Using Inspection Data to Forecast Test Defects

John T. Harding
Software Technology Transition

Some organizations have applied software inspections well but may not be using the data to improve the inspection process and to make business trade-offs based on the inspection data. This article describes how to use inspection data from code inspections to forecast the number of defects that remain in the product and how to forecast the number of defects that need to be removed during each test activity.

What Is the Difference Between an Inspection and a Peer Review?

Within the programming community, the term *peer review* can refer to one of many different types of reviews. Some are consistent, rigorous, and have associated data, but more often they are informal, their effectiveness is highly dependent on the people involved, and they rarely have associated data. To eliminate misunderstandings, this article will use the term *inspection* to refer to consistent and rigorous "peer reviews" that have associated data.

What Are Software Inspections?

An inspection is a review of a work product, led by a moderator who is not the producer, that seeks and records defects in that work product using standardized checklists and techniques. The inspection process initiates rework as necessary, initiates re-review, passes the work product based on exit criteria, and adds to the base of historical data.

What Is Needed to Get Good Data for Analysis?

As with any measurement, the data must be consistent and accurate or you will be comparing apples and oranges; however, this does not mean the data has to be precise. Typically, organizations at lower levels of process maturity cannot be as precise as organizations at higher maturity levels. Do not let this stop you from using the data. It is more important to understand the relationships and be in the right order of magnitude when you start to use the data than to focus on getting to the nth digit to the right of the decimal point. Merely focus on consistent data, which requires a consistent process, and accurate data, which requires understanding some common definitions.

Consistent processes require a definition of both the artifacts to be inspected and of the process to create them. This consistent process ensures that the inspected work products are similar. Most organizations evolve to rather than start at this point. Criteria for work products, including coding standards, design standards, guidelines, and templates, can help ensure that the various artifacts are similar, regardless of which person in the organization worked on the artifact.

The need for different people to review each other's artifacts during inspections helps ensure that these criteria and enablers for consistency evolve, especially with respect to design and coding standards that improve consistency and readability. This also requires that the inspection process be consistent, which may be the easiest aspect for many organizations since it is a "single" process. Some of the key aspects are training, project-specific checklists, criteria for reinspection, data definitions, scenarios, establishing project-specific inspection rates, and data-capture mechanisms.

Accurate data requires that some primary data elements be defined and commonly understood, especially the definitions of major defects, size, and time. Even after these elements are defined, people will still need to make value judgments to record data. These judgements are not likely to produce consistent, useful data until there is a common understanding of how the data will be analyzed and used. Most organizations have difficulty getting consensus on what a major defect is until people realize what trade-offs will be made based on the reported data.

If I Have a Consistent Inspection Process and Good Data, How Can I Start to Forecast the Number of Defects Test Needs to Find?

The following example illustrates how this could be done. If you have done some code inspections, the data might look as follows:

- Product contains 10,000 lines of code (LOC).
- Two thousand LOC were inspected.
- Fifty major defects were found in the inspections.

After examining the inspection rates, you determined that they were all reasonable and that the inspections appeared to be consistent. Although there is not one set of numbers applicable to every project, some relationships appear to be consistent across many different software organizations. For example, groups that are starting to do consistent inspections typically discover about 50 percent of defects present in the product. Rarely do they start out finding as many as 60 percent or as few as 40 percent. With this industry norm and the project-specific data, you can project the following:

- Fifty defects in 2,000 LOC is a defect density of 25 defects per thousand LOC (KLOC).
- Assuming 50 percent inspection effectiveness, the product defect density is 50 defects per KLOC.

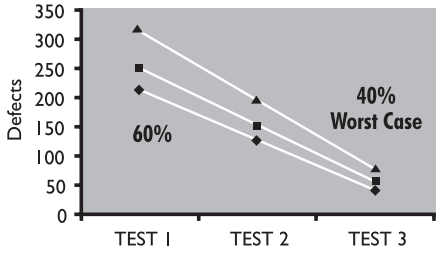

Figure 1. *Estimating remaining defects.*

- The product contains 50 defects per KLOC x 10 KLOC, or 500 defects.
- Fifty defects were removed during the inspections, so there are 450 defects remaining.

You now have a quantifiable target for the number of defects that should be found in test. There are, however, some additional calculations to perform. Since inspections on most projects are initially between 40 percent and 60 percent effective, you can redo the calculations for the 50 percent effectiveness for both of these limits to show a range of defects for each test stage, then expect the actual number to be somewhere in between (see Figure 1).

For the 40 percent calculation:

- Inspection defect density is 25 major defects per KLOC.
- Product defect density is 62.5 defects per KLOC (25/.4).
- There are 570 major defects remaining (10 KLOC x 62 defects per KLOC = 620 defects minus 50).

For the 60 percent calculation:

- Inspection defect density is 25 major defects per KLOC.
- Product defect density is 41.6 defects per KLOC (25/.6).
- There are 370 major defects remaining (10 KLOC x 42 defects per KLOC = 420 defects minus 50).

Now, when the test defect data comes in, you will better understand what questions need to be asked.

Since it is nearly always more cost-effective to find defects in the early testing activities, you probably want to find the largest number of defects in the first test stage and less in each subsequent testing stage. There are any number of techniques to do this, including various deprecation algorithms. You could also just do it manually, such as was done for Figure 1, just by drawing a straight line. As you are doing this, it is OK to round up or down—this is "blackboard" mathematics.

If there are roughly 450 defects remaining, you can decide how to remove them by comparing the time and cost to remove them using inspections with the time and cost to remove them using testing. Typically, inspections will be much more cost-effective. Therefore, you might decide to perform more inspections for the other 80 percent of the product that has not yet been inspected to reduce the number of defects going into test. If you choose to go directly to test, you could estimate how many defects you need to find during each test activity (see Figure 1).

Inspection Data from Additions and Modifications

If you are adding or modifying functions in an existing product, you will need to gather some additional data. When you inspect changes, you will typically have to examine some of the unchanged code to understand how the changed code works. This analysis will require that you count both the total LOC inspected and the new or modified LOC inspected. The major defects will also need to be collected at the same level of granularity, which helps you understand both the LOC and the defects associated with the added or modified functions. The example in this article assumes that the LOC and major defects are both associated exclusively with the count of new and changed lines of code.

Defect Density Considerations

There is one additional calculation you should consider. What if the code that was inspected is from the least defect-dense parts of the product? What if it is from the most defect-dense parts of the product? How many of your products have a constant defect density across all parts of the product? You should try to understand if the inspection data represents a sample of the product with the same distribution of defect density as the entire product. If you have complete data regarding customer, inspection, and test defects, this can be calculated easily. If you do not have this data you may need to do some additional "sampling" inspections to determine which parts of the product are high and which are low with respect to defect density.

If there is no defect density information available, a simple approach is to divide the product into low and high defect-dense areas based on asking the programmers in which category each file or module belongs. These "sampling" inspections also provide you with the defect density values. Then you could redo each of the calculations (40 percent, 50 percent, 60 percent), and instead of assuming a constant defect density, you could apportion the product into different defect densities and calculate the number of defects in each different portion of the product. Table 1 illustrates how this could be done at 50 percent effectiveness. The total defects remaining is 410 (+ 280 from low density + 180 from high density - 50 found during inspections). Please note that if you have a significant amount of defect density data, you can break your product into 10 decile ranges for this calculation.

The Bottom Line

When you first start to do this type of analysis, you may feel a little uncomfortable using inexact numbers. You need to ask yourself whether you are better off

Table 1. *Defects apportioned by low and high density, assuming 50 percent effectiveness.*

Defect Density Group	Lines of Code	Defects per KLOC (from inspections)	Defects per KLOC (for group)	Total Defects
Low	7,000	20	40	280 (7 KLOC x 40 defects per KLOC)
High	3,000	30	60	180 (3 KLOC x 60 defects per KLOC)
Total	**10,000**	**50**	**100**	**460**

trying to show business value with these rough numbers. I believe most executives would prefer to use inexact data and try to refine it over time. Software inspections are valuable, but they take time; it is important to maximize the business value your organization can achieve from them.

How Do I Optimize the Time Spent on Inspections?

For most software development projects, the single largest cost factor is labor. If doing inspections takes time, optimizing the time spent on inspections means getting the most impact from each inspection by finding the most defects per unit of time expended. Although this may appear obvious, many organizations are not doing this.

Another aspect is to focus the inspection process on finding the most significant product defects. "Significant" would have to be defined by members of the project to include cost to fix, critical functions, or other similar attributes. Analysis of the defect data from inspections, test, and customer-discovered defects can help effectively focus the inspection effort. This is done by understanding and analyzing the types and characteristics of these defects and then modifying the inspection process, e.g., updating checklists, additions, or modifications to scenarios. Inspection data can also help determine the optimum planning rates for the inspections, provide insights into areas for process improvement, and help to build defect removal models.

Additional Elements for Successful Inspections

Champions

As people try to implement any new technology, they need a focal point in the organization who can help them tailor the new technology to meet their project-specific needs. This person also ensures that commonly occurring problems are solved in a consistent manner. The champion may also have to keep the organization focused on the new technology to keep it "alive" until it is a permanent part of the infrastructure.

Moderators

Skilled moderators are key to ensuring consistent control and focus across different inspections. They must be able to manage their peers and keep the discussions focused on finding defects.

Metrics Policy

Because the goal of inspections is to find defects, then to analyze the defect data, there is a need to "decriminalize" defects. A metrics policy statement may help accomplish this.

Time

Given the time required to perform truly effective inspections, the time and resources must be planned, or the inspections are unlikely to happen. You must also be selective in what you inspect so that you focus your efforts on the areas with the highest potential payback.

Data Analysis and Feedback

If people think data are being collected but not used, you may get "garbage in" data. People will do a better job capturing and recording data if they know how the data will be used and how it will impact them. Data recording often requires some degree of interpretation, and the associated value judgements will be much more consistent and accurate if the people doing the data gathering are closely involved with the analysis and its resulting decisions.

Some Basic Analysis Concepts

Although this article is not long enough to fully elaborate on the analysis concepts, a few aspects should be highlighted:

Consistent Inspections and Consistent Data

This is critical if the data are to be used to help make decisions. Consistent processes and data are key to having consistent inspections.

Questions

The key to data analysis is to help people on the project understand what questions they need to ask. In many cases, the data will not tell you what to do, but rather where you need to probe for more information.

Accuracy vs. Precision

Many people with technical backgrounds are uncomfortable using imprecise data. Accuracy means the data truly represent what you believe they represent. The focus on accuracy and not on precision should be publicly discussed so that everyone understands the objective.

Timely Feedback

Analysis and decisions need to be made during the project—do not wait until the end of the project, when it is too late to make decisions that can impact the project. When an inspection effort is first undertaken, the results should be quantified and presented publicly while the project is still in progress.

Feedback and "Feedforward"

Most people are familiar with feedback, which is typically how lessons learned are captured for application to *future* projects. Feedforward is the use of data from an *early* process activity to adjust how the work is done in subsequent process activities on the *same* project. For example, if design inspections discover that a new or modified function is more defect dense than the other functions in that release, more emphasis can be placed on the defect-dense function during code inspections and test.

Ratios

This is similar to the way financial analysts look at business data. They do not look at any single number to determine a company's financial health, but instead look at a large set of numbers. More important, they examine the ratios. For inspections, the following key ratios should be examined.

- **Units of size per hour of inspection.** This could be pages, LOC, or diagrams per hour, or whatever is appropriate for the situation. These rates should be examined to determine if they are reasonable for the type of material being inspected. This data can also be used in subsequent analysis, but only if it is accurate and reasonable. I have seen organizational

data that suggest inspections were held at rates much too fast for optimum efficiency, sometimes approaching 1,000 LOC per hour. In some cases, the inspectors are just recording the total size of the material, not the amount of material inspected. Rates need to make sense, which requires consistent ways to count both the time and the size. You can use standard reading rates to determine if the rates observed are reasonable.

- **Defect density.** This could be major defects per page, per KLOC, or whatever is appropriate. For organizations just starting to do code inspections, defects per KLOC is usually appropriate. As organizations reach improved quality levels, some are starting to measure defect density per million LOC. If LOC is the measure, the data should exclude comments or anything that would not change the compiler-generated object code. Defect density can be considered the "yield" from inspections. Since finding defects is the goal, finding more of them should be an objective. Defect density information can help organizations determine where to inspect.
- **Hours per major defect.** Since labor is the largest cost factor for most software projects, measure the hours it takes to find and fix problems—this figure will be more meaningful to the workers and managers. If you know the hours, you can apply labor rates to arrive at dollars. The ratio would be the sum of all hours expended for the inspection divided by major defects. Typically, groups starting out will find major defects at a rate between two and four hours per major defect. That rate may go much lower, especially for code inspections.
- **Major defects to minor defects.** This ratio can help you better understand if the focus of the inspections is on major or minor problems. It may also highlight recording problems.

Basic Analysis Techniques to Optimize the Time Spent During Inspections

Of all the techniques available, scatter charts are probably the easiest to use and may provide the most useful information. The key is to keep things as simple as possible and not look for complex relationships. Because inspection data represent an intellectual activity and not a mechanical operation, the data will have a higher degree of variance than typical factory data.

Avoid overly sophisticated tools. Many textbooks suggest that linear regression techniques require high correlation coefficients if a relationship exists between the two variables; however, you may find that for inspections and other software-related data, the coefficients may be lower than what the statistical books recommend. Does this mean you should not use the data and the relationships? No, it means you need to be careful. The common relationships that appear in most projects are discussed below. Consider starting to use scatter charts at a project level where the data represent a somewhat consistent entity. Consider starting to analyze the following:

- Major defects per KLOC vs. LOC per hour (do one for inspection time and another for preparation time).
- Hours per major defect vs. LOC per hour (for inspection).
- Size of material inspected vs. hours per defect.
- Size of material inspected vs. defects per KLOC.

Defect Removal Models

Once you have optimized the inspection process, consider developing a defect removal model to help with planning, tracking, and identifying areas for software process improvement. A defect removal model will allow you to understand the process capability for defect removal during each process stage.

Won't Analyzing the Data Only Take More Time?

You could just fix the defects found during inspections and not record data, but there are many important uses for inspection data, many of which are explained throughout this article. Not the least of these uses is to maintain management support for inspections. Look at how much time you may be investing in the inspections alone:

- The typical rate for code inspection is 125 LOC per hour.
- The typical number of inspectors is four.
- Preparation time ideally equals inspection time.

Although these values may make inspection appear more time-consuming than expected, data from many different types of organizations show these values fall into acceptable ranges. The key is for each project team to identify the optimum rates for its project. This means that if four people inspect 1,000 LOC at the above rate (4 x (1,000/125)), it will take 32 hours for inspection and another 32 for preparation, which is 64 hours or approximately 1.5 people weeks. This is not a trivial investment, especially if a significant percentage of the artifacts for a software development project are to be inspected.

When you look at these numbers, you may be ready to cancel your inspection program, but wait—the cost of removing defects through inspections needs to be compared with the cost of removing the same defects in test or having to fix them when they are found by customers. Many articles have shown that organizations that analyze their data can find and fix defects much more cheaply with inspections than during subsequent process activities. Many organizations have shown that they can find and fix the average defect in two to four person hours or less using inspections, which is typically much faster than most testing processes. Although your numbers may not be the same, you will not know unless you analyze your data.

It should only take a couple of days to do this analysis, which is well worth the time for the improvements that can be effected based on this analysis—especially given the amount of time spent on inspections. And the bottom line is managers and executives are unlikely to continue spending money on

see HARDING, page 24

The difference in a Y2K application failure situation is a matter of degree, not of kind. The level of daily faults will reach a point that will overwhelm the support staff; the contingency backup support, which is designed for isolated crises, will also be overwhelmed by simultaneous crisis calls from too many sites. The faults will come in waves as critical dates are reached for each application, and the faults will build to a peak around the end of 1999 and the beginning of 2000. Faults will start to recede after March 1, 2000, although new failures will continue for some time. We are already seeing a few cases of significant Y2K faults, although so far none have been overwhelming.

If a testing project fails to complete full testing, it does not *necessarily* follow that the renovated application will fail in production. In some cases, the level of undetected faults will be containable in practice. In other cases, undetected faults will not be containable, and damage to the business will result. The business purpose behind significant testing projects is to take chance out of the equation and to provide an insurance policy against damage. In this sense, the cost of the testing project can be considered the premium on an insurance policy.

The complete version of this article details what is required to achieve risk minimization using conventional testing methods, how to proceed in a risk and cost optimization testing project using conventional testing, and a discussion of some innovative technical approaches to introduce economies of scale by automating the process of testing. Where applicable, automated testing can allow a testing project to move significantly closer to the risk minimization model within the limits of what is practical and affordable. ◆

About the Author

Don Estes is chief technology officer for 2000 Technologies Corporation, for whom he has designed and implemented both a data encapsulation and an automated testing system. He also works closely with vendors of limited window-

ing, program encapsulation, and object code remediation systems. He has been involved with COBOL and database applications for 25 years and database and mainframe performance tuning for 10 years. For the last seven years, he has helped design and execute projects for the mass modification of large bodies of source code, primarily for platform migration, using state-of-the-art automated source language transformation technologies and automated testing methods. He is a regular contributor to Peter de Jager's Year 2000 mail list, where he is known for his contributions relating to Y2K rapid compliance strategies and automated testing. Estes is a graduate of Massachusetts Institute of Technology in physics, with a postgraduate degree from the University of Texas in educational psychology.

2000 Technologies Corporation
114 Waltham Street, Suite 19
Lexington, MA 02173
Voice: 781-860-5277, 800-756-8046
E-mail: info@2000technologies.com

HARDING, from page 22

inspections unless they start seeing value for the dollars spent. They need to see the business payback in quantitative terms. The business value analysis should compare the cost (hours per major defect) for defects found in inspections with the cost for each test activity. If you have not been collecting this data for the test activities, you may need to have developers and testers estimate the number of hours they believe it takes to find defects during each test activity. ◆

About the Author

John T. Harding is one of the founding partners of Software Technology Transition, which provides training and implementation in the Software Engineering Institute (SEI) Capability Maturity Model (CMM) and CMM-Based Appraisal for Internal Process Improvement method and in software inspections, metrics, and project management. Other work includes the International Organization for Standardization (ISO) gap analysis, helping organizations develop business and software baselines, and action planning for software process improvement. He was a visiting scientist at the SEI, was the metrics mission manager for Groupe Bull, and held various technical and managerial positions in software development with IBM and the Bank of Boston. He has a master's degree in business administration from Boston University and a bachelor's degree from Rensselaer Polytechnic Institute (RPI) and is a member of the Association for Computing Machinery and the Institute of Electrical and Electronics Engineers.

Software Technology Transition
60 Elm Street
Andover, MA 01810
Voice and fax: 978-475-5432
E-mail: johntharding@compuserve.com

Recommended Reading

1. Ebenau, R.G. and S.H. Strauss, *Software Inspection Process*, McGraw-Hill, New York, 1994.
2. Gilb, T. and Dorothy Graham, *Software Inspections*, Addison-Wesley, Reading, Mass., 1993.
3. Weller, E.F., "Lessons From Three Years of Inspection Data," *IEEE Software*, September 1993, pp. 38-45.
4. Weller, E.F., "Using Metrics to Manage Software Projects," *IEEE Computer*, September 1994, pp. 27-33.
5. Grady, Robert B. and Deborah L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, N.J., 1987.
6. Grady, Robert B., *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
7. Ishikawa, Kaoru, *Guide to Quality Control*, Asian Productivity Organization, Tokyo, 1976.
8. Burr, Adrian and Mal Owen, *Statistical Methods for Software Quality*, International Thomson Computer Press, London, 1996.