Seven Truths About Peer Reviews

Karl E. Wiegers

Process Impact

www.processimpact.com

The term "testing" conjures an image of executing software in a prescribed way to see whether it functions as intended. An alternative form of testing (or, more precisely, quality control) is to invite some colleagues to examine your work products for defects and improvement opportunities: a peer review. Whereas traditional testing is limited to executable code, you can apply peer reviews to any software deliverable, design, or document. Peer reviews have long been recognized as a powerful way to improve quality, yet few software organizations have systematic and effective review programs in place. This article presents seven facts about peer reviews that any organization concerned about quality needs to know. To begin, keep in mind that...

Peer reviews can take many forms

A quality-driven organization will practice a variety of peer review methods, spanning a spectrum of formality, rigor, effectiveness, and cost. Let's look at descriptions of some common review approaches.

- An ***inspection*** is the most systematic and rigorous type of peer review. Inspection follows a well-defined multistage process with specific roles assigned to individual participants. Inspections are more effective at finding defects than are informal reviews. For example, inspections held on Motorola's Iridium project detected 80% of the defects present, whereas less formal reviews discovered only 60% of the defects [2].
- ***Team reviews*** are a type of "inspection-lite," being planned and structured but less formal and less rigorous than inspections. Typically, the overview and follow-up inspection stages are simplified or omitted, and some participant roles may be combined (e.g., moderator and reader).
- A ***walkthrough*** is an informal review in which the work product's author describes it to some colleagues and solicits comments. Walkthroughs differ significantly from inspections because the author takes the dominant role; other specific review roles are usually not defined. Walkthroughs are informal because they typically do not follow a defined procedure, do not specify exit criteria, require no management reporting, and generate no metrics.
- In ***pair programming***, two developers work on the same program simultaneously at a single workstation, continuously reviewing their joint work. Pair programming lacks the outside perspective of someone who is not personally attached to the code that a formal review brings.
- In a ***peer deskcheck***, only one person besides the author examines the work product. A peer deskcheck typically is an informal review, although the reviewer could employ defect checklists and specific analysis methods to increase effectiveness.
- A ***passaround*** is a multiple, concurrent peer deskcheck, in which several people are invited to provide comments. The passaround mitigates two major risks of a peer deskcheck: the reviewer failing to provide timely feedback and the reviewer doing a poor job.

The project team should select the cheapest review method that will reduce the risk associated with defects remaining in a given deliverable to an acceptable level. Use inspections for high-risk work products, and rely on cheaper techniques for components that have lower risk.

Although there are many ways to get your colleagues to help improve the deliverables you create, only...

Inspections are a software industry best practice

Inspection has been identified as one of the top nine software industry best practices [1]. Nearly 30 years of experience show that inspections are an efficient and effective way to discover defects in any work product, as the following examples illustrate:

- Hewlett-Packard's inspection program measured a return on investment of 10 to 1, saving an estimated $21.4 million per year. Design inspections reduced time to market by 1.8 months on one project [5].
- Inspections contributed to a ten-fold improvement in quality and a 14 percent increase in productivity at AT&T Bell Laboratories [7].
- Inspecting 2.5 million lines of real-time code at Bell Northern Research prevented an average of 33 hours of maintenance effort per defect discovered [11].
- IBM reported that each hour of inspection saved 20 hours of testing and 82 hours of rework effort had the defects found by inspection remained in the released product [6].
- At Imperial Chemical Industries, the cost of maintaining a portfolio of about 400 programs that had been inspected was one-tenth the cost per line of code of maintaining a similar set of 400 uninspected programs [4].

All types of peer review yield additional benefits that are difficult to quantify. They spread product, project, and technical knowledge among the team members, which enables someone other than the author to maintain a work product. Reviews reveal assumptions, help the team establish shared expectations, and create a common understanding of technical work products. Reviews help practitioners build products that are simple and understandable, which reduces maintenance and enhancement costs.

The maximum inspection benefits come from process improvements that prevent defects in future work. One way to achieve this benefit is to inspect large work products when they are perhaps ten percent complete, rather than waiting until a large body of work has been finished, possibly incorrectly. The insights gained permit the remaining 90 percent of the work to be done better.

The widespread fear that inspections will simply slow the project down assumes that the time you invest in inspections yields no return on investment. However, inspection authority Tom Gilb reports that each major defect found by inspection saves an average of nine labor hours in avoided downstream rework [4]. If your inspectors take less than nine labor hours to find and fix a major defect, you come out ahead. It's that simple.

Yes, inspection is a powerful technique for quality improvement and knowledge exchange. But which inspection method should you use? Remember that...

There is no one true inspection method

Several inspection processes are in widespread use. The most popular are Michael Fagan's original method [3,10] and the variation developed by Tom Gilb and Dorothy Graham [4]. Although these methods use different terminology and define different ways to perform various inspection steps (see Table 1), their similarities outweigh their differences. Either method, as well as additional inspection techniques described in the literature, can help any software organization improve its product quality and development productivity. Beginning to practice any inspection method in your organization is better than endlessly debating the "right" approach or inventing Yet Another Inspection Model.

Table 1. Comparison of Fagan and Gilb/Graham Inspection Methods

Element	Fagan Method	Gilb/Graham Method
Process Steps	Planning Overview Preparation Inspection Meeting Rework Follow-up Causal Analysis	Planning Kickoff Meeting Individual Checking Logging Meeting Editing Follow-up Process Brainstorming Meeting
Roles	Author Moderator Reader Recorder Inspector	Author Inspection Leader (not used) Scribe Checker
Defect-Detection Techniques	defect checklists	rule sets and checklists
Emphasis	removing defect	document quality measurement process improvement

Debates rage about certain aspects of the inspection process. Should the author lead the inspection meeting or just serve as another inspector and listen to comments the other participants make? Do you really need a reader to present the work product being inspected to the other participants one small bit at a time? Do you even need an inspection meeting? These are legitimate debates, and there are no unambiguous answers. Study the arguments on both sides [12], try different approaches for yourself, and see which ones work best for you.

If inspections are so great, should you shut down your testing department? No, because...

Peer reviews complement testing

Do not expect to replace testing with peer reviews; rather, add reviews to your quality tool kit. The two techniques find different kinds of defects, can be applied at different stages of the project, and demand different skills. Automated testing is vastly more efficient than manual review, and the notion of "regression reviewing" gives me the creeps. Testing demonstrates the actual behavior of the system in operation, not the imagined behavior that reviewers deduce from studying the code. Design and code reviews won't tell you how fast the system will operate, although if the performance is sluggish you need to review the code to figure out why and where to tune it. And the hands-on experience of actually exercising the product will reveal usability issues that no user interface review would find.

A seminar student once protested that code reviews were unnecessary, insisting that she could find all of the errors faster by testing. This is a common misconception. Bell Northern Research discovered that finding defects through inspection was two to four times faster than revealing them through testing [11]. At IBM's Santa Teresa laboratory, 3.5 labor hours were needed to find a major defect by code inspection, versus 15 to 25 hours of testing [9]. A single testing stage is unlikely to remove more than 35 percent of the defects in the tested work product, whereas design and code inspections typically find 50 to 70 percent of the defects [8].

If a test reveals a failure, you have to go on a debugging expedition to find the underlying defect. During a review, though, you are looking directly at the problem, not at an indirect symptom of it. Reviews also expose defects that testing tools might be blocked from detecting. For example, if a button on a dialog box doesn't function properly, you have to fix it before you can test the display that's supposed to appear when the user clicks on that button. However, you can review the code for that second display in spite of the button bug. Peer reviews are the best available tool for finding defects in requirements and design specifications, project plans, user manuals, and test documentation. In fact, reviews constitute a type of static testing for these work products.

Testing doesn't tell you anything about the clarity, maintainability, or efficiency of the code, but a developer can judge these qualities during a review. A human reviewer can spot unclear error messages, inadequate comments, hard-coded variable values, and repeated code patterns that could be consolidated. Testing won't reveal unnecessary code, although code analyzers can flag unreachable code that a person might not notice. You could use a test coverage analyzer to identify untested code, such as exception handlers that are hard to trigger, then focus your manual review on those code segments.

You can augment your manual code reviews with automated code analyzers such as Gimpel Software's Flexelint, ParaSoft's Code Wizard, and NuMega CodeReview (see http://www.processimpact.com /pr_goodies.shtml#pr_tools for links to these and similar tools) The efficient programmer will run code through these products prior to submitting it for manual examination by human eyes. Code analyzers can find subtle syntax errors and likely problem areas such as uninitialized variables that the human eye might not see, but they won't detect logic errors in the implementation. Similarly, a word processor's spell-checker will catch misspelled words in a requirements specification, but it cannot spot erroneous, ambiguous, or missing requirements or incorrect word usage.

The benefits that peer reviews can provide are indisputable. But it's important to note that...

Peer reviews are both technical and social activities

Most software testing activities are technical in nature. However, peer reviews include a strong cultural and interpersonal component. Asking someone else to identify errors in your work is a learned—not instinctive —behavior. Instilling a review program into an organization requires an understanding of that organization's culture and the values that its members hold. Managers need to believe that the time spent on reviews is a

sound business investment so they make time available for the team to hold reviews. You need to know why certain people resist submitting their work to scrutiny by their colleagues and address that resistance. You must educate the team and its managers about the peer review process, appropriate behavior during a review, and the benefits that getting a little help from their friends can provide to both individuals and the organization. The social issues become even more challenging when reviews involve participants from diverse national or cultural backgrounds, some of whom might be reluctant to point out errors in a colleague's work.

The dynamics between the reviewers and the author are a sensitive issue. The reviewers aren't there to show that they're smarter than the author, and the author isn't there to justify every bit of the work product and rationalize away problems. Reviewers should thoughtfully select the words they use to raise an issue, making their observations about the product and not about the author. An author who walks out of a review meeting feeling embarrassed, personally attacked, or professionally insulted will not voluntarily submit work for review again.

The cultural aspect of peer reviews also means that...

Managers can make or break a review program

Management plays an essential role in determining how well an organization's review activities work. Key issues include management participation in reviews, the use (and misuse) of data collected from reviews, and demonstrating management commitment to a review program.

Management Participation

The conventional wisdom is that managers may not inspect products created by people who report to them. The rationale is that managers will be tempted to evaluate authors based on defects identified during the inspection. A second risk is that other participants might hesitate to point out problems if they suspect the manager is keeping mental records to use at performance evaluation time.

I believe that the issue of whether to include managers in an inspection is less clear-cut than simply saying "No!" Management participation depends on the mutual respect and trust between the manager and the other team members. The author's first-level manager may participate at the author's invitation, provided that the manager has the requisite technical knowledge. When I managed a small group, the other team members often invited me to inspect their work. I interpreted this to mean that they respected my ability to improve their products, and they trusted me not to think less of them if we did find defects.

Managers need to have their own work products, such as project charters and development plans, inspected. A technical manager who solicits peer review is leading by example and creating an open culture of constructive criticism and continuous learning.

Data Abuse

Using defect data from reviews to evaluate the performance individuals is a classic culture killer. It can lead to *measurement dysfunction*, in which measurement motivates people to behave in a way that produces results inconsistent with the desired goals. I encountered one company in which a manager declared that finding more than five defects during a code inspection would count against the author's performance evaluation. This misguided strategy could lead developers to not submit their work for review or to inspect only small chunks of work to avoid finding too many defects in any inspection. Inspectors might point out

defects to authors off-line, rather than during the inspection, and the organization's culture might emphasize *not* finding defects during inspections. Such evaluation criminalizes the mistakes that we all make and motivates participants to manipulate the process to avoid being hurt by it.

Management Commitment

Managers often say that they "support" an organization's quality or process improvement initiatives. I am not interested in management "support," which generally translates into simple acceptance of the new approach or permission for the team members to pursue it. However, I'm extremely interested in management *commitment*. Following are several actions that software managers can take that demonstrate their commitment to a peer review program.

- Provide the resources and time to develop, implement, and sustain an effective review process. Ensure that project schedules include time for reviews.
- Set policies, expectations, and goals about review practice. Hold people accountable for participating in reviews and for contributing constructively to them.
- Make training available to the participants and attend the training themselves.
- Run interference with other managers and customers who challenge the need for reviews.
- Ask for status reports on how the program is working, what it costs, and the team's benefits from reviews.

If managers aren't willing to provide this level of commitment to their review program, don't be surprised if it stumbles and falls. Another essential point for managers to remember is that...

A peer review program doesn't run itself

A serious review program requires several elements to succeed. These include appropriate process assets (procedures, forms, checklists, and the like), staff resources to install and sustain the program, training for review participants and their managers, and—most essential—time to actually perform reviews.

Process Assets

Don't expect your team members to perform peer reviews effectively unless they have a written process description and supporting work aids, collectively termed *process assets*. The process must be clear and easy to apply, or people won't use it. Figure 1 identifies some items to include in your review process. The process overview provides a high-level description of the activities involved in any type of review. One of those steps is to select the appropriate review method for each situation, so some risk assessment guidance is included. The process should include procedures with step-by-step instructions for performing inspections and perhaps other types of reviews.

You may download a sample peer review process description from the author's web site at http://www.processimpact.com/pr_goodies.shtml#pr_assets. Also available there are other work aids, including review forms, defect checklists for several types of software work products, and some simple spreadsheets to jumpstart your collection and analysis of inspection data.

The Process Owner

Too often, process improvement actions screech to a halt after a process working group has developed some new procedures; busy practitioners aren't going to change how they work just because some of their colleagues recommended a different approach. Every software organization should therefore identify a

process owner for its peer review program, whose typical responsibilities are summarized in Figure 2. The peer review process owner is an enthusiastic and committed champion, a manager who strongly believes in reviews and is willing to devote energy to making the program succeed. The process owner provides continuity, serving as the point of contact for requests to improve the review procedures, forms, and checklists. Without ownership, processes can decay over time and fail to meet the organization's current needs. This wastes the investment made in developing the processes and frustrates the team members.

The Peer Review Coordinator

Each organization should also identify an individual to coordinate the review program on a day-to-day basis. Typically, this is a part-time responsibility performed by a software quality engineer. Because this role will consume considerable time, it should be defined as part of the coordinator's primary job description.

The ideal coordinator is an experienced inspector and moderator who thoroughly understands the technical and social aspects of reviews. He coaches team members who need help performing them and makes training sessions available. He is the custodian of the organization's inspection database. Based on analysis of inspection metrics data, the coordinator offers process improvement suggestions. The peer review coordinator should observe reviews periodically to see how the process works in practice, and he works with moderators to diagnose and rectify inspection problems.

Making peer reviews work for you

I began practicing peer reviews on software projects in 1988. Today I would never consider working in an organization that did not include reviews as a cornerstone of its software engineering culture. The foundation of a successful review program is a shared philosophy that "We prefer to have a peer, rather than a customer, find a defect" [12]. Every development organization can reap the benefits that well-conducted peer reviews can provide.

References

1. Brown, Norm. "Industrial-Strength Management Strategies," *IEEE Software*, Vol. 13, No. 4 (July 1996), pp. 94-103.
2. Brown, Norm. "High-Leverage Best Practices: What Hot Companies Are Doing to Stay Ahead," *Cutter IT Journal*, Vol. 12, No. 9 (September 1999), pp. 4-9.
3. Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3 (1976), pp. 182-211.
4. Gilb, Tom, and Dorothy Graham. *Software Inspection*. Wokingham, England: Addison-Wesley, 1993.
5. Grady, Robert B., and Tom Van Slack. "Key Lessons in Achieving Widespread Inspection Use," *IEEE Software*, Vol. 11, No. 4 (July 1994), pp. 46-57.
6. Holland, Dick. "Document Inspection as an Agent of Change," *Software Quality Professional*, Vol. 2, No. 1 (December 1999), pp. 22-33.
7. Humphrey, Watts S. *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley, 1989.
8. Jones, Capers. *Applied Software Measurement: Assuring Productivity and Quality, 2nd Edition.* New York: McGraw-Hill, 1996.
9. Kaplan, Craig A. "Secrets of Software Quality," *Proceedings of the Fifth International Conference on Software Quality*, October 24-26, 1995, Austin, Texas, pp. 15-27.
10. Radice, Ronald A. *High Quality Low Cost Software Inspections*. Andover, Massachusetts: Paradoxicon Publishing, 2002.

11. Russell, Glen W. "Experience with Inspection in Ultralarge-Scale Developments," *IEEE Software*, Vol. 8, No. 1 (January 1991), pp. 25-31.
12. Wiegers, Karl E. *Creating a Software Engineering Culture*. Dorset House, 1996.
13. Wiegers, Karl E. *Peer Reviews in Software: A Practical Guide*. Addison-Wesley, 2002.

[This paper was originally published in *Cutter IT Journal*, July 2002. It is reprinted with permission from Cutter Information LLC. It is adapted from *Peer Reviews in Software: A Practical Guide* by Karl E. Wiegers (Addison-Wesley, 2002).]