BEST PRACTICES WHITE PAPER

Establishing a Measurement Program

Jenny Stuart, Vice President of Consulting, Construx Software

Version 1, September 2008

Contributors

Pam Perrott, Professional Software Engineer

Steve Tockey, Principal Consultant

When beginning a measurement program it can be confusing to determine what to measure, how to measure it, and where to start measuring. A number of common measures are used within the software development industry, but not all measures are useful and appropriate for all organizations. As a measurement program is established, the organization must determine its specific improvement goals. These goals help it determine the questions that need to asked to determine whether it is moving towards or away from its goals and the specific measures that need to collected to answer the questions.

Contents

Introduction

Using measures based on objective data gathering provides the foundation for quantitatively managing processes, as well as measuring the effectiveness of process improvement efforts. A measurement program facilitates

- Centralizing knowledge of software development measures used within an organization.
- Providing a clearinghouse for accessing measurement data on projects.
- Providing a focal point for standardizing measurements across the organization.
- Specialization of measurement skills, such as data collection techniques, use of measurement tools, and statistical analysis skills. Measurement experts can then assist others within the organization with the use of measures.

Ultimately, Construx believes the success of measures and the data upon which a program is based depends heavily on a grass-roots buy-in. Thus, it is important to identify measurements that are implementable, do not distort reality, and will not cause undesirable behavior as staff try to optimize their behavior based on the measures (which they will do).

Deciding What to Measure

The final adoption and subsequent evolution of any organization's measures should be driven by an iterative Goal, Question, Metric (GQM) approach, where measures are selected to answer specific questions required to identify whether goals are being met. As discussed by Vic Basili, the GQM approach includes developing a set of business and measurement goals, generating a set of questions that objectively and quantitatively define the goals, developing mechanisms to collect the data for the measures, and using the data on projects.

A measurement program should begin by defining the goals for improvement. Construx recommends these goals focus on overall organizational improvement objectives or on specific improvement activities that are occurring. Goals usually fall into one of these four general categories:

- Improve software quality
- Improve schedule accuracy
- Improve productivity
- Maximize customer satisfaction

Once the business goals have been established, the organization determines the questions that will quantify the business goals. Some examples of goals and questions are outlined in Table 1.

Table 1 *Example Goals and Questions*

Goal	Questions
Shorter schedules by reducing the duration of system testing phases	■ What is the current percentage of the schedule allocated to system testing? ■ Is the current percentage increasing or decreasing?
Improved development responsiveness as a result of quicker turnaround times	■ How much calendar time was spent on regression testing because of a change request? ■ Is this average turnaround time improving, i.e., getting shorter?
Improved user satisfaction with system quality and team responsiveness	■ How do users rate the development team's responsiveness, etc.? ■ If we take surveys each quarter, will they show user satisfaction increasing or decreasing?
Reduced system testing costs	■ How many hours were spent on system testing relative to overall project hours? ■ Is this ratio affected by the size of the project?

Example Measures

While each organization implementing a measurement program will identify the unique measures that best meet its needs, a typical approach is to include a combination of both quantitative (hard data) and qualitative (subjective data) information. This section outlines the more common measures that organizations implement.

A trap organizations often fall into with measurement programs is to try to measure too much too fast. It is better to start with a small number of measures that can be used to answer key GQM questions. We are not recommending that a company adopt all the measures described in this paper—they are starting points for selecting fewer measures to begin with. Rather, organizations should implement a measurement program incrementally. They should also analyze results regularly to refine goals, improve measures, and ensure effectiveness.

Quantitative Measures

Most organizations find that collecting data on size, effort expended, schedule, and quality is a good starting point for a measurement program. Measurements that most organizations find useful are calculated from these base measures and used for monitoring and measuring software process improvement. Each of these base measures is valuable for a different reason. Size is useful to normalize other measures, such as defects.

The absolute count of defects isn't as useful as measures such as defects/function point or defects/thousands lines of code (KLOC).

Having a measure of size also enables organizations to improve estimation efforts because estimation requires size to be known to estimate effort and schedule. Effort is the biggest determiner of cost in software projects. In fact, it often represents 100% of the cost, and thus size is an important factor for any organization seeking to qualitatively measure its productivity and effectiveness. Schedule shows how effective the organization is in delivering on time. Measuring defects in addition to the average cost to fix them reflects effort spent in rework, which is a major contributor to project cost and can affect the expected level of customer satisfaction with the product.

The common measures derived from size, effort, schedule, and quality that many organizations implement include the following:

- **Effort per function point/KLOC.** Measures productivity in either hours per function point or hours per thousand lines of code (KLOC) produced by a project. It is sometimes captured as lines of code per person per month or function points per person per month. This productivity metric is extremely useful in improving estimation and in measuring whether or not a process change improves productivity. It is important to note that many factors affect productivity, so only projects with similar technologies and of similar size should be compared when looking for productivity improvements.
- **Defects found in system test.** Measures quality by understanding the number of defects found in system test. This is measured as defects per KLOC or defects per function point. Beyond measurement in system test, highly effective processes also measure the defects found before system testing through reviews and unit testing. This supports measurement on the quality of software entering the system test and, with post-release defects, measures the percentage of defects that enter system test that are then found during the system test phase.
- **Post-release defects.** Measures quality by understanding the number of defects found after release. This is often measured as defects per KLOC or defects per function point.
- **Schedule/effort accuracy.** Measures predictability and estimation/process accuracy by understanding the difference between the estimated and actual schedule and the effort needed to complete the project. This often includes the number of times the project recalibrated the estimate. This factor can be measured as an absolute or as an average percent of schedule slippage and effort changes. Initially, this is a measure of how good the organization is at estimating; later, when estimation is more accurate and repeatable, this can be used to measure predictability.

- **Earned value.** Measures predictability by understanding the actual project work completed as compared to the plan. Earned value is measured as a ratio of the cost of the work actually done at a certain date to the cost of the work planned to be complete by that date. This measure requires a detailed work breakdown structure with fairly small tasks, from approximately ½ week to 2 weeks of effort. And each small task must have an estimated date it will be finished on and an estimate of effort. Then the tasks actually done and the actual effort to finish them are compared to the estimates.

- **Rework rate.** Measures productivity and process efficiency by understanding the total project effort spent redoing work later in the project, i.e. fixing requirements defects found during development, fixing coding defects found in testing, etc. As rework is reduced, total productivity should rise. This measure requires a time recording system that can be tailored so that rework, as distinguished from forward work, can be recorded, as well as a training program to instruct staff on what kinds of work constitute rework. Typical software development is 30 to 80% rework, so attacking rework can greatly improve productivity.

- **Requirements delivery rate.** Measures process efficiency by understanding the number of initial requirements the project committed to and the final number delivered. Requirements delivery rate can be measured as an absolute or as a percentage of delivered requirements. Because function points measure requirements from the customer's point of view, it is possible to substitute function points planned and delivered for this measure.

- **Defect-phase containment.** Measures quality and process efficiency by understanding the defect counts by phase and by origin, i.e., count of requirements defects detected at design time, coding time, system testing time, beta test, post-release, etc. Adding cost-to-fix data to this measure makes the information even more interesting. Industry data shows that, on average, a requirements defect detected during the requirements phase is much cheaper to fix than a requirements defect detected later—for example, in the system test.

  Defect-phase containment is a more sophisticated measure than the other measures described in this section. It requires recording defects found throughout software development, including defects found in phases such as unit test, which are usually not recorded. Also, each defect needs to be analyzed to record the phase where it originated and the phase where it was found, and ideally the cost to fix as well. Defects found after release need to be recorded as well for some predetermined period of time. Then the percentage of requirements defects detected in requirements, in design, in coding, in testing, and after release can be calculated. The same calculation can be done for defects originating in design and originating in code. Some defects originate in defect fixes, so that is another category. An extremely efficient process finds 95% of defects in the same phase in which they are inserted, when they are cheapest to find and fix.

Although this list provides some common measures to measure productivity and quality, Construx recommends that business goals always be used to establish the specific measures that make sense for each organization. It is also important to understand the current measurement baseline and add measurements incrementally and appropriately. For example, earned value is a sophisticated measure that requires the organization to have effective project tracking already in place.

Qualitative Measurement

In addition to quantitative measures, it is useful to collect qualitative data from the staff about their perceptions of improvements and the usefulness of a measurement. Qualitative data can prove quite effective for answering difficult questions related to software process improvement goals. This is particularly true of efficiency/productivity, where it is notoriously difficult to come up with effective hard measures of the output of software processes.

Standardized, repeatable, anonymous surveys can be an excellent tool for monitoring trends in an organization. One task the measurement program could facilitate is a short semi-annual survey that asks software development staff questions about their perception of process improvement.

Individuals could be asked to disagree or agree on a scale of 1 through 5 for a series of statements about their perception of the organization's effectiveness. The survey should at least be anonymous at the individual level, possibly at the team level, or even across an entire department or organization. Some of the questions in a survey like this should be based on improvement goals. Some examples of disagree/agree statements include

- "My personal productivity has risen in the last 6 months."
- "My team's productivity has risen in the last 6 months."
- "Our productivity has risen in the last 6 months."
- "The requirements I have received have improved in the last 6 months."
- "The requirements I receive are adequate for me to efficiently complete tasks."
- "I have a reasonable number of task commitments each week."
- "I have the resources to complete my work effectively."
- "I feel pressured to take shortcuts to meet project schedules."

Once tabulated, the results of the survey can be illuminating on their own. They are most interesting for trend analysis, in which standardized questions are compared over time.

Morale is a huge driver of individual productivity and is also affected by an individual's perception of his or her productivity (most engineers want to feel productive). Providing recommendations for improving morale is beyond the scope of this white paper, but regularly measuring morale would provide management with a good indication of trends in morale, and thus related productivity trends.

Starting a Measurement Program

An effective software measurement program can be instrumental in guiding projects to success. But many measurement programs are ineffective or are outright failures. Only about 20% of measurement programs exist two years after they are started. Without a knowledgeable selection of measures and a well-supported deployment of data collection measures, measurement programs are likely to produce data that doesn't match the on-the-ground truth of processes and perceptions within your organization. Keys to successfully starting a measurement program include the following:

- Have high-level sponsorship for the program. Implementing measurement, more than most software process improvements, needs a high-level sponsor for the effort to be successful, as it has a wide and deep impact in the organization.
- Charter a virtual or actual team with stakeholders and project staff to determine what will be measured, how it will be measured, and how the measurements will be used. The team does not have to be co-located, but it does need to meet regularly, both to plan the initial rollout and scope and to monitor the ongoing program. During deployment, the team is available to receive suggestions from the organization about changes that should be made to the data being gathered, the measurements, or the processes for collecting data.
- Establish a goal for the measurement program of providing managers and teams with useful and usable information about project and product delivery. The measurement program itself should not be responsible for dictating the use of measures in software development projects. Rather, it should facilitate the implementation of measurement-based management. Thus, if management wants a certain measure adopted across the organization, the mandate for use of that measure should come from management. Support for collecting data and interpreting results can come from the measurement program.
- Use measurement for process improvement only. The measures should never be used for staff reviews, promotions, or salary decisions. Although it can be tempting to use the data for these purposes, organizations that have attempted this find that this is one of the fastest ways to ensure the program is not successful. When people fear that measures will be used against them or that they or their project won't measure up compared to others, they will be resistant to the program and may not provide data that reflects the truth about their projects and the organization.

- Clearly communicate the goals of the measurement effort to the staff. It is common for the staff to be concerned when measurement is implemented. They might be afraid they won't measure up individually or that their productivity or their project's productivity will compare unfavorably with colleagues and other projects. It's important that this underlying fear is recognized and understood, and that the organization clearly communicate that the measurement effort is not to gather individual data but to improve performance based on the organization's goals.
- Recognize that measurement by itself only provides data. It doesn't change processes, except that staff will try to get better at whatever is being measured. The organization needs to decide on the software process improvements to implement and to use measurement to decide if the improvements are helping it achieve its goals.
- Use a Goal Question Metric (GQM) approach, and gather only data that you have a real, immediate use for. Gathering data that is not used by the organization tends to make people cynical about a measurement program. It is better to choose a few measures that will actually be used in decision making than to gather a broad data set that in which some data is never used. Once the organization is collecting and using the initial set of measures, it can incrementally add more measures to answer the complete set of questions necessary to understand if it is achieving its goals.
- Look for quick wins during the program implementation. Managers and teams within the organization should be encouraged and rewarded for experimentation with data collection and metric analysis in a grass-roots fashion. Management doesn't need to wait for the measurement program to be fully implemented before looking for ways that data would be useful on projects.
- Feed back the information to the people who provided it. In many organizations, the measurements are gathered but the data is not sent back to the teams. By publishing the measurements (at least in summary form), the organization lets people know that the data is at least being paid attention to. More important is actually publishing management-level decisions based on the measurements—for example, "We saw this trend in that measurement, so we decided to do that differently." If people see that the data is being used to make important decisions, they not only have an incentive to keep providing the data, they also have an incentive to provide the right data.
- When starting the program, do not expect to gather data that is 100% accurate. Realize that some proposed measures may be difficult to gather with 100% accuracy, and it may be necessary to sacrifice some accuracy to make the data collection manageable.

About Construx

This white paper was created by Construx Software Builders, Inc. Construx Software is the market leader in software development best practices training and consulting. Construx was founded in 1996 by Steve McConnell, respected author and thought leader on software development best practices. Steve's books *Code Complete*, *Rapid Development*, and other titles are some of the most accessible books on software development with more than a million copies in print in 20 languages. Steve's passion for advancing the art and science of software engineering is shared by Construx's team of seasoned consultants. Their depth of knowledge and expertise has helped hundreds of companies solve their software challenges by identifying and adopting practices that have been proven to produce high quality software—faster, and with greater predictability.

Steve McConnell, CEO/Chief Software Engineer

steve.mcconnell@construx.com
+1(425) 636-0100

Jenny Stuart, VP Consulting

jenny.stuart@construx.com
+1(425) 636-0108

Matt Peloquin, CTO

matt.peloquin@construx.com
+1(425) 636-0104

Steve Tockey, Principal Consultant

steve.tockey@construx.com
+1(425) 636-0106

Mark Nygren, COO/VP Sales

mark.nygren@construx.com
+1(425) 636-0110

For more information about Construx's support for software development best practices, please see our website at www.construx.com, contact us at consulting@construx.com, or call us at +1(866) 296-6300.

Construx Software Builders, Inc.
10900 NE 8th Street, Suite 1350
Bellevue, WA 98004
U.S.A.