Combat Resistance to Software Measurement by Targeting Management Expectations

Carol A. Dekkers
Quality Plus Technologies, Inc.

The software industry has been slow to embrace measurement practices even when software managers recognize the benefits it can deliver. This article seeks to overcome management resistance to software measurement by addressing management expectations. It includes a discussion of the human and technical factors involved in software measurement success.

When it comes to measurement, the software industry responds inconsistently. Even though other industries have long depended on measurable outcomes to gauge their profitability and control their processes' progress, the information technology industry has been slow to embrace software measurement. Even when software managers recognize that software measurement can deliver benefits and is a critical component in achieving a Software Engineering Institute Capability Maturity Model for Software (SW-CMM) Level 3 or higher, their expectations are often unrealistic.

While advancements have been made to implement software measurement, specifically to the Department of Defense (DoD) in such initiatives as the Practical Software Measurement Initiative and others, for the most part software measurement is not well understood or used in the software industry. Traditionally, when the software industry has produced metrics, they are typically isolated operational measures (such as transactions per hour or million-instructions per second) or physical measures (such as source lines of code) that tell little about the effectiveness or efficiency of the development process.

Additionally, many software managers seek a silver-bullet metric, which not only answers development questions but does so with several-decimal-point accuracy. Because there is no silver bullet, measurement falls short of these expectations and metrics programs are abandoned before they deliver a return on investment. Such outcomes do not need to happen, in fact, software measurement can deliver value even when the chosen measures are subjective (as with customer satisfaction) or when the measures are imperfect (as with defect tracking).

This article seeks to overcome management resistance to software measurement by addressing management expectations for silver-bullet metrics. Realistic management expectations for measurement provide a chance for measurement to survive. It is a critical prerequisite for DoD agencies contemplating process improvement based on software measurement.

Not an Exact Science

For engineers, computer scientists, and other information technology professionals, it is natural to expect that measurement can be made into an exact science (recall college labs where data outliers on research graphs were too difficult to explain and therefore were erased). In the real world of information technology, however, measurement does not always translate into predictable outcomes, and not everything that can be measured necessarily should be.

Measurement consists of taking a series of observations about a process or product and analyzing the data to indicate where positive changes might be made. It is important to realize that just because something can be measured to the *nth* degree of accuracy does not make it valuable to measure – there needs to be a purpose and a method behind the measure before it will be useful.

> *"... there needs to be a purpose and a method behind the measure before it will be useful."*

The first step in creating a successful measurement program is to realign your and your company's expectations about software measurement. The following sections describe how to do that.

Goal-Question-Metric Approach

Follow the Goal-Question-Metric (GQM) approach to software measurement introduced by Victor Basili of the University of Maryland[1]. This approach forces companies to clearly identify their strategic goals and to pose questions to track whether or not the goals are being met. Only then are the metrics needed to answer the questions identified, and data collection mechanisms put into place. The resulting metrics necessarily depend on the specific goals and questions of the organization. Within the SW-CMM are a number of Level 2 and Level 3 key process areas that can form the basis of an organization's goals/questions/metrics.

The importance of planning when implementing measurement is an area that is often glossed over in an organization's rush to quickly establish a solid metrics program. As such, it is not uncommon for senior management to initiate a software metrics program by collecting metrics without first having identified the goals or the questions into which the metrics should fit. Consequently, after six months of data collection, the lack of planning often becomes clear as management and their developers try to fit the metrics into a cohesive program to support their goal/question/decision-making needs. Without planning, the collected metrics often do not fit together properly, nor do they answer the questions that management needs answered to gauge whether their goals are being met.

Planning for measurement (by identifying your critical goals and questions that measurement must support) is as important a prerequisite as are requirements in software development. To be successful, measurement implementation should follow a project plan and consist of allocated and scheduled resources to perform the measurement requirements (GQM). An analysis of how measurement will be done should address the six W's of data collection and measurement:

- What processes will be impacted?
- What measures are needed?
- Who will participate in metrics design, collection, data analysis, and reporting?
- When will metrics be collected, including collection frequency (how often), life-cycle phase, data entry, etc?
- Where will metrics be collected: centralized/decentralized, from all projects/some projects, etc.?

- Why are metrics being gathered? (The purpose of each metric can change what is reported.)

Metrics without goals and pertinent questions are meaningless. For example, if a manager asks me for my project hours, and I think the purpose is to determine my paycheck, the answer will be 40 hours per week. If the purpose is to analyze how much time is actually being spent on the project by phase so that the process can be improved, my answer will be the accurate 50 hours per week.

While it may be tempting to rush directly to the design and selection of metrics, *do not* skip proper planning! Instead, take the time and energy to develop and produce a measurement project plan. This project plan is critical to aligning management's expectations because it will identify the resources, the time frame, and the coordination needed to implement measurement. In the same way that skipping software requirements leads to products that do not meet customer needs, skipping measurement program requirements – GQM – will lead to a measurement program that does not meet its customer needs.

The aforementioned book[1] outlines more details about the steps to take in planning a GQM-based measurement program, including checklists, tasks, recommended time frames, and resource levels. While it is not the only model for implementing software measurement, (others include the Balanced Scorecard), GQM is a rational approach that aligns metrics to the business goals, which in turn, will lead to higher success rates for measurement programs.

No Silver Bullet

Communicate early and often that there is no silver-bullet software metric, just as there is no silver-bullet accounting metric. Defects, functional size, project duration, and work effort all measure a different aspect of software development; they are not interchangeable. No single measure or single combination metric will satisfy all goals or answer all measurement questions – you must choose the metric suitable for each specific question. Once the specific, measurable GQMs have been identified, select the most appropriate metric. In the same way that a toolbox contains many tools, each specifically designed to serve a particular use, a measurement toolbox should contain specific measures selected to suit your specific needs.

For example, if the goals were to increase user satisfaction and software product quality, the questions would include the following: "What was the level of customer satisfaction with the product before implementing change?" "What is the new level of customer satisfaction?" "How has product quality improved (percentage increase in product quality levels)?" The contributing metrics would then consist of the following: customer satisfaction rating (using a numerically scored customer satisfaction survey) and defect density (measured using defects per function point or other software sizing measure).

There is no Swiss army knife of metrics – you need to select the measure(s) that best fits the purpose, be it defects, function points, number of objects, lines of code, customer satisfaction, work effort, etc. – each is intended to measure a different aspect of software development.

"Metrics without goals and pertinent questions are meaningless."

Learn About Metrics

Learn about the available metrics and what they mean before implementing them in an organization. For example, work effort is a function of many variables, including software size, implementation technology, development tools, skills, hardware platforms, degree of reuse, tasks to be done, and many others. As such, no single variable can accurately predict work effort; yet, there is often an expectation that a single variable (for example, degree of reuse) can accurately predict work effort.

If one of your goals is to increase estimating capability, it is also wise to research the available automated tools on the market and talk to actual users (not just tool vendors) about how their chosen tools work within their particular environment. Note that not all estimating tools address the same problem – some provide probable estimates of work effort and cost, while others provide hourly breakdowns of predicted work effort. Which one will best suit your needs? It depends on your goals and questions.

Use Metrics Properly

Plan a measurement program by using metrics and measures in their intended manner, and ensure that there is a common understanding of the chosen measures. For example, functional size reflects software size based on its functional user requirements, not its physical size. (Physical size of software is often expressed in lines of code.) Together with other variables, functional size can be used as a technology independent measure of software size in order to predict effort or cost in software estimation models.

However, functional size is not the right measure for predicting direct access storage device space requirements. These requirements depend on the physical space taken up by the software and the volume of data and are better measured with other units. For example, 50,000 COBOL lines of code take up more space than the equivalent lines of Java code. And the user requirement to store 50 million transactions takes up more physical space than it does to store a tenth of that.

There is an abundance of information on the Internet about various software metrics from organizations such as the Quality Assurance Institute <www.qaiusa.com>, the American Society for Quality <www.asq.org>, and the International Function Point Users Group <www.ifpug.org>.

Realize True Accuracy

Remember that the accuracy of a metric is a function of the least accurate component measure it involves. People often run into measurement difficulty when they assign several decimal places of accuracy to metrics that are derived from a series of relatively inaccurate or imprecise measures. For example, the function point (FP) count of a project is calculated by summing up discrete values of its component functions, none of which is more granular than three FP. To then calculate defect density and report it with multiple decimal places leads to the mistaken conclusion that the metric is exact.

The same situation arises when sophisticated estimating models produce effort estimates to 15-minute accuracy based on input variables that may have been guesses (e.g., project risk on a one-to-five scale). We all know intuitively that estimates based on a myriad of input variables cannot accurately predict schedules to the closest 15 minutes (let alone the number of hours). Yet I routinely encounter professionals who cite hour estimates with at least one decimal place. (Does this imply that your estimate is accurate to the closest tenth of an hour, or six minutes?)

Correlate With Common Sense

Use common sense and statistics to correlate col-

lected data, and question figures that seem out of line. Do not accept data purely at face value without verifying its consistency or accuracy. Many companies collect work effort data on completed projects, but the definition of project work effort can vary widely across different teams (e.g., overtime recorded/not recorded, resources included, work breakdown structure, commencement/finish points, etc).

Be careful not to compare data that appears comparable because of common units (e.g., hours) that is actually based on different measurement criteria. For example, two projects may report 100 development hours, but one project included overtime and user training hours while the other did not. Although the units are the same, the hours are not comparable. Project hours has no industry-wide definition and can vary widely. Ensure that your organization has established a consistent definition for collecting and reporting project hours for any projects included within the scope of data collection.

Additionally, it is important to apply common sense when establishing the frequency and granularity (unit size) for metrics data collection. For example, while it might be ideal from a theoretical point of view to collect work effort metrics to the closest 0.5 hour broken down by a work breakdown structure task level, it may require more administrative changes and double data entry effort, eliminating potential gain. If your current work effort reporting provides for the developers to enter their project hours into an automated system to the closest hour and broken down by phase on a weekly basis, it would likely prove counterproductive to ask them to re-enter hours a different way just to populate the metrics database. Work with and leverage your existing processes – your developers will appreciate it *and* will more readily buy in to participating in the metrics collection process.

The frequency and granularity of your metrics collection process will depend on your chosen metrics (in support of the goals and questions) and the scope of your measurement program. If your goal is to improve a particular process (e.g., Capability Maturity Model® key processes) for which there has never been any data collected, do not structure the data collection process to impede the overall development processes. Measurement should always be the means to an end – not an end in itself. In other words, measurement must support and provide the opportunity to improve a particular process, *not* to take the place of the development process itself. Measurement should not interfere with the business of developing software. If we focus on measurement to the detriment of developing software, our business will cease to be viable; it will no longer be a matter of measurement, it will become a matter of survival.

Conclusion

These are a few of the factors, both human and technical, that can lead to software measurement success. There is a great deal to be gained by tracking and controlling software development through measurement – if only management would realign their measurement expectations of what the particular measures can provide, rather than seeking a non-existent silver bullet that will solve all of their measurement needs.◆

Note

1. McGraw-Hill published a book featuring a foreword by Victor Basili: The Goal/Question/Metric Method by Rini van Solingen and Egon Berghout.

About the Author

Carol A. Dekkers is a leading software measurement authority and president of Quality Plus Technologies, Inc., which provides professional software measurement training and consulting services as well as International Function Point Users Group (IFPUG) certified function point training, mentoring, and consulting services. She is past president of the IFPUG and was named by the American Society for Quality as one of the 21 New Faces of Quality for the 21st Century. Additional measurement articles by Dekkers, including how to set up measurement programs, are available by e-mail or by accessing an article request form at <www.qualityplustech.com>.

Quality Plus Technologies, Inc.
8430 Egret Lane
Seminole, FL 33776
Phone: (727) 393-6048
Fax: (727) 393-8732
E-mail: dekkers@qualityplustech.com

COMING EVENTS

August 10-13
Third International Conference on Intelligent Systems Design and Applications
Tulsa, OK
http://isda03.softcomputing.net

August 18-21
2nd Annual C4ISR Summit
Danvers, MA
www.paulrevereafa.horizons.com

August 25-29
QAI's Annual eXtreme Conference
Las Vegas, NV
www.qaiusa.com

September 8-12
International Conference on Practical Software Testing Techniques
Minneapolis, MN
www.psqtconference.com

September 14-19
International Function Point Users Group Annual Conference
Scottsdale, AZ
www.ifpug.org/conferences

September 22-25
AUTOTESTCON 2003
Anaheim, CA
www.autotestcon.com

October 21-24
18th International Forum on COCOMO and Software Cost Modeling
Los Angeles, CA
http://cse.usc.edu

November 17-21
International Conference on Software Process Improvement
Washington, DC
www.software-process-institute.com

April 19-22, 2004
Software Technology Conference 2004

STC

Salt Lake City, UT
www.stc-online.org