

Software Metrics: Ten Traps To Avoid¹

Karl E. Wiegers

Process Impact
www.processimpact.com

As software development gradually evolves from art toward engineering, more and more developers appreciate the importance of measuring the work we do. While software metrics can help you understand and improve your work, implementing a metrics program is a challenge. Both the technical and the human aspects of software measurement can be difficult to manage.

According to metrics guru Howard Rubin, up to 80 percent of software metrics initiatives fail within two years. In this article we will examine ten traps that can sabotage the unsuspecting metrics practitioner. Several symptoms of each trap are described, along with some possible solutions. By being aware of these common risks, you can chart a course toward successful measurement of your software development activities.

Trap #1: Lack of Management Commitment

Symptoms: As with most improvement initiatives, management commitment is essential for a metrics effort to succeed. The most obvious symptom that commitment is lacking is when your management actively opposes measurement. More frequently, management claims to support measurement, and effort is devoted to designing a program, but practitioners do not collect data because management hasn't explicitly required them to.

Another clue that managers aren't fully committed is that they charter a metrics program and planning team, but then do not assist with deploying the program into practice. Managers who are not committed to software measurement will not use the available data to help them do a better job, and they won't share the data trends with the developers in the group.

Solutions: To persuade managers of the value of software measurement, educate them! A good resource is *Practical Software Metrics for Project Management and Process Improvement*, by Robert Grady (PTR Prentice-Hall, 1992). Tie the metrics program to the managers' business goals, so they see that having good data available is the only way to tell if the software organization is becoming more effective. You need their input to help design the metrics program, to ensure that it will meet their needs.

If you cannot obtain commitment from senior management, turn your attention to the project and individual practitioner level. There are many valuable metrics that developers and project teams can use to understand and improve their work, so focus your energy at those who are willing to try. As with any improvement initiative, grass roots efforts can be effective locally, and you can use positive results to encourage a broader level of awareness and commitment..

Trap #2: Measuring Too Much, Too Soon

Symptoms: Hundreds of aspects of software products, projects, and processes can be measured. It is easy to select too many different data items to be collected when beginning a metrics program. Until a measurement mindset is established in the organization, expect resistance to both

¹ This paper was originally published in *Software Development*, October 1997. It is reprinted (with modifications) with permission from *Software Development* magazine.

the concept of measuring software and the time required to collect, report, and interpret the data. A long list of metrics might scare off some of the managers and practitioners whose participation is required for the program to succeed.

Solutions: Begin growing your measurement culture by selecting a fairly small, balanced set of software metrics. By balanced, I mean that you are measuring several complementary aspects of your work, such as quality, complexity, and schedule. As your team members learn what the metrics program is all about, and how the data will (and will not) be used, you can gradually expand the suite of metrics being collected. Start simple and build on your successes.

The participants in the metrics program must understand why the requested metrics are valuable before they will be willing to do their part. For each metric you propose, ask, “What can we do differently if we have this data?” If you can’t come up with an answer, perhaps you don’t need to measure that particular item right now. Once the participants are in the habit of using the data they collect to help them understand their work and make better decisions, the program can be expanded.

Trap #3: Measuring Too Little, Too Late

Symptoms: Some programs start by collecting just a few measures, which don’t provide enough useful information to let people understand what’s going on and make better decisions. This might lead stakeholders to conclude that the metrics effort is not worthwhile, so they terminate it prematurely. Another obstacle to getting adequate and timely data is the resistance many software people exhibit toward measurement. Participants who are more comfortable working undercover may drag their feet on collecting data.. They may report only a few of the requested data items, or only report data points that make them look good, or turn in their data long after it is due. If your metrics effort is having trouble getting off the launch pad because of some resisters, you may be a victim of this trap.

A metrics program has the potential to do actual damage if too few dimensions of your work are being measured. People sometimes change their behavior in reaction to what is being measured, which can have unfortunate and unanticipated side effects. For example, if you are measuring productivity but not software quality, some people may change their programming style to generate more lines of code and therefore look more productive. I can write code very fast if the quality is not important.

Solutions: As with Trap #2, the balanced set of metrics is critical to success. Measure several aspects of your product size, work effort, project status, quality of product, or customer satisfaction. You don’t need to start with all of these at once, but select a small suite of key measures that will help you understand your group’s work better, and begin collecting them right away. Since software metrics are often a lagging indicator of what is going on, the later you start, the farther off-track your project might stray before you realize it. Avoid choosing metrics that might tempt program participants to optimize one aspect of their performance at the expense of others.

Trap #4: Measuring the Wrong Things

Symptoms: If the data items being collected do not obviously relate to the key success strategies for your business, you may be measuring the wrong things. If your managers are not obtaining the timely information they need to do a better job of managing the projects and people, it’s time to reevaluate your metrics suite. Another symptom of this trap is that inappropriate surrogate measures are being used. One example is attempting to measure actual project work effort using an accounting system that insists upon 40 labor hours per week per employee.

Solutions: Select measures that will help you steer your process improvement activities, by showing whether process changes are having the desired effect. For example, if you're taking steps to reduce the backlog of change requests, measure the total number of requests submitted, the number open each week, and the average days each request is open. To evaluate your quality control processes, count the number of defects found in each test and inspection stage, as well as the defects reported by customers. As you design the program, leverage from what individuals or project teams are already measuring.

Make sure you know who the audience is for the metrics data, and make sure the metrics being collected will accurately answer their questions. The goal/question/metric (GQM) paradigm works well for selecting the metrics that will let you answer specific questions associated with defined organizational or project goals. A good example of applying GQM is found in "A Practical View of Software Measurement and Implementation Experiences Within Motorola" by Michael K. Daskalantonakis, in *IEEE Transactions on Software Engineering* (November 1992).

Trap #5: Imprecise Metrics Definitions

Symptoms: Vague or ambiguous metric definitions allow every practitioner to interpret them differently. One person counts an unnecessary software feature as a defect, while someone else does not. Time spent fixing a bug found by testing is classified as test effort by one person, coding effort by another, and rework by a third. Trends in measures tracked over time may show erratic behavior because individuals are not collecting, reporting, or charting their results in the same way.

You'll have a clue that your metrics definitions are inadequate if participants are frequently puzzled about what exactly they are being expected to measure. If you keep getting questions like, "Do I count unpaid overtime in the total work effort?" you may be falling into this trap.

Solutions: A complete and consistent set of definitions for the things you are trying to measure is essential if you wish to combine data from several individuals or projects. For example, the definition of a line of code is by no means standard even for a single programming language. Standardize on a single tool for collecting metrics based on source. Automate the measurement process where possible, and use standard calibration files to make sure all participants have configured their tools correctly.

Those designing the metrics program must create a precise definition for each data item being collected, as well as for other metrics computed from combinations of these data items. This is much harder than you might suspect. Plan to spend considerable time agreeing on definitions, documenting them as clearly as possible, and writing procedures to assist practitioners with collecting the data items easily and accurately.

Trap #6: Using Metrics Data to Evaluate Individuals

Symptoms: The kiss of death for a software metrics initiative is to use the data for input into an individual's performance appraisal. Using metrics data for either reward or punishment, such as rank ordering programmers based on their lines of code generated per day, is totally inappropriate. When someone knows that the numbers she reports might be held against her, she'll either stop reporting numbers at all, or only report numbers that make her look good. Fear of the consequences of reporting honest data is a root of many metrics program failures.

Solutions: Management must make it clear that the purpose of the metrics program is to understand how software is being built, to permit informed decisions to be made, and to assess the impact of process changes on the software work. The purpose is NOT to evaluate individual team

members. Control the scope of visibility of different kinds of data; if individual names are not attached to the numbers, no individual evaluations can be made. However, it is appropriate to include the *activity* of collecting and using accurate (and expected) data in an individual's performance evaluation.

Certain metrics should be private to the individual; an example is the number of defects found by unit testing or code review. Others should remain private to a project team, such as the percentage of requirements successfully tested and the number of requirements changes. Some metrics should have management visibility beyond the project, including actual versus estimated schedule and budget, and number of reported and open customer-reported defects. The best results come about when individuals use their private metrics data to judge and correct themselves, thereby improving their personal software process.

Trap #7: Using Metrics to Motivate, Rather than to Understand

Symptoms: When managers attempt to use a measurement program as a tool for motivating desired behaviors, they may reward people or projects based on their performance with regard to just one or two metrics. Public tracking charts may be pointed out as showing desirable or undesirable results. This can cause practitioners who are using the charts to understand what's up with their software to hide their data, thereby avoiding the risk of public management scrutiny. Managers may focus on getting "the numbers" where they want them to be, instead of really hearing what the data is telling them. As with Trap #3, the behavioral changes stimulated by motivational measurement may not be the ones you really want.

Solutions: Metrics data is intrinsically neither virtuous nor evil, simply informative. Using metrics to motivate rather than to learn has the potential of leading to dysfunctional behavior, in which the results obtained are not consistent with the goals intended by the motivator. Metrics dysfunction can include inappropriately optimizing one software dimension at the expense of others, or reporting fabricated data to tell managers what they want to hear.

Stress to the participants that we must have accurate data if we are to understand the current reality and take appropriate actions. Use the data to understand discrepancies between your quality and productivity goals and the current reality, so you can improve your processes accordingly. The process improvement program is the tool to drive desired behaviors, and the metrics program is the way to see if we are getting the results we want. If you choose to use measurement to help motivate desired behaviors, be *very* careful.

Trap #8: Collecting Data That Is Not Used

Symptoms: The members of your organization may diligently collect the data and report it as requested, yet they never see evidence that the data is being used for anything. People may grumble that they spend precious time measuring aspects of their software, but they don't have any idea of what the benefits are. The required metrics forms are submitted to some collection center, which stores them in a write-only database. Your management doesn't seem to care whether the data is reported or not (related to Trap #1).

Solutions: Software engineering should be a data-driven profession, but it cannot be if the available data disappears from sight. Project leaders and upper management need to close the loop and share results with the team. They need to relate the benefits of having the data available, and how the information helped managers make decisions and take appropriate actions. Selected public metrics trends must be made visible to all stakeholders, so they can share in the successes and choose how to address the shortcomings.

Today's current data becomes tomorrow's historical data, and future projects can use previous results to improve their estimating capability. Make sure your team members know how the data is used, and give them access to the public metrics repository so they can view it—and use it—themselves.

Trap #9: Lack of Communication and Training

Symptoms: You may be falling into this trap if the participants in the metrics program don't understand what is expected of them, or if you hear a lot of opposition to the program. Fear of measurement is a classic sign that the objectives and intent of the program need to be better communicated. If people do not understand the measurements and have not been trained in how to perform them, they won't collect reliable data at the right times .

Solutions: Create a short training class to provide some basic background on software metrics, describe your organization's program, and clarify each participant's role. Explain the individual data items being collected and how they will be used. Defuse the fear of measurement by stressing that individuals will not be evaluated on the basis of any software metrics data. Develop a handbook and web site with detailed definitions and procedures for each of the requested data items. Top-down communication from management should stress the need for data-driven decision-making, and the need to create a measurement-friendly culture.

Trap #10: Misinterpreting Metrics Data

Symptoms: A metric trend that jumps in an undesirable direction can stimulate a knee-jerk response to take some action to get the metric back on track. Conversely, metrics trends can reveal warning signs of serious problems that may be ignored by those who don't want to hear bad news. For example, if your defect densities increase despite quality improvement efforts, you might conclude the "improvements" are doing more harm than good, and be tempted to revert to old ways of working. In reality, improved testing might well find a larger fraction of the defects that are present—this is good!

Solutions: Monitor the trends that key metrics exhibit over time, and don't overreact to single data points. Make sure you understand the error bars around each of your measures, so you can tell whether a trend reversal is significant. If you can figure out why, say, your post-release bug correction effort has increased in two successive calendar quarters , you can decide if corrective action is required. Allow time for the data you're collecting to settle into trends, and make sure you understand what the data is telling you before you change your course of action.

Many of us struggle with how to implement a sensible metrics program that gives us the information we need to manage our projects and organizations more effectively. By staying alert to the ten risks described here, you can increase the chance of successfully implementing a software metrics initiative in your organization.