12 Steps to Useful Software Metrics
by Linda Westfall

email: lwestfall@westfallteam.com
phone (work & cell): 972-867-1172
website: Live Courses (softwareexcellenceacademy.com)

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.

Introduction to the Twelve Steps

There are multitudes of possible software metrics based on all of the possible software entities and all the possible attributes of each of those entities. How do we pick the metrics that are right for our organizations? The first four steps defined in this article 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, measurement function, measurement method, decision criteria, reporting mechanisms, and additional qualifiers. The last two steps deal with implementation issues, including data collection and managing the impact of human factors on metrics.

When I 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 method usually involved collecting and reporting on the current "hot" metrics measurement 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 problems with both of these methods. The problem with the first method is that if we consider all of the possible software entities and their possible attributes that can be measured, there are 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 products, processes, and services. Organizational, project, and task goals are determined in advance, and 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] b

Measuring software is a powerful way to track progress towards our 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 strategy changes." [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 need the information supplied by the metric.

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

Functional Management: These people are interested in applying greater control to the software development process, reducing risk, and maximizing return on investment.

Project Management: These people are 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 who do software development are 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 verification and validation activities are interested in finding as many new defects as possible in the time allocated to testing and 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) are interested in quantitative information upon which they can base their decisions, finding, and recommendations.

Customers/Users: The people who purchase and use the software are interested in the on-time delivery of high-quality software products and reducing the overall ownership cost.

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 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 significantly 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. The Goal/Question/Metric paradigm concept is to identify our goals, determine the questions that need to be answered to determine if we are meeting or moving towards those goals, and then select metrics that provide information to help answer those questions. As illustrated in Figure 1, a goal may spawn more than one questions, a question can relate to more than one goal, and a metric can provide information to answer more than one question.

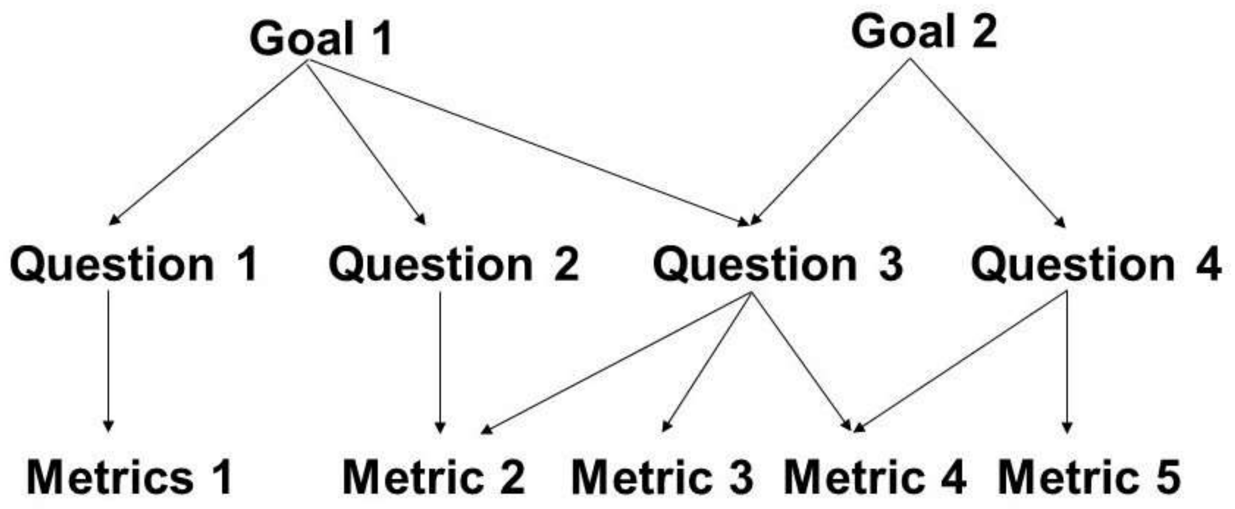


Figure 3: Goal/Question/Metric Paradigm

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 targets. We typically look at goals that emphasize project management and control issues or project level requirements and objectives at the project level. 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.

We consider goals that emphasize task success factors at the specific task level. These are often expressed in terms of the entry and exit criteria for the task.

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 to ensure that each goal is 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?

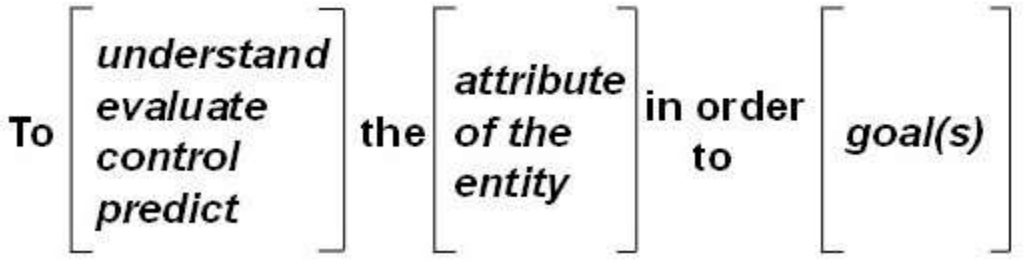
Step 4 – Select Metrics

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 moving toward our goals or meeting our goals.

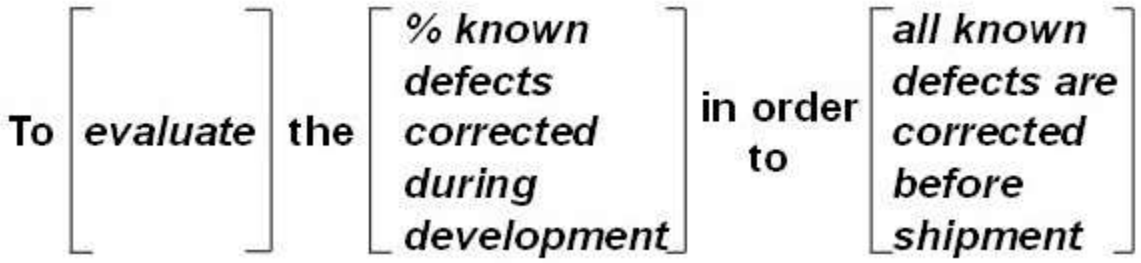
When selecting metrics, we must be practical, realistic, and pragmatic. Avoid the "ivory-tower" perspective completely removed from the existing software-engineering environment. Start with what is possible within the current process. Once we have a few successes, our customers will be open to more radical 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 c***ontrol*** resources and processes used to produce our software. Metrics can be used to ***predict*** attributes of software entities in the future. [based on Humphrey-89]. A comprehensive metric program includes metrics that perform all of these functions.



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


To
evaluate
the
% known defects corrected during development
in order to
all known defects are corrected before shipment

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 measurement goal. This statement leads to the following metrics requirements statement template. 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 may interpret these words in their context with meanings that 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. Unfortunately, they may also refer to different entities. One external customer may use a *customer call report* to refer to their complaint and *problem report* as the description of the defect in the software. In contrast, another customer may use a *problem report* for the initial complaint. Differing interpretations of terminology may be one of the most significant barriers to understanding.

Unfortunately, there is little standardization in the software 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 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 your organizational objectives or use them as a basis for creating your 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 will calculate the metric. Some metrics called base measures, direct 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. More complex metrics, called derived measures, are calculated 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 which is calculated as the number of lines of code reviewed divided by the number of preparation hours.

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

- Does the function 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 measurement function: use an existing function or create a new one. In many cases, there is no need to "re-invent the wheel." Many software metrics functions 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 functions with little or no adaptation to match our own environment.

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

To illustrate the selection of a function, let's consider a metric for the duration of unplanned system outages. If we evaluate a software system installed at a single site, a simple function 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 function such as minutes of outage per 100 operation months. If we wanted to focus on our customers' impact, we might select minutes of outage per site per year.

Step 7 – Establish a Measurement Method

The seventh step in designing a metric is breaking the function down into its lowest level base measures (metric primitives) and defining the measurement method used to assign a value to each base measure. The measurement method defines the mapping system used to assign numbers or symbols to the attributes.

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, simple counts of items with certain characteristics. For example, if the metric is the problem report arrival rate per month, we could count all of the problem reports in the database that had an open date 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 but < 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, a specific measurement method must be defined. This method should also accompany the metric in all reports and analyses 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 to be collected to implement the metric. To illustrate this, let's use the function of minutes of system outage per site per year. One of the base measures for this model is the number of sites. At first, counting this base measure 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 during 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 software releases, 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 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, the decision criteria provide the guidance that will help the metrics customers interpret the measurement results.

Going back to Humphrey's four reasons to measure (i.e., control, evaluate, understand and predict). [Humphry-89] Control metrics are used to monitor our software projects, processes, products, and services and identify areas where corrective or management action may be required. Metrics used for control 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-17]

- Thresholds: a value that, when crossed, indicates that an out-of-control condition may exist. For example, if we planned our project assuming a 15% staff turnover, we could track our actual staff turnover against a 15% threshold.
- Variances: the difference between two values that, when it exceeds a specific value, indicates an out-of-control condition may exist. For example, we could track the variance between our budget and actual expenditures.
- Control limits: specific upper and lower boundary values used to indicate that an out-of-control condition might exist. For example, the upper and lower control limits in a statistical process control chart.

Evaluate type metrics are used to examine and analyze the measurement information as part of our decision-making processes. For metrics used to evaluate, decision criteria help define "what good." For example, suppose we want to make a 15% return on investment (ROI) for our project. In that case, the decision criteria could indicate that the benefit to cost ratio must be at least 1.15, or we shouldn't initiate the project. We might also establish decision criteria for our exit from software qualification testing. If decision criteria are not met, then testing should continue. Examples of these software qualification test decision criteria could include:

- At least X% of all planned test cases must be executed, and at least Y% of those must have passed.
- Zero non-closed critical problems can exist, no more than X major non-closed problems can exist, all of which have workarounds, and no more than X minor non-closed problems can exist. (I use the term non-closed instead of open because of the possible

ambiguity of the word open. For example, developers may consider a problem no longer "open" when they have corrected it, but it may not htegrated into the product and retested).

- 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 metrics used to understand and predict, 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 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 step includes defining the report format, data extraction and reporting cycle, reporting mechanisms, distribution, and availability.

The report format defines what the information product looks like. Is the information product a table with multiple measurement values? Is the new measure 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, control values, or other decision criteria 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 measurement. The reporting cycle defines how often the report is generated and when it is due for distribution. For example, root cause analysis measurements may be triggered by some event, like the completion of a phase/iteration in the software development process. Other metrics like the defect arrival rate may be extracted and reported daily during testing and extracted monthly 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

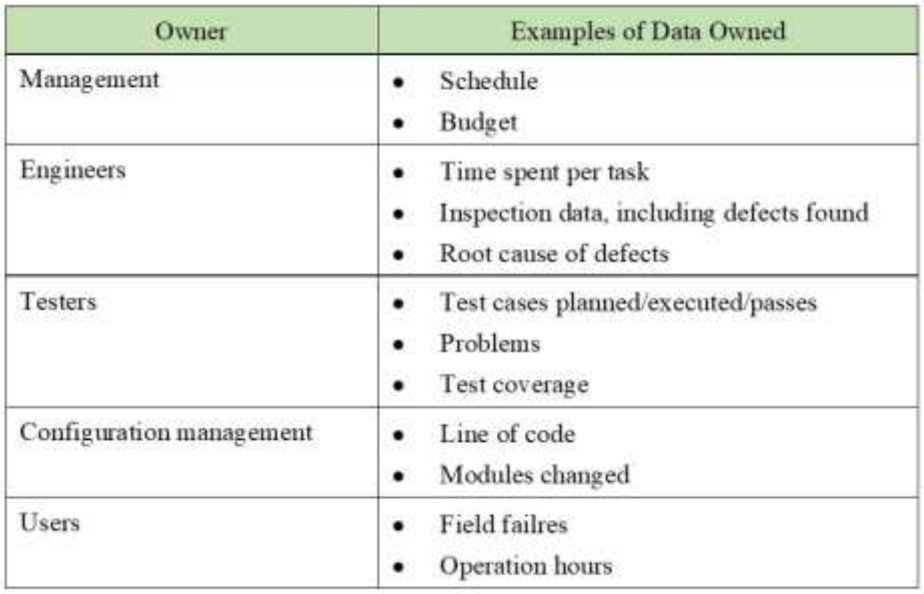
The tenth step in designing a metric is determining the additional metric qualifiers. A good metric is a generic metric. That means the metric is valid for an entire hierarchy of additional qualifiers. For example, we can talk about the duration of unplanned outages for an entire product line, an individual 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 "what data to collect?" was answered in steps 7 and 10 of the 12 steps. The answer is to collect all 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 "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 responsible for generating the data. Table 1 illustrates the owners of various kinds of data.



Owner	Examples of Data Owned
Management	• Schedule • Budget
Engineers	• Time spent per task • Inspection data, including defects found • Root cause of defects
Testers	• Test cases planned/executed/passes • Problems • Test coverage
Configuration management	• Line of code • Modules changed
Users	• Field failres • Operation hours

Table 1: Examples of Data Owners

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 data 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 needs to support the program by giving these people the time and resources required to perform data collection activities. Support staff must also be available to answer questions and deal with data and data collection problems and issues.

A training program should be provided to help ensure 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. These courses can be as short as one hour for simple collection mechanisms. I have found that hands-on, interactive training provides the best results, where the group works actual data collection examples.

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 shared understanding of when and how to collect the data. This understanding reduces the risk of collecting invalid and inconsistent data.

If the correct 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 sound 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 the workflow.
- Accessible: Easy access to the data is required for data to be useful and used. Therefore, 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 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 expenses 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 looking at how measurements affect people and people affect measurements. Whether a metric is ultimately valuable for an organization depends upon the people's attitudes 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 negative behaviors might result from using or misusing the metric. For example, management uses the metrics to prod or punish individuals or teams. Another example would be people trying to manipulate the measurement results. One of my favorite measurement quotes is, "Don't underestimate the intelligence of your engineers. For any one metric you come up with, they will find at least two ways to beat it" [Unknown]

This concern does not mean that we give up on a metrics program because people might misuse the metrics. It means we have to be conscious of potential problems and proactively plan how to deal with them. As the last step in the "12 Steps to Useful Software Metrics", we determine the positive behaviors that we want as a result of implementing the metric and the possible negative behaviors that might result from its implementation. We can then create an action plan on how we will emphasize the positives and eliminate or at least minimize the negatives.

The best way I have found to deal with human factors issues in working with metrics is to follow some basic rules:

Don't measure individuals: The state-of-the-art in software metrics is not up to this yet. Individual productivity measures are the classic example of this mistake. Remember that we often give our best people the most challenging 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 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, product (including quality), and schedule attributes to meet the customer's needs. Focusing on any single metric can cause the attribute to be measured to improve at the expense of other attributes, resulting in an anorexic software development process.

Select metrics based on goals: Metrics act as a big spotlight focusing attention on the measured area. By aligning our metrics with our goals, we focus 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 being used, they are more likely to consider data collection important.
- If team members are kept informed about precisely how the data is used, they are less likely to become suspicious or fearful that it is being used against them.
- 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 feel ownership. Participating in the definition of the metrics will enhance this feeling of ownership. In addition, the people who work with a process daily will have intimate knowledge of that process. This participation gives them a valuable perspective on how the process can best be measured to ensure accuracy and validity and best interpret the measured result to maximize usefulness.

Conclusion

A metrics program based on an organization's goals 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 essential. A practical, systematic, start-to-finish method of selecting, designing, and implementing software metrics is valuable in ensuring the consistent collection, reporting, and use of the measurement information.

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.

[IFPUG-90] International Function Point User's Group; www.ifpug.org.

[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.

[McCabe-82] Thomas J. McCabe, Structured Testing: *A Software Testing Methodology Using the Cyclomatic Complexity Metric*, NBS Special Publication, National Bureau of Standards, 1982.

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

[Westfall-03] Linda Westfall, *Are We Doing Well or Are We Doing Poorly*, 2003 Applications in Software Measurement (ASM) Conference, available for download: http://www.westfallteam.com/software_metrics,_measurement_&_analytical_methods.htm