# 12 Steps to Useful Software Metrics

**Linda Westfall**

**The Westfall Team**

**westfall@idt.net**

**PMB 101, 3000 Custer Road, Suite 270
Plano, TX 75075**

**972-867-1172 (voice)
972-943-1484 (fax)**

**Abstract:** ***12 Steps to Useful Software Metrics*** introduces the reader to a practical process for establishing and tailoring a software metrics program that focuses on goals and information needs. The process provides a practical, systematic, start-to-finish method of selecting, designing and implementing software metrics. It outlines a cookbook method that the reader can use to simplify the journey from software metrics in concept to delivered information.

**Bio:** Linda Westfall is the President of The Westfall Team, which provides Software Metrics and Software Quality Engineering training and consulting services. Prior to starting her own business, Linda was the Senior Manager of the Quality Metrics and Analysis at DSC Communications where her team designed and implemented a corporate wide metric program. Linda has more than twenty-five years of experience in real-time software engineering, quality and metrics. She has worked as a Software Engineer, Systems Analyst, Software Process Engineer and Manager of Production Software.

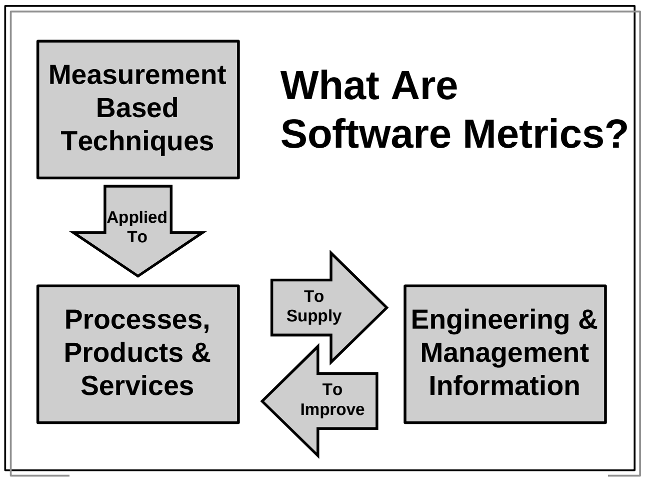
Very active professionally, Linda Westfall is a past chair of the American Society for Quality (ASQ) Software Division. She has also served as the Software Division's Program Chair and Certification Chair and on the ASQ National Certification Board. Linda is a past-chair of the Association for Software Engineering Excellence (ASEE) and has chaired several conference program committees.

Linda Westfall has an MBA from the University of Texas at Dallas and BS in Mathematics from Carnegie-Mellon University. She has her Professional Engineer (P.E.) license in software engineering from the State of Texas. She is an ASQ Certified Software Quality Engineer (CSQE) and Certified Quality Auditor (CQA).

**Key Words/Phrases:** Software Metrics, Software Measurement, Data Collection

# 12 Steps to Useful Software Metrics

## What Are Software Metrics?



Measurement Based Techniques
What Are Software Metrics?
Applied To
Processes, Products & Services
To Supply
To Improve
Engineering & Management Information

Figure 1: What Are Software Metrics?

Software metrics are an integral part of the state-of-the-practice in software engineering. More and more customers are specifying software and/or quality metrics reporting as part of their contractual requirements. Industry standards like ISO 9000 and industry models like the Software Engineering Institute's (SEI) Capability Maturity Model Integrated (CMMI®) include measurement. Companies are using metrics to better understand, track, control and predict software projects, processes and products.

The term *software metrics* means different things to different people. When we buy a book or pick up an article on software metrics, the topic can vary from project cost and effort prediction and modeling, to defect tracking and root cause analysis, to a specific test coverage metric, to computer performance modeling. These are all examples of *metrics* when the word is used as a noun.

I prefer the activity based view taken by Goodman. He defines software metrics as, "The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products." [Goodman-93] Figure 1, illustrates an expansion of this definition to include software-related services such as installation and responding to customer issues. Software metrics can provide the information needed by engineers for technical decisions as well as information required by management.

If a metric is to provide useful information, everyone involved in selecting, designing, implementing, collecting, and utilizing it must understand its definition and purpose. This paper outlines twelve steps to selecting, designing and implementing software metrics in order to insure this understanding.

## Some Basic Measurement Theory

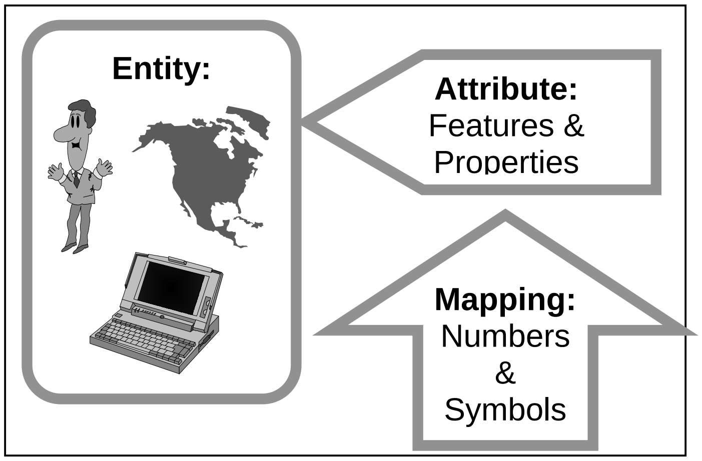


Figure 2: Measurement Defined

The use of measurement is common. We use measurements in everyday life to do such things as weigh ourselves in the morning or when we check the time of day or the distance we have traveled in our car. Measurements are used extensively in most areas of production and manufacturing to estimate costs, calibrate equipment, assess quality, and monitor inventories. Science and engineering disciplines depend on the rigor that measurements provide, but what does measurement really mean?

According to Fenton, "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 describe them according to clearly defined rules" [Fenton-91]. An entity is a person, place, thing, event or time period. An attribute is a feature or property of the entity.

To measure, we must first determine the entity. For example, we could select a car as our entity. Once we select an entity, we must select the attribute of that entity that we want to describe. For example, the car’s speed or the pressure in its tires would be two attributes of a car. Finally, we must have a defined and accepted mapping system. It is meaningless to say that the car’s speed is 65 or its tire pressure is 75 unless we know that we are talking about miles per hour and pounds per square inch, respectively.

We will use the basic process model of input - process - output to discuss software entities. Software entities of the input type include all of the resources used for software research, development, and production. Examples of input entities include people, materials, tools, and methods. Software entities of the process type include software-related activities and events and are usually associated with a time factor. Examples of process entities include defined activities such as developing a software system from requirements through delivery to the customer, the inspection of a piece of code, or the first 6 months of operations after delivery. Process entities also include time periods, which do not necessarily correspond to specific activities. An example would be the period between 1/1/93 and 2/1/93. Software entities of the output type are the products of the software process. These include all the artifacts, deliverables, and documents that are produced. Examples of software output entities include requirements documentation, design specifications, code (source, object & executable), test documentation (plans, scripts, specifications, cases, reports), project plans, status reports, budgets, problem reports, and software metrics.

Each of these software entities has many properties or features that we might want to measure. We might want to examine a computer's price, performance, or usability. We could look at the time or effort that it took to execute a process, the number of incidents that occurred during the process, its cost, controllability, stability, or effectiveness. We might want to measure the complexity, size, modularity, testability, usability, reliability, or maintainability of a piece of source code.

One of the challenges of software metrics is that few standardized mapping systems exist. Even for seemingly simple metrics like the number of lines of code, no standard counting method has been widely accepted. Do we count physical or logical lines of code? Do we count comments or data definition statements? Do we expand macros before counting and do we count the lines in those macros more than once? Another example is engineering hours for a project – besides the effort of software engineers, do we include the effort of testers, managers, secretaries, and other support personnel? A few metrics, which do have standardized counting criteria include McCabe’s Cyclomatic Complexity and the Function Point Counting Standard from the International Function Point User Group (IFPUG). However, the selection, definition, and consistent use of a mapping system within the organization for each selected metric are critical to a successful metrics program.

## Introduction to the Twelve Steps

Based on all of the possible software entities and all the possible attributes of each of those entities, there are multitudes of possible software metrics. How do we pick the metrics that are right for our organizations? The first four steps defined in this paper will illustrate how to identify metrics customers and then utilize the goal/question/metric paradigm to select the software metrics that match the information needs of those customers. Steps 5-10 present the process of designing and tailoring the selected metrics, including definitions, models, counting criteria, benchmarks and objectives, reporting mechanisms, and additional qualifiers. The last two steps deal with implementation issues, including data collection and the minimization of the impact of human factors on metrics.

When I first started doing software metrics there seemed to be two schools of thought. The first said collect data on everything and then analyze the data to find correlation, meaning or information. The second school of thought was what I call the shotgun method of metrics. This usually involved collecting and reporting on whatever the current "hot" metrics were or using whatever data was available as a by-product of software development to produce metrics.

These methods are both what I call the Jeopardy approach to metrics. You know the game show Jeopardy – where they start with the answer and the contestants try to guess what the question is. In the Jeopardy approach to metrics we start with the metric and try to guess what it tells us about our software processes, products or services.

There are serious problems with both of these methods. The problem with the first method is that if we consider all of the possible software entities and all of their possible attributes that can be measured, there are just too many measures.  It would be easy to drown an organization in the enormity of the task of trying to measure everything.  One of my favorite quotes talks about "spending all of our time reporting on the nothing we are doing because we are spending all of our time reporting."  The problem with the second method can be illustrated in Watts Humphrey's quote, "There are so many possible measures in a complex software process that some random selection of metrics will not likely turn up anything of value." [Humphrey-89]  In other words, Murphy's Law applies to software metrics.  The one item that is not measured is the one item that should be measured.

There has been a fundamental shift in the philosophy of software metrics.  Software metrics programs are now being designed to provide the specific information necessary to manage software projects and improve software engineering processes and services.  Organizational, project and task goals are determined in advance and then metrics are selected based on those goals.  These metrics are used to determine our effectiveness in meeting our goals.  The foundation of this approach is aimed at making practitioners ask not so much "What should I measure?" but "Why am I measuring?" or "What business needs does the organization wish its measurement initiative to address?" [Goodman-93]

Measuring software is a powerful way to track progress towards project goals.  As Grady states, "Without such measures for managing software, it is difficult for any organization to understand whether it is successful, and it is difficult to resist frequent changes of strategy." [Grady-92]  Appropriately selected metrics can help both management and engineers maintain their focus on their goals.

## Step 1 – Identify Metrics Customers

The first step of the "12 Steps to Useful Software Metrics" is to identify the customers for each metric.  The customer of the metric is the person (or people) who will be making decisions or taking action based upon the metric; the person/people who needs the information supplied by the metric.

There are many different types of customers for a metrics program.  This adds complexity to the program because each customer may have different information requirements.  Customers may include:

- ***Functional Management:*** Interested in applying greater control to the software development process, reducing risk and maximizing return on investment.
- ***Project Management:*** Interested in being able to accurately predict and control project size, effort, resources, budgets and schedules.  Interested in controlling the projects they are in charge of and communicating facts to their management.
- ***Software Engineers/Programmers:*** The people that actually do the software development.  Interested in making informed decisions about their work and work products.  These people are responsible for collecting a significant amount of the data required for the metrics program.
- ***Test Managers/Testers:***  The people responsible for performing the verification and validation activities.  Interested in finding as many new defects as possible in the time allocated to testing and in obtaining confidence that the software works as specified.  These people are also responsible for collecting a significant amount of the required data.
- ***Specialists:*** Individuals performing specialized functions (e.g., Marketing, Software Quality Assurance, Process Engineering, Software Configuration Management, Audits and Assessments, Customer Technical Assistance).  Interested in quantitative information upon which they can base their decisions, finding and recommendations.
- ***Customers/Users:*** Interested in on-time delivery of high quality software products and in reducing the over-all cost of ownership.

If a metric does not have a customer, it should not be produced.  Metrics are expensive to collect, report, and analyze so if no one is using a metric, producing it is a waste of time and money.

The customers' information requirements should always drive the metrics program.  Otherwise, we may end up with a product without a market and with a program that wastes time and money.  By recognizing potential customers and involving those customers early in the metric definition effort, the chances of success are greatly increased.

## Step 2 – Target Goals

Basili and Rombach [Basili-88] define a Goal/Question/Metric paradigm that provides an excellent mechanism for defining a goal-based measurement program. Figure 3 illustrates the Goal/Question/Metric paradigm.

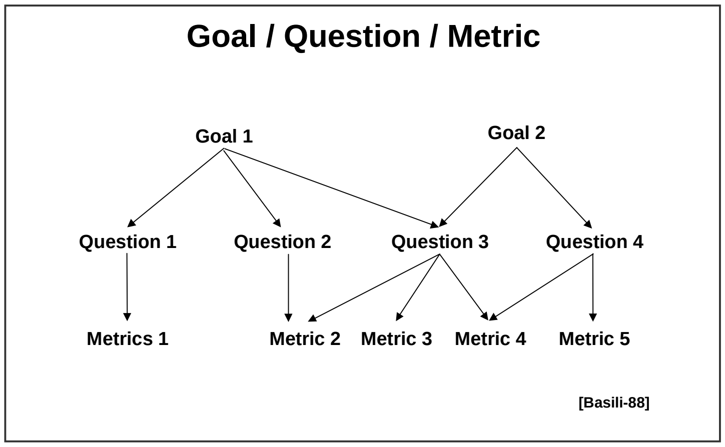


Figure 3: Goal/Question/Metric Paradigm

The second step in setting up a metrics program is to select one or more measurable goals. The goals we select to use in the Goal/Question/Metric will vary depending on the level we are considering for our metrics. At the organizational level, we typically examine high-level strategic goals like being the low cost provider, maintaining a high level of customer satisfaction, or meeting projected revenue or profit margin target. At the project level, we typically look at goals that emphasize project management and control issues or project level requirements and objectives. These goals typically reflect the project success factors like on time delivery, finishing the project within budget or delivering software with the required level of quality or performance. At the specific task level, we consider goals that emphasize task success factors. Many times these are expressed in terms of the entry and exit criteria for the task.

Software metrics programs must be designed to provide the specific information necessary to manage software projects and improve software engineering processes and services. Organizational, project, and task goals are determined in advance and then metrics are selected based on those goals. The metrics are used to determine our effectiveness in meeting these goals.

When talking to our customers, we may find many of their individual needs are related to the same goal or problem but expressed from their perspective or in the terminology of their specialty. Many times, what we hear is their frustrations.

For example, the Project Manager may need to improve the way project schedules are estimated. The Functional Manager is worried about late deliveries. The practitioners complain about overtime and not having enough time to do things correctly. The Test Manager states that by the time the test group gets the software it's too late to test it completely before shipment.

When selecting metrics, we need to listen to these customers and, where possible, consolidate their various goals or problems into statements that will help define the metrics that are needed by our organization or team.

In our example, all these individuals are asking for an improved and realistic schedule estimation process.

## Step 3 – Ask Questions

The third step is to define the questions that need to be answered in order to ensure that each goal is being obtained. For example, if our goal was to ship only defect-free software, we might select the questions:

- Is the software product adequately tested?
- How many defects are still undetected?
- Are all known defects corrected?

## Step 4 – Select Metrics

The fourth step is to select metrics that provide the information needed to answer these questions. Each selected metric now has a clear objective -- to answer one or more of the questions that need to be answered to determine if we are meeting our goals.

When we are selecting metrics, we must be practical, realistic, and pragmatic.  Avoid the "ivory-tower" perspective that is completely removed from the existing software-engineering environment.  Start with what is possible within the current process.  Once we have a few successes, our customers will be open to more radical ideas -- and may even come up with a few of their own.

Also, remember software metrics don't solve problems.  People solve problems.  Software metrics act as indicators and provide information so people can make more informed decisions and intelligent choices.

An individual metric performs one of four functions.  Metrics can help us ***Understand*** more about our software products, processes, and services.  Metrics can be used to ***Evaluate*** our software products, processes, and services against established standards and goals.  Metrics can provide the information we need to ***Control*** resources and processes used to produce our software.  Metrics can be used to ***Predict*** attributes of software entities in the future. [Humphrey-89].  A comprehensive Software Metric program would include metrics that perform all of these functions.

A requirements statement for each metric can be formally defined in terms of one of these four functions, the attribute of the entity being measured and the goal for the measurement.  This leads to the following metrics objective template:

$$\textbf{To}\begin{bmatrix}\textbf{\textit{understand}}\\ \textbf{\textit{evaluate}}\\ \textbf{\textit{control}}\\ \textbf{\textit{predict}}\end{bmatrix}\textbf{the}\begin{bmatrix}\textbf{\textit{attribute}}\\ \textbf{\textit{of the}}\\ \textbf{\textit{entity}}\end{bmatrix}\begin{matrix}\textbf{in order}\\ \textbf{to}\end{matrix}\begin{bmatrix}\textbf{\textit{goal(s)}}\end{bmatrix}$$

An example of the use of this template for the “percentage of known defects corrected” metric would be:

$$\textbf{To}\begin{bmatrix}\textbf{\textit{evaluate}}\end{bmatrix}\textbf{the}\begin{bmatrix}\textbf{\textit{\% known}}\\ \textbf{\textit{defects}}\\ \textbf{\textit{corrected}}\\ \textbf{\textit{during}}\\ \textbf{\textit{development}}\end{bmatrix}\begin{matrix}\textbf{in order}\\ \textbf{to}\end{matrix}\begin{bmatrix}\textbf{\textit{all known}}\\ \textbf{\textit{defects are}}\\ \textbf{\textit{corrected}}\\ \textbf{\textit{before}}\\ \textbf{\textit{shipment}}\end{bmatrix}$$

Having a clearly defined and documented requirements statement for each metric has the following benefits:

- Provides a rigor and discipline that helps ensure a well-defined metric based on customer goals
- Eliminates misunderstandings about how the metric is intended to be used
- Communicates the need for the metric, which can help in obtaining resources to implement the data collection and reporting mechanisms
- Provides the basis for the design of the metric

## Step 5 – Standardize Definitions

The fifth step is to agree to standard definitions for the entities and their measured attributes.  When we use terms like *defect, problem report, size,* and even *project*, other people will interpret these words in their own context with meanings that may differ from our intended definition.  These interpretation differences increase when more ambiguous terms like *quality*, *maintainability*, and *user-friendliness* are used.

Additionally, individuals may use different terms to mean the same thing.  For example, the terms *defect report, problem report, incident report, fault report,* or *customer call report* may be used by various organizations to mean the same thing, but unfortunately they may also refer to different entities.  One external customer may use *customer call report* to refer to their complaint and *problem report* as the description of the defect in the software, while another customer may use *problem report* for the initial complaint.  Differing interpretations of terminology may be one of the biggest barriers to understanding.

Unfortunately, there is little standardization in the industry of the definitions for most software attributes.  Everyone has an opinion and the debate will probably continue for many years.  Our metrics program cannot wait that long.  The approach I suggest is to adopt standard definitions within your organization and then apply them consistently.  You can use those definitions that do exist within the industry as a foundation to get you started. For example, definitions from the IEEE Glossary of Software Engineering Terminology [IEEE-610] or those found in software engineering and metrics literature. Pick the definitions that match with your organizational objectives or use them as a basis for creating your own definition.

## Step 6 – Choose a Measurement Function

The sixth step is to choose a measurement function for the metric. In simple terms, the measurement function defines how we are going to calculate the metric. Some metrics, called base measures or metric primitives, are measured directly and their measurement function typically consists of a single variable. Examples of base measures include the number of lines of code reviewed during an inspection or the hours spent preparing for an inspection meeting. Other more complex metrics, called derived measures are modeled using mathematical combinations (e.g., equations or algorithms) of base measures or other derived measures. An example of a derived measure would be the inspection's preparation rate, modeled as the number of lines of code reviewed divided by the number of preparation hours.

Many measurement models include an element of simplification. This is both the strength and the weakness of using modeling. When we create a model to use as our measurement function, we need to be pragmatic. If we try to include all of the elements that affect the attribute or characterize the entity, our model can become so complicated that it's useless. Being pragmatic means not trying to create the most comprehensive model. It means picking the aspects that are the most important. Remember that the model can always be modified to include additional levels of detail in the future. Ask yourself the questions:

- Does the model provide more information than we have now?
- Is the information of practical benefit?
- Does it tell us what we want to know?

There are two methods for selecting a model to use as our measurement function: use an existing model or create a new one. In many cases, there is no need to "re-invent the wheel." Many software metrics models exist that have been used successfully by other organizations. These are documented in the current literature and in proprietary products that can be purchased. With a little research, we can utilize these models with little or no adaptation to match our own environment.

The second method is to create our own model. The best advice here is to talk to the people who are actually responsible for the product or resource or who are involved in the process. They are the experts. They know what factors are important. If we create a new model for our metric, we must ensure the model is intelligible to our customers and we must prove it is a valid model for what we are trying to measure. Often, this validation can occur only through application of statistical techniques.

To illustrate the selection of a model, let's consider a metric for the duration of unplanned system outages. If we are evaluating a software system installed at a single site, a simple model such as minutes of outage per calendar month may be sufficient. If our objective is to compare different software releases installed on varying numbers of sites, we might select a model such as minutes of outage per 100 operation months. If we wanted to focus in on the impact to our customers, we might select minutes of outage per site per year.

## Step 7 – Establish a Measurement Method

The seventh step in designing a metric is to break the function down into its lowest level base measures (metric primitives) and define the measurement method used to assign a value to each of those base measures. The measurement method defines the mapping system that is used to assign numbers or symbols to the attributes. The base measures and their counting criteria define the first level of data that needs to be collected in order to implement the metric.

Some measurement methods are established by using standardized units of measure (e.g., inches, feet, pounds, liters, dollars, hours, days). Other measurement methods are based on counting criteria, which are simple counts of items with certain characteristics. For example, if the metric is the problem report arrival rate per month, we could simply count all of the problem reports in the database that had an open date during each month. However, if we wanted defect counts instead of just problem report counts we might exclude all the reports that didn't result from a product defect (e.g., works as designed, user error, withdrawn). Other rules may also be used for the measurement method. For example, what rules does your organization use for assigning severity to a problem report? These rules might include judging how much of the software's functionally is impacted (e.g., more than 50%, >=50% but < 20%, >= 20%) or the duration of that impact (for more than 60 second, >=60 seconds < 15 seconds, >- 15 seconds) or other criteria.

The importance of the need for defining a measurement method can be illustrated by considering the lines of code metric. The lines of code measure is one of the most used and most often misused of all of the software metrics. The problems, variations, and anomalies of using lines of code are well documented [Jones-86], and there is no industry-accepted standard for counting lines of code. Therefore, if you are going to use a metric based on lines of code, it is critical that specific measurement method be defined. This method should also accompany the metric in all reports and analysis so that metrics customers can understand the definition of the metric. Without this, invalid comparisons with other data are almost inevitable.

The base measures and their measurement methods define the first level of data that needs to be collected in order to implement the metric. To illustrate this, let's use the model of minutes of system outage per site per year. One of the metrics primitives for this model is the number of sites. At first, counting this primitive seems simple. However, when we consider the dynamics of adding new sites or installing new software on existing sites, the counting criteria become more complex. Do we use the number of sites on the last day of the period or calculate some average number of sites for the period? Either way, we will need to collect data on the date the system was installed on the site. In addition, if we intend to compare different releases of the software we will need to collect data on what releases have been installed on each site and when each was installed.

## Step 8 – Define Decision Criteria

The eighth step in designing a metric is defining decision criteria. Once we have decided what to measure and how to measure it, we have to decide what to do with the results. According to the ISO/IEC 15939 Software Engineering -- Software Measurement Process standard, decision criteria are the "thresholds, targets, or patterns used to determine the need for action or further investigation, or to describe the level of confidence in a given result". [ISO/IEC-15929] In other words, you need decision criteria to obtain guidance that will help you interpret the measurement results.

Control type metrics are used to monitor our software processes, products and services and identify areas where corrective or management action is required. Control type metrics act as "red-flags" to warn us when things are not as expected or planned. If all is going well, the decision should be "everything is fine and therefore no action is required." The decision criteria for control type metrics usually take the form of: [Westfall-03]

- Thresholds
- Variances
- Control limits

Evaluate type metrics are used to examine and analyze the measurement information as part of our decision-making processes. For evaluate type metrics, decision criteria help define "what good." For example, if we want to make a 15% return on investment (ROI) for our project, the benefit to cost ration would have to be at least 1.15. We might also establish decision criteria for our exit from system test that include:

- At least X% of all planned test cases must be executed and at least Y% of those must have passed
- No non-closed critical problems can exist, no more than X major non-closed problems can exist all of which have work-arounds and no more than X minor non-closed problems can exist
- The arrival rate of new problem reports must be decreasing towards zero with no new critical problems reported in the last X number of days

For understand and predict type metrics, it is typically the “level of confidence in a given result” portion of the ISO 15939 definition that applies. How confident are we that the understanding we have gained from the metric reflects reality? How confident are we that our prediction reflects what the actual values will be in the future. One method we can use is to calculate statistical confidence intervals. However, we can also obtain a more subjective confidence level by considering factors including:

- The completeness of the data used. For example, if we are trying to understand the amount of effort we expend on software development does our time reporting system include all the time spent including unpaid overtime?
- Is the data used subjective or objective? Has human or other types of bias crept into the data?
- What is the integrity and accuracy of the data? For example, if we again consider time card data are people completing their time cards as they complete tasks or are they waiting until the end of the week and then estimating how much time they spend on various projects.
- How stable is the product process or service being measured? For example, if we are using a new process or a new programming language, we may not be very confident in a prediction we make based on our historic data or we may not have any relevant historic data upon which to base our predictions.
- What is the variation within the data set? For example, if we look at the distribution in a data set that has very little variance, we can be fairly confident that the mean (average) of that data set accurately represents that data sent. However, if the data set has a large amount of variance or a bi-modal distribution, our confidence level that the mean accurately represents the data set is decreased.

## Step 9 – Define Reporting Mechanisms

The ninth step is to decide how to report the metric. This includes defining the report format, data extraction and reporting cycle, reporting mechanisms, distribution, and availability.

The report format defines what the report looks like. Is the metric included in a table with other metrics values for the period? Is it added as the latest value in a trend chart that tracks values for the metric over multiple periods? Should that trend chart be a bar, line, or area graph? Is it better to compare values using stacked bars or a pie chart? Do the tables and graphs stand alone, or is there detailed analysis text included with the report? Are goals or control values included in the report?

The data extraction cycle defines how often the data snap-shot(s) are required for the metric and when they will be available for use for calculating the metric. The reporting cycle defines how often the report is generated and when it is due for distribution. For example, root cause analysis metrics may be triggered by some event, like the completion of a phase in the software development process. Other metrics like the defect arrival rate may be extracted and reported on a daily basis during system test and extracted on a monthly basis and reported quarterly after the product is released to the field.

The reporting mechanism outlines the way that the metric is delivered (i.e., hard copy report, email, on-line electronic data).

Defining the distribution involves determining who receives regular copies of the report or access to the metric. The availability of the metrics defines any restrictions on access to the metric (i.e., need to know, internal use only) and the approval mechanism for additions and deletions to access or standard distribution.

## Step 10 – Determine Additional Qualifiers

The tenth step in designing a metric is determining the additional metric qualifiers. A good metric is a generic metric. That means the metric is valid for an entire hierarchy of additional qualifiers. For example, we can talk about the duration of unplanned outages for an entire product line, an individual product, or a specific release of that product. We could look at outages by customer or business segment. Alternatively, we could look at them by type or cause.

The additional qualifiers provide the demographic information needed for these various views of the metric. The main reason additional qualifiers need to be defined as part of the metrics design is that they determine the second level of data collection requirements. Not only is the metric primitive data required, but data also has to exist to allow the distinction between these additional qualifiers.

## Step 11 – Collect Data

The question of "what data to collect?" was actually answered in steps 7 and 10 above. The answer is to collect all of the data required to provide the metrics primitives and the additional qualifiers.

In most cases, the "owner" of the data is the best answer to the question of "who should collect the data?" The data "owner" is the person with direct access to the source of the data and in many cases is actually responsible for generating the data. Table 1 illustrates the owners of various kinds of data.

Benefits of having the data owner collect the data include:

- Data is collected as it is being generated, which increases accuracy and completeness
- Data owners are more likely to be able to detect anomalies in the data as it is being collected, which increases accuracy
- Human error caused by duplicate recording (once by data recorder and again by data entry clerk) is eliminated, which increases accuracy

Once the people who gather the data are identified, they must agree to do the work. They must be convinced of the importance and usefulness of collecting the data. Management has to support the program by giving these people the time and resources required to perform data collection activities. A support staff must also be available to answer questions and to deal with data and data collection problems and issues.

| Owner | Examples of Data Owned |
|---|---|
| Management | • Schedules<br>• Budgets |
| Engineers | • Time spent per task<br>• Inspection data including defects<br>• Root cause of defects |
| Testers | • Test cases planned/executed/passed<br>• Problem reports from testing<br>• Test coverage |
| Configuration Management Specialists | • Lines of code<br>• Modules changed |
| Users | • Problem reports from operations<br>• Operational hours |

Table 1: Examples of Data Ownership

A training program should be provided to help insure that the people collecting the data understand what to do and when to do it. As part of the preparation for the training program, suitable procedures must be established and documented. For simple collection mechanisms, these courses can be as short as one hour. I have found that hands-on, interactive training, where the group works actual data collection examples, provides the best results.

Without this training, hours of support staff time can be wasted answering the same questions repeatedly. An additional benefit of training is that it promotes a common understanding about when and how to collect the data. This reduces the risk of collecting invalid and inconsistent data.

If the right data is not collected accurately, then the objectives of the measurement program cannot be accomplished. Data analysis is pointless without good data. Therefore, establishing a good data collection plan is the cornerstone of any successful metrics program. Data collection must be:

- Objective: The same person will collect the data the same way each time.

- Unambiguous: Two different people, collecting the same measure for the same item will collect the same data.
- Convenient: Data collection must be simple enough not to disrupt the working patterns of the individual collecting the data. Therefore, data collection must become part of the process and not an extra step performed outside of the workflow.
- Accessible: In order for data to be useful and used, easy access to the data is required. This means that even if the data is collected manually on forms, it must ultimately be included in a metrics database.

There is widespread agreement that as much of the data gathering process as possible should be automated. At a minimum, standardized forms should be used for data collection, but at some point the data from these forms must be entered into a metrics database if it is to have any long-term usefulness. I have found that information that stays on forms quickly becomes buried in file drawers never to see the light of day again.

Dumping raw data and hand tallying or calculating metrics is another way to introduce human error into the metrics values. Even if the data is recorded in a simple spreadsheet, automatic sorting, data extraction, and calculation are available and should be used. Using a spreadsheet or database also increases the speed of producing the metrics over hand tallies.

Automating metrics reporting and delivery eliminates hours spent standing in front of copy machines. It also increases usability because the metrics are available on the computer instead of buried in a pile of papers on the desk. Remember, metrics are expensive. Automation can reduce the expense, while making the metrics available in a timelier manner.

## Step 12 – The People Side of the Metrics Equation

No discussion on selecting, designing and implementing software metrics would be complete without a look at how measurements affect people and people affect measurements. Whether a metric is ultimately useful to an organization depends upon the attitudes of the people involved in collecting the data, calculating, reporting, and using the metric. The simple act of measuring will affect the behavior of the individuals being measured. When something is being measured, it is automatically assumed to have importance. People want to look good; therefore, they want the measures to look good. When creating a metric, always decide what behaviors you want to encourage. Then take a long look at what other behaviors might result from the use or misuse of the metric. The best way I have found to avoid human factors problems in working with metrics is to follow some basic rules:

**Don't measure individuals:** The state-of-the-art in software metrics is just not up to this yet. Individual productivity measures are the classic example of this mistake. Remember that we often give our best people the hardest work and then expect them to mentor others in the group. If we measure productivity in lines of code per hour, these people may concentrate on their own work to the detriment of the team and the project. Even worse, they may come up with unique ways of programming the same function in many extra lines of code. Focus on processes and products, not people.

**Never use metrics as a "stick":** The first time we use a metric against an individual or a group is the last time we get valid data.

**Don't ignore the data:** A sure way to kill a metrics program is to ignore the data when making decisions. "Support your people when their reports are backed by data useful to the organization" [Grady-92]. If the goals we establish and communicate don't agree with our actions, then the people in our organization will perform based on our behavior, not our goals.

**Never use only one metric:** Software is complex and multifaceted. A metrics program must reflect that complexity. A balance must be maintained between cost, quality and schedule attributes to meet all of the customer's needs. Focusing on any one single metric can cause the attribute being measured to improve at the expense of other attributes.

**Select metrics based on goals:** Metrics act as a big spotlight focusing attention on the area being measured. By aligning our metrics with our goals, we are focusing people's attention on the things that are important to us.

**Provide feedback:** Providing regular feedback to the team about the data they help collect has several benefits:

- It helps maintain focus on the need to collect the data. When the team sees the data actually being used, they are more likely to consider data collection important.
- If team members are kept informed about the specifics of how the data is used, they are less likely to become suspicious about its use.
- By involving team members in data analysis and process improvement efforts, we benefit from their unique knowledge and experience.
- Feedback on data collection problems and data integrity issues helps educate team members responsible for data collection. The benefit can be more accurate, consistent, and timely data.

**Obtain "buy-in":** To have 'buy-in" to both the goals and the metrics in a measurement program, team members need to have a feeling of ownership. Participating in the definition of the metrics will enhance this feeling of ownership. In addition, the people who work with a process on a daily basis will have intimate knowledge of that process. This gives them a valuable perspective on how the process can best be measured to ensure accuracy and validity, and how to best interpret the measured result to maximize usefulness.

## Conclusion

A metrics program that is based on the goals of an organization will help communicate, measure progress towards, and eventually attain those goals. People will work to accomplish what they believe to be important. Well-designed metrics with documented objectives can help an organization obtain the information it needs to continue to improve its software products, processes, and services while maintaining a focus on what is important. A practical, systematic, start-to-finish method of selecting, designing, and implementing software metrics is a valuable aid.

## References

[Basili-88] V. R. Basili, H. D. Rombach, 1988, The TAME Project: Towards Improvement-Oriented Software Environments. In *IEEE Transactions in Software Engineering* 14(6) (November).

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

[Goodman-93] Paul Goodman, 1993, *Practical Implementation of Software Metrics,* McGraw Hill, London.

[Grady-92] Robert B. Grady, 1992, *Practical Software Metrics for Project Management and Process Improvement,* Prentice-Hall, Englewood Cliffs.

[IEEE-610] *IEEE Standard Glossary of Software Engineering Terminology,* ANSI/IEEE Std 610-1990, The Institute of Electrical and Electronics Engineers, New York, NY, 1990.

[ISO/IEC-15939] ISO/IEC 15939:2002 (E), International Standard, Software Engineering – Software Measurement Process.

[Jones-86] Capers Jones, 1986, *Programming Productivity,* McGraw Hill, New York.

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

[Schulmeyer-98] G. Gordon Schulmeyer, James I. McManus, *Handbook of Software Quality Assurance, 3rd Edition*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.

## About the Author

Linda Westfall is the President of The Westfall Team which provides Software Metrics and Software Quality Engineering training and consulting services.   Prior to starting her own business, Linda was the Senior Manager of the Quality Metrics and Analysis at DSC Communications where her team designed and implemented a corporate wide metric program.  Linda has twenty years of experience in real time software engineering, quality and metrics.  She has worked as a Software Engineer, Systems Analyst, Software Process Engineer and Manager of Production Software.

Very active professionally, Linda Westfall is Chair of the American Society for Quality (ASQ) Software Division.  She has also served as the Software Division's Program Chair and Certification Chair and on the ASQ National Certification Board. Linda has her Professional Engineer (PE) license in Software Engineering from the State of Texas and is an ASQ Certified Software Quality Engineer (CSQE) and Certified Quality Auditor (CQA).