Software Project and Process Measurement

Dr. Christof Ebert
Vector Consulting Services

Software measurement is the discipline that ensures that we stay in control and can replicate successful processes. It applies to products (e.g., ensuring performance and quality), processes (e.g., improving efficiency), projects (e.g., delivering committed results), and people (e.g., evolving competence). This article discusses software measurement and provides practical guidance for project and process measurement.

Goals that are measured will be achieved. In almost every area of life, setting goals and monitoring achievement are the foundations for success. Take a relay team in track or swimming. What drives these athletes to continuously improve? There are always competitors to beat, a number 1 ranking to achieve, or a record to break. Many of the same drivers apply to the software industry. Software development is a human activity and, as such, demands that organizations continuously improve their performance in order to stay in business. Software measurement is a primary approach to control and manage projects and processes and to track and improve performance.

The way software measurement is used in an organization determines how much business value that organization actually realizes. Software measurements are used to:

- Understand and communicate.
- Specify and achieve objectives.
- Identify and resolve problems.
- Decide and improve.

Today, improving a business (and its underlying processes) is impossible without continuous measurements and control. For some industries and organizations, this is demanded by laws (such as the Sarbanes-Oxley Act) and liability regulations. For all organizations, it is a simple question of accounting and finance control. The measurement process is an inherent part of a business process (see Figure 1). Therefore, software engineering—as part of the product development business process—also needs controlling and measurement. Software measurements include performance measurement, project control, and process efficiency and effectiveness.

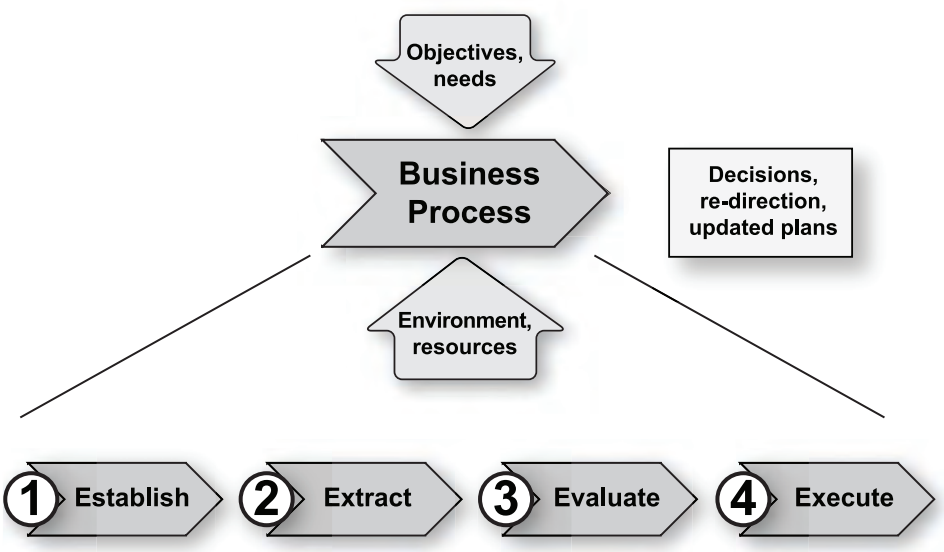
Measurements are management tools. Consequently, measurements must be governed by goals such as reducing project risks or improving test efficiency, otherwise they simply build up into *data cemeteries*. Figure 1 shows this objective-driven generic measurement process, known as E-4 [1]:

1. **Establish** concrete improvement or control objectives and the necessary measurement activities.
2. **Extract** measurements for the established performance control needs.
3. **Evaluate** this information in view of a specific background of actual status and objectives.
4. **Execute** decisions and actions to reduce the differences between status and objectives.

The E-4 measurement process is based on the Deming Cycle (Plan, Do, Check, Act) and extends classic measurement paradigms, such as the goal-question-metric approach, by adding an immediate action-focus. It follows the observation that to effectively and continuously improve a process, it is important to first stabilize it, keep it in its specification limits, and continuously improve it [2]. The major difference is E-4's clear focus on goal-oriented measurement and execution of decisions at the end of the four steps.

The E-4 measurement process is governed by ISO/IEC 15939 [3], a standard that summarizes how to plan and implement a measurement process. It consists of two parts: The first part establishes the measurement program and prepares it within the project or organization. The second part is about execution: You extract data, evaluate it, and execute corrective actions based on the outcome of the measurement analyses. The second part—driven by the first part—sets the standards and defines what to do with the measurements. It is not enough to simply collect numbers and report them. Measurements are only effective if they are embedded as tools to support decision-making and to drive the implementation of decisions. For the fundamentals of software measurement and practical solutions, I refer to the book "Software Measurement" [1].

Figure 1: *A Generic Measurement Process*



Project Measurement

Project measurement is the major application of software measurement. Software measurement is an absolutely necessary precondition for project success, but only one-third of all software engineering companies systematically utilize techniques to measure and control their product releases and development projects [4].

The goal of project measurement is to master the project and finish it according to commitments. What is needed is a way to determine if a project is on track or

not. Whether it is embedded software engineering, the development of a software application, or the introduction of a managed IT-service, demand outstrips the capacity of an organization. As a result, I've observed the acceptance of impossible constraints in time, content, and cost; as a direct consequence of acceptance comes an increase in churn and turmoil—as well as budget overruns, canceled projects, and delays. Still, far too many projects fail to deliver according to initial commitments simply because of insufficient or no adequate measurement.

While many organizations claim that project work is difficult due to changing customer needs and high technology demands, the simple truth is that they do not understand the basics: estimation, planning, and determining progress during the project. Consequently, they fully lose control once customer requirements change.

Project control can help in avoiding these problems by answering a few simple questions derived from the following management activities:

- **Decision-making.** What should I do?
- **Attention directing.** What should I look at?
- **Performance evaluation.** Are we doing a good or bad job?
- **Improvement tracking.** Are we doing better or worse than in the last period?
- **Target setting.** What can we realistically achieve in a given period?
- **Planning.** What is the best way to achieve our targets?

Project monitoring and control is defined as a control activity concerned with identifying, measuring, accumulating, analyzing, and interpreting project information for strategy formulation, planning, and tracking activities, decision-making, and cost accounting. As such, it is the basic method for gaining insight into project performance and is more than just ensuring the overall technical correctness of a project. The most important elements are the existence of a closed loop between the object being controlled, the actual performance measurements, and a comparison of targets against actuals. Figure 2 shows this control loop. The project with its underlying engineering processes delivers results, such as work products. It is influenced and steered by the project goals. Project control captures the observed measurements and risks and relates them to the goals. By analyzing these differences, specific actions can be taken to get back on track or to ensure that the project remains on track. These actions serve as an additional input to the project beyond the original project targets. Making the actions effective is the role of the project manager.

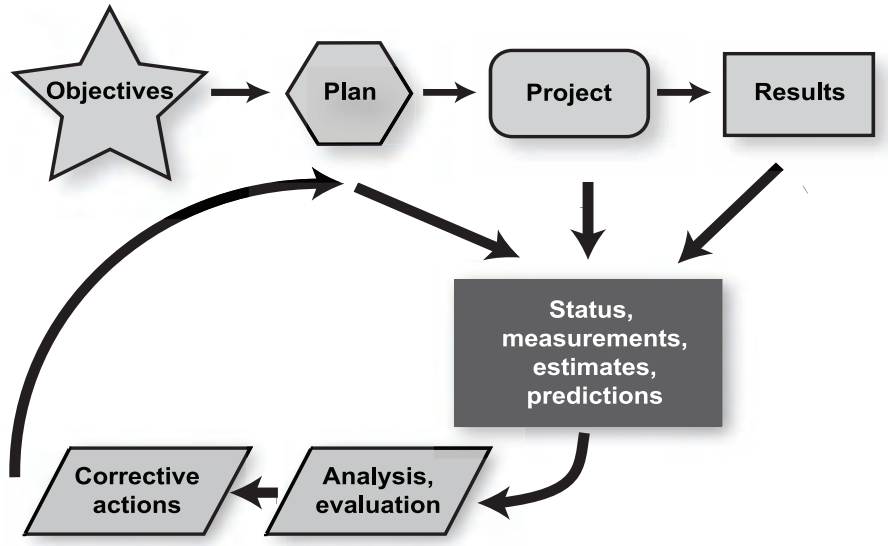


Figure 2: *Measurements Provide a Closed Feedback Loop for Effective Corrective Action*

A client recently asked us for help in improving engineering efficiency at their organically grown embedded software business. Software costs initially did not matter when there were only a few engineers. As things progressed, software dominated hardware and cost increases were out of control. Customers liked the technology but increasingly found quality issues and, when the customers made audits, did not really find a decent engineering process. Our Q&A with the client went like this:

Us: "How do you set up a software project?"

The Client: "Well, we take the requirements, build a team of engineers, and follow our V-shaped master plan."

Us: "Is the planning typically accurate?"

The Client: "No, but we don't have any better basis."

Us: "How do you make sure you have the right quality level?"

The Client: "Well, we do a unit test, integration test, and system test. But, in fact, we test too much and too late."

Us: "Is your cost in line with expectations or could you do better?"

The Client: "We really don't know about how this all is measured and what to learn from measurements. That's why we invited you."

In terms of software and technology, this client was way above average. However, it was unclear to management and engineers how to assess status, mitigate risks, and improve performance. The project managers had no history database to decide on release criteria. Requirements were changing, but without any control. It is exactly this pattern of exciting technology and skills—combined with fast growth but insufficient engineering and management processes—that eventually hits many companies in embedded software development. Finally, the client's customers concluded that despite all of the technical advantages, processes and practices were below current professional software engineering expectations. A natural first step on our side was to introduce a lean yet effective measurement program.

To get started without much overhead, our team recommended a lean set of project indicators [1, 5, 6]. They simplified the selection by reducing the focus on project tracking, contractor oversight, and program management perspective. Here is our short list of absolutely necessary project measurements:

- **Requirements status and volatility.** Requirements status is a basic ingredient to tracking progress based on externally perceived value. Always remember that you are paid for implementing requirements, not just generating code.
- **Product size and complexity.** Size can be measured as either functional size in function points or code size in lines of code or statements. Be prepared to distinguish according to your measurement goals for code size between what is new and what is reused or automatically generated code.
- **Effort.** This is a basic monitoring parameter to assure that you stay within budget. Effort is estimated up-front for the project and its activities. Afterwards, these effort elements are tracked.
- **Schedule and time.** The next basic

monitoring measurement is ensuring that you can keep the scheduled delivery time. Similar to effort, time is broken down into increments or phases that are tracked based on what is delivered so far. Note that milestone completion must be lined up with defined quality criteria to avoid poor quality being detected too late.

- **Project progress.** This is the key measurement during the entire project execution. Progress has many facets: Simply look to deliverables and how they contribute to achieving a project's goals. Typically, there are milestones for the big steps, and earned value and increments for the day-to-day operational tracking. Earned value techniques evaluate how results (such as implemented and tested requirements or closed work packages) relate to the effort spent and elapsed time. This then allows for estimating the cost to complete and remaining time to complete the project.
- **Quality.** This is the most difficult measurement, as it is hardly possible to accurately forecast whether the product has already achieved the right quality level that is expected for operational usage. Quality measurements need to predict quality levels and track how many defects are found compared to estimated defects. Reviews, unit test, and test progress and coverage are the key measurements to indicate quality. Reliability models are established to forecast how many defects still need to be found. Note that quality attributes are not only functional but also relate to performance, security, safety, and maintainability.

These measurements must be actively used for the weekly tracking of status, risks, and progress (e.g., increment availability, requirements progress, code delivery, defect detection), while others are used to build up a history database (e.g., size, effort). Most of these measurements, such as defect tracking, are actually by-products from operational databases. This ensures sufficient data quality to compare project status across all projects of a portfolio. External benchmarks (such as provided in [1,6]) help in bootstrapping a measurement program as they immediately point towards inefficiency and below-average performance.

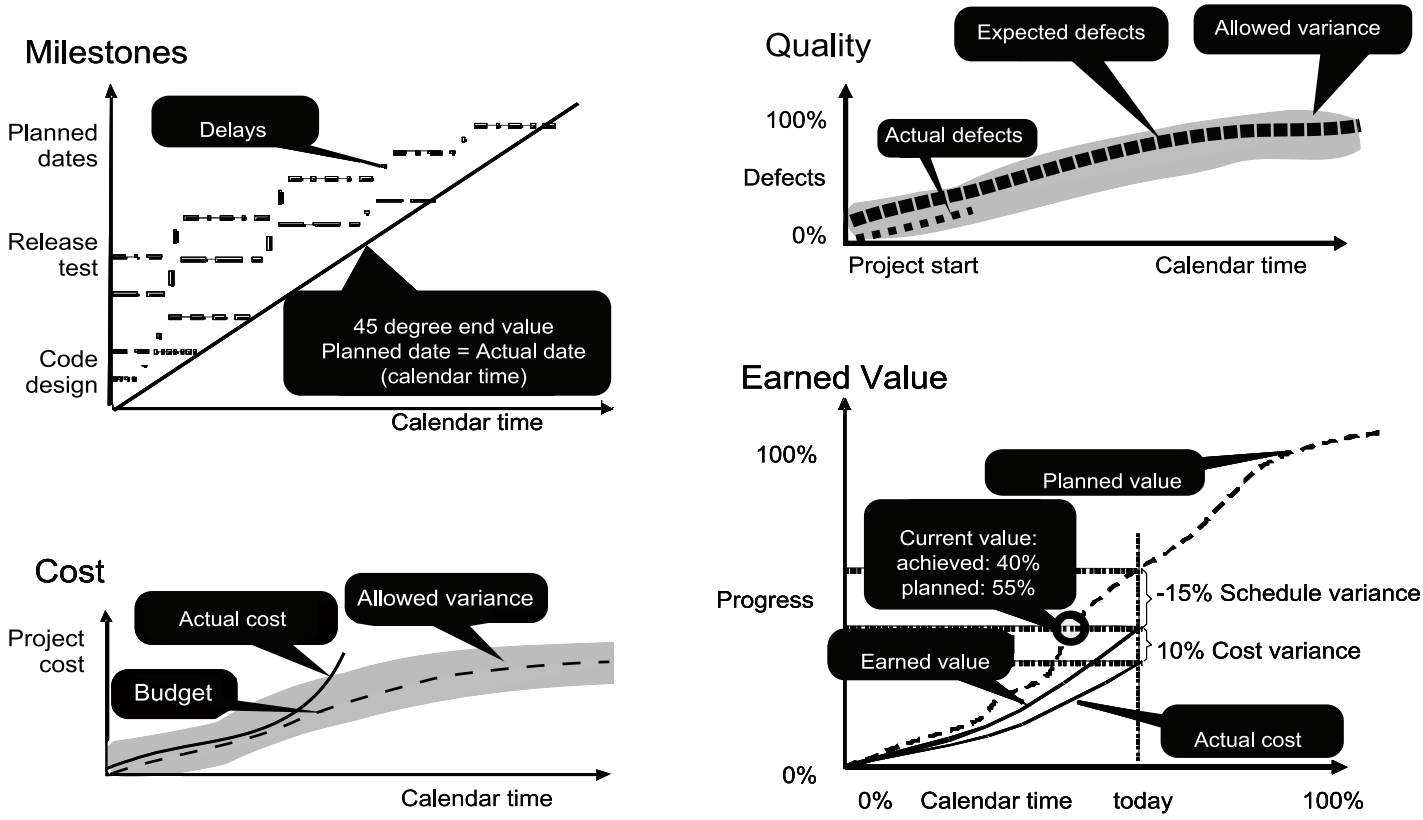
To ease monitoring and avoid getting lost in the fog of numbers, projects should aggregate the relevant information in a standardized dashboard. Figure 3 shows a simplified dashboard of how we have utilized it with many clients. It includes milestone tracking, cost evolution, a selection of process measurements, work product deliveries, and faults with status information. There can be both direct measurements (e.g., cost) as well as indirect measurements and predictions (e.g., cost to complete). Such dashboards provide information in a uniform way across all projects, thus not overloading the user with different representations and semantics to wade through. All projects within the organization must share the same set of consistent measurements presented in a unique dashboard. A lot of time is actually wasted by reinventing spreadsheets and reporting formats when the project team should be focused on creating value.

Measurements—such as schedule and budget adherence, earned value, or quality level—are typical performance indicators that serve as *traffic lights* on the status of the individual project. Only projects in *amber* and *red* status that run out of agreed variance (which, of course, depends on the maturity of the organization) would be investigated. The same dashboard is then allowed to drill down to more detailed measurements to identify root causes of problems. When all projects follow a defined process and utilize the same type of reporting and performance tracking, it is easy to determine status, identify risks, and resolve issues—without getting buried in the details of micromanaging the project.

Process Measurement

Software engineering processes determine the success of organizations as well as how they are perceived in the marketplace.

Figure 3: *Measurement Dashboard Overview*



Purchasing organizations worldwide are using tools such as CMMI to evaluate their suppliers, be it in automotive, information/communication technologies, or governmental projects [1, 7]. Markets have recognized that continuous process improvement contributes substantially to cost reductions and quality improvement. An increasing number of companies are aware of these challenges and are proactively looking at ways to improve their development processes [8]. Suppose a competitor systematically improves productivity at a relatively modest annual rate of 10 percent (as we can see in many software and IT companies). After only three years, the productivity difference is $(1.1^3) = 1.33$, or a 33 percent advantage. This directly translates into more capacity for innovation, higher margins, and improved market positioning.

Software measurement is a necessary precondition to performance improvement. The concept of objective-driven process improvement will focus processes on the objectives they must achieve. Processes are a means to an end and need to be lean, pragmatic, efficient, and effective—or they will ultimately fail, despite all the push one can imagine. Figure 4 shows this goal-driven relationship from business objectives to concrete annual performance objectives (on an operational level) to specific process performance measurements.

What follows are eight integral software measurement steps, showing how objective-driven process improvement is translated into concrete actions, and how software measurements are used to improve processes. Each step includes an example—all taken from the same medium-sized company that Vector recently consulted—showing how we put each step into action.

Step 1: Identify the organization's improvement needs from its business goals

These business goals provide the guidance for setting concrete engineering performance improvement objectives on a short-term basis. Example: The business goal with our client was to improve revenues and cash flows and to reduce cost of non-performance from schedule delays.

Step 2: Define and agree on the organization's key performance indicators (KPIs)

Such KPIs are standardized across the organization and ensure visibility, accountability, and comparability. Naturally, KPIs must relate to the business goals. Measurements should drive informed decision-making. They must be used for effectively communicating status and progress against the objectives set for the business, process, or project. Measurements, both direct and indirect, should be periodically evaluated in conjunction with the driving objectives and to identify problems or derive decisions. Example: The selected performance indicator was schedule predictability, measured as the normalized delays compared to the originally agreed deadline. Schedule changes (after a project had started) were not considered in this measurement. This avoided the argument that a specific delay was justified due to changing requirements. Once such an excuse is accepted, delays typically increase—as do costs (due to the changes). In this case, we found almost unanimous agreement from product managers and business owners who found that the lack of schedule changes helped avoid the downward spiral of requirements changes, delays, and cost overruns.

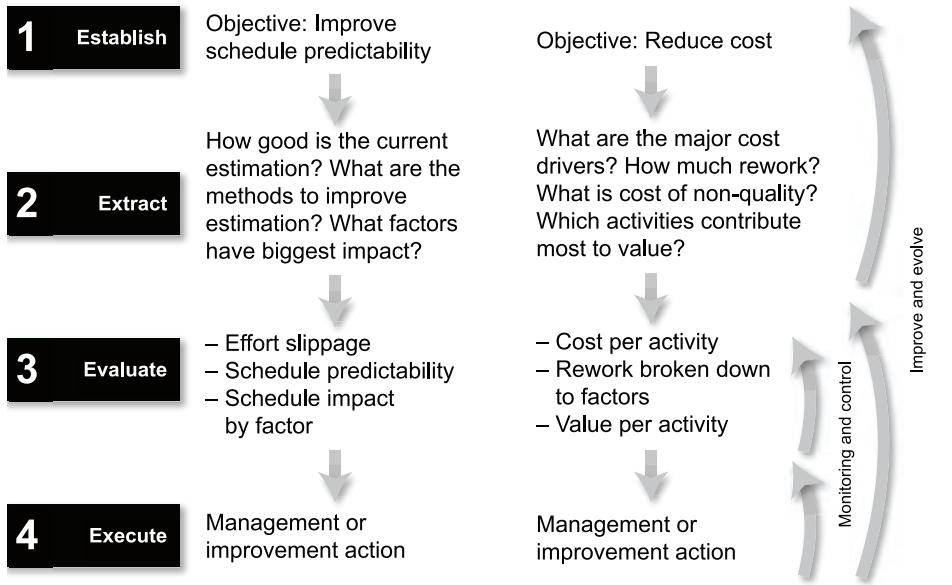


Figure 4: *Applying the E-4 Measurement Process to Achieve Improvements*

Step 3: Identify the organization's hot spots, such as areas where they are feeling the most pain internally or from customers

Typical techniques include root cause analysis of defects and customer surveys. Example: Average schedules in the company were 45 percent behind initial commitments. A root cause analysis of delays was performed: 40 percent of delays came from insufficient project management; 30 percent from changing requirements; 20 percent from supplier delays; and 10 percent from other causes. Looking to the highest-ranking root causes, we found insufficient planning and control within the project. Often, requirements were added or changed following a customer question—rather than including the customer in the trade-off and impact analysis. When speaking to client-customers, this became even more evident: Customers perceived such ad-hoc changes as a weakness and missed clear guidance and leadership on feature content.

Step 4: Commit to concrete improvement objectives

These improvement objectives should directly address the mentioned weaknesses and simultaneously support the overarching business objective. We use the acronym SMART to outline good objectives:

- **S**pecific (precise).
- **M**easurable (tangible).
- **A**ccountable (in-line with individual responsibilities).
- **R**ealistic (achievable).
- **T**imely (suitable for current needs).

These objectives are reviewed and approved by upper-level managers before proceeding further. This ensures that priorities are appropriate and that nothing relevant has been overlooked or misunderstood. Example: Two improvement objectives, improving estimations and improving requirements development, were agreed upon. The respective performance targets were agreed to in a management seminar to achieve buy-in. Each project was required to have two estimates where the first was allowed to deviate by 20 percent and the second to deviate by 10 percent. After the project started, the requirements change rate was required to be under 20 percent—except that customers would, after a clear decision-making process, agree on trade-offs and pay for the changes. Time-boxing and incre-

mental development with prioritized requirements was then introduced to achieve improvement objectives. Requirements priorities were agreed upon across impacted stakeholders from product management, engineering, and marketing/sales before each new project started.

Step 5: Identify specific levers[1] to start improvements and connect them to ROI planning

This is typically best done when using a process improvement framework such as CMMI [6]. This framework provides the necessary guidance regarding which best practices to apply and how processes relate to each other. Without the right levers, chances are high that objectives will not be reached. Example: Focus was on requirements development, requirements management, technical solutions, project planning, and project monitoring and control. Project managers were educated in project management techniques and negotiation skills.

Step 6: Perform a brief gap analysis of the selected process areas to identify strengths and weaknesses

This systematic look at weaknesses helps to focus limited engineering resources where it matters most. Example: Requirements were collected rather than developed; requirements management satisfied the basic need for change management; engineering was too technology-driven and was extended to capture business reasoning; project planning showed severe weaknesses in estimation and feasibility analysis; and project monitoring and control showed weaknesses in getting stakeholder agreement on changes.

Step 7: Develop a concrete action plan for the identified weaknesses

Avoid trying to change all weaknesses at the same time. Use increments to subdivide bigger changes. Consider available resources and skills and get external support if you lack competencies, such as change management. Example: The most urgent need was project planning. A dedicated one-month initiative was launched right away to install a suitable estimation method and to train people on it. A tool for feasibility analysis was introduced in parallel because not much of the organization's own historical data was available. A historical database was installed for a set of key project measurements. In a second phase, earned value analysis was introduced. After three months, requirements development was launched under the leadership of product management.

Step 8: Implement improvements and deliver tangible results

Implement the agreed-upon changes to operational projects and measure progress against the committed improvement objectives. Example: Performance measurements were collected from all ongoing projects. There weren't reprimands for insufficient performance, but it was carefully analyzed. We found with this client that too many requirements changes lacked a clear specification, analysis, and commitment by the product manager. Consequently, a strong focus was given towards change management and change review boards[2]. A weekly project review was introduced after a few weeks, enhanced with daily Scrum meetings of development teams. Requirements changes passing the change review board had to have their proper business case or were not accepted. Although the results proved valid, marketing and sales were unhappy because they wanted flexibility and no accountability for the changes. This improvement reduced requirements changes (within the first three months) substantially. The first few projects had 20 percent schedule overrun (compared to the previous average of 45 percent) and two of them were even close to 10 percent. These two were further evaluated to identify best practices.

As it turned out, the project managers demanded requirements reviews by product managers and testers before accepting requirements to change review boards. The quality of requirements substantially improved. This change was immediately pushed forward by senior management for all projects. As it turns out, testers were unhappy because they didn't have the time scheduled for doing the reviews. Rather than demanding overtime, senior management asked for *five percent slots* (two hours) each week; this proved to be sufficient time to review one to three requirements with the necessary depth.

Getting More From Your Measurements

Measurement programs mostly fail because they are disconnected from actual business. Too often, things are measured that do not matter at all and there is usually no clear improvement objective behind what is measured. Often the collected measurements and resulting reports are useless, only creating additional overhead. In the worst cases, they hide useful and necessary information. We have seen reports on software programs with more than 50 pages full of graphs, tables, and numbers. When asked about topics such as cost to complete, expected cost of non-quality after handover, or time-to-profit, the organization did not have a single number. Sometimes numbers are even created to hide reality and attract attention to what looks good. Be aware that measurements are sometimes abused to obscure and confuse reality.

The primary question with software measurement is not "What measurements should I use?" but rather "What do I need to improve?" It is not about having many numbers but rather about having access to the exact information needed to understand, manage, and improve your business. This holds true for both project and process measurement.

As a professional in today's fast-paced and ever-changing business environment, you need to understand how to manage projects on the basis of measurements and forecasts. You need to know which measurements are important and how to use them effectively. Here are some success factors to pragmatically use measurements:

- Estimate project time and effort and realistically set deadlines.
- Check feasibility on the basis of given requirements, needs, and past project performance (productivity, quality, schedule, and so on). Do not routinely overcommit.
- Manage your requirements and keep track of changes. Measure requirements changes and set thresholds of what is allowed.
- Know what *value* means to your client or customer. Track the earned value of your project.
- Understand what is causing delays and defects. Do not let delays accumulate. Remove the root causes and do not simply treat the symptoms.
- Be decisive and communicate with a fact-based approach. Avoid disputes with management, clients, and users.
- Deliver what matters. Use measurements to keep commitments.
- Continuously improve by analyzing your measurements and then implementing respective objective-driven changes. Follow through until results are delivered.

Practitioners should help management so that they make decisions on the basis of measurements. This will give management the necessary information

before and after the decision so that they can follow the effects and compare them to the goals. Managers should ensure that decisions are made based on facts and analyses. They should always consider number 4 in the E-4 process (executing decisions and actions), and make sure that decisions move the organization towards agreed-upon objectives.

Accountability means setting realistic and measurable objectives. Objectives such as *reduce errors* or *improve quality by 50 percent* are not useful. The objective should be clear and tangible: *Reduce the number of late projects by 50 percent for this year compared to the previous year.* These objectives must end up in a manager's key performance indicators. Projects using unclear oral, memo, or slide reports will most certainly fail to deliver according to commitments. Projects using a standard spreadsheet-based progress reporting—related to requirements, test cases, and defects versus plans—will immediately receive the necessary attention if they deviate. With this attention, corrective action will follow, and the project has a good chance to recuperate because of sufficiently early timing.

Goals that are measured will be achieved. The reason is very simple: Once you set a SMART objective and follow it up, you make a commitment to both yourself and your team or management group. As humans—and specifically as engineers—we want to deliver results. Measurement provides the stimulus and direction to reach our goals.◆

References

1. Ebert, Christof, and Reiner Dumke. Software Measurement. New York/Heidelberg: Springer-Verlag, 2007.
2. Juran, Joseph M., and A. Blanton Godfrey. Juran's Quality Handbook. New York: McGraw-Hill Professional, 2000.
3. ISO/IEC. ISO/IEC 15939:2002. Software Engineering – Software Measurement Process. 2002.
4. Kraft, Theresa A. "Systematic and Holistic IT Project Management Approach for Commercial Software With Case Studies." Information System Education Journal 63.6: 1-15. 23 Dec. 2008 <http://isedj.org/6/63/>.
5. Carleton, Anita D., et al. "Software Measurement for DoD Systems: Recommendations for Initial Core Measures." Technical Report CMU/SEI-92-TR-19. Sept. 1992 <www.sei.cmu.edu/pub/documents/92.reports/pdf/tr19.92.pdf>.
6. Ebert, Christof, and Capers Jones. "Embedded Software: Facts, Figures and Future." IEEE Computer 42.4: 42-52, Apr. 2009.
7. Chrissis, Mary Beth, Mike Konrad, and Sandy Shrum. CMMI: Guidelines for Process Integration and Product Improvement. 2nd ed. Boston: Addison-Wesley Professional, 2006.
8. Gibson, Diane L., Dennis R. Goldenson, and Keith Kost. "Performance Results of CMMI-Based Process Improvement." Technical Report CMU/SEI-2006-TR-004. Aug. 2006 <www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr004.pdf>.

Notes

1. "Levers" in this case mean to set up the improvement project in a way that the different changes follow some order, won't come ad-hoc and isolated, and would thus meet objectives.
2. Change review boards are staffed with engineering and product managers and ensure that both technical and business rationale and impacts of changes are considered. They then make informed and firm decisions.

About the Author

Christof Ebert, Ph.D., is managing director and partner at Vector Consulting Services. He is helping clients worldwide to improve technical product development and to manage organizational changes. Prior to working at Vector, he held engineering and management positions for more than a decade in telecommunication, IT, aerospace, and transportation. A measurement practitioner who has worked for *Fortune* 500 companies, Ebert authored "Software Measurement," published by Springer, now in its third fully revised edition.

Vector Consulting Services
Ingersheimer Straße 24
D-70499 Stuttgart
Germany
Phone: +49-711-80670-0
E-mail: christof.ebert@vector-consulting.de