

SEVEN STEPS TO DESIGNING A SOFTWARE METRIC

Linda L. Westfall
 Principle
 Software Measurement Services
 Plano, TX 75075

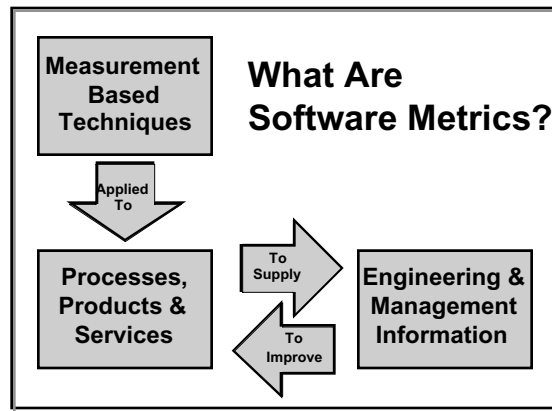
SUMMARY

This paper outlines a practical, step by step process for designing effective software metrics.

KEY WORDS: software, metrics, measurement, data collection

INTRODUCTION

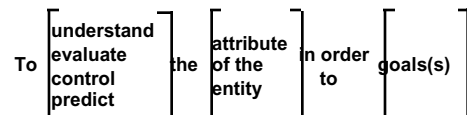
Software metrics are an integral part of the state-of-the-practice in software engineering. Goodman (1993, 6) defines software metrics as: "The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products". Figure 1, illustrates an expansion of this definition to emphasize the inclusion of software related services such as installation and responding to customer issues. It also emphasizes that software metrics provide the information needed by engineers for technical decisions.



If the metric is to provide useful information, everyone involved in designing, implementing, collecting data for and utilizing a software metrics must understand its definition and purpose. This paper outlines seven steps to documenting the design of software metrics in order to insure this understanding.

Step 1 - Objective Statement: Software metrics can perform 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 1989)

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



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

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

Having a clearly defined and documented objective 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 base requirements statement for the design of the metric.

Step 2- Clear Definitions: The second step in designing a metric is to agree to a standard definition for the entities and their attributes being measured.

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 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 suggestions from the industry as a foundation to get you started. Pick and choose the definitions that match with your organizational objectives or use them as a basis for creating your own definition.

Step 3 - Define the Model: The next step is to derive a model for the metric. In simple terms, the model defines how we are going to calculate the metric. Some metrics, called metric primitives, are measured directly and their model typically consists of a single variable. Other more complex metrics are modeled using mathematical combinations of metrics primitives or other complex metrics.

Most modeling includes an element of simplification. When we create a software measurement model 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 its useless. Being pragmatic means not trying to create the perfect model. Pick 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. In many cases there is no need to "re-invent the wheel". There are many software metrics models that have been used successfully by other organizations. These can be found 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.

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 like minutes of outage per 100 operation months. Or if we wanted to focus in on the impact to our customers, we might select minutes of outage per site per year.

Step 4 - Establish Counting Criteria: The next step in designing a metric is to break the model down into its lowest level metric primitives and define the counting criteria used to measure each primitive. This defines the mapping system for the measurement of each metric primitive.

The importance of the need for defining counting criteria can be illustrated by considering the lines of code metric. Lines of code is one of the most used and most often mis-used of all of the software metric. The problems and variations and anomalies of using lines of code are well documented [Jones 1986]. However, there is still no industry accepted standard for counting lines of code. Therefore, if you are going to use a lines of code metric, it is critical that specific counting criteria be defined.

The metric primitives and their counting criteria define the first level of data needs to be collected in order to implement the metric. To illustrate this, let's use the model minutes of 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. But 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. And 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 5 - Decide What's "Good": The fourth step in designing a metric is defining what's "Good". Once you have decided what to measure and how to measure it, you have to decide what to do with the results. Is 10 too few or 100 too many? Should the trend be up or down? What do the metrics say about whether or not the product is ready to ship?

One of the first places to start looking for "what's good" is the customer. Many times, user requirements dictate certain values for some metrics. There may be product reliability levels that must be met. The customer may have a defined expectation of defect reduction from release to release or a required repair time for discovered defects. Another source of information is the metrics literature. Research and studies have helped establish industry accepted norms for standard measures. Modules should have a McCabe's Cyclomatic Complexity of ≤ 10 . Code can be effectively inspected at a rate of approximately 125 lines per hour.

The best source of information on "what's good" is your own data. Processes vary from group to group. Many metrics do not have industry accepted counting criteria (i.e., lines of code), and most documentation does not include how the metrics were calculated. Therefore, comparing your values to published standards may result in erroneous interpretations. Whenever possible, use your organization's own data to determine baselines. If historic data is not available, wait until enough data is collected to reasonably establish current values.

Once you have decided "what's good", you can determine whether or not action is needed. If you are "meeting or exceeding" desired values, no corrective action is necessary. Management can either turn its attention elsewhere or establish some maintenance level actions to insure that the value stays at acceptable levels.

However, if improvements are needed, goals can be established to help drive and monitor improvement activities. When setting metrics goals remember four things:

- The goal must be reasonable. It's all right to establish a stretch goal but if the goal is unrealistic, everyone will just ignore it.
- The goal should be associated with a time frame. Nothing happens overnight. To say a 50% backlog reduction without a "by when" is meaningless.
- The goal should be based on supporting actions. It may be reasonable to set a goal of 50% backlog reduction for defects if a special team is created to concentrate on fixing problems. If everything is "business as usual", do not expect the setting of the goal to have any affect.
- The goal should be broken down into small incremental pieces. If the "by when" is a long way off, it is only human nature to procrastinate. If there is a year to accomplish the improvement, it will be easy to put it on the back burner in preference to more pressing concerns. If we divide the same goal into 12 smaller end of month goals, actions are more likely to be taken now.

Step 6 - Metrics Reporting: The next step is to decide how to report the metric. This includes defining the report format, data extraction and reporting cycle, reporting mechanisms and distribution and availability.

Report format is designing what does the report look 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 those data items will be available snap-shots(s) available for use for calculating the metric. The reporting cycle defines how often the report generated and when is it 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 metric delivery mechanism (i.e., hard copy report, 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 this access and to the standard distribution.

Step 7 - Additional Qualifiers: The final step in designing a metric is determining the additional metric qualifiers. A good metric is a generic metric. That means that the metric is valid for an entire hierarchy of additional extraction 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. Or 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 that the 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 has to exist to allow the distinction between these additional qualifiers.

Human Factors: No discussion on selecting and designing software metrics would be complete without a look at how measurements affect people and people affect measures. Whether a

metric is ultimately useful to an organization depends upon the attitudes of the people involved. The people involved in collecting the data, in calculating and reporting the metrics and in using the metric. The simple act of measuring will effect 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 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. Or 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 metric program is to ignore the data when making decisions. "Support your people when their reports are backed by data useful to the organization" (Grady 1992, 114). If the goals we establish and communicate don't agree with our actions, then the people in our organization will perform based in 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 customers 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 objectives: To have a metrics program that meets our information needs, we have to select the metrics that provide information to answer our questions.

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 insure 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 the organizations will help communicate those goals. People will work to accomplish what they believe to be important. Well designed metrics with documented objectives can help our organization obtain the information it needs to continue

to improve its software products, processes and services while maintaining a focus on what is important to that organization.

REFERENCE LIST

Basili, V. R., Rombach H. D. 1988. The TAME Project: Towards Improvement-Oriented Software Environments. In *IEEE Transactions in Software Engineering* 14(6) (November), 758-773.

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

Goodman, Paul. 1993. *Practical Implementation of Software Metrics*. London: McGraw Hill.

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

Jones, Capers. 1986. *Programming Productivity*. New York: McGraw Hill.

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