Seven Deadly Sins of Software Reviews

Karl Wiegers
Process Impact
716-377-5110
www.processimpact.com

A quarter-century ago, Michael Fagan of IBM developed the software inspection technique, a method for finding defects through manual examination of software work products by a group of the author's peers. Many organizations have achieved dramatic results from inspections, including IBM, Raytheon, Motorola, Hewlett Packard, and Bull HN. However, other organizations have had difficulty getting any kind of software review process going. Considering that effective technical reviews are one of the most powerful software quality practices available, all software groups should become skilled in their application.

This article describes seven common problems that can undermine the effectiveness of software reviews of any type (inspections being a specific type of formal review). I describe several symptoms of each problem, and I suggest several possible solutions that can prevent, or correct, the problem. By laying the foundation for effective software technical reviews and avoiding these common pitfalls, you too can reap the benefits of this valuable quality practice.

Participants Don't Understand the Review Process

Symptoms: Software engineers don't instinctively know how to conduct and contribute to software reviews. Review participants may have different understandings of their roles and responsibilities, and of the activities conducted during a review. Team members may not know which of their software work products should be reviewed, when to review them, and what review approach is most appropriate in each situation.

Team members may not understand the various types of reviews that can be performed. The terms "review", "inspection", and "walkthrough" are often used interchangeably, although they are not the same beast. A lack of common understanding about review practices can lead to inconsistencies in review objectives, review team size and composition, forms used, recordkeeping, and meeting approaches. Too much material may be scheduled for a single review, because participants are not aware of realistic review rates. It may not be clear who is running a review meeting, and meetings may lose their focus, drifting from finding defects to solving problems or challenging the author's programming style. Results from these points of confusion are typically missed defects, frustration, and an unwillingness to participate in future reviews.

Solutions: Training is the best way to ensure that your team members share a common understanding of the review process. For most teams, four to eight hours of training will be sufficient, though you wish to obtain additional specialized training for those who will play the role of moderator in formal inspections. Training can be an excellent team-building activity, as all members of the group hear the same story on some technical topic and begin with a shared understanding and vocabulary.

Your group should also adopt some written procedures for how reviews are to be conducted. These

procedures will help review participants understand their roles and activities, so they can consistently practice effective and efficient reviews. Your peer review process should include procedures for both formal and informal reviews. Not all work products require formal inspection (though inspection is indisputably the most effective review method), so a palette of procedural options will let team members choose the most appropriate tool for each situation. Adopt standard forms for recording issues found during review meetings, and for recording summaries of the formal reviews that were conducted. Good resources for guidance on review procedures and forms are *Software Inspection Process* by Robert Ebenau and Susan Strauss (McGraw-Hill, 1994) and *Handbook of Walkthroughs, Inspections, and Technical Reviews, 3rd Edition* by Daniel Freedman and Gerald Weinberg (Dorset House, 1990).

Reviewers Critique the Producer, Not the Product

Symptoms: Initial attempts to hold reviews sometimes lead to personal assaults on the skills and style of the author. A confrontational style of raising issues exacerbates the problem. Not surprisingly, this makes the author feel beaten down, defensive, and resistant to legitimate suggestions that are raised or defects that are found. When authors feel personally attacked by other review participants, they will be reluctant to submit their future products for review. They may also look forward to reviewing the work of their antagonists as an opportunity for revenge.

Solutions: When helping your team begin reviews, emphasize that the correct battle lines pit the author and his peers against the defects in the work product. A review is not an opportunity for a reviewer to show how much smarter he is than the author, but rather a way to use the collective wisdom, insights, and experience of a group of peers to improve the quality of the group's products. Try directing your comments and criticisms to the product itself, rather than pointing out places the author made an error. Practice using the passive voice: "I don't see where these variables were initialized, "not "You forgot to initialize these variables"

In an inspection, the roles of the participants are well defined. One person-not the author-is the moderator and leads the inspection meeting. In the review courses I teach, students often ask why the author does not lead a formal inspection meeting. One reason is to remove the confrontational nature of describing a defect directly to the person who is responsible for the defective work product. I have found the best results come when both reviewers and author check their egos (and weapons!) at the door and focus on improving the quality of a work product.

Reviews Are Not Planned

Symptoms: On many projects, reviews do not appear in the project's work breakdown structure or schedule. If they do appear in the project plan, they may be shown as milestones, rather than as tasks. Because milestones take zero time by definition, the non-zero time that reviews actually consume may make the project appear to slip its schedule because of reviews. Another consequence of failing to plan the reviews is that potential review participants do not have time to take part when one of their peers asks them to join in.

Solutions: A major contributor to schedule overruns is inadequate planning of the tasks that must be performed. Not thinking of these tasks doesn't mean that you won't perform them; it simply means that

when you do perform them, the project will wind up taking longer than you expected. The benefits of well-executed software technical reviews are so great that project plans should explicitly show that key work products will be reviewed at planned checkpoints.

When planning a review, estimate the time required for individual preparation, the review meeting (if one is held), and likely rework. (The unceasing optimism of software developers often leads us to forget about the rework that follows most quality activities.) The only way to create realistic estimates of the time needed is to keep records from your reviews of different work products. For example, you may find that your last 20 code reviews required an average of 6 labor-hours of individual preparation time, 8 labor-hours of meeting time, and 3 labor-hours of rework. Without collecting such data, your estimates will forever remain guesses, and you will have no reason to believe that you can realistically estimate the review effort on your future projects.

Review Meetings Drift Into Problem-Solving

Symptoms: Software developers are creative problem solvers by nature. We enjoy nothing more than sinking our cerebrums into sticky technical challenges, exploring elegant solutions to thorny problems. Unfortunately, this is not the behavior we want during a technical review. Reviews should focus on finding defects, but too often an interesting defect triggers a spirited discussion about how it ought to be fixed.

When a review meeting segues into a problem-solving session, the progress of examining the product slows to a halt. Participants who aren't equally fascinated by the problem at hand may become bored and tune out. Debates ensue as to whether a proposed bug really is a problem, or whether an objection to the author's coding style indicates brain damage on the part of the reviewer. Then, when the reviewers realize the meeting time is almost up, they hastily regroup, flip through the remaining pages quickly, and declare the review a success. In reality, the material that is glossed over likely contains some major problems that will come back to haunt the development team in the future.

Solutions: The kind of reviews I'm discussing in this article have one primary purpose: to find defects in a software work product. Solving problems is usually a distraction that siphons valuable time away from the focus on error detection. One reason inspections are more effective than less formal reviews is that they have a moderator who controls the meeting, including detecting when problem-solving is taking place and bringing the discussion back on track. Certain types of reviews, such as walkthroughs, may be intended for brainstorming, exploring design alternatives, and solving problems. This is fine, but don't confuse a walkthrough with a defect-focused review such as an inspection.

My rule of thumb is that if a problem can be solved with no more than 1 minute of discussion, go for it. You have the right people in the room and they're focused on the issue. But if it looks like the discussion will take longer, remind the recorder to note the item and ask the author to pursue solutions off-line, after the meeting.

Rarely, you may encounter a show-stopper defect, one that puts the whole premise of the product being reviewed into question. Until that issue is resolved, there may be no point in completing the review. In such a case, you may choose to switch the meeting into a problem-solving mode, but then don't pretend that you completed the review as intended.

Reviewers Are Not Prepared

Symptoms: You come into work at 7:45AM and find a stack of paper on your chair with a note attached: "We're reviewing this code at 8:00AM in conference room B." There's no way you can properly examine the work product and associated materials in 15 minutes. If attendees at a review meeting are seeing the product for the first time, they may not understand the intent of the product or its assumptions, background, and context, let alone be able to spot subtle errors. Other symptoms of inadequate preparation are that the work product copies brought to the meeting aren't marked up with questions and comments, and some reviewers don't actively contribute to the discussion.

Solutions: Since about 75% of the defects found during inspections are located during individual preparation, the review's effectiveness is badly hampered by inadequate preparation prior to the meeting. This is why the moderator in an inspection begins the meeting by collecting the preparation times from all participants. If the moderator judges the preparation time to be inadequate (say, less than half the planned meeting time), she should reschedule the meeting. Make sure the reviewers receive the materials to be reviewed at least two or three days prior to the scheduled review meeting.

When reviews come along, most people don't want to interrupt their own pressing work to carefully study someone else's product. Try to internalize the fact that the time you spend reviewing a co-worker's product will be repaid by the help you'll get from your friends when your own work comes up for review. Use the average collected preparation times to help reviewers plan how much time to allocate to this important stage of the review process.

The Wrong People Participate

Symptoms: If the participants in a review do not have appropriate skills and knowledge to find defects, their review contributions are minimal. Participants who are there only to learn may benefit, but they aren't likely to improve the quality of the product. Management participation in reviews may (but doesn't always) also lead to poor review results. If the team feels the manager is counting the bugs found to hold against the author at performance appraisal time, they may hesitate to raise issues during the discussion that might make their colleague look bad.

Large review teams can also be counterproductive. I once participated in a review (ironically, of a draft peer review process) that involved 14 reviewers. A committee of 14 can't agree to leave a burning room, let alone agree on what's an error and how a sentence should be phrased. Large review teams can generate multiple side conversations that do not contribute to the review objectives and slow the pace of progress.

Solutions: Review teams having 3 to 7 participants are most effective. The reviewers should include the work product's author, the author of any predecessor or specification document, and anyone who will be a victim of the product. For example, a design review should include the designer, the author of the requirements specification, the programmer, and whoever is responsible for integration testing. On small projects, one person may play all these roles, so ask some of your peers to represent the other perspectives. It's okay to include some participants who are there primarily to learn (an important side benefit of software reviews), but focus on people who will spot bugs.

I'm not dogmatic on the issue of management participation. As a group leader, I also wrote software, so I needed to have it reviewed (thereby setting an example for the rest of the team), and I was able to contribute usefully to reviews of other team members' products. This is very much a cultural issue, dependent on the mutual respect and attitudes of the team members. A good rule of thumb is that only a first-line manager is permitted in a review, and only if it is acceptable to the author. Managers can never join in the review "just to see what's going on."

Reviewers Focus on Style, Not Substance

Symptoms: Whenever I see a defect list containing mostly style issues, I'm nervous that substantive errors have been overlooked. When review meetings turn into debates on style and the participants get heated up over indentation, brace positioning, variable scoping, and commenting, they aren't spending energy on finding logic errors and missing functionality.

Solutions: Style *can* be a defect, if excessive complexity, obscure variable names, and coding tricks make it hard to understand and maintain the code. This is obviously a value judgment: an expert programmer can understand complex and terse programs more readily than someone who has less experience. Control the style distraction by adopting standard templates for project documents and coding standards or guidelines. These will help make the evaluation of style conformance more objective. Use code reformatting tools to enforce the standards, so people can program the way they like, then convert the result to the established group conventions.

Be flexible and realistic about style. It's fine to share your ideas and experiences on programming style during a review, but don't get hung up on minor issues and don't impose your will on others. Programming styles haven't come to Earth from a font of Universal Truth, so respect individual differences when it is a matter of preference, rather than clarity.

Any software engineering practice can be performed badly, thereby failing to yield the desired benefits. Technical reviews are no different. Many organizations have enjoyed excellent results from their inspection programs, with returns on investment of up to 10 to 1. Others, though, perceive reviews to be frustrating wastes of time, usually because of these seven deadly sins, not because review are inherently time-wasters. By staying alert to these common problems of reviews and applying the solutions I have suggested here, you can help make your technical review activities be a stellar contributor to your software quality program.

(This article was originally published in *Software Development*, March 1997. It is reprinted here with permission from *Software Development* magazine.)