Metrics for Software Testing: Managing with Facts: Part 2: Process Metrics

Provided by Rex Black Consulting Services (www.rbcs-us.com)

Introduction

In the previous article in this series, I offered a number of general observations about metrics, illustrated with examples. We talked about the use of metrics to manage testing and quality with facts. We covered the proper development of metrics, top-down (objective-based) not bottom-up (tools-based). We looked at how to recognize a good set of metrics.

In the next three articles in the series, we'll look at specific types of metrics. In this article, we will take up process metrics. Process metrics can help us understand the quality capability of the software engineering process as well as the testing capability of the software testing process. Understanding these capabilities is a pre-requisite to rational, fact-driven process improvement decisions. In this article, you'll learn how to develop and understand good process metrics.

The Uses of Process Metrics

As I wrote above, process metrics help us understand process capability. They provide a means of measuring a process. As mentioned in the previous article, we can talk about metrics as relevant to effectiveness, efficiency, and elegance.

Effectiveness process metrics measure the extent to which a process produces a desired result. Efficiency process metrics measure the extent to which a process produces that desired result in a way that is not wasteful and, ideally, minimizes the resources used. Elegance process metrics measure the extent to which a process achieves effectiveness and efficiency in a graceful, well-executed fashion.

Process metrics provide practical insights about your current capabilities. By benchmarking your process metrics against industry norms, you can see where your test process stands compared with other companies. This can give you good ideas about which parts of the process to improve. It can also help you avoid trying to improve the parts of your process where your capabilities are already sufficiently good.

It's important to remember that process metrics measure process capabilities. It's a classic worst practice of metrics to assume that process metrics measure team or individual capabilities. Most of the factors that control a process's capabilities are under control of management, not under control of the team or individual carrying out that process. If such metrics are used to reward or punish teams or individuals, then they will often find ways to distort the metrics in order to maximize rewards or minimize punishments. You will then lose the ability to truly measure the process, and thus lose the opportunity to identify ways to improve the process.

Developing Good Process Metrics

In the previous article, I defined a process for defining good metrics. Let's review that process here, with an emphasis on process metrics:

- Define objectives for the process.
- Consider questions about the effectiveness, efficiency, and elegance with which the process realizes those objectives.
- Devise measurable metrics, either direct or surrogate, for each effectiveness, efficiency, and elegance question.
- Determine realistic goals for each metric, based on an understanding of industry best practices and current process capability.
- As appropriate, implement process improvements that increase effectiveness, efficiency, or elegance as measured by the metrics.

The typical objectives for the test process vary, but often include finding bugs and especially finding important bugs. Let's use these as examples to illustrate the process.

Let's start by developing a metric that evaluates the test process's effectiveness at accomplishing its bug finding objective. Since we can think of testing as a bug filter, a good effectiveness question is: "What percentage of the bugs present in a system during testing are found by the testing process?". The metric we can use to answer this question is called either defect detection effectiveness (DDE) or defect detection percentage (DDP). I prefer defect detection effectiveness, so I'll use that name here.[1]

The general formula for defect detection effectiveness, for any testing activity at any point in the software development lifecycle, is as follows:

$$DDE = \frac{defects\ detected}{defects\ present}$$

Usually, the denominator (*defects present*) is approximated by counting all defects found in and subsequent to the testing activity in question. This leaves out any defects which are never detected. Such defects might matter in terms of intrinsic code quality, but testing (which evaluates system behavior) can't detect such defects directly. Therefore, defects which never result in improper behavior don't matter in terms of test process effectiveness.

[1] I use the words *bug* and *defect* synonymously. I prefer the word *bug*, but use the phrase *defect detection effectiveness* for this metric due to the terms widespread usage, as opposed to *bug detection effectiveness*, which I've never heard in use.

So, for testing at the final stage of the lifecycle, prior to release, we can calculate defect detection effectiveness as follows:

$$DDE\ (final\ test) = \frac{test\ defects}{test\ defects + production\ defects}$$

Because defects typically are measured during system development (at least during the formal testing stages) and after a system is put into production, this metric is measurable.

With our metric in place, let's move on to setting goals. Based on our assessments of a number of clients around the world, typical defect detection effectiveness is 85%. Some teams do much better, though no test process finds 100% of the bugs.

If you measure your defect detection effectiveness and find it below 85%, then you should move to the next step of the process given above, instituting test process improvements. (Of course, you might want to improve the effectiveness even if it is above 85%.) You should analyze the causes for ineffectiveness at finding defects first, then develop a process improvement plan that addresses those causes. As I noted above, do not assume that the process problems arise from individual or team capability problems, but rather use data to check for the true causes.

Before we move on to the next example metric, you might have noticed that this first example process metric shares a similar underlying objective as the project metric example shown in the previous article. There certainly can be and often is overlap between the objectives that underlie process metrics, project metrics, and product metrics, but often the specific metrics and measures will differ, along with the timescales and way of presenting the metrics. This example illustrated those differences.

Let's move on to developing a metric for the other objective, the proper focus of bug finding, which is finding important bugs. In terms of our effectiveness at doing so, we can ask, "Do we find more important bugs than bugs which are less important?" The metric we can use to answer this question involves using our defect detection effectiveness (DDE) metric again. We simply calculate the defect detection effectiveness for all bugs and for only important bugs (however we make that distinction), and check the relationship between them as follow:

$$DDE\ (important\ bugs) > DDE\ (all\ bugs)$$

With our metric in place, what should our goal be? Since we want to focus on important bugs, the percentage of more important defects we find should be greater than the overall percentage of defects we find.

Based on our assessments of a number of clients around the world, it is often the case that this metric in fact shows that the defect detection effectiveness for all bugs is higher than that of important bugs. This is backwards, and can occur for various reasons. However, it often occurs when the test process does not use a proper risk based testing strategy as part of its approach. Investigating the causes for this imbalance, and, if appropriate, instituting risk based testing, can be a good process improvement to resolve this problem.[2]

Let me make a final observation about this metric. You might ask why the metric checks the relationship, rather than the difference in percentages. Consider the following metric, the defect detection difference (DDD):

$$DDD = DDE\ (important\ bugs) - \ DDE\ (all\ bugs)$$

This might seem to make sense, because we could try to maximize the defect detection difference to increase our focus on important bugs. The problem is that metrics influence behavior, often in unexpected and undesirable ways. Testers might choose to increase the defect detection difference by ignoring (or at least not reporting) less-important bugs, but that would interfere with another important goal of the testing process, which is to produce useful information. Though reports on less-important bugs are, of course, less important, the project team is still better off knowing about such bugs.

Note that defect detection effectiveness is a direct metric, not a surrogate metric. We are interested in percentage of defects found during testing, and the emphasis on finding important bugs, and that's what we're measuring. Remember from the previous article that a surrogate metric is one that measures something related to the area of interest. Since we can find a direct metric which is easily measurable in this case, we need not develop a surrogate metric.

Understanding Trade-offs in Process Metrics

Process metrics are useful to decide on process improvements, and initially most organizations find they can improve their process in a way that pushes all their process metrics upwards, towards some desirable level. However, at some point in the process improvement effort, effectiveness, efficiency, and elegance become trade-offs. In other words, to increase effectiveness, efficiency, or elegance beyond a certain point, some less-important attributes must be reduced.

Let me illustrate with an example. I talked earlier about defect detection effectiveness as a useful metric. I mentioned that 85% is a typical industry average for this metric. While we find that many of our clients can increase this metric to 95% without any reduction in efficiency, increasing defect detection effectiveness beyond 95% often increases the cost per bug found. Many organizations want testing to be effective at finding bugs, but not if the cost per bug found starts to climb beyond some threshold of value per bug found. Only your organization can determine where that threshold is, but generally we can say that, once the cost per bug found in testing starts to approach the cost per bug found in production, the economic benefits of testing start to decline. Therefore, carefully balancing of

[2] To learn more about risk based testing strategies, you can read my article "A Case Study in Risk Based Testing," at www.rbcs-us.com/images/documents/A-Case-Study-in-Risk-Based-Testing.pdf or view the series of video on risk based testing at www.rbcs-us.com/software-testing-resources/library/digital-library.

the effectiveness of testing against the efficiency of testing, in terms of bugs found and their relative costs, is important.[3]

How Test Process Metrics Can Measure the Software Process

The two examples of test process metrics that we saw earlier in this article measure the test process capability. However, test process metrics can also measure the overall software process. Let's look at two examples.

First, let's consider the time required for the entire bug lifecycle, from discovery to resolution. Generally, the longer a bug report takes to go from discovery to resolution, the higher the level of project risk associated with the underlying bug. Unresolved bugs may make testing and development less efficient, delay testing (by blocking other tests), and in some cases even prevent the delivery of the software to customers. So, we should strive to resolve bugs as quick as possible after they are reported.

A good metric for bug resolution is the defect closure period (DCP). For each bug report, we can calculate the closure period as follows:

$$DCP = date\ (discovery) - \ date\ (resolution)$$

This metric gives the defect closure period in days. It measures the time required to repair a bug and the time required to confirmation test the bug repair.

While the metric is interesting for individual bug reports, I have found it more productive to look at two metrics derived from this metric. The first is the daily closure period. This is the average defect closure period for all bugs resolved on a given day. The second is the rolling closure period, which is the average defect closure period for all bugs resolved on or before a given day. A graph of these two metrics for an example project is shown in Figure 1.

[3] For more information about how to calculate the costs of bugs found in testing versus bugs found in production, you can see my article "Testing ROI" which you can find at www.rbcs-us.com/images/documents/What-IT-Managers-Should-Know-about-Testing-ROI.pdf.

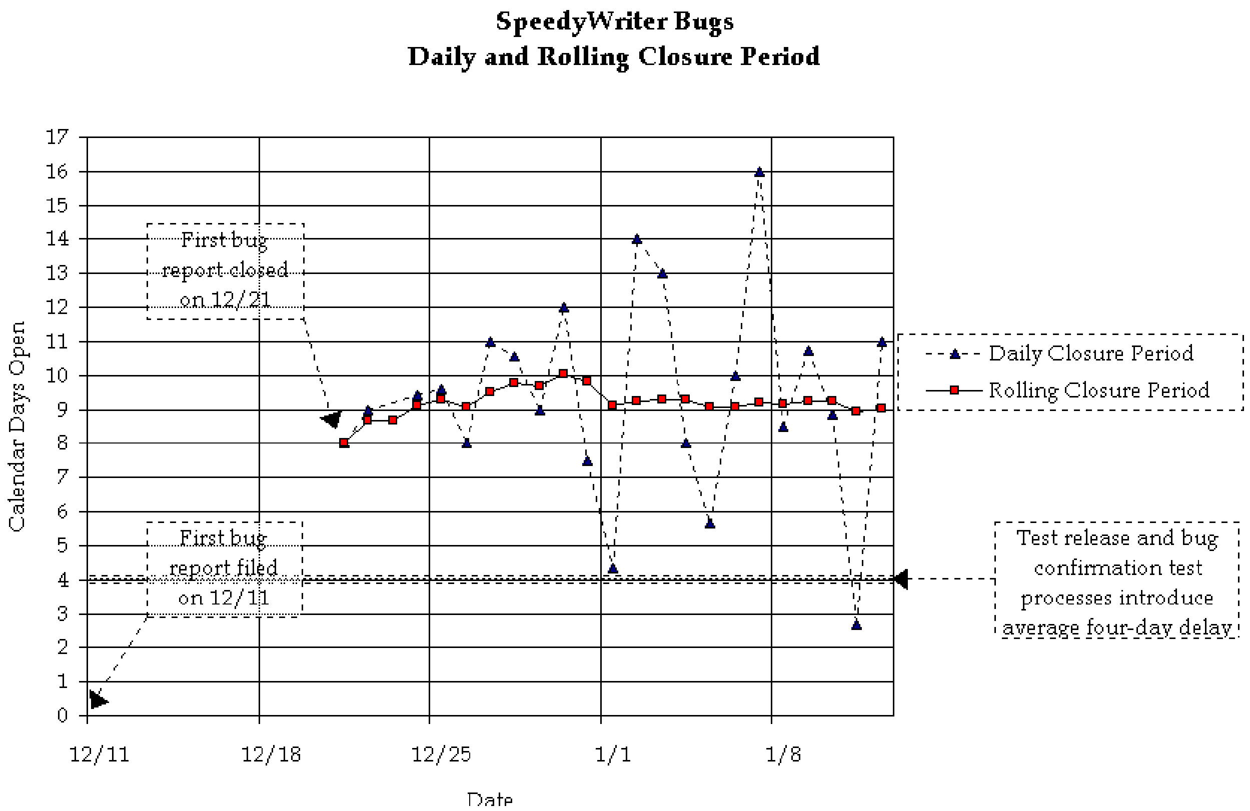

Figure 1: Daily and Rolling Closure Period

What we want to see is a process that is both stable and acceptable. A stable closure period shows a relatively low variance from one day to another, with the slope of the rolling closure curve remaining almost constant and close to zero. In addition, to the extent that the daily closure period changes, it fluctuates randomly around the rolling closure curve, staying within a few days in either direction. For an acceptable closure period, both the daily closure period and rolling closure period curves fall within acceptable upper and lower limits. These should be set in the project plan or test plan for bug turnaround time.

Although the pressures of the typical project might make it hard to believe, there is indeed a lower limit for an acceptable closure period. Bugs deferred the day they are opened pull the daily closure curve toward zero, but the bug remains in the product. Bugs fixed too quickly tend to be fixed poorly. I audited a project once where the test team took test releases two or three times a day for bugs identified just hours before. Although the closure period was very quick (a little over a day), that project had a significant number of bug reports that had been reopened multiple times, one of them 10 times.

On that note, let's look at another test process metric that can tell us about the success rate of bug resolution for the development process. This metric, the bug opened count, tracks the number of times each bug report is opened. The count is set to one (1) when the report is first submitted, and incremented each time (if any) the report is reopened due to a failure of the confirmation test of the supposed bug fix. (Note that the bug tracking system must be configured to gather this data.) Ideally, each bug report is opened only once, and fixed the first time development attempts to fix it. Any reopen of the bug report is a failure to fix the bug properly, which results in inefficiency and a project risk of schedule delay.

Figure 2 shows the histogram of this data for the audited project mentioned earlier. The x-axis shows the count of the number of bug reports opened a given number of times, with the number of times shown on the y-axis. The x-axis is on a logarithmic scale to make the count of reopened bug reports more visible.

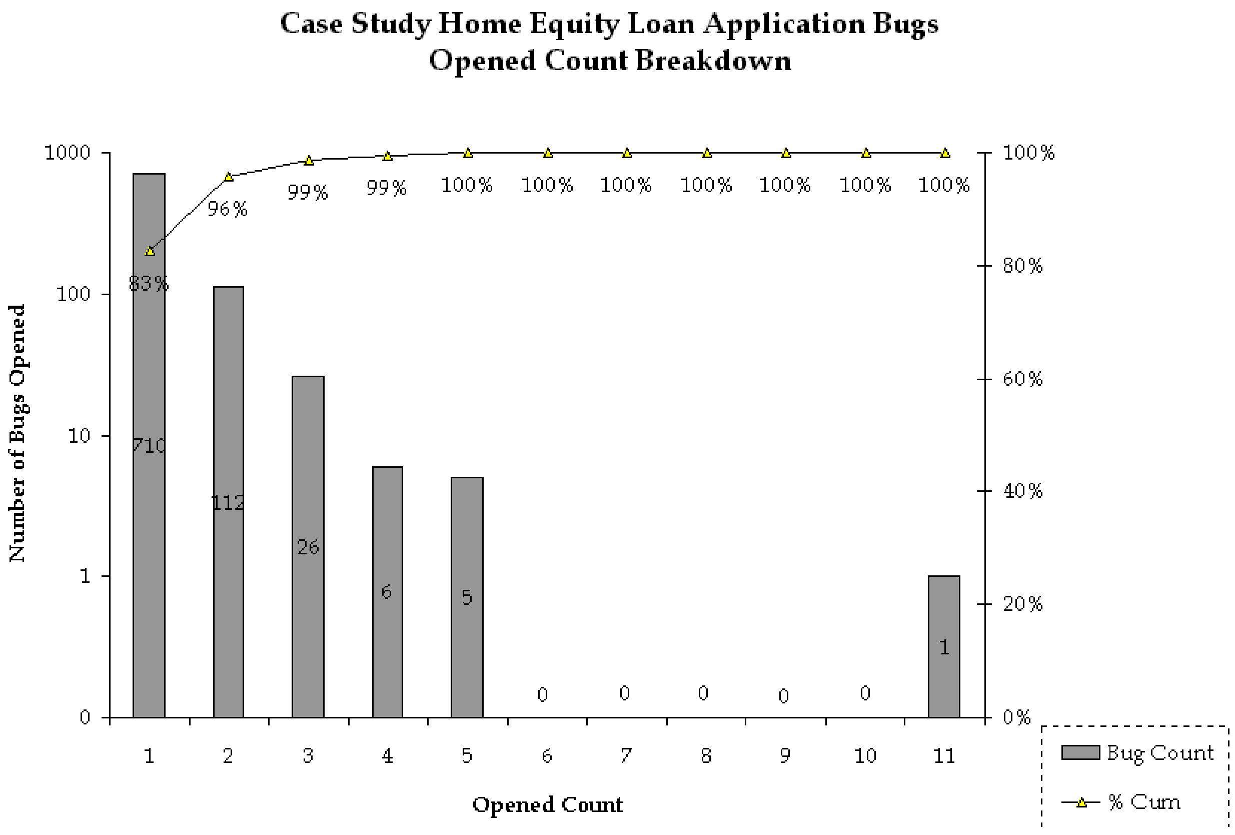

Figure 2: Bug Fix Inefficiency Due to Reopen

In Figure 2, you can see that 17% of the bug reports failed the confirmation test at least once and were reopened, since only 83% of the bug reports had an opened count of one (1). If we assume that each additional confirmation test and regression test associated with each bug fix required one (1) person-hour of effort, we can calculate the inefficiency associated with the rework as follows:

$$Rework\ Inefficiency = 1 \times 112 + 2\ \times 26 + 3\ \times 6 + 4\ \times 5 + 10\ \times 1 = 212$$

This figure, 212 person-hours, is quite significant, since the original test effort was scheduled for ten (10) people to last six (6) weeks. In other words, almost nine percent (9%) of the planned test effort was consumed by an unplanned inefficiency related to problems in the software development process.[4]

Process Metrics in Test Dashboards and Assessments

So far in this article, I've shown you a number of ways to use test process metrics to understand test process capability and software development process capability. As such, you might imagine that such metrics would figure prominent in any testing dashboard or assessment. However, they typically do not.

Certainly test process metrics can be useful in test and even project dashboards. Imagine the use of the chart shown in Figure 2 during projects to monitor for unexpectedly delays and escalating project risks associated with failed bug fixes. Many times, having clear insight into the test process's operations and capabilities is useful and enlightening, but it's surprising how few organizations do so.

The same with various test process assessments. While some test assessments do rely on metrics, most do not. RBCS' own Critical Testing Process assessment includes interviews and a complete set of metrics to evaluate the test process, but other test assessments and maturity measures rely primarily on subjective interviews to check whether certain practices are carried out without evaluating how effectively, efficiently, or elegantly they are carried out. This is inadequate. Consider, for example, a test team that has a good bug tracking tool and which follows a proper process for bug management, but which has a very low defect detection effectiveness. Without metrics to evaluate the effectiveness of the bug management process, an assessor might conclude that the process is effective.[5] As I said in the first article in this series, there is no understanding, and no basis for rational decisions, without metrics.

Moving on to the Project

In this article, I discussed the use of process metrics. I've illustrated the application of the metrics development process discussed in the previous article, as applied to process metrics. Using examples of process metrics from industrial use, we've seen how these metrics allow us to understand the capabilities of our testing process, and the capabilities of the software development process, too. We've discussed the use and potential misuse of process metrics.

In the next two articles in the series, we'll look at two more specific types of metrics. Next, we'll move on to project metrics. These are the most commonly used testing metrics, but they are often misunderstood and misused. The next article will give you some ideas on how to use test project metrics properly. See you in the next issue!

---

4 Both of these metrics are discussed further in my book *Managing the Testing Process, 3e*, and supporting spreadsheets to produce these graphs are found at www.rbcs-us.com/software-testing-resources/library.

5 See www.rbcs-us.com/test-assessment for more details on the Critical Testing Processes assessment model, or check out my book of the same name.