Lightweight Tool Support For Effective Code Reviews

Karl Wiegers
Process Impact
http://www.processimpact.com

Peter Moore
Cenqua Pty Ltd
http://www.cenqua.com

Every programmer has had the experience of struggling to find a small but elusive bug. No matter how much you study the code, you simply can't see it. So you ask a colleague to come look over your shoulder. As you explain the problem you're trying to solve, one of two things happens. Either you spot the problem yourself while explaining it, or after you describe the problem, your colleague immediately finds the likely cause. It was there all the time, staring you in the face, but you just couldn't see it. Often we're too close to our own work to spot the defects. We need to get a little help from our friends through the simple technique called peer review.

The benefits of software peer reviews have been known for decades, yet many developers don't perform them, for a variety of reasons. This white paper describes several types of software peer reviews, the benefits they can provide to any organization, and some of the reasons why developers don't perform such reviews routinely. The paper also describes the characteristics of a tool that can facilitate the way a software development team collaborates through effective peer reviews. Combining the tool with effective review techniques can help any software team improve its development effectiveness and efficiency by the early detection and removal of code errors.

What Is a "Peer Review"?

A peer review refers to any activity in which someone other than the author of a software work product examines that work product with the specific intent of finding defects. A key aspect of software peer review is that the *author* of the work is never evaluated; only the work itself is examined to find ways to make it better. Although source code is the most obvious candidate for review from the programmer's perspective, any deliverable created on a software project could potentially undergo peer review. Such deliverables include requirements specifications, architecture descriptions, designs, user interfaces, test procedures and test cases, and any kind of project plan. In short, any work product that has the potential for containing an error is a candidate for being reviewed.

The earlier in the process that a defect is discovered, the less damage it does and the cheaper it is to correct. Therefore, one of the highest leverage quality practices available to the software industry is thorough review of requirements specifications, because an error in a requirement has such far-reaching ramifications.

There are several different types of peer review approaches. They range from the very informal, ad hoc review described above—asking a colleague to help you find one specific problem—to the full rigor of a formal software inspection. Figure 1 places several peer review methods along a formality scale.[1] The most formal peer reviews, such as inspections, have several characteristics:

- Defined review objectives
- Participation by a trained team and leadership by a trained moderator
- Specific participant roles and responsibilities

[1] Wiegers, Karl E. 2002. *Peer Reviews in Software: A Practical Guide*. Boston, Mass: Addison-Wesley.

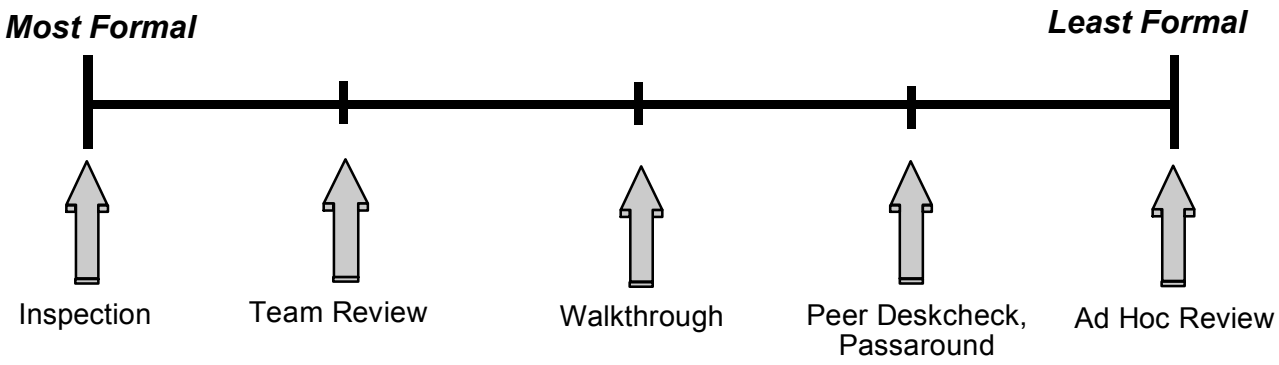

Figure 1. The peer review formality spectrum.

- A documented review procedure with explicit entry and exit criteria
- Reporting of results to management
- Tracking of defects to closure
- Recording of process and quality data

Following are brief descriptions of these different types of reviews:

Inspection:	An *inspection* follows a well-defined procedure that includes six stages: planning, overview, individual preparation, inspection meeting, rework, and follow-up. Certain participants have assigned roles: author, moderator, reader, and recorder. The reviewers use work aids such as checklists of the types of defects commonly found in certain work products to carefully examine the work product for possible errors. Data collected is used to improve the organization's inspection process and software engineering processes.
Team Review:	*Team reviews* are planned and structured but are less formal and less rigorous than inspections. The overview and follow-up inspection stages are typically omitted, and some participant roles may be combined. The author might lead a team review and the reader role is omitted. Instead, the moderator asks the reviewers if they have any issues on specific sections of the work product.
Walkthrough:	A *walkthrough* is an informal review in which the author of a work product describes it to a group of colleagues and solicits comments. The Author takes the dominant role; other specific review roles usually are not defined. Walkthroughs are informal because they typically do not follow a defined procedure, do not specify entry or exit criteria, require no reporting, and generate no metrics.
Peer Deskcheck:	During a *peer deskcheck* only one person besides the author examines the work product. A peer deskcheck can be fairly formal if the reviewer uses defect checklists, analysis methods, and standard record forms.
Passaround:	A multiple, concurrent peer deskcheck is called a *passaround*. The author delivers copies of the work product to several reviewers and collates their feedback.

Informal reviews are fully appropriate in certain situations. They are quick and cheap, do not require advanced planning or preparation, demand no organizational infrastructure, and can help the work product's author proceed on an improved course. It's important to select a review technique for each situation that fits the project's culture, time constraints, and review objectives. Making a review more complicated and time-consuming than necessary is not cost effective. Conversely, only superficially reviewing a critical piece of code could miss

major problems. It's a good idea to consider risk when selecting an appropriate review technique. Consider both the probability that a specific body of work contains defects and how severe the impact of an undiscovered defect could be. As a general guideline, use more formal reviews for high-risk work products and less formal reviews for those having lower risk.

Benefits of Peer Reviews

Many companies have learned that the results yielded by a good peer review process far outweigh the costs of performing them, particularly for inspections. As an example, the Jet Propulsion Laboratory estimated a net savings of $7.5 million from 300 inspections performed on software produced for NASA.[2] Another large company estimated an annual savings of $2.5 million due to their inspection activities, based on costs of $146 to fix a major defect found by inspection and $2,900 to fix one found by the customer. Hewlett-Packard's inspection program yielded a return on investment of ten to one.[3] Companies as diverse as Aetna Insurance Company and AT&T Bell Laboratories have reported increases in coding productivity after introducing inspections.[4] The productivity increase comes about because detecting coding errors early means that it costs less to fix them than if they weren't discovered until much later in the development process or even after delivery.

In addition to revealing defects that need to be corrected, reviews also suggest improvements the author could make in the work product. The author might not incorporate every improvement suggestion, but the review input lets him improve how he performs similar work in the future.

One of the great side benefits of reviews is education.. Code reviews enable teams to share knowledge of how their software works between developers working on different projects. This leads to better cross-fertilization of ideas and quicker identification of blind alleys that developers might have gone down. A reviewer who is not already intimately familiar with the code can often ask the naïve questions that challenge assumptions and discover shortcomings that people immersed in the project just didn't see. such informal knowledge exchange enables developers to move more readily from one project to another. There's a smaller learning curve if the developer already has had a role as a reviewer on the new project. Giving team members a look under the hood at someone else's code makes it easier for them if they need to maintain or extend that code in the future.

Reviews can be a valuable mentoring technique. Less experienced developers who have their code reviewed can learn about common gotchas to avoid, inefficiencies, and good coding idioms. Reviews also provide a technique for bringing new developers up to speed with the local coding culture, such as naming conventions and layout style.

Barriers to Performing Reviews

If peer reviews are so wonderful, why don't all developers do them already? Asking your colleagues to tell you what you've done wrong is a learned behavior, not an instinctive behavior. Having someone point out an error you've made is a threat to your ego, and developers instinctively want to protect their egos. During a review meeting, egos can get in the way if authors are defensive about possible defects that are raised or if reviewers aren't

---

[2] Ebenau, Robert G., and Susan H. Strauss. 1994. *Software Inspection Process*. New York: McGraw-Hill.

[3] Grady, Robert B., and Tom Van Slack. 1994. "Key Lessons in Achieving Widespread Inspection Use," *IEEE Software* 11(4): 46-57.

[4] Humphreys, Watts S. 1989. *Managing the Software Process*. Reading, Mass.: Addison-Wesley.

thoughtful about how they present their observations about the work product. Inappropriate behavior during a review meeting can lead to raging debates and hurt feelings, instead of quality improvements in the code.

If the author takes the lead role during a review meeting he can dominate it, turning the meeting into a presentation rather than an open and honest quest for errors. Some developers are unwilling to challenge the work done by more experienced developers, particularly in the semi-public setting of a review meeting. This is a particular problem if the author is an aggressive and domineering sort who isn't receptive to input from his colleagues. If the participants in a review have widely different skill and experience levels, some might not be able to contribute effectively while others are bored and feel their time is being wasted.

The rationale behind holding a review meeting is that the interaction between reviewers can lead the team to discover defects that no individual reviewer found on his own. This so-called synergy effect has been termed the "phantom inspector,"[5] reflecting the notion that the whole is greater than the sum of its parts. However, one of the biggest problems with review meetings is that they can easily turn into problem-solving discussions. After all, software developers are problem-solving kinds of people who love a technical challenge. Every review minute spent fixing a problem, though, is a minute not spent looking for the next problem. It's up to the review moderator to keep the meeting discussion focused on its primary objective of discovering as many potential defects as possible.

Often it's difficult for reviewers to coordinate their schedules and locations to hold a traditional review meeting. This is particularly true with geographically distributed teams. In these situations, asynchronous reviews that do not require a face-to-face meeting can work well. For example, in the review technique called a passaround, multiple reviewers contribute their comments on the work product at their own convenience. The author then reconciles comments from different reviewers and requests follow-up discussion or clarification when necessary. A tool that facilitates asynchronous review and pulls together input from multiple reviewers into a single view helps coordinate the work of developers and reviewers working in different places.

Busy developers sometimes regard code reviews as a waste of time. They might welcome the input from their colleagues, and everyone wants his work to be as good as possible. However, time spent in meetings can seem overly time-consuming and bureaucratic. Also, using paper forms to record issues brought up during the meeting seems archaic and inefficient, particularly if information from the forms must be transcribed into a database, perhaps to analyze defect patterns.

In addition, traditional peer reviews involve some process overhead. Someone must be responsible for gathering the artifacts to be reviewed and any supporting documentation and distributing the review package to the participants. Reviewers need to receive the information a few days prior to the review meeting so they can perform their individual preparation, where most defects are discovered. Meetings must be scheduled, rooms booked, and input from multiple reviewers collated. Although not overwhelming, this overhead effort inhibits some teams from performing reviews as frequently as they should.

The Tool Solution

As with many software development challenges, tools are available to assist with code reviews. Given the intrinsically collaborative nature of the code review process, it is important to be cognizant of how tools subtly (and not so subtly) change emphasis and

[5] Fagan, Michael E. 1986. "Advances in Software Inspections," *IEEE Transactions on Software Engineering* 12(7): 744-51.

behavior. Online review processes can be as varied as the offline processes being supported or replaced. Tool-assisted code reviews are probably not going to be a one-for-one replacement of a formal code inspection process, but this isn't necessarily a downside of using review tools.

Good tools can increase both the quantity and quality of reviews. To achieve this they need to be flexible and lightweight so team members don't view the tool as being part of the problem instead of part of the solution. The tools should minimize the effort devoted to administrative aspects such as scheduling meetings, encouraging attendance, and recording review comments. It should be trivial to gather and distribute artifacts, invite reviewers, and assign reviewer roles. Instead of recording issues on separate log forms, the tool should let reviewers insert their comments in context, right next to the line of code in question. The tool should also facilitate discussions among the reviewers on issues that are brought up. The tool should support—but not restrict—your organization's code review process. An effective code review support tool will include the following characteristics and capabilities:

- Make it easy to begin holding reviews.
- Impose a very low administrative burden.
- Notify participants of forthcoming reviews and review results.
- Easily collect review comments in context.
- Enable threaded discussions.
- Facilitate an established formal review process.
- Provide customizable metrics generation.
- Provide comprehensive reporting of review results.

Gathering a group of engineers in a room to constructively criticize an individual's work can be a Petri dish for self-aggrandizement and bruised egos. Moving the process online reduces the opportunities for grandstanding and domination. An individual is still able to rant or even flame, but other reviewers can skip to the next comment, thereby saving their time and emotional energy. An electronic medium does not stop people from making unconstructive or offensive comments. However, the time needed to type comments sometimes provides a chance for reflection so the reviewer can phrase his input in a more thoughtful and sensitive fashion. This helps all participants on both the giving and receiving end of the review focus on the product instead of the producer. Retaining an accurate history of the comments made and evolving lines of thought is also valuable. A permanent record discourages excessive harshness in comments.

A common problem with meeting-based reviews is that reviewers often fail to do adequate individual preparation prior to the review meeting. Hence they become passive observers or, worse, cause the review meeting to degenerate into a walkthrough that emphasizes understanding more than defect-detection. In an online review, in contrast, the preparation and reviewing activities are combined. Online reviews require each reviewer to actively examine and comment on the artifacts. When examining the list of review comments it's immediately apparent if an invited reviewer didn't contribute much input.

A key benefit of asynchronous tool-mediated reviews is the ability to perform reviews at a time and place that is convenient for each participant. This is an increasingly important benefit in this age of outsourcing, offshoring, and teleworking. Online code reviews are also a great way to fill in a few minutes when your brain is processing your direct issues. This is the antithesis of a scheduled meeting interrupting your train of thought. This encourages more reviews with a wider range of stakeholders.

A nice bonus of electronic reviews is the ability to easily record metrics that classify the kinds of defects that are raised. A tool also provides a place to store data related to the time that reviewers spend on different aspects of the review activity, including rework that the author performs to correct defects. A good tool should present statistics and give you the

ability to analyze your metrics in order to improve the review as well as your development process. Metrics are implicitly linked to the reviewees and reviewers, as well as to work products. These metrics can aid in root cause analysis, in addition to measuring the review process itself.

It is important to note that using a code review tool does not preclude holding traditional review meetings. Quite to the contrary, it should highlight opportunities where a real-time discussion will have the most benefit. These face-to-face meetings can focus on deeper issues because the simple problems have already been found and the participants are prepared. In face-to-face review meetings, the discussion often moves from reviewing to problem solving (or various other off-topic talk). In a review tool it is more obvious when a thread is moving off topic, raise that as a non-defect outcome (e.g., review of architecture required), and stay focused on reviewing the current artifacts.

Adopting a review tool might lead your team to dilute the purity of traditional meeting-driven code inspection. But is also likely to encourage people to perform more reviews. You might find that you frequently end up supplementing the online review with a traditional meeting to follow up on specific issues, drill down into a rigorous examination of one section of the code, or even to explore potential solutions. Supplementing an effective code review process with an appropriate streamlining tool can help make your reviews more effective, more efficient, and maybe even more fun.