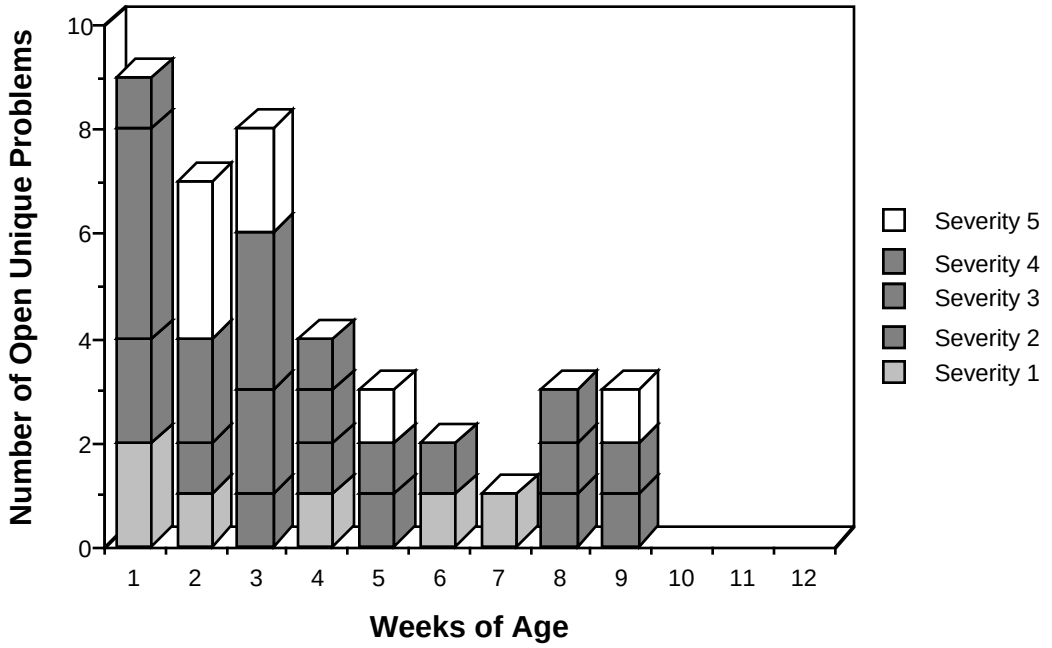


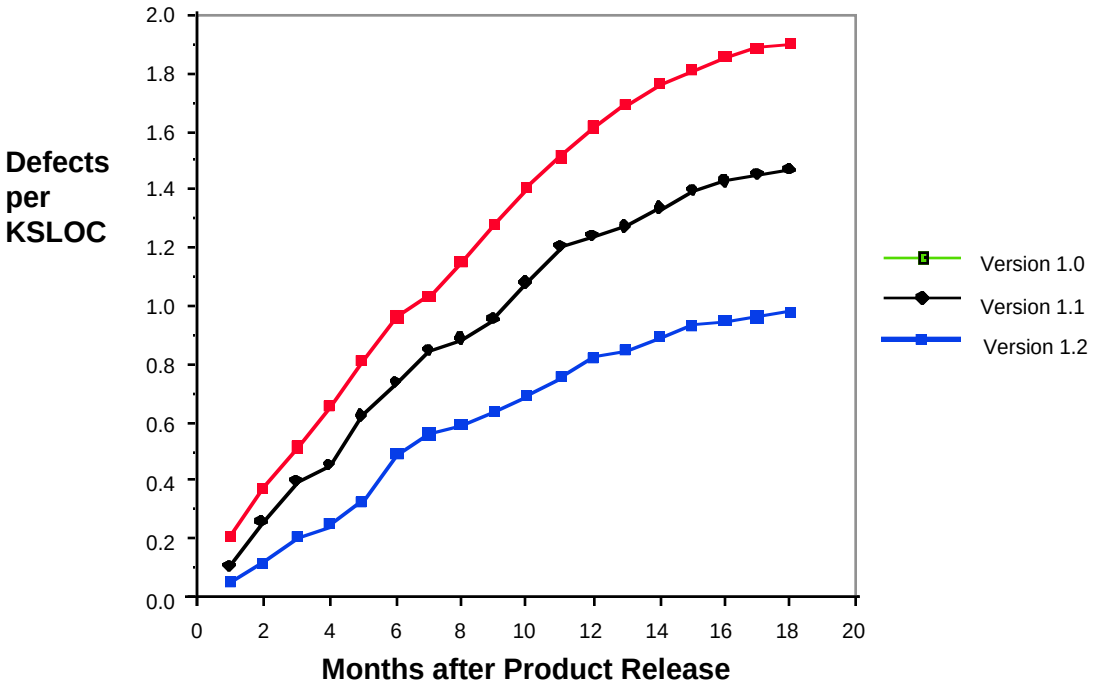

200
100
0
Source Lines
(thousands)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Contract Month
Programmed
Modified
Copied


Number
of Unique
Problems
20
18
16
14
12
10
8
6
4
2
0
1 2 3 4 5 6 7 8 9 10
Customer ID
Rolling three month totals


Number of Problems
80
60
40
20
0
0
5
10
15
Number of Weeks
Total
Closed
Unevaluated
Open


Number of Unique Problems
120
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Months after Product Release
Hardware failure
Operational mistake
User mistake
Enhancement request
Cause unknown
Documentation defect
Software failures

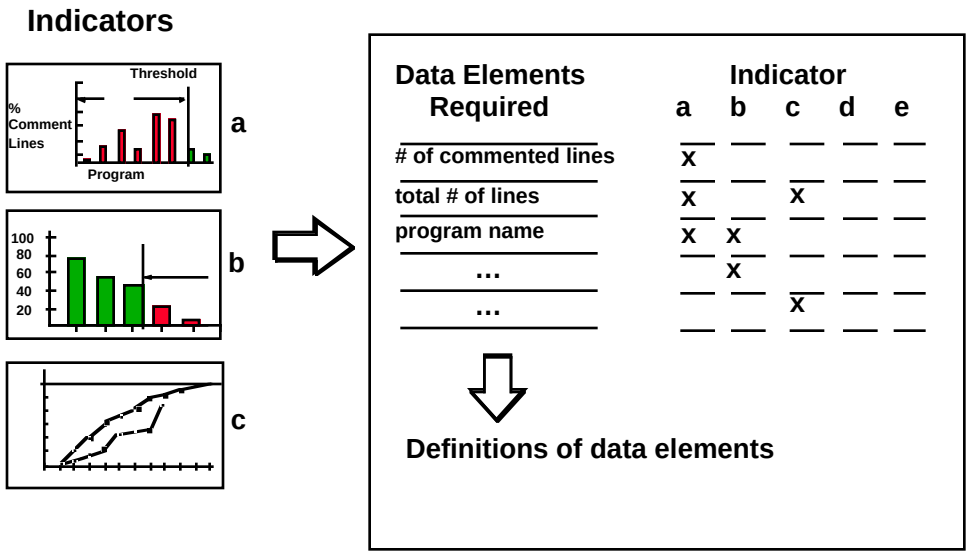

Number of Open Unique Problems
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
10
11
12
Weeks of Age
Severity 5
Severity 4
Severity 3
Severity 2
Severity 1


Defects per KSLOC
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14
16
18
20
Months after Product Release
Version 1.0
Version 1.1
Version 1.2

Exercise 7: Identifying Data Elements

The objectives of this exercise are to identify

1. the data elements that you must collect to construct the indicators that you identified in the previous exercise
2. how you want the data elements to be defined, so that the indicators will show what they purport to show



Directions:

1. Review your results from Exercise 6 (the questions and indicators you identified).
2. Identify the data elements that you will have to collect to construct your indicators.
3. Use the worksheet on the next page to list the data elements and map them back to your indicators.

Worksheet: List of Required Data Elements

What data are required?	Indicator a	b	c	d	e
1 _____	_____	_____	_____	_____	_____
2 _____	_____	_____	_____	_____	_____
3 _____	_____	_____	_____	_____	_____
4 _____	_____	_____	_____	_____	_____
5 _____	_____	_____	_____	_____	_____
6 _____	_____	_____	_____	_____	_____
7 _____	_____	_____	_____	_____	_____
8 _____	_____	_____	_____	_____	_____
9 _____	_____	_____	_____	_____	_____
10 _____	_____	_____	_____	_____	_____
11 _____	_____	_____	_____	_____	_____
12 _____	_____	_____	_____	_____	_____
13 _____	_____	_____	_____	_____	_____
14 _____	_____	_____	_____	_____	_____
15 _____	_____	_____	_____	_____	_____

Exercise 8: Defining Measures

The objective of this exercise is to construct operational definitions for the data elements you will be collecting. You should tell both the collectors and the users of the data exactly what is included in and what is excluded from the measured values, as well as how the data elements are collected.

Checklists and forms that can help you define some of the potentially useful measures are presented in Appendix B. The use of these checklists and forms is discussed and illustrated in [Florac 92], [Goethert 92], and [Park 92]. These reference materials also contain examples of forms that can be used to record and report measurement results.

Directions:

1. Choose one of your indicators for explicit definition.
2. If checklists and forms exist for defining the data elements that you will use to construct your indicators, then use them to create definitions (specifications) for the data that you will collect to meet the requirements for this indicator.

   Alternatively, if checklists and supporting forms do not exist, create them (or construct other structured formats) as necessary, to ensure that you have operational definitions for all data to be collected. This means that people collecting the data must know exactly what is to be included in and excluded from measured values. When the methods used to collect the data could change either the results or the interpretation of the results, ensure that these methods are described.

   (Note: When constructing checklists and supporting forms for defining measures, we have found it most productive to focus not on telling others what to do, but on identifying what you and others need to know to use the data correctly. Once you do this, it is easy to turn the process around to tell others how to collect the data you want.)
3. Repeat Steps 1 and 2 until you have defined explicit data collection rules for all of your data elements.

Exercise 9: Analysis, Diagnosis, Action

The objective of this exercise is to identify the actions that must be taken to get your measurement process up and running. The results will help you prepare a measurement plan for your organization.

Directions:

1. Start with the indicators and data elements that you identified and defined in Exercises 6, 7, and 8.
2. Analyze the extent to which your organization meets the measurement needs for these data elements now.
3. Identify (diagnose) what else your organization must do to meet these needs.
4. Prepare a summary of your analysis, diagnosis, and action-planning status. Use the outlines, questions, and worksheets that follow as guides.

Outline

Analysis

- What data elements are required? (List the data elements your team identified in Exercise 8.)
- What data are currently being collected that would support your needs? Assess the availability and source for each data element. Organize and summarize your results. The figures below show possible ways to do this. A template for the second layout is attached (Worksheet #1).

What software processes are sources for data?

	Planned	Actual
Size	✔	Configuration Management
Effort	✔	Labor Tracking
Quality	✔	Problem Tracking
Schedule	✔	Configuration Management

✔ = estimates from project management

Data Element	Indicator a	b	c	d	e
1	X				X
2	X		X		
3		X			X
4		X		X	
5			X		
6				X	

Avail	Source
+	QA
O	CM
-	?
OO	Etc.
+	•
- -	•

Code	Meaning
+	Available
0	Not explicitly available - can be derived from other data
00	- can be obtained via minor effort
-	Not available now
- -	Impossible to obtain or extremely difficult

- What processes are used to collect the data? How are the data being collected now?
- Are the data stored and retained? Is the information accessible? Who is responsible?
- What tools are available or used to collect, aggregate, and report the data?

Diagnosis

- Does the collected data satisfy your requirements? What's missing or needs improvement?
- Can your existing measurement process(es) be used?
- How frequently should your data be collected?
- Will data collection procedures and recording forms be required?
- Will new or additional tools be required to collect or use this data?

Action

Based on your analysis and diagnosis of the measurement needs in your organization and on your assessment of your current status, identify the tasks that must be accomplished to establish a defined measurement process in your organization. Use Worksheet #2 (or an alternative format) to summarize your status. The figure below gives an example.

	Data element						
Planning tasks	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data elements defined	**Y**	**N**	**60%**	**Not Doc'd**	**Y?**		
Data collection frequencies and points in the software process defined	**50%**	**N**	**60%**	**Not Doc'd**			
Timelines defined for getting measurement results to databases and users	**N**	**N**	**30%**	**Not Doc'd**			
Data collection forms defined	**N**	**N**	**N**	**N**			
Data collection procedures defined	**N**	**N**					
Data storage, database design, and data retention responsibilities defined	**N**	**N**					
Who will collect and who will access the data identified	**N**	**N**					
Analysis processes defined	**N**	**N**					
Reporting processes defined	**N**	**N**					
Supporting tools identified and made available	**N**	**N**					
Process guide for data definition and collection prepared	**Y**						

Worksheet #1: Summary of Data Needs and Availability

Data Element	Indicators a	b	c	d	e	f	Avail.	Source
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								

Availability Codes

Code	Meaning
+	Available
0 00	Not explicitly available - can be derived from other data - can be obtained via minor effort
–	Not available now
– –	Impossible to obtain or extremely difficult

Worksheet #2: Status

Planning tasks	Data element 1	2	3	4	5	6	7
Data elements defined							
Data collection frequencies and points in the software process defined							
Timelines defined for getting measurement results to databases and users							
Data collection forms defined							
Data collection procedures defined							
Data storage, database design, and data retention responsibilities defined							
Who will collect and who will access the data identified							
Analysis processes defined							
Reporting processes defined							
Supporting tools identified and made available							
Process guide for data definition and collection prepared							

Exercise 10: Preparing Your Measurement Plan

The objective of this exercise is to produce an action plan that implements the measures you have defined. A generic outline for the major topics that your plan should address was presented in Section 4.9. You may use that outline as a template for your plan, or you may select an alternative structure that better meets your needs. You will, of course, follow through by getting approval for the plan and by implementing and tracking the measurement and management actions that are called for in the plan.

Directions:

1. Write a plan for implementing the measures you have defined. Use the results of Exercise 9 as your starting point and the template in Section 4.9, together with your results from Exercises 1 through 8, as your guides.
2. Get approval, endorsement, and resources for your plan from your senior managers.
3. Implement the plan.

Appendix B: Checklists and Forms for Defining Measures

This appendix contains examples of checklists and forms that many organizations have found useful for creating and communicating their definitions of some frequently used software measures. These materials are reproduced here so that they will be easily accessible to you when you are accomplishing Exercise 8. You may copy these materials or adapt them to suit your own needs.

For explanations of these materials and for guidelines and examples of their use, please see [Park 92], [Goethert 92], and [Florac 92]. We suggest that your teams obtain copies of these reports and read them, so that they can take advantage of the ideas that are discussed and illustrated therein. This will help you avoid many of the pitfalls of pseudodefinitions that have trapped other organizations.

Problem Count Definition Checklist-1				
Software Product ID [ ]				
Definition Identifier: [ ]		**Definition Date [ ]**		
Attributes/Values	**Definition [X]**		**Specification [ ]**	
Problem Status	**Include**	**Exclude**	**Value Count**	**Array Count**
Open				
Recognized				
Evaluated				
Resolved				
Closed				
Problem Type	**Include**	**Exclude**	**Value Count**	**Array Count**
Software defect				
Requirements defect				
Design defect				
Code defect				
Operational document defect				
Test case defect				
Other work product defect				
Other problems				
Hardware problem				
Operating system problem				
User mistake				
Operations mistake				
New requirement/enhancement				
Undetermined				
Not repeatable/Cause unknown				
Value not identified				
Uniqueness	**Include**	**Exclude**	**Value Count**	**Array Count**
Original				
Duplicate				
Value not identifed				
Criticality	**Include**	**Exclude**	**Value Count**	**Array Count**
1st level (most critical)				
2nd level				
3rd level				
4th level				
5th level				
Value not identified				
Urgency	**Include**	**Exclude**	**Value Count**	**Array Count**
1st (most urgent)				
2nd				
3rd				
4th				
Value not identified				

The use of this checklist is discussed in CMU/SEI-92-TR-22 [Florac 92].
This work has been sponsored by the U.S. Department of Defense. Page 1

Problem Count Definition Checklist–2				
Software Product ID [ ]				
Definition Identifier: [ ]		**Definition Date [ ]**		
Attributes/Values	**Definition [X]**		**Specification [ ]**	
Finding Activity	**Include**	**Exclude**	**Value Count**	**Array Count**
Synthesis of				
Design				
Code				
Test procedure				
User publications				
Inspections of				
Requirements				
Preliminary design				
Detailed design				
Code				
Operational documentation				
Test procedures				
Formal reviews of				
Plans				
Requirements				
Preliminary design				
Critical design				
Test readiness				
Formal qualification				
Testing				
Planning				
Module (CSU)				
Component (CSC)				
Configuration item (CSCI)				
Integrate and test				
Independent verif. and valid.				
System				
Test and evaluate				
Acceptance				
Customer support				
Production/deployment				
Installation				
Operation				
Undetermined				
Value not identified				
Finding Mode	**Include**	**Exclude**	**Value Count**	**Array Count**
Static (non-operational)				
Dynamic (operational)				
Value not identified				

Problem Count Request Form

Product ID, Ver/Rel: [ ] **Problem Count Def ID: [ ]**

Date of Request: [ ] **Requester's Name or ID: [ ]**

Date Count to be made: [ ]

Time Interval for Count: From [ ] To [ ]

Aggregate Time By:	Day	Week	Month	Year
Date opened				
Date closed				
Date evaluated				
Date resolved				
Date/time of occurence				

Report Count By:	Attribute Sort Order	Select Value, Sort Order	Special Instructions or Comments
Originator			
Site ID			
Customer ID			
User ID			
Contractor ID			
Specific ID(s) list			
Environment		Sort Order	
Hardware config ID			
Software config ID			
System config ID			
Test proc ID			
Specific ID(s) list			

Defects Found In:				
Select a configuration	Type of Artifact			
component level:	Requirement	Design	Code	User Document
Product (CSCI)				
Component (CSC)				
Module (CSU)				
Specific (list)				

Changes Made To:				
Select a configuration	Type of Artifact			
component Level:	Requirement	Design	Code	User Document
Product (CSCI)				
Component (CSC)				
Module (CSU)				
Specific (list)				

The use of this checklist is discussed in CMU/SEI-92-TR-22 [Florac 92].

This work has been sponsored by the U.S. Department of Defense.

Problem Status Definition Rules

Product ID: **Status Definition ID:**

Finding Activity ID: **Definition Date:**

Section I

When is a problem considered to be Open? A problem is considered to be Open when all the attributes checked below have a valid value:		When is a problem considered to be Closed? A problem is considered to be Closed when all the attributes checked below have a valid value:	
	Software Product Name or ID		Date Evaluation Completed
	Date/Time of Receipt		Evaluation Completed By
	Date/Time of Problem Occurence		Date Resolution Completed
	Originator ID		Resolution Completed By
	Environment ID		Projected Availability
	Problem Description (text)		Released/Shipped
	Finding Activity		Applied
	Finding Mode		Approved By
	Criticality		Accepted By

Section II

What Substates are used for Open?

#	Name	#	Name
1		6	
2		7	
3		8	
4		9	
5		10	

The use of this checklist is discussed in CMU/SEI-92-TR-22 [Florac 92].

This work has been sponsored by the U.S. Department of Defense.

Problem Status Definition Form-2

Product ID: **Status Definition ID:**

Finding Activity ID: **Definition Date:**

Section III

What attributes are unconditionally required to have values as criteria for each substate?

Attribute		Substate Number									
#	Name	1	2	3	4	5	6	7	8	9	10
1	Problem ID										
2	Software Product Name or ID										
3	Date/Time of Receipt										
4	Date/Time of Problem Occurence										
5	Originator ID										
6	Environment ID										
7	Problem Description (text)										
8	Finding Activity										
9	Finding Mode										
10	Criticality										
11	Problem Type										
12	Uniqueness										
13	Urgency										
14	Date Evaluation Completed										
15	Evaluation Completed By										
16	Date Resolution Completed										
17	Resolution Completed By										
18	ID of Original Problem										
19	Changes Made To										
20	Related Changes										
21	Defect Found In										
22	Defects Caused By										
23	Projected Availability										
24	Released/Shipped										
25	Applied										
26	Approved By:										
27	Accepted By:										

Section IV

List the substates with conditional attribute values		Substates affected	
Substate #	Conditional Attribute/Value	Substate #	Attribute Numbers

Staff-Hour Definition Checklist

Definition Name: _____ Date: _____

Originator: _____

Page: 1 of 3

Type of Labor	Totals include	Totals exclude	Report totals
Direct			
Indirect			
Hour Information			
Regular time			
Salaried			
Hourly			
Overtime			
Salaried			
Compensated (paid)			
Uncompensated (unpaid)			
Hourly			
Compensated (paid)			
Uncompensated (unpaid)			
Employment Class			
Reporting organization			
Full time			
Part time			
Contract			
Temporary employees			
Subcontractor working on task with reporting organization			
Subcontractor working on subcontracted task			
Consultants			
Labor Class			
Software management			
Level 1			
Level 2			
Level 3			
Higher			
Technical analysts & designers			
System engineer			
Software engineer/analyst			
Programmer			
Test personnel			
CSCI-to-CSCI integration			
IV&V			
Test & evaluation group (HW-SW)			
Software quality assurance			
Software configuration management			
Program librarian			
Database administrator			
Documentation/publications			
Training personnel			
Support staff			

The use of this checklist is discussed in CMU/SEI-92-TR-21 [Goethert 92].

This work has been sponsored by the U.S. Department of Defense.

Definition Name: _____ Page: 2 of 3

	Totals include	Totals exclude	Report totals
Activity			
Development			
Primary development activity			
Development support activities			
Concept demo/prototypes			
Tools development, acquisition, installation, & support			
Nondelivered software & test drivers			
Maintenance			
Repair			
Enhancements/major updates			
Product-Level Functions			
CSCI-Level Functions (Major Functional Element)			
Software requirements analysis			
Design			
Preliminary design			
Detailed design			
Code & development testing			
Code & unit testing			
Function (CSC) integration and testing			
CSCI integration & testing			
IV&V			
Management			
Software quality assurance			
Configuration management			
Documentation			
Rework			
Software requirements			
Software implementation			
Redesign			
Recoding			
Retesting			
Documentation			
Build-Level Functions (Customer Release)			
(Software effort only)			
CSCI-to-CSCI integration & checkout			
Hardware/software integration and test			
Management			
Software quality assurance			
Configuration management			
Documentation			
IV&V			

Definition Name: _____ Page: 3 of 3

Product-Level Functions continued	Totals include	Totals exclude	Report totals
System-Level Functions			
(Software effort only)			
System requirements & design			
System requirements analysis			
System design			
Software requirements analysis			
Integration, test, & evaluation			
System integration & testing			
Testing & evaluation			
Production and deployment			
Management			
Software quality assurance			
Configuration management			
Data			
Training			
Development employees			
Customer			
Support			

Supplemental Information Form

Staff-Hours Measurement

Definition Name: _____

Project Name: _____

Hour Information

Indicate the length of the following:

	Hours
Standard work day	
Standard work week	
Standard labor month	

Labor Class Information

Describe the typical responsibilities and duties for the labor categories indicated.

Labor Class	**Description**
Software Management	
Level 1	
Level 2	
Level 3	
Level 4	
Technical analysts and designers	
Programmer	
Test personnel	
Others	

Product-Level Functions

Describe at what level(s) (major functional element, customer release, and/or system) staff hours are counted for the functions indicated.

Function	**Level**
Management	
Software quality assurance	
Configuration management	
Documentation	
Other	

The use of this checklist is discussed in CMU/SEI-92-TR-21 [Goethert 92].
This work has been sponsored by the U.S. Department of Defense.

Schedule Checklist
Part A: Date Information

Date: _____
Originator: _____

Project will record planned dates: Yes _____ No _____
If Yes, reporting frequency: Weekly _____ Monthly _____ Other: _____
Project will record actual dates: Yes _____ No _____
If Yes, reporting frequency: Weekly _____ Monthly _____ Other: _____

Number of builds _____

Milestones, Reviews, and Audits	Include	Exclude	Repeat each build	Relevant dates reported*
System-Level				
System requirements review				
System design review				
CSCI-Level				
Software specification review				
Preliminary design review				
Critical design review				
Code complete				
Unit test complete				
CSC integration and test complete				
Test readiness review				
CSCI functional & physical configuration audits				
System-Level				
Preliminary qualification test				
Formal qualification test				
Delivery & installation				
Other system-level: _____				

*Key to indicate "relevant dates reported" for reviews and audits
1 - Internal review complete
2 - Formal review with customer complete
3 - Sign-off by customer
4 - All high-priority action items closed
5 - All action items closed
6 - Product of activity/phase placed under configuration management
7 - Inspection of product signed off by QA
8 - QA sign-off
9 - Management sign-off
10 - _____
11 - _____

The use of this checklist is discussed in CMU/SEI-92-TR-21 [Goethert 92].
This work has been sponsored by the U.S. Department of Defense.

Part A: Date Information (cont.)

Deliverable Products	Include	Exclude	Repeat each build	Relevant dates reported*
System-Level				
Preliminary system specification				
System/segment specification				
System/segment design document				
Preliminary interface requirements spec.				
Interface requirements specification				
Preliminary interface design document				
Interface design document				
Software development plan				
Software test plan				
Software product specification(s)				
Software user's manual				
Software programmer's manual				
Firmware support manual				
Computer resources integrated support doc.				
Computer system operator's manual				
CSCI-Level				
Preliminary software requirements spec(s)				
Software requirements specification(s)				
Software preliminary design document(s)				
Software (detailed) design document(s)				
Software test description(s) (cases)				
Software test description(s) (procedures)				
Software test report(s)				
Source code				
Software development files				
Version description document(s)				

*Key to indicate "relevant dates reported" for deliverable products

1 - Product under configuration control
2 - Internal delivery
3 - Delivery to customer
4 - Customer comments received
5 - Changes incorporated
6 - Sign-off by customer
7 - _____
8 - _____

Schedule Checklist, cont.

Part B: Progress/Status Information

Project will record planned progress: Yes _____ No _____
If Yes, reporting frequency: Weekly _____ Monthly _____ Other: _____
Project will record actual progress: Yes _____ No _____
If Yes, reporting frequency: Weekly _____ Monthly _____ Other: _____

Activities	Work Units	Work Unit Completion Criterion*
CSCI requirements analysis	Requirements documented or specified	
CSCI preliminary design	Requirements allocated to CSCs	
	CSCs designed	
CSCI detailed design	CSUs designed	
CSU coding and unit testing	Lines coded	
	Lines unit tested	
	Number CSUs coded	
	Number CSUs unit tested	
	Number lines unit tested	
CSCI integration	Number of CSUs integrated	
	Number of lines integrated	
CSCI testing	Number of tests passed	

*Key to indicate "Work Unit Completion Criterion"

1 - None specified
2 - Peer review held
3 - Engineering review held
4 - QA sign-off
5 - Manager or supervisor sign-off
6 - Inspected
7 - Configuration controlled
8 - Entry in employee status report
9 - No known deficiencies
10 - Reviewed by customer
11 - All relevant action items closed
12 - _____
13 - _____

Definition Checklist for Source Statement Counts

Definition name: _____ Date: _____

_____ Originator: _____

Measurement unit:		**Physical source lines** ☐					
		Logical source statements ☐					
Statement type	**Definition** ☐	**Data array** ☐				**Includes**	**Excludes**
	When a line or statement contains more than one type, classify it as the type with the highest precedence.						
1	Executable		**Order of precedence ->**		1		
2	Nonexecutable						
3	Declarations				2		
4	Compiler directives				3		
5	Comments						
6	On their own lines				4		
7	On lines with source code				5		
8	Banners and nonblank spacers				6		
9	Blank (empty) comments				7		
10	Blank lines				8		
11							
12							
How produced	**Definition** ☐	**Data array** ☐				**Includes**	**Excludes**
1	Programmed						
2	Generated with source code generators						
3	Converted with automated translators						
4	Copied or reused without change						
5	Modified						
6	Removed						
7							
8							
Origin	**Definition** ☐	**Data array** ☐				**Includes**	**Excludes**
1	New work: no prior existence						
2	Prior work: taken or adapted from						
3	A previous version, build, or release						
4	Commercial, off-the-shelf software (COTS), other than libraries						
5	Government furnished software (GFS), other than reuse libraries						
6	Another product						
7	A vendor-supplied language support library (unmodified)						
8	A vendor-supplied operating system or utility (unmodified)						
9	A local or modified language support library or operating system						
10	Other commercial library						
11	A reuse library (software designed for reuse)						
12	Other software component or library						
13							
14							
Usage	**Definition** ☐	**Data array** ☐				**Includes**	**Excludes**
1	In or as part of the primary product						
2	External to or in support of the primary product						
3							

The use of this checklist is discussed in CMU/SEI-92-TR-20 [Park 92].

This work has been sponsored by the U.S. Department of Defense.

Definition name: _____

_____

Delivery	Definition ☐	Data array ☐	Includes	Excludes
1 Delivered				
2 Delivered as source				
3 Delivered in compiled or executable form, but not as source				
4 Not delivered				
5 Under configuration control				
6 Not under configuration control				
7				

Functionality	Definition ☐	Data array ☐	Includes	Excludes
1 Operative				
2 Inoperative (dead, bypassed, unused, unreferenced, or unaccessed)				
3 Functional (intentional dead code, reactivated for special purposes)				
4 Nonfunctional (unintentionally present)				
5				
6				

Replications	Definition ☐	Data array ☐	Includes	Excludes
1 Master source statements (originals)				
2 Physical replicates of master statements, stored in the master code				
3 Copies inserted, instantiated, or expanded when compiling or linking				
4 Postproduction replicates—as in distributed, redundant, or reparameterized systems				
5				

Development status	Definition ☐	Data array ☐	Includes	Excludes
Each statement has one and only one status, usually that of its parent unit.				
1 Estimated or planned				
2 Designed				
3 Coded				
4 Unit tests completed				
5 Integrated into components				
6 Test readiness review completed				
7 Software (CSCI) tests completed				
8 System tests completed				
9				
10				
11				

Language	Definition ☐	Data array ☐	Includes	Excludes
List each source language on a separate line.				
1 _____				
2 Job control languages _____				
3 _____				
4 Assembly languages _____				
5 _____				
6 Third generation languages _____				
7 _____				
8 Fourth generation languages _____				
9 _____				
10 Microcode _____				
11				

Definition name: _____		Includes	Excludes
Clarifications (general)	**Listed elements are assigned to statement type –>**		
1 Nulls, continues, and no-ops			
2 Empty statements (e.g., ";;" and lone semicolons on separate lines)			
3 Statements that instantiate generics			
4 Begin…end and {…} pairs used as executable statements			
5 Begin…end and {…} pairs that delimit (sub)program bodies			
6 Logical expressions used as test conditions			
7 Expression evaluations used as subprogram arguments			
8 End symbols that terminate executable statements			
9 End symbols that terminate declarations or (sub)program bodies			
10 Then, else, and otherwise symbols			
11 Elseif statements			
12 Keywords like procedure division, interface, and implementation			
13 Labels (branching destinations) on lines by themselves			
14			
15			
16			
Clarifications (language specific)			
Ada			
1 End symbols that terminate declarations or (sub)program bodies			
2 Block statements (e.g., begin…end)			
3 With and use clauses			
4 When (the keyword preceding executable statements)			
5 Exception (the keyword, used as a frame header)			
6 Pragmas			
7			
8			
9			
Assembly			
1 Macro calls			
2 Macro expansions			
3			
4			
5			
6			
C and C++			
1 Null statement (e.g., ";" by itself to indicate an empty body)			
2 Expression statements (expressions terminated by semicolons)			
3 Expressions separated by semicolons, as in a "for" statement			
4 Block statements (e.g., {…} with no terminating semicolon)			
5 "{", "}", or "};" on a line by itself when part of a declaration			
6 "{" or "}" on line by itself when part of an executable statement			
7 Conditionally compiled statements (#if, #ifdef, #ifndef)			
8 Preprocessor statements other than #if, #ifdef, and #ifndef			
9			
10			
11			
12			

Definition name: _____		Includes	Excludes
CMS-2	**Listed elements are assigned to**		
1 Keywords like SYS-PROC and SYS-DD	**statement type –>**		
2			
3			
4			
5			
6			
7			
8			
9			
COBOL			
1 “PROCEDURE DIVISION”, “END DECLARATIVES”, etc.			
2			
3			
4			
5			
6			
7			
8			
9			
FORTRAN			
1 END statements			
2 Format statements			
3 Entry statements			
4			
5			
6			
7			
8			
JOVIAL			
1			
2			
3			
4			
5			
6			
7			
8			
Pascal			
1 Executable statements not terminated by semicolons			
2 Keywords like INTERFACE and IMPLEMENTATION			
3 FORWARD declarations			
4			
5			
6			
7			
8			
9			

Definition name: _____		Includes	Excludes
	Listed elements are assigned to statement type –>		
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

Summary of Statement Types

Executable statements

Executable statements cause runtime actions. They may be simple statements such as assignments, goto's, procedure calls, macro calls, returns, breaks, exits, stops, continues, nulls, no-ops, empty statements, and FORTRAN's END. Or they may be structured or compound statements, such as conditional statements, repetitive statements, and "with" statements. Languages like Ada, C, C++, and Pascal have block statements [begin…end and {…}] that are classified as executable when used where other executable statements would be permitted. C and C++ define expressions as executable statements when they terminate with a semicolon, and C++ has a <declaration> statement that is executable.

Declarations

Declarations are nonexecutable program elements that affect an assembler's or compiler's interpretation of other program elements. They are used to name, define, and initialize; to specify internal and external interfaces; to assign ranges for bounds checking; and to identify and bound modules and sections of code. Examples include declarations of names, numbers, constants, objects, types, subtypes, programs, subprograms, tasks, exceptions, packages, generics, macros, and deferred constants. Declarations also include renaming declarations, use clauses, and declarations that instantiate generics. Mandatory begin…end and {…} symbols that delimit bodies of programs and subprograms are integral parts of program and subprogram declarations. Language superstructure elements that establish boundaries for different sections of source code are also declarations. Examples include terms such as PROCEDURE DIVISION, DATA DIVISION, DECLARATIVES, END DECLARATIVES, INTERFACE, IMPLEMENTATION, SYS-PROC, and SYS-DD. Declarations, in general, are never required by language specifications to initiate runtime actions, although some languages permit compilers to implement them that way.

Compiler Directives

Compiler directives instruct compilers, preprocessors, or translators (but not runtime systems) to perform special actions. Some, such as Ada's pragma and COBOL's COPY, REPLACE, and USE, are integral parts of the source language. In other languages like C and C++, special symbols like # are used along with standardized keywords to direct preprocessor or compiler actions. Still other languages rely on nonstandardized methods supplied by compiler vendors. In these languages, directives are often designated by special symbols such as #, $, and {$}.

Rules for Counting Physical Source Lines

For each source language to which the definition applies, provide the following information:

Language name:

Note: This information is required only for statement types that are excluded from counts or for which individual counts are recorded.

Executable lines: List the rules used to identify executable lines. If special rules are used for constructs such as block statements, embedded statements, empty statements, or embedded comments, describe them.

Declarations: List the rules used to identify declaration lines. Explain how declarations are distinguished from executable statements.

Compiler directives: List the rules used to identify compiler directives.

Comments: List the rules used to identify beginnings and endings of comments.

Modified comments: If separate counts are made for modified lines, list the rules used to keep modifications to comments on lines with other code from being classified as modified statements of higher precedence.

Special rules: List any special rules that are used to classify the first or last statements of any sections of code.

The use of this form is discussed in CMU/SEI-92-TR-20 [Park 92].

This work has been sponsored by the U.S. Department of Defense.

Rules for Counting Logical Source Statements

For each source language to which this definition applies, provide the following information:

Language name:

Executable statements: List all rules and delimiters used to identify beginnings and endings of executable statements. If special rules are used for constructs such as block statements, embedded statements, empty statements, expression statements, or subprogram arguments, describe them.

Comments: If comments are counted, list the rules used to identify beginnings and endings of comment *statements*. Explain how, if at all, comment *statements* differ from physical source lines.

Declarations: List the rules and delimiters used to identify beginnings and endings of declarations. Explain how declarations are distinguished from executable statements.

Special rules: List any special rules or delimiters that are used to identify the first or last statements of any sections of code.

Compiler directives: List the rules and delimiters used to identify beginnings and endings of compiler directives.

Exclusions: List all keywords and symbols that, although set off by statement delimiters, are not counted as logical source statements.

The use of this form is discussed in CMU/SEI-92-TR-20 [Park 92].

This work has been sponsored by the U.S. Department of Defense.

Practices Used to Identify Inoperative Elements

List or explain the methods or rules used to identify:
Intentionally bypassed statements and declarations

Unintentionally included dead code

A. Unreachable, bypassed, or unreferenced elements (declarations, statements, or data stores) within modules:

B. Unused, unreferenced, or unaccessed modules or include files in code libraries:

C. Unused modules, procedures, or functions, linked into delivered products:

The use of this form is discussed in CMU/SEI-92-TR-20 [Park 92].
This work has been sponsored by the U.S. Department of Defense.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **AGENCY USE ONLY** (leave blank)	2. **REPORT DATE** August 1996	3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** Goal-Driven Software Measurement —A Guidebook		5. **FUNDING NUMBERS** C — F19628-95-C-0003
6. **AUTHOR(S)** Robert E. Park, Wolfhart B. Goethert, William A. Florac		
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-96-HB-002
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116		10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. **SUPPLEMENTARY NOTES**		
12.a **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS		12.b **DISTRIBUTION CODE**

13. **ABSTRACT** (maximum 200 words)

The materials in this guidebook are designed to help you identify, select, define, and implement software measures to support your business goals. The measures that result are traceable back to your business goals, so that data collection efforts are better able to stay focused on their intended objectives.

14. **SUBJECT TERMS**			15. **NUMBER OF PAGES** 212 16. **PRICE CODE**
17. **SECURITY CLASSIFICATION OF REPORT** UNCLASSIFIED	18. **SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	19. **SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	20. **LIMITATION OF ABSTRACT** UL