Measuring the Benefits of Peer Review

Collaborator

by SMARTBEAR

It's no secret that peer reviews of code and artifacts can improve product quality and team skill levels. What's less apparent is how you can quantitatively measure the benefit you derive from adhering to the practice. This e-book will walk you through the kinds of measurements that can be truly useful in calculating the derived benefit from peer review, as well as give some practical advice on how to use that data to successfully implement peer review in your organization.

This information is adapted from Karl Weigers' book *Peer Reviews in Software* (Addison-Wesley, 2002),

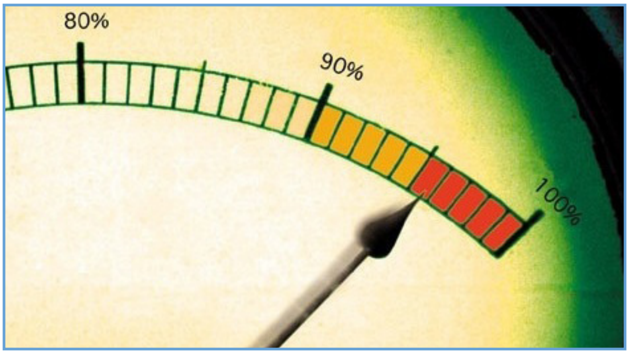
How Peer Review Metrics Create a Measurement Culture 3
- Why Collect Data? 4
- Some Measurement Caveats 6

Which Peer Review Metrics are Worth Collecting? 8
- Base Metrics and Derived Metrics 8
- Suggested Peer Review Derived Metrics 10
- The Peer Reviews Database 11

Getting the Absolute Most out of your Peer Review Data 12

Data Analysis 13
- Scatter Charts 13
- Statistical Process Control 15
- Measuring the Impact of Inspections 16
- Effectiveness 16
- Efficiency 18
- Why a Tool is the Easy and (Right Way) to Get There 19

How Peer Review Metrics Create a Measurement Culture



My friend Nick was a quality manager at a company that is respected for its superior software development and quality practices. Nick once said to me, “We only found two major defects in our latest code inspection, but we expected to find between four and six. We’re trying to figure out what’s going on. Did we miss some, or was this code particularly clean for some reason?”

Few organizations can make such precise statements about the quality of their software products. Nick’s organization has stable, repeatable development processes in place, and they’ve accumulated inspection data for several years. Analyzing historical data lets Nick predict the likely defect density in a given deliverable. When a specific inspection’s results depart significantly from the norm, Nick can ask probing questions to understand why. Did the inspectors prepare at the optimum rate, based on the organization’s experience? Did they use suitable analysis techniques? Were they adequately trained and experienced in inspection? Was the author more or less experienced than average? Was the product more or less complex than average? You can’t reach this depth of understanding without data.

It doesn’t take as much effort as you might fear to collect, analyze and interpret metrics from your peer reviews. It’s more a matter of establishing a bit of infrastructure to store the data, then making it a habit for review participants to record just a few numbers from each review experience. In fact, I think that peer review metrics provide an easy way to begin growing a measurement culture in your organization.

Why Collect Data?

Recording data about the review process and product quality is a distinguishing characteristic of formal peer reviews, such as the type of rigorous peer reviews called inspections. Data answers important questions, provides quantifiable insights and historical perspective, and lets you base decisions on facts instead of perceptions, memories or opinions.

For example, one organization learned that it could inspect requirements specifications written by experienced business analysts twice as fast as those written by novices because they contained fewer defects. This data revealed the need to train and mentor novice BAs. Another organization improved its development process by studying data on defect injection rates and the types of defects their inspections did not catch. This example illustrates the value of recording the life-cycle activities during which each defect is created and discovered.

One way to choose appropriate metrics is the Goal-Question-Metric, or GQM, technique.

1. First, state your business or technical goals.
2. Next, identify *questions* you need to answer to tell if you are reaching those goals.
3. Finally, select *metrics* that will let you answer those questions.

One goal might be to reduce your rework costs through peer reviews. Answers to the following questions could help you judge whether you're reaching that worthy goal:

- What percentage of each project's development effort is spent on rework?
- How much effort do our reviews consume? How much do they save?
- How many defects do we discover by review? What kind? How severe? At what life-cycle stage?
- What percentage of the defects in our products do our reviews remove?
- Do we spend less time testing, debugging, and maintaining products that we reviewed than those we did not?

Figure 1 (below) shows a progression of benefits that peer review measurements can provide to your organization. The individual base metrics you begin collecting from each review don't tell you much by themselves. Tracking some derived *metrics* calculated from those base *metrics*—often as simple sums or ratios—reveals averages and trends of your team's preparation and inspection rates, defect densities, inspection effectiveness, and other parameters.

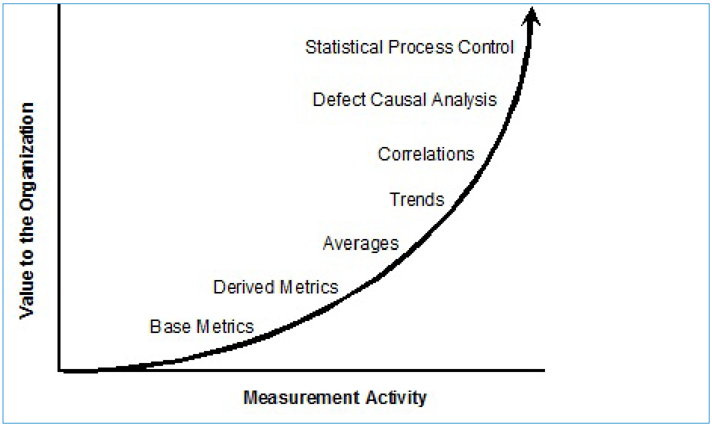


Figure 1. The value of various peer review measurement analyses

These trends help you detect anomalous inspection results, provided that your development and inspection processes are consistent. They also help you estimate how many additional defects you can expect to find in the remaining life-cycle phases or in operation. Correlations between pairs of metrics help you understand, say, how increasing the preparation rate affects inspection efficiency. For the maximum rewards, use defect causal analysis and statistical process control to guide improvements in your development processes and quality management methods. It takes a while to reach this stage of peer review measurement sophistication.

Don't make your measurement process so elaborate that it inhibits the reviews themselves. Establishing a peer review culture and finding defects is more important than meticulously recording masses of data. One of my groups routinely conducted various types of peer reviews, recorded numerous base metrics, and tracked several derived metrics. The entire team recognized the value we obtained from the reviews; hence, they became an ingrained component of our software engineering culture.

Some Measurement Caveats

Software measurement is a sensitive subject. It's important to be honest and nonjudgmental about metrics. Data is neither good nor bad, so a manager must neither reward nor punish individuals for their metrics results. The first time a team member is penalized for some data he reported is the last time that person will submit accurate data. Defects found prior to peer review should remain private to the author. Information about defects found in a specific peer review should be shared only with the project team, not with its managers. You can aggregate the data from multiple reviews to monitor averages and trends in your peer review process without compromising the privacy of individual authors. The project manager should share aggregated data with the rest of the team so they see the insights the data can provide and recognize the peer review benefits.

Beware the phenomenon known as *measurement dysfunction*. Measurement dysfunction arises when the measurement process or the ways in which managers use the data lead to counterproductive behaviors by the people providing the data. People behave in the ways for which they are rewarded; they usually avoid behaviors that could have unpleasant consequences. Some forms of measurement dysfunction that can arise from peer reviews are: inflating or deflating defect severities; marking as closed defects that really aren't resolved; and distorting defect densities, preparation times, and defect discovery rates to look more favorable.

There's a natural tension between a work product author's desire to create defect-free products and the reviewers' desire to find lots of bugs. Evaluating either authors or reviewers according to the number of defects found during a review will lead to conflict. If you rate reviewers based on how many defects they find, they'll report many defects, even if it means arguing with the author about whether every small issue truly is a defect. It's not necessary to know who identified each defect or to count how many each reviewer found. What is important is that all team members participate constructively in peer reviews. Help managers avoid the temptation to misuse the data for individual performance evaluation by not making individual defect data available to them.

It's tempting to overanalyze the review data. Avoid trying to draw significant conclusions from data collected shortly after launching your peer review program. There's a definite learning curve as software people beginning participating in systematic reviews and figure out how to do them effectively and constructively. If you begin tracking a new metric, give it time to stabilize and make sure you're getting reliable data before jumping to any conclusions. The trends you observe are more significant than any single data point.

Now that we have a basic foundation of peer review metrics principles, the next sections will get into some specific metrics to track and how to analyze the data.

Which Peer Review Metrics are Worth Collecting and Calculating?

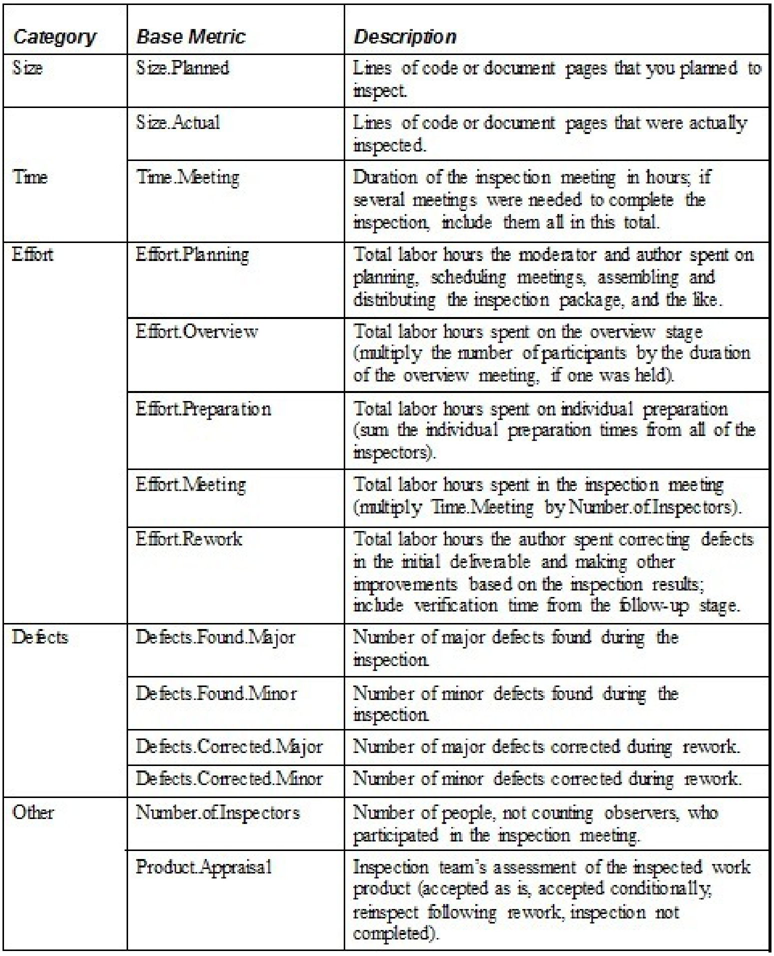
The first section provided some basic concepts and principles about measuring aspects of your peer review program. This section recommends several base metrics to count or measure directly, as well as a number of *derived metrics* that you can calculate from those base metrics to see what's really happening in your peer review program.

Base Metrics and Derived Metrics

The basic dimensions of software measurement are size, time, effort, and quality. Although you could measure dozens of peer review data items in these categories, the base metrics listed below will give you a solid start on review measurement.

(Note that the terminology in the metrics tables refers to inspections instead of the more generic peer reviews. Measurement like this is most common when performing those formal peer reviews termed inspections.)

If you're using standard forms for recording peer review results, include spaces on the forms to capture these values. Whenever possible, use tools to count objects such as lines of code consistently, according to your organization's conventions.

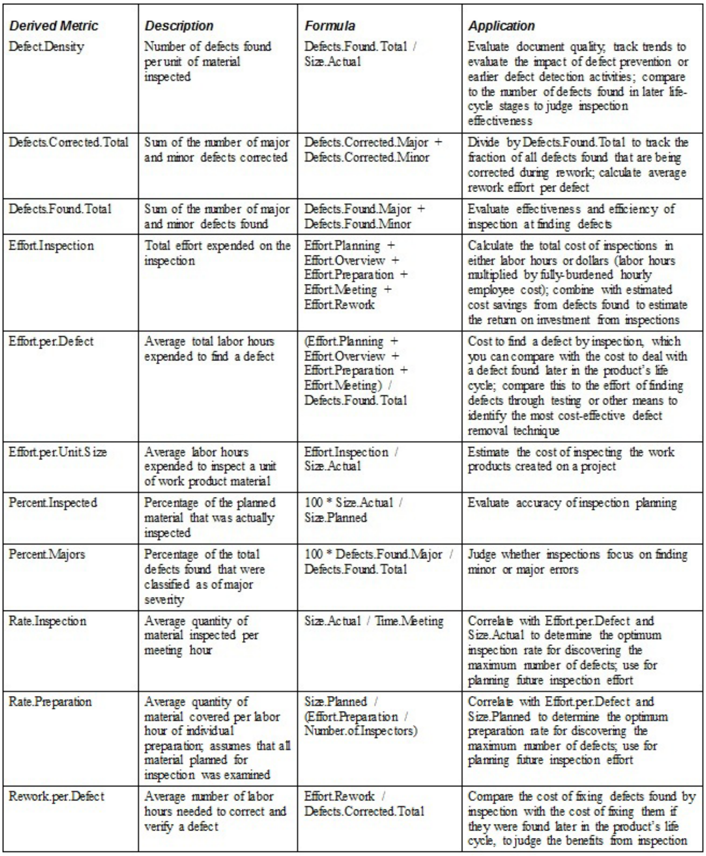


Category	*Base Metric*	*Description*
Size	Size.Planned	Lines of code or document pages that you planned to inspect.
	Size.Actual	Lines of code or document pages that were actually inspected.
Time	Time.Meeting	Duration of the inspection meeting in hours; if several meetings were needed to complete the inspection, include them all in this total.
Effort	Effort.Planning	Total labor hours the moderator and author spent on planning, scheduling meetings, assembling and distributing the inspection package, and the like.
	Effort.Overview	Total labor hours spent on the overview stage (multiply the number of participants by the duration of the overview meeting, if one was held).
	Effort.Preparation	Total labor hours spent on individual preparation (sum the individual preparation times from all of the inspectors).
	Effort.Meeting	Total labor hours spent in the inspection meeting (multiply Time.Meeting by Number.of.Inspectors).
	Effort.Rework	Total labor hours the author spent correcting defects in the initial deliverable and making other improvements based on the inspection results; include verification time from the follow-up stage.
Defects	Defects.Found.Major	Number of major defects found during the inspection.
	Defects.Found.Minor	Number of minor defects found during the inspection.
	Defects.Corrected.Major	Number of major defects corrected during rework.
	Defects.Corrected.Minor	Number of minor defects corrected during rework.
Other	Number.of.Inspectors	Number of people, not counting observers, who participated in the inspection meeting.
	Product.Appraisal	Inspection team's assessment of the inspected work product (accepted as is, accepted conditionally, reinspect following rework, inspection not completed).

As you gain experience, you might elect to subdivide some of these items. You could separate rework effort from follow-up effort; distinguish reviews of new, modified, and reused code, or separate the metrics for major and minor defects. You should only increase the measurement complexity when you have specific questions that require more detailed data. I recommend you begin by collecting all the items listed in the table above. Not only is the effort needed to capture and store this information relatively small, but it's impossible to reconstruct the data if you decide later that you want it.

You can calculate several derived metrics from the data items in the table above that will give you insight into your peer review process. The next table describes several derived metrics for reviews and shows how to calculate them from the base metrics. The computations are very simple.

Suggested Peer Review Derived Metrics



Derived Metric	***Description***	***Formula***	***Application***
Defect.Density	Number of defects found per unit of material inspected	Defects.Found.Total / Size.Actual	Evaluate document quality, track trends to evaluate the impact of defect prevention or earlier defect detection activities; compare to the number of defects found in later life-cycle stages to judge inspection effectiveness
Defects.Corrected.Total	Sum of the number of major and minor defects corrected	Defects.Corrected.Major + Defects.Corrected.Minor	Divide by Defects.Found.Total to track the fraction of all defects found that are being corrected during rework; calculate average rework effort per defect
Defects.Found.Total	Sum of the number of major and minor defects found	Defects.Found.Major + Defects.Found.Minor	Evaluate effectiveness and efficiency of inspection at finding defects
Effort.Inspection	Total effort expended on the inspection	Effort.Planning + Effort.Overview + Effort.Preparation + Effort.Meeting + Effort.Rework	Calculate the total cost of inspections in either labor hours or dollars (labor hours multiplied by fully-burdened hourly employee cost); combine with estimated cost savings from defects found to estimate the return on investment from inspections
Effort.per.Defect	Average total labor hours expended to find a defect	(Effort.Planning + Effort.Overview + Effort.Preparation + Effort.Meeting) / Defects.Found.Total	Cost to find a defect by inspection, which you can compare with the cost to deal with a defect found later in the product's life cycle; compare this to the effort of finding defects through testing or other means to identify the most cost-effective defect removal technique
Effort.per.Unit.Size	Average labor hours expended to inspect a unit of work product material	Effort.Inspection / Size.Actual	Estimate the cost of inspecting the work products created on a project
Percent.Inspected	Percentage of the planned material that was actually inspected	100 * Size.Actual / Size.Planned	Evaluate accuracy of inspection planning
Percent.Majors	Percentage of the total defects found that were classified as of major severity	100 * Defects.Found.Major / Defects.Found.Total	Judge whether inspections focus on finding minor or major errors
Rate.Inspection	Average quantity of material inspected per meeting hour	Size.Actual / Time.Meeting	Correlate with Effort.per.Defect and Size.Actual to determine the optimum inspection rate for discovering the maximum number of defects; use for planning future inspection effort
Rate.Preparation	Average quantity of material covered per labor hour of individual preparation; assumes that all material planned for inspection was examined	Size.Planned / (Effort.Preparation / Number.of.Inspectors)	Correlate with Effort.per.Defect and Size.Planned to determine the optimum preparation rate for discovering the maximum number of defects; use for planning future inspection effort
Rework.per.Defect	Average number of labor hours needed to correct and verify a defect	Effort.Rework / Defects.Corrected.Total	Compare the cost of fixing defects found by inspection with the cost of fixing them if they were found later in the product's life cycle, to judge the benefits from inspection

The Peer Reviews Database

You'll need a repository in which to store your review data, along with query, reporting and charting tools to monitor averages and trends in the metrics for a series of inspections. To get started quickly, use the Excel spreadsheets available from processimpact.com. The spreadsheet is not a robust database, just a simple tool to help you begin accumulating and analyzing the data easily.

The inspection data spreadsheet contains three linked worksheets that accommodate the base metrics from *Table 1* and the derived metrics from *Table 2*.

- The **Inspection Info** worksheet contains descriptive information about each inspection, the data items in the "Other" category and the major calculated metrics.
- Enter the time and effort base metrics into the **Effort** worksheet.
- The defect counts in various categories go into the **Defects** worksheet, along with the number of defects the author corrected during rework.
- A **Help Info** worksheet contains instructions for using the spreadsheet and inserting new rows by using a macro.

This approach stores the data for individual inspections in reverse chronological order, with the most recent entries at the top. You can modify the spreadsheet if, for example, you wish to focus only on major defects or to exploit the charting and statistical analysis features of Excel.

The units used for measuring work product size are different for code and for other documents. Therefore, you'll need separate spreadsheets for each major type of work product being inspected: source code, requirements specifications, design documents, test cases and so on. The website contains one spreadsheet for code inspections, another for inspections of other documents, and a third with some sample code inspection data so you can see how it all works.

Now that you know what metrics are worth collecting and calculating, the final article in this series will discuss how to analyze your peer review data. I'll also show how you can use the metrics to judge the effectiveness and efficiency of your organization's peer reviews.

Getting the Absolute Most Out of Your Peer Review Data

In the first two sections, I discussed some [concepts and principles about measuring peer reviews,] and identified a number of [base and derived metrics] to calculate from each review. This final section describes how to analyze all of that peer review data.

One aspect of benefit is *effectiveness*, or yield: The percentage of the defects present in a work product that your reviews discover. You'd like this to be close to 100%. Another aspect is *efficiency*: the effort required to find a defect by peer review. Less effort per defect indicates higher efficiency. As before, the discussion below refers to inspections as the type of formal peer reviews from which data typically is collected.

Scatter Charts

As you accumulate data from multiple inspections of a particular type of work product, the spreadsheet I described in the previous article will calculate the averages of the metrics. You can then use scatter charts to look for correlations between pairs of metrics. The spreadsheets do not contain any scatter charts, but it wouldn't be difficult to add those that you find informative.

Figure 1 shows a sample scatter chart. This chart plots **Effort.per. Defect** (average labor hours of review needed to find one defect) against **Rate.Preparation** (average lines of code per hour studied during individual preparation) for a set of code inspections. Each data point represents a separate inspection. For example, the highlighted point in Figure 1 marks a inspection that had a preparation rate of 100 lines per hour and found one defect for every half hour of preparation time.

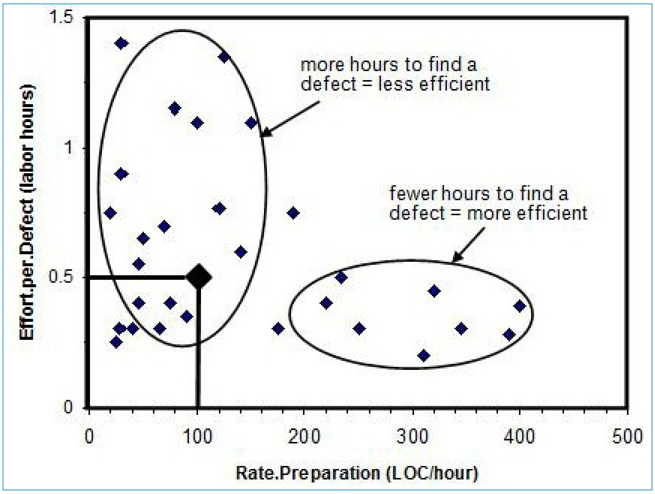


Figure 1. Sample scatter chart of two inspection metrics

This chart shows that inspections having preparation rates slower than about 200 lines of code per hour are less efficient than those with higher preparation rates. That is, it takes more hours of work, on average, to find each defect if the inspectors prepare slowly. This doesn't necessarily mean that you should race through the material to try to boost efficiency, though.

In figure 2, A plot of **Defect.Density** (the number of defects found per thousand lines of code) versus **Rate.Inspection** shows that *effectiveness* decreases with higher inspection rates *(Figure 2)*. That is, the faster you go, the fewer defects you find. (Or, is that discovering more defects forces you to cover the material more slowly?) This type of data analysis helps you judge the optimum balance between efficiency and effectiveness.

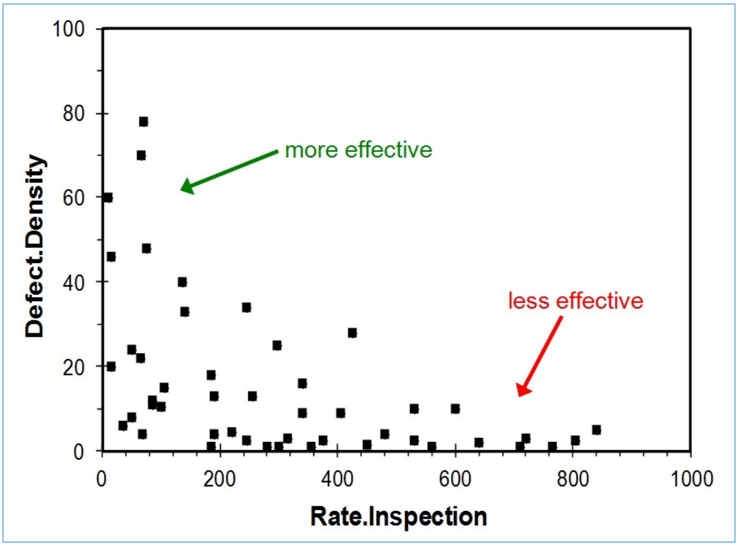


Figure 2. Relationship between defect density and code inspection rate

Statistical Process Control

As your organization establishes repeatable development and inspection processes, you can use statistical process control (SPC) to monitor key inspection parameters. SPC is a set of analytical techniques for measuring the stability of a process and identifying when an individual performance of the process falls outside the expected range of variation, or control limits. Points that lie beyond the control limits stimulate an inquiry to understand why. The historical data trends from a process that is in control facilitate predicting future outcomes of the process. Inspection metrics that are amenable to SPC include **Rate.Preparation**, **Rate. Inspection**, **Size.Actual**, and **Defect.Density**.

As a simple illustration of SPC, *Figure 3* depicts a *control chart* containing preparation rates (lines of code per hour) from 25 code inspections. The control chart plots specific data (**Rate.Preparation**) from the process being observed (inspection preparation) versus the set of observations (inspection number). The average preparation rate was 240 lines of code per hour, shown with the solid horizontal line. The *upper control limit*, shown as the dashed line at 440 LOC/hour, is a statistic that attempts to discriminate normal variation or noise in the process from variations that can be attributed to some assignable cause.

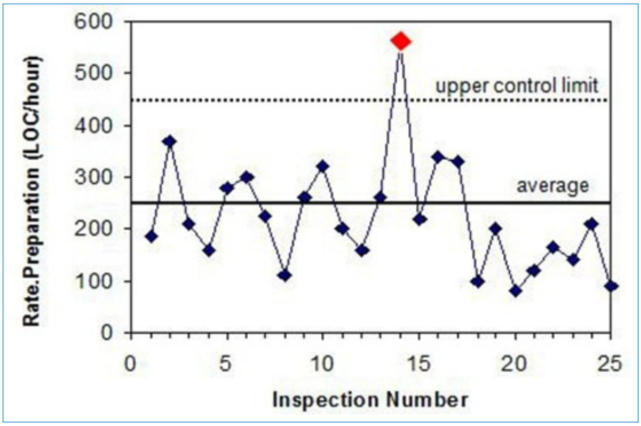


Figure 3. Sample inspection data control chart

Figure 3 shows that the preparation rate for inspection 14 (shown in red) was an abnormally high 560 LOC/hour, which lies well beyond the upper control limit. Because inspection 14 departed from the expected preparation rate range, its results should be viewed with caution. Did inspectors cut their preparation shorter than the organization's historical data recommends for optimum defect discovery? Was there something unusual about the work product—particularly straightforward or clean code, reused code—that justified the increase in preparation rate? Was the inspection team particularly experienced and efficient?

Correlating defect densities with preparation rates for a repeatable process lets you judge whether a code component that had an abnormally high preparation rate might suffer future quality problems because of residual defects. This is the kind of data analysis that allowed my friend Nick, the quality manager mentioned in the first article in this series, to predict how many major defects he expected to find when inspecting a new code module.

Measuring the Impact of Inspections

Participants can subjectively judge the value of inspections. Sometimes, though, a more quantitative benefit analysis is desired. Ideally, you will be able to demonstrate that inspections are saving your project team, company, or customers more time than they consume.

Effectiveness

Inspection effectiveness is a lagging indicator: you can't measure it at the time of the inspection, only later. Calculating effectiveness requires that you know both how many defects your inspections discover and how many were found later in development, during testing, and by customers. Defects found by customers within three to six months following product release provide a reasonable indication of how many significant defects eluded your quality nets. To illustrate an inspection effectiveness calculation, consider the following sample data for a single code module:

Defects found during code inspection:	7
Defects found during unit testing:	3
Defects found during system testing:	2
Defects found by customer:	1
Total defects initially present in the code:	13
Code inspection effectiveness:	100 * (7 / 13) = 54%

If you know your inspection effectiveness, you can estimate how many defects remain in a document following inspection. Suppose that your average effectiveness for inspecting requirements specifications is 60 percent. You found 16 major defects while inspecting a particular specification. Assuming that the average applies to this inspection, you can estimate that the document originally contained about 27 major defects, of which 11 remain to be discovered later. Without knowing your inspection effectiveness, you can't make any claims about the quality of a document following inspection.

Suppose you inspect just a small sample of a document. Combining your known inspection effectiveness, the document size, and the number of defects found in the sample lets you estimate the total defects in the rest of the document. This is simplistic because it assumes that the sample is representative of the whole document, which might not be true. To estimate the potential cost of those remaining defects, multiply the defect count estimate by your average cost of correcting defects found through testing or by customers. (If you don't know those average costs, begin collecting them now!) Compare that projected cost with the cost of inspecting the rest of the document to judge whether such inspection is warranted economically.

Using some typical numbers, the following example suggests that it would be cheaper to remove the remaining defects by inspection (90 labor hours) than by system testing (270 labor hours). This illustration assumes that testing and inspection find the same types of bugs, but that also might not be the case.

Defects found in a 2-page sample of 20 pages of code:
2

Estimated defects in the remaining 18 pages:
18

Effort spent inspecting the 2-page sample:
10 labor hours

Estimated effort to inspect the remaining 18 pages:
5 labor hours/page * 18 pages = 90 labor hours

Average effort to find and correct a defect in system test:
15 labor hours

Estimated effort needed to find and correct the remaining 18 defects by system testing:
15 * 18 = 270 labor hours

Efficiency

The more efficient your inspections are, the more defects they discover per hour of effort. Your efficiency will increase with experience. When you begin holding inspections, strive to maximize efficiency, which reduces the average cost of finding a defect.

There's a paradox here, though. A successful inspection program leads to process improvements that reduce the number of errors developers make. This leaves fewer defects in the deliverables to be discovered by inspection or testing. So, as your product quality improves, the cost of discovering each defect will *increase*, and the trends in your metrics will become harder to interpret. Monitor both inspection efficiency and effectiveness to understand whether a trend toward decreasing efficiency truly indicates higher product quality, or if it means your inspections are not working as well as they should be. You might reach a point where the increased cost of using inspections to hunt for the few defects present in the product exceeds the business risks of shipping the product with those defects still present.

Peer review measurement, like everything else you do on a project, is not free. If you're serious about quality, though, you'll find that the small investment you make in metrics is well repaid by the insights you gain. You'll learn how your peer review program is—and is not—working and how to improve it. And you'll be able to convince skeptics that peer reviews are indeed a valuable investment that improves both product quality and team productivity.

Why a Tool is the Easy (and Right) Way to Get There

If it seems daunting to capture and review all of these metrics, don't worry. Tool-assisted code review solves a lot of problems for you by capturing this data objectively, without the ability of any one team member skewing the metrics, and provides the most important derived metrics for you.

At SmartBear, we want to make sure that teams can adhere to this kind of best practice and rigorous measurement without loss of productivity or staff increases. We also recognize that what is best practice for code is also best practice for other artifacts in the software development cycle. That's why we made Collaborator by SmartBear agnostic when it comes to the type of artifact being reviewed. Whether you are engaging in peer review for code, requirements, test plans, user documentation, or any other kind of document, you can still capture and report on the most important metrics for measuring quality:

- Defect Density
- Defects found during peer review
- Peer inspection rate
- Peer inspection effectiveness

Quality begins at the beginning of the development cycle and your processes and metrics should reflect that. We believe our tools are the best in the world to help you achieve success at all stages of your development process, including peer review of any artifact.

About SmartBear Software

More than one million developers, testers and operations professionals use SmartBear tools to ensure the quality and performance of their APIs, desktop, mobile, Web and cloud-based applications. SmartBear products are easy to use and deploy, are affordable and available for trial at the website. Learn more about SmartBear, the company's award-winning tools or join the active user community at http://www.smartbear.com, on Facebook or follow us on Twitter @smartbear and Google+.

SMARTBEAR