Humanizing Peer Reviews[1]

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

Peer review—an activity in which people other than the author of a software deliverable examine it for defects and improvement opportunities—is one of the most powerful software quality tools available. Peer review methods include inspections, walkthroughs, peer deskchecks, and other similar activities. After experiencing the benefits of peer reviews for nearly fifteen years, I would never work in a team that did not perform them.

However, many organizations struggle to implement an effective review program. Many of the barriers to successful peer reviews are social and cultural in nature, not technical. This article explores some of the social and psychological aspects of having people review each other's work, ways to overcome resistance to reviews, and issues regarding management involvement. The suggestions provided here might help your peer review program succeed where others have failed.

Scratch Each Other's Back

Asking your colleagues to point out errors in your work is a learned—not instinctive—behavior. We all take pride in the work we do and the products we create. We don't like to admit that we make mistakes, we don't realize how many we make, and we don't like to ask other people to find them. Holding successful peer reviews requires us to overcome this natural resistance to outside critique of our work.

Busy practitioners are sometimes reluctant to spend time examining a colleague's work. They might be leery of a co-worker who asks for a review of his code. Questions arise: Does he lack confidence? Does he want you to do his thinking for him? "Anyone who needs their code reviewed shouldn't be getting paid as a software developer," scoff some potential review resisters. These resisters don't appreciate the value that multiple pairs of eyes can add.

In a healthy software engineering culture, team members engage their peers to improve the quality of their work and increase their productivity. They understand that time spent looking at a colleague's deliverable isn't time wasted, especially when other team members willingly reciprocate. The best software engineers I have known actively sought out reviewers, having learned early on how much they can help. Indeed, the input from many reviewers over their careers was part of what made these developers the best at what they do.

Gerald Weinberg introduced the concept of "egoless programming" in *The Psychology of Computer Programming*, originally published in 1971 and re-released in 1998. Weinberg recognized that people tie much of their perceived self-worth to their work. You can interpret a fault that someone finds in an item you created as a shortcoming in yourself as a software

[1] This paper was originally published in *STQE*, March/April 2002. It is reprinted (with modifications) with permission from Software Quality Engineering. It is adapted from *Peer Reviews in Software: A Practical Guide* by Karl E. Wiegers (Addison-Wesley, 2002).

developer, and perhaps even as a human being. To guard your ego, you don't want to know about all the errors you've made, and you might rationalize possible bugs as product features. Such staunch ego-protection presents a barrier to effective peer review, leads to an attitude of private ownership toward individual contributions within a team project, and can result in a poor quality product. Egoless programming enables an author to step back and let others point out places where improvement is needed.

Note that the term is "egoless programming," not "egoless programmer." Developers need a robust enough ego to trust and defend their work, but not so much ego that they reject suggestions for better solutions. Similarly, the egoless reviewer should have compassion and sensitivity for his colleagues, if only because their roles will be reversed one day.

A broad peer review program can only succeed in a work culture that values quality in its many dimensions, including freedom from defects, satisfaction of customer needs and business objectives, timeliness of delivery, and the possession of desirable product functionality and attributes. Recognizing that team success depends on helping each other do the best job possible, members of a healthy culture prefer to have peers—not customers—find defects. They understand that reviews are not meant to identify scapegoats for quality problems. Having a co-worker locate a defect is regarded as a "good catch," not a personal failing. In fact, reviews can motivate authors to practice superior craftsmanship because they know their colleagues will closely examine their work.

Tips for the Reviewer

The dynamics between the reviewers and the work product's author are a critical aspect of peer reviews. The author must trust and respect the reviewers to be receptive to their comments. Conversely, the reviewers must respect the author's talent and hard work. The ways in which reviewers speak to authors indicate whether their culture favors respectful collaboration or competitive antagonism.

Reviewers should focus on what they observed about the product, thoughtfully selecting the words they use to raise an issue. Saying, "I didn't see where these variables were initialized" is likely to elicit a constructive response; the more accusatory "You didn't initialize these variables" might get the author's hackles up. You might phrase your comments in the form of a question: "Are we sure that another component doesn't already provide that service?" Or, identify a point of confusion: "I didn't see where this memory block is deallocated." Direct your comments to the work product, not to the author. For example, say "This specification is missing Section 3.5 from the template" instead of "You left out section 3.5." Reviewers and authors must work together outside the reviews, so each needs to maintain a level of professionalism and mutual respect to avoid strained relationships.

While a draft of my book *Peer Reviews in Software* was undergoing peer review (naturally!), one reviewer expressed a concern by writing, "How in the world have you managed to" miss some point he thought was important. Then he added, "Good grief, Karl." I respect this reviewer's experience and value his insights, but perhaps he could have phrased that bit of feedback more tactfully. Expressing incredulity at the author's lack of understanding is not likely to make the author receptive to a reviewer's suggestion. While a reviewer might let an inappropriate comment slip out accidentally during a discussion, it's inexcusable in written feedback.

You do not want your reviews to create authors who look forward to retaliating against their tormentors. Moreover, an author who walks out of a review meeting feeling personally attacked or professionally insulted will not voluntarily submit his work for review again. Bugs are the bad guys in a review, not the author or the reviewers. The leaders of the review initiative should strive to create a culture of constructive criticism, in which team members seek to learn from their peers and do a better job the next time. Managers should encourage and reward those who initially participate in reviews and make useful contributions, regardless of the review outcomes.

Problem: Why Don’t People Do Reviews?

Lack of knowledge about reviews, cultural issues, and simple resistance to change (which often masquerades as excuses) contribute to the underuse of reviews. Many people don’t understand what peer reviews are, why they are valuable, the differences between formal and informal reviews, or when and how to perform them. Some developers and project managers don’t think their project is large enough or critical enough to need reviews, yet any piece of work can benefit from an outside perspective. There’s also a widely held perception that reviews take too much time and slow the project down. In reality, any technique that facilitates early quality improvements should shorten schedules, reduce maintenance costs, and enhance customer satisfaction. The misperception that testing is always superior to manual inspection also leads some practitioners to shun reviews.

Several cultural issues can create unpleasant review experiences. The fear of management retribution if defects are discovered can make an author reluctant to let others examine his work. Another cultural barrier is the reviewer’s attitude that the author is the most qualified person to work on his part of the system—who am I to look for errors in his work? This is a common reaction from new developers who are invited to review an experienced colleague’s deliverables. There is also the paradox that many developers are reluctant to try a new method unless the technique has been proven to work, yet the developers don’t believe the new approach works until they’ve successfully done it themselves. They don’t want to take anyone else’s word for it.

Cultural biases that run deeper than workplace attitudes can play against review participation. For instance, our educational system grades people primarily on individual performance, so collaborating is sometimes viewed as cheating. There’s an implication that if you need help, then you must not be very smart. Therefore, it’s not surprising that developers often resist asking for help through reviews. We have to overcome the ingrained culture of individual achievement and embrace the value of collaboration. In addition, authors who submit their work for scrutiny might feel their privacy is being invaded, being forced to air the internals of their work for all to see. This is threatening to some people, which is why the culture must emphasize the value of reviews as a collaborative, nonjudgmental tool for improved quality and productivity.

Then there are the excuses. People who don’t want to do reviews will expend considerable energy explaining why reviews don’t fit their culture, needs, or time constraints. Resistance often appears as NAH (Not Applicable Here) Syndrome: we don’t need no stinkin’ reviews. There is the attitude that some people’s work does not need reviewing. Some team members can’t be bothered to look at a colleague’s work. “I’m too busy fixing my own bugs to waste time finding someone else’s. Aren’t we all supposed to be doing our own work correctly?” Other developers imagine that their software prowess has moved them past the point of peer reviews. “Inspections have been around for 25 years; they’re obsolete. Our high-tech group only uses leading-edge technologies.” These excuses could reflect the team’s cultural attitudes toward quality, indicate resistance to change, or reveal a fear of peer reviews. You must address these existing barriers to establish a successful review program.

Strategy: Overcoming Resistance to Reviews

Lack of knowledge is easy to correct if people are willing to learn. A one-day class that includes a practice review session gives team members a common understanding about the process. Managers who also attend the class send a powerful signal about their commitment to reviews. Management attendance says to the team, "This is important enough for me to spend time on it, so it should be important to you, too" and "I want to understand reviews so I can help make this effort succeed."

Dealing with cultural issues requires that you understand your team's culture and how best to steer the team members toward improved software engineering practices. What values do they hold in common? Is there a shared understanding of, and commitment to, quality? What previous change initiatives have succeeded and why? Which have struggled and why? Who are the opinion leaders in the group and what are their attitudes toward reviews?

Larry Constantine described four cultural paradigms found in software organizations: *closed*, *open*, *synchronous*, and *random* (see "Work Organization: Paradigms for Project Management and Organization," *Comm. ACM*, October 1993). Understanding which paradigm your team's culture most closely resembles can give you some clues about how to introduce a peer review program.

A **closed** culture has a traditional hierarchy of authority. You can introduce peer reviews in a closed culture through a management-driven process improvement program, perhaps based on one of the Software Engineering Institute's capability maturity models.

Innovation, collaboration, and consensus decision-making characterize an **open** culture. Members of an open culture want to debate the merits of peer reviews and participate in deciding when and how to implement them. Respected leaders who have had positive results with reviews in the past can influence the group's willingness to adopt them. Such cultures might favor review meetings in which discussion of proposed solutions is common, although the inspection process emphasizes finding—not fixing—defects during meetings.

Members of a **synchronous** group are well aligned and comfortable with the status quo. Because they recognize the value of coordinating their efforts, they are probably already performing at least informal reviews. A comfort level with informal reviews makes it easier to implement a more formal inspection program.

Entrepreneurial, fast-growing, and leading-edge technology companies often develop a **random** culture populated by autonomous individuals. In random organizations, individuals who have performed peer reviews in the past might continue to hold them. The other team members probably don't have the patience for reviews, but they might change their minds if quality problems from the resulting chaos burn them badly enough.

Whichever category best describes your work culture, people will want to know what benefits any new process will provide to them personally. Table 1 lists some of the benefits that various project team members might reap from reviewing major life cycle deliverables. Not only will the team benefit, but the customers come out ahead as well. They'll receive a timely product that is more robust and reliable, better meets their needs, and increases their own productivity.

Table 1. Benefits from peer reviews for different project roles.

Project Role	***Possible Benefits from Peer Reviews***
Developer	• Less time spent performing rework • Increased programming productivity • Better techniques learned from other developers • Reduced unit testing and debugging time • Less debugging during integration and system testing • Exchanging of information about components and overall system with other team members
Project Manager	• Shortened product development cycle time • Increased chance of shipping the product on schedule • Reduced field service and customer support costs • Reduced lifetime maintenance costs, freeing resources for new development projects • Improved teamwork, collaboration, and development effectiveness • Reduced impact from staff turnover through cross-training of team members • Better and earlier insight into project risks and quality issues
Maintainer	• Fewer production support demands, leading to a reduced maintenance backlog • More robust designs that tolerate change • Conformance of work products to team standards • Better structured and documented work products that are easy to understand and modify • Better understanding of the product from having participated in design and code reviews during development
Quality Assurance Manager	• Ability to judge testability of product features under development • Shortened system-testing cycles and less retesting • Ability to use review data when making release decisions • Education of quality engineers about the product • Ability to anticipate quality assurance effort needed
Requirements Analyst	• Earlier correction of missing or erroneous requirements • Fewer infeasible and untestable requirements because of developer and test engineer input during reviews
Test Engineer	• Ability to focus on finding subtle defects because product is of higher initial quality • Fewer defects that block continued testing • Improved test design and test cases that smooth out the testing process

Although there are many individual rewards from conducting peer reviews, also address the larger question of "What's in it for *us*?" Sometimes when you're asked to change the way you work, your immediate personal reward is small, although the team as a whole might benefit in a big way. I might not get three hours of benefit from spending three hours reviewing someone else's code. However, the other developer might avoid ten hours of debugging effort later in the project and we might ship the product sooner than we would have otherwise.

Influential resisters who come to appreciate the value of reviews might persuade other team members to try them, too. A quality manager once encountered a developer named Judy who was opposed to "time-sapping" inspections. After participating under protest, Judy quickly saw the power of the technique and became the group's leading convert. Since Judy had some influence with her peers, she helped turn developer resistance toward inspections into acceptance. Judy's project team ultimately asked the quality manager to help them hold even *more* inspections. Engaging developers in an effective inspection program helped motivate them to try some other software quality practices, too.

Management Involvement

The attitudes and behaviors that managers exhibit affect how well reviews will work in an organization. While managers want to deliver quality products, they also feel pressure to release products quickly. Managers need to learn about peer reviews and their benefits so they can build the reviews into project plans, allocate resources, and communicate their commitment to reviews to the team.

Watch out for culture killers, such as singling out certain developers for the humiliating "punishment" of having their work reviewed. For example, my colleague Phil once told me that his manager demanded code reviews whenever a project was in trouble, with the unstated objective of finding a scapegoat. Unfortunately, Phil was the first to fall victim to such a review. The review team found only minor issues, but the manager then went around complaining that the project was late because Phil's code was full of bugs! Understandably, Phil soon quit this job.

On a similar note, I recently heard from a quality manager at a company that had operated a successful inspection program for two years. Then the development manager announced that finding more than five bugs during an inspection would count against the author at performance evaluation time. Naturally, this made the development team members very nervous. It conveyed the erroneous impression that the point of the inspection was to punish people for making mistakes. Such evaluation criminalizes the mistakes that we all make and pits team members against each other. This misapplication of inspection data could lead to numerous dysfunctional outcomes:

1. Developers might not submit their work for inspection to avoid being punished for their results. They might refuse to inspect a peer's work to avoid contributing to someone else's punishment.

2. Inspectors might not point out defects during the inspection, instead reporting them to the author offline so they aren't tallied against the author. This undermines the open focus on quality that should characterize peer review.

3. Inspection teams might endlessly debate whether something really is a defect, because defects count against the author while issues or simple questions do not.

4. The team's inspection culture will develop an unstated goal of finding few defects, rather than revealing as many as possible.

5. Authors might inspect very small pieces of work so they don't find more than five bugs in any one inspection. This leads to inefficient and time-wasting inspections.

When presented with these risks, reasonable managers will rethink their intention to misuse the review data. If an unreasonable manager insists on using the data in this way, the team won't be successful with reviews. Managers may legitimately expect developers to submit their work for review and to review deliverables that others create. However, managers must not evaluate individuals based on the number of defects found during those reviews.

Without visible and sustained commitment to peer reviews from management, only those practitioners who believe reviews are important will perform them. Management commitment goes far beyond providing verbal support or giving team members permission to hold reviews. Figure 1 lists several clear signs of management commitment to peer reviews. If too many of these indicators are missing in your organization, your review program will likely struggle.

Figure 1. Ten Signs of Management Commitment to Peer Reviews.

1. Providing the resources and time to develop, implement, and sustain an effective review process.

2. Setting policies, expectations, and goals about review practice.

3. Ensuring that project schedules include the time needed to perform reviews.

4. Making training available to the participants and attending the training themselves.

5. Never using review results to evaluate the performance of individuals.

6. Holding people accountable for participating in reviews and for contributing constructively to them.

7. Publicly rewarding the early adopters of reviews to reinforce desired behaviors.

8. Running interference with other managers and customers who challenge the need for reviews.

9. Respecting the judgment of an inspection team's appraisal of a document's quality.

10. Asking for status reports on how the program is working, what it costs, and the team's benefits from reviews.

Review Your Way to Success

If you're serious about the quality of your work, you'll accept that you make mistakes, seek the counsel of your compatriots in finding them, and willingly review your colleagues' work products. You will set aside your ego so you can benefit from the experience and perspective of your technical associates. When you have internalized the benefits of peer reviews, you won't feel comfortable unless someone else carefully examines any significant deliverable you create.

On the other hand, even if you care about quality, you'll be reluctant to participate in reviews if your environment is not supportive of such quality practices. Perhaps you can lead by example, inviting other team members to look at your deliverables. Maybe you can contribute to leading a nascent review program and help steer managers and practitioners to the effective and routine application of reviews. But if holding reviews would be too unpleasant in a hostile environment, you might want to find a culture that better supports your personal quality philosophy.

[SIDEBAR] Reviews without Borders

Increasingly, software projects involve teams that collaborate across multiple corporations, time zones, continents, organizational and national cultures, and native languages. The review issues include both communication logistics and cultural factors; the latter pose the greater challenge. Different cultures have different attitudes toward critiquing work performed by another team member or by a supervisor. People from certain nations or geographical regions are comfortable with a more aggressive interaction style than are others, who avoid confrontation. A review participant from the more reserved community might feel that someone from the assertive domain is dominating the review, while an assertive participant wonders why his quiet counterpart isn't contributing to the discussion. If you face such a multicultural challenge, learn about ways to get members of different cultures to collaborate and adjust your expectations about peer reviews.

One company encountered cultural barriers on a project that involved collaborating development teams in Singapore and the Netherlands (see Erik P.W.M. Van Veenendaal's "Practical Quality Assurance for Embedded Software," *Software Quality Professional*, June 1999). Developers in Singapore were not accustomed to having others comment on their work. They could take well-meaning comments personally, especially if they were presented semi-publicly during an inspection meeting. To deal with this, the company matched a co-author from the Netherlands with each work product from Singapore and held all inspections in the Netherlands. This approach succeeded, but it sidestepped the underlying cultural issue and essentially permitted the Singapore developers to avoid engagement in the inspections.

When you plan reviews for cross-cultural development projects, be aware of these interaction differences and consider which review approaches will work best. Discuss these sensitive issues with review participants to make everyone aware of how their differences will affect the review process. If the participants are geographically separated, hold an initial face-to-face training session to surface the cultural and communication factors so the team can determine how best to function when the team members are separated.