When Two Eyes Aren't Enough[1]

Karl E. Wiegers
Process Impact
www.processimpact.com

"Hey, Maria, do you have a minute? I can't find a silly little bug in this program. Would you take a look at this code for me, please?"

"Sure, Phil. What's it doing wrong?"

"It's not aligning these images correctly. They're all supposed to be left justified, but each one is indented farther in. I'm pretty sure the problem's in the DisplayImage method, but I've been looking at it for 15 minutes and just can't see anything wrong."

"Hmmm, let's see here. It looks like...." [mutter, mutter, mutter] "No, that part looks fine. But look at the top of this loop." [Maria points to the screen.] "I think this parenthesis is in the wrong place. If you move the index variable inside that paren, I don't think the images will be stair-stepped any more."

Phil smacks his forehead with his palm. "You're right! Thanks a lot, Maria. I can't believe I didn't see that."

"No problem, Phil. I'm glad we found it now rather than later."

Most programmers have asked their colleagues to help them find elusive problems in their code through an ad hoc review like the one Phil and Maria performed. Often you're too close to your own work to spot errors you've made. As you study the code, your brain just recites what you created earlier (or what you intended to create) because you're following the same reasoning you used when you made the mistake. You need a fresh perspective, another pair of eyes that haven't seen the code before, and another brain that thinks in a different way.

In a *peer review*, someone other than the author of a work product examines that product for defects and improvement opportunities. There are several distinct peer review approaches, but software practitioners use conflicting terminology to describe them. This article describes several kinds of peer reviews that span a range of formality and structure. It also provides guidance for selecting an appropriate review technique for a given situation.

Figure 1 places several common review methods on a formality spectrum. The most formal reviews, exemplified by inspections, have several characteristics:

- Defined objectives
- Participation by a trained team
- Leadership by a trained moderator
- Specific participant roles and responsibilities
- A documented procedure for conducting the review
- Reporting of results to management
- Explicit entry and exit criteria for each work product
- Tracking defects to closure
- Recording process and quality data to improve both the review process and the software development process.

---

[1] This paper was originally published in *Software Development*, October 2001. It is reprinted (with modifications) with permission from *Software Development* magazine.

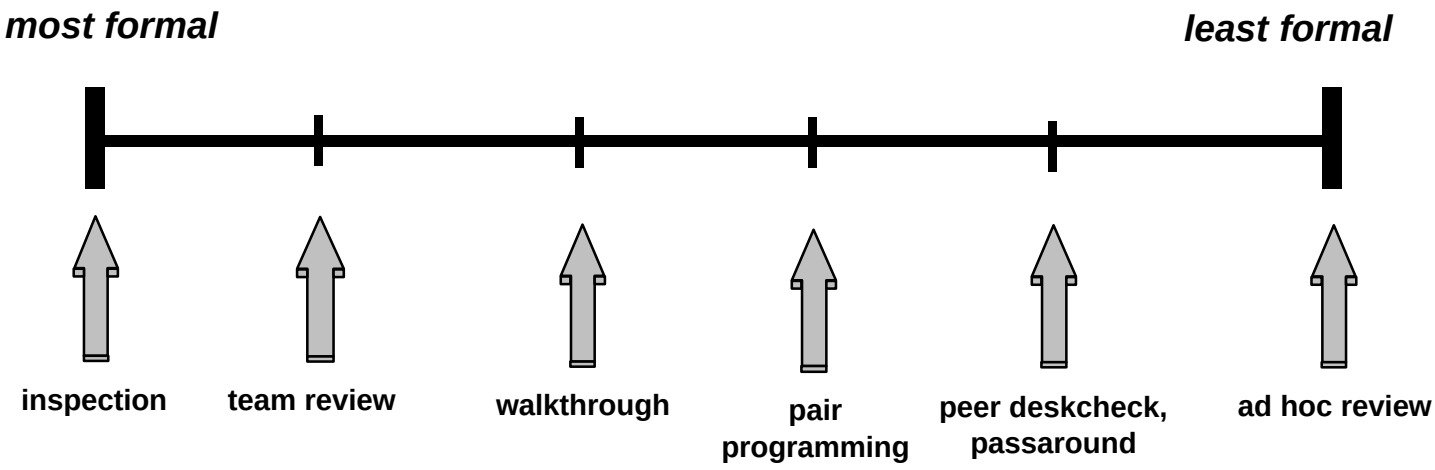

Figure 1. Peer review formality spectrum.

Even informal reviews are worth doing, and they may meet your needs in certain situations. You should recognize the strengths and limitations of the various approaches so you can select a review process for each situation that fits your culture, time constraints and business objectives.

All peer reviews involve some combination of planning, studying the work product, holding a review meeting, correcting errors and verifying the corrections. Table 1 shows which of these activities are typically included in each of the review types shown in Figure 1.

Table 1. Activities Included in Different Types of Software Peer Reviews

	Review Activity				
Review Type	**Planning**	**Preparation**	**Meeting**	**Correction**	**Verification**
Inspection	Yes	Yes	Yes	Yes	Yes
Team Review	Yes	Yes	Yes	Yes	No
Walkthrough	Yes	No	Yes	Yes	No
Pair Programming	Yes	No	Continuous	Yes	Yes
Peer Deskcheck, Passaround	No	Yes	Possibly	Yes	No
Ad Hoc Review	No	No	Yes	Yes	No

Inspection

An inspection is the most systematic and rigorous type of peer review. Inspection—a software industry best practice—follows a well-defined, multistage process with specific roles assigned to individual participants. The most common inspection process includes seven stages: planning, overview, preparation, meeting, rework, follow-up and causal analysis. For maximum effectiveness, inspectors should be trained in the inspection process and be able to perform all of

the different roles. Inspections rely on checklists of common defects found in different types of software work products, rules for constructing work products and various analytical techniques to search for bugs.

A fundamental aspect of inspection isthat participants other than the work product author lead the meeting (moderator), present the material to the inspection team (reader) and record issues as they are brought up (recorder). Participants prepare for the inspection meeting by examining the material on their own to understand it and to find problems. During the meeting, the reader presents the material a small portion at a time to the other inspectors, who raise issues and point out possible defects. The reader helps the team reach the same interpretation ofeach portion of the product, because the inspectors can compare their understanding to that expressed by the reader. At the end of the meeting, the team agrees on an appraisal of the work product and decides how to verify the changes that the author will make during rework.

Inspections are more effective at finding defects than are informal review techniques. One telecommunications company, I'm told, detected an average of 16 to 20 defects per thousand lines of code by inspection, compared to only three defects per thousand lines of code using informal reviews. The close scrutiny of an inspection provides a thorough test of a product's understandability and maintainability, and reveals subtle programming problems.

Different inspectors spot different kinds of problems, but this contribution does not increase linearly with additional participants. When one inspector notes a defect, it's common to hear another participant say, "I saw that, too." The interactions between inspectors during the meeting can reveal new bugs as one person's observation stimulates another's thinking. You lose this collaborative benefit if your peer review does not include a meeting, although some studies have questioned whether the synergy adds any real value (see Lawrence G. Votta Jr.'s, "Does Every Inspection Need a Meeting?" *Proceedings of the First ACM SIGSOFT Symposium on Software Development Engineering* [ACM Press, 1993], and Adam A. Porter and Philip M. Johnson's "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies," *IEEE Transactions on Software Engineering*, Mar. 1997).

Team Review

Team reviews are a type of "inspection-lite": though planned and structured, they're a bit less formal and less comprehensive. The team review goes by many names, including "structured walkthrough" and simply "review."

As I've practiced them, team reviews follow several of the steps found in an inspection. The participants receive the review materials several days prior to the review meeting, and are expected to study the materials on their own. The team collects data on the review effort and the number of defects found. However, the overview and follow-up inspection stages are simplified or omitted, and some participant roles may be combined. The author may lead the team review, whereas in an inspection, the author may not serve as the moderator. In contrast to most inspections, the reader role is omitted. Instead of having one participant describe a small chunk of the product at a time, the review leader asks the participants if they have any comments on a specific section or page.

Team reviews cost more than having a single colleague perform a peer deskcheck, but the different participants will find different sets of errors. As with any meeting, the team can get sidetracked in tangential discussions, so the review leader must keep the meeting on course. The recorder or scribe captures issues as they're raised during the discussion, using standard forms the organization has adopted.

While little data is available, one industry study (see Erik P.W.M. Van Veenendaal's "Practical Quality Assurance for Embedded Software," *Software Quality Professional*, June 1999) found that this type of peer review discovered only two-thirds as many defects per hour of effort as inspections revealed. IBM Federal Systems Division measured coding productivity for projects that practiced team reviews at about half the productivity of a set of similar projects that used inspections. Team reviews are suitable for a team or product that doesn't require the full rigor of the inspection process.

Walkthrough

A walkthrough is an informal review in which the author of a work product describes it to a group of peers and solicits their comments. Walkthroughs differ significantly from inspections because the author takes the dominant role; there are no other specific reviewer roles. Whereas an inspection is intended to meet the team's quality objectives, a walkthrough principally serves the needs of the author. Table 2 points out some differences among the ways that inspections, team reviews and walkthroughs are usually performed.

Table 2. Comparison of Some Inspection, Team Review and Walkthrough Characteristics

Characteristic	*Inspection*	*Team Review*	*Walkthrough*
Leader	Moderator	Moderator	Author
Material presenter	Reader	Moderator	Author
Granularity	Small chunks	Pages or sections	Author's discretion
Recorder used	Yes	Yes	Maybe
Follows a documented procedure	Yes	Maybe	Maybe
Defined participant roles	Yes	Yes	No
Defect checklists used	Yes	Yes	No
Data collected and analyzed	Yes	Maybe	No
Product appraisal determined	Yes	Yes	No

Walkthroughs typically do not follow a defined procedure, require no management reporting and generate no metrics. They can be an efficient way to review products modified during maintenance because the author can draw the reviewers' attention to those portions of the deliverables that were changed. However, this runs the risk of overlooking other sections that should have been changed but were not. Because records are rarely kept, there is little data about how effective walkthroughs are at detecting bugs. Inspections at Ford Motor Company discovered 50 percent more defects per thousand lines of code than did walkthroughs (see Kirk Bankes's and Fred Sauer's "Ford Systems Inspection Experience," *Proceedings of the 1995 International Information Technology Quality Conference*, Quality Assurance Institute, Orlando, Fla.).

When walkthroughs do not follow a defined procedure, people perform them in diverse ways. In a typical walkthrough, the author presents a code module or design component to the participants, describing what it does, how it is structured and performs its tasks, the logic flow, and its inputs and outputs. Finding problems is one goal; reaching a shared understanding of, and

a consensus on, the component's purpose, structure and implementation is another. Design walkthroughs provide a way to assess whether or not the proposed design is sufficiently robust and appropriate to solve the problem.

If you don't fully understand some information presented in a walkthrough, you might gloss over it and assume the author is right. It's easy to be swayed into silence by an author's rationalization of his approach, even if you aren't convinced. Sometimes, though, a reviewer's confusion is a clue that a defect lurks nearby. Walkthroughs have a place in your peer review toolkit, but the domination by the author and the unstructured approach render them less valuable than either inspections or team reviews as a defect filter.

Pair Programming

In pair programming—a practice made popular by Extreme Programming—two developers work on the same product simultaneously at a single workstation. This facilitates communication and affords an opportunity for continuous, incremental and informal review of each person's ideas. Every line of code is written by two brains driving a single set of fingers, which leads to superior work products by literally applying the adage, "two heads are better than one." Pair programming can be used to create any software work product, not just code.

Culturally, pair programming promotes collaboration, an attitude of collective ownership of the team's code base and a shared commitment to the quality of each component. At least two team members become intimately familiar with every piece of code, which reduces the knowledge lost through staff turnover. The pair can quickly make course corrections because of the real-time review.

I classify pair programming as a type of informal review because it is unstructured and involves no process, preparation or documentation. It lacks the outside perspective of someone who is not personally attached to the code that a formal review brings. Nor does it include the author of the parent work product as a separate perspective. Pair programming is not specifically a review technique, but rather a software development strategy that relies on the synergy of two focused minds to create products superior in design, execution and quality. However, pair programming is a major culture change in the way a development team operates, so it's not a simple replacement for traditional peer reviews in most situations.

Peer Deskcheck

In a peer deskcheck, only one person besides the author examines the work product. The author might not know how the reviewer approached the task or how comprehensively the review was done. A peer deskcheck depends entirely on the reviewer's knowledge, skill and self-discipline, so expect wide variability in results. Peer deskchecks can be fairly formal if the reviewer employs defect checklists, specific analysis methods and standard forms to keep records. Upon completion, the reviewer can deliver a defect list to the author or simply hand the author his marked-up work product.

The peer deskcheck is the cheapest review approach. It involves only one reviewer's time plus perhaps a follow-up discussion with the author to explain the reviewer's findings. This method is suitable for low-risk work products, if you have colleagues who are skilled at finding defects on their own or if you have severe time and resource restrictions. A peer deskcheck can be more comfortable for the author than an intimidating group review. However, the only errors found will be those the one reviewer is best at spotting. Also, the author is not present to answer questions and hear discussions that can help him find additional defects that no one else sees.

Peer deskchecks provide a good way to begin developing a review culture. Find a colleague whom you respect professionally and trust personally, and offer to exchange work products for peer deskchecks. This is also a good mentoring method, providing it's done with sensitivity.

Passaround

A passaround is a multiple, concurrent peer deskcheck. Instead of asking a single colleague for input, you deliver a copy of the product to several people and collate their feedback. The passaround mitigates two major risks of a peer deskcheck: the reviewer failing to provide timely feedback and the reviewer doing a poor job. You can engage more reviewers through a passaround than you could conveniently assemble in a meeting. However, a passaround still lacks the mental stimulation that a group discussion can provide. Once I used a passaround approach to have other team members review my development plan for a new project. Unfortunately, none of us noticed that some important tasks were missing from my work breakdown structure. We thought of these missing tasks later during a team meeting, which makes me suspect that we would have found them if we had used a team review instead of the passaround.

As an alternative to distributing physical copies of the document to reviewers, one of my groups placed an electronic copy of the document in a shared folder on our server. Reviewers contributed their comments in the form of document annotations, such as Microsoft Word comments or PDF notes, for a set period of time. Then the author reconciled conflicting inputs from the reviewers, making obvious corrections and ignoring unhelpful suggestions. Only a few issues remained that required the author to sit down with a particular reviewer for clarification or brainstorming.

This passaround method allows each reviewer to see the comments others have already written, which minimizes redundancy and reveals differences of interpretation. Watch out for debates that might take place between reviewers in the form of document comments; those are better handled through direct communication. Document reviews of this type are a good choice when you have reviewers who cannot hold a face-to-face meeting because of geographical separation or time limitations.

Ad Hoc Review

This article opened with an example of one programmer asking another to spend a few minutes helping to track down an elusive bug. Such spur-of-the-moment reviews should be a natural part of software team collaboration. They provide a quick way to get another perspective that often finds problems we just cannot see ourselves. Ad hoc reviews are the most informal type of review. They might solve the immediate problem, but they have little impact beyond that.

Choosing a Review Approach

One way to select the most appropriate review method for a given situation is based on risk, the likelihood of a work product containing defects and the potential for damage if it does. Use inspections for high-risk work products, and rely on cheaper techniques for components that have lower risk. Some factors that increase risk include:

- Use of new technologies, techniques or tools

- Complex logic or algorithms that are difficult to understand but must be accurate and optimized
- Excessive developer schedule pressure
- Inadequate developer training or experience
- Mission- or safety-critical portions of the product with severe failure modes
- Key architectural components that provide a base for subsequent product evolution
- A large number of exception conditions or failure modes that must be handled, particularly if they are difficult to stimulate during testing
- Components that are intended to be reused
- Components that will serve as models or templates for other components
- Components with multiple interfaces that affect various parts of the product.

You can also select a review technique based on your stated objectives for the review. Table 3 suggests which review approaches are best suited to achieve specific objectives. The best way to judge which peer review method to use in a given situation is to keep records of review effectiveness and efficiency in your own organization. You might find that inspections work well for code, but that team reviews or walkthroughs work best for design documents. Data provides the most convincing argument, but even subjective records of the kind of reviews you did, the work products that you examined and how well the reviews worked will be useful. Most importantly, begin developing a culture that embraces peer review as a powerful contributor to increased software quality, productivity and individual learning.

Table 3. Suggested Review Methods for Meeting Various Objectives

Review Objectives	**Inspection**	**Team Review**	**Walkthrough**	**Pair Programming**	**Peer Deskcheck**	**Passaround**
Find implementation defects	X	X	X	X	X	X
Check conformance to specifications	X	X			X	
Check conformance to standards	X				X	
Verify product completeness and correctness	X		X			
Assess understandability and maintainability	X	X		X	X	X
Demonstrate quality of critical or high-risk components	X					
Collect data for process improvement	X	X				
Measure document quality	X					
Educate other team members about the product		X	X			X
Reach consensus on an approach		X	X	X		
Explore alternative approaches			X	X		
Ensure that changes or bug fixes were made correctly		X	X		X	
Simulate execution of a program			X			
Minimize review cost					X	
Provide progress check to management and customers					X	X