Chapter 2. A Little Help from Your Friends

Asking other people to point out errors in your work is a learned—not instinctive—behavior. We all take pride in the work we do. We don't like to admit we make mistakes, we don't realize how many we make (or we would correct them ourselves), and we don't like to ask someone else to find them. If you're going to hold successful peer reviews, you have to overcome this natural resistance to outside critique of your work.

Peer reviews are as much a social interaction as a technical practice. Instilling a review program into an organization requires an understanding of that organization's culture and the values its members hold. Managers need to believe that the time spent on reviews is a sound business investment so they will make time available for the team to hold reviews. You need to understand why certain people resist submitting their work to scrutiny by their colleagues and address that resistance. You also must educate the team and its managers about the peer review process, appropriate behavior during reviews, and the benefits that getting a little help from their friends can provide both to individuals and to the organization.

Scratch Each Other's Backs

Busy practitioners are sometimes reluctant to spend time examining a colleague's work. You might be leery of a coworker who asks you to review his code. Does he lack confidence? Does he want you to do his thinking for him? "Anyone who needs his code reviewed shouldn't be getting paid as a software developer," scoff some review resisters.

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 the time they spend looking at a colleague's work product is repaid when other team members examine their own deliverables. The best software engineers I have known actively sought out reviewers. Indeed, the input from many reviewers over their careers was part of what made these developers the best.

Gerald Weinberg introduced the concept of "egoless programming" in 1971 in *The Psychology of Computer Programming*, which was reissued in 1998 (Weinberg 1998). Egoless programming addresses the fact that people tie much of their perception of self-worth to their work. You can interpret a fault someone finds in an item you created as a shortcoming in yourself as a software 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. Your ego might be so protective that you deny the possibility that you made errors and attempt to rationalize possible bugs into features.

Such staunch ego-protection presents a barrier to effective peer review, leads to an attitude of private ownership of an individual's contributions to 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 in a product he created. Practitioners of egoless programming also understand that their products should be easy for others to understand. In contrast, some programmers enjoy writing obscure, clever code that only they can understand, with the notion that this makes them somehow superior to others who struggle to comprehend it. A manager who values egoless programming will encourage a culture of collaborative teamwork, shared rewards for success, and the open exchange of knowledge among team members.

Note that the term is “egoless programming,” not “egoless programmer.” People are entitled to protect their egos. Developers need a robust enough ego to trust and defend their work, but not so much ego that they reject suggestions for better solutions. Software professionals take pride in the things they create. However, they also recognize that people make mistakes and can benefit from outside perspectives. The egoless reviewer has compassion and sensitivity for his colleagues, if for no reason other than that their roles will be reversed one day.

Reviews and Team Culture

While individual participants can always benefit from a peer review, a broad review program can succeed only in a culture that values quality. “Quality” has many dimensions, including freedom from defects, satisfaction of customer needs, timeliness of delivery, and the possession of desirable product functionality and attributes. Members of a software engineering culture regard reviews as constructive activities that help both individuals and teams succeed. They understand that reviews are not intended to identify inferior performers or to find scapegoats for quality problems.

Reviews can result in two undesirable attitudes on the part of the work product’s author. Some people become lax in their work because they’re relying on someone else to find their mistakes, just as some programmers expect testers to catch their errors. The author is ultimately responsible for the product; a review is just an aid to help the author create a high-quality deliverable. Sometimes when I’m reading a draft of an article or book chapter I’ve written, I hear a little voice telling me that a section is incorrect or awkwardly phrased. I used to tell myself, “I’ll give it to the reviewers and see what they think.” Big mistake: the reviewers invariably disliked that clumsy section. Now whenever I hear that little voice, I fix the problem before I waste my reviewers’ time.

The other extreme to avoid is the temptation to perfect the product before you allow another pair of eyes to see it. This is an ego-protecting strategy: you won’t feel embarrassed about your mistakes if no one else sees them. I once managed a developer who refused to let anyone review her code until it was complete and as good as she could make it—fully implemented, tested, formatted, and documented. She regarded a review as a seal of approval rather than as the in-process quality-improvement activity it really is.

Such reluctance has several unfortunate consequences. If your work isn’t reviewed until you think it’s complete, you are psychologically resistant to suggestions for changes. If the program runs, how bad can it be? You are likely to rationalize away possible bugs because you believe you’ve finished and you’re eager to move on to the next task. Relying on your own deskchecking and unit testing ignores the greater efficiency of a peer review for finding many defects.

At the same time, the desire to show our colleagues only our best side can become a positive factor. Reviews motivate us to practice superior craftsmanship because we know our coworkers will closely examine our work. In this indirect way, peer reviews lead to higher quality. One of my fellow consultants knows a quality engineer who began to present his team with summaries of defects found during reviews, without identifying specific work products or authors. The team soon saw a decrease in the number of bugs discovered during reviews. Based on what he knew about the team, my colleague concluded that authors created better products after they learned how reviews were being used on the project and knew what kinds of defects to

look for. Reviews weren't a form of punishment but stimulated a desire to properly complete a body of work.

The Influence of Culture

In a healthy software engineering culture, a set of shared beliefs, individual behaviors, and technical practices define an environment in which all team members are committed to building quality products through the effective application of sensible processes (Wiegers 1996a). Such a culture demands a commitment by managers at all levels to provide a quality-driven environment. Recognizing that team success depends on helping each other do the best possible job, members of a healthy culture prefer to have peers, not customers, find software defects. Having a coworker locate a defect is regarded as a "good catch," not as a personal failing.

Peer reviews have their greatest impact in a healthy software culture, and a successful review program contributes strongly to creating such a culture. Prerequisites for establishing and sustaining an effective review program include

- Defining and communicating your business goals for each project so reviewers can refer to a shared project vision
- Determining your customers' expectations for product quality so you can set attainable quality goals
- Understanding how peer reviews and other quality practices can help the team achieve its quality goals
- Educating stakeholders within the development organization—and, where appropriate, in the customer community—about what peer reviews are, why they add value, who should participate, and how to perform them
- Providing the necessary staff time to define and manage the organization's review process, train the participants, conduct the reviews, and collect and evaluate review data

The dynamics between the work product's author and its reviewers are critical. The author must trust and respect the reviewers enough to be receptive to their comments. Similarly, the reviewers must show respect for the author's talent and hard work. Reviewers should thoughtfully select the words they use to raise an issue, focusing on what they observed about the product. Saying, "I didn't see where these variables were initialized" is likely to elicit a constructive response, whereas "You didn't initialize these variables" might get the author's hackles up. The small shift in wording from the accusatory "you" to the less confrontational "I" lets the reviewer deliver even critical feedback effectively. Reviewers and authors must continue to work together outside the reviews, so they all need to maintain a level of professionalism and mutual respect to avoid strained relationships.

An author who walks out of a review meeting feeling embarrassed, personally attacked, or professionally insulted will not voluntarily submit work for peer review again. Nor do you want reviews to create authors who look forward to retaliating against their tormentors. The bad guys in a review are the bugs, not the author or the reviewers, but it takes several positive experiences to internalize this reality. 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 to do a better job the next time. To

accelerate this culture change, managers should encourage and reward those who initially participate in reviews, regardless of the review outcomes.

Reviews and Managers

The attitude and behavior that managers exhibit toward reviews affect how well the reviews will work in an organization. Although managers want to deliver quality products, they also feel pressure to release products quickly. They don't always understand what peer reviews or inspections are or the contribution they make to shipping quality products on time. I once encountered resistance to inspections from a quality manager who came from a manufacturing background. He regarded inspections as a carryover from the old manufacturing quality practice of manually examining finished products for defects. After he understood how software inspections contribute to quality through early removal of defects, his resistance disappeared.

Managers need to learn about peer reviews and their impact on the organization so they can build the reviews into project plans, allocate resources for them, and communicate their commitment to reviews to the team. If reviews aren't planned, they won't happen. Managers also must be sensitive to the interpersonal aspects of peer reviews. Watch out for known culture killers, such as managers singling out certain developers for the humiliating "punishment" of having their work reviewed.

Without visible and sustained commitment to peer reviews from management, only those practitioners who believe reviews are important will perform them. Management commitment to any engineering practice is more than providing verbal support or giving team members permission to use the practice. Figure 2–1 lists eleven signs of management commitment to peer reviews.

Eleven 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. Maintaining the practice of reviews even when projects are under time pressure
4. Ensuring that project schedules include time for reviews
5. Making training available to the participants and attending the training themselves
6. Never using review results to evaluate the performance of individuals
7. Holding people accountable for participating in reviews and for contributing constructively to them
8. Publicly rewarding the early adopters of reviews to reinforce desired behaviors
9. Running interference with other managers and customers who challenge the need for reviews
10. Respecting the review team's appraisal of a document's quality
11. Asking for status reports on how the program is working, what it costs, and the team's benefits from reviews

Figure 2–1. Eleven Signs of Management Commitment to Peer Reviews

To persuade managers about the value of reviews, couch your argument in terms of what outcomes are important to the manager's view of success. Published data

convinces some people, but others want to see tangible benefits from a pilot or trial application in their own organization. Still other managers will reject both logical and data-based arguments for reviews and simply say no. In this case, keep in mind one of my basic software engineering cultural principles—"Never let your boss or your customer talk you into doing a bad job"—and engage your colleagues in reviews anyway (perhaps quietly, to avoid unduly provoking your managers).

A dangerous situation arises when a manager wishes to use data collected from peer reviews to assess the performance of the authors (Lee 1997). Software metrics must *never* be used to reward or penalize individuals. The purpose of the data you collect from reviews is to better understand your development and quality processes, to improve processes that aren't working well, and to track the impact of process changes. Using defect data from inspections to evaluate 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 (Austin 1996).

I recently heard from a quality manager at a company that had operated a successful inspection program for two years. The development manager had just announced his intention to use inspection data as input to the performance evaluations of the work product authors. Finding more than five bugs during an inspection would count against the author. Naturally, this made the development team members very nervous. It conveyed the erroneous impression that the purpose of inspections is to punish people for making mistakes or to find someone to blame for troubled projects. This misapplication of inspection data could lead to numerous dysfunctional outcomes, including the following:

1. To avoid being punished for their results, developers might not submit their work for inspection. 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 telling the author about them offline so they aren't tallied against the author. Alternatively, developers might hold "pre-reviews" to filter out bugs unofficially before going through a punitive inspection. This undermines the open focus on quality that should characterize inspection. It also skews any metrics you're legitimately tracking from multiple inspections.

3. Inspection teams might debate whether something really is a defect, because defects count against the author, and issues or simple questions do not. This could lead to glossing over actual defects.

4. The team's inspection culture might develop an implicit goal of finding few defects rather than revealing as many as possible. This reduces the value of the inspections without reducing their cost, thereby lowering the team's return on investment from inspections.

5. Authors might hold many inspections of small pieces of work to reduce the chance of finding more than five bugs in any one inspection. This leads to inefficient and time-wasting inspections. It's a kind of gamesmanship, doing the minimum to claim you have had your work inspected but not properly exploiting the technique.

These potential problems underscore the risks posed to an inspection program by using inspection data to evaluate individuals. Such evaluation criminalizes the mistakes we all make and pits team members against each other. It motivates participants to manipulate the process to avoid being hurt by it. If I were a developer in this situation, I would encourage management to have the organization's peer review coordinator (see Chapter 10) summarize defects collected from multiple inspections so the defect counts aren't linked to specific authors. If management insisted on using defect counts for performance appraisal, I would refuse to participate in inspections. Managers may legitimately expect developers to submit their work for review and to review deliverables that others create. However, a good manager doesn't need defect counts to know who the top contributors are and who is struggling.

When inspection metrics were introduced into one organization, a manager happily exclaimed, "This data will help me measure the performance of my engineers!" After the inspection champion explained the philosophy of software measurement to him, the manager agreed not to see the data from individual inspections. He publicly described the inspection process as a tool to help engineers produce better products. He told the engineers he would not view the individual inspection measures because he was interested in the big picture, the overall efficiency of the software engineering process. This manager's thoughtful comments helped defuse resistance to inspection measurement in his organization.

Why People Don't Do Reviews

If peer reviews are so great, why isn't everybody already doing them? Factors that contribute to the under use of reviews include lack of knowledge about reviews, cultural issues, and simple resistance to change, often masquerading as excuses. If reviews aren't a part of your organization's standard practices, understand why so you know what must change to make them succeed.

Many people don't understand what peer reviews are, why they are valuable, the differences between informal reviews and inspections, or when and how to perform reviews. Education can solve this problem. Some developers and project managers don't think their projects are large enough or critical enough to need reviews. However, any body of work can benefit from an outside perspective.

The misperception that testing is always superior to manual examination also leads some practitioners to shun reviews. Testing has long been recognized as a critical activity in developing software. Entire departments are dedicated to testing, with testing effort scheduled into projects and resources allocated for testing. Organizations that have not yet internalized the benefits of peer reviews lack an analogous cultural imperative and a supporting infrastructure for performing them.

A fundamental cultural inhibitor to peer reviews is that developers don't recognize how many errors they make, so they don't see the need for methods to catch or reduce their errors. Many organizations don't collect, summarize, and present to all team members even such basic quality data as the number of errors found in testing or by customers. Authors who submit their work for scrutiny might feel that their privacy is being invaded, that they're 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.

Previous unpleasant review experiences are a powerful cultural deterrent. The fear of management retribution or public ridicule if defects are discovered can make authors reluctant to let others examine their work. In poorly conducted reviews, authors can feel as though they—not their work—are being criticized, especially if personality conflicts already exist between specific individuals. Another cultural barrier is the attitude that the author is the most qualified person to examine his part of the system ("Who are you to look for errors in my work?"). Similarly, a common reaction from new developers who are invited to review the work of an experienced and respected colleague is, "Who am I to look for errors in his work?"

Traditional mechanisms for adopting improved practices are having practitioners observe what experienced role models do and having supervisors observe and coach new employees. In many software groups, though, each developer's methods remain private, and they don't have to change the way they work unless they wish to (Humphrey 2001). Paradoxically, many developers are reluctant to try a new method unless it has been proven to work, yet they don't believe the new approach works until they have successfully done it themselves. They don't want to take anyone else's word for it.

And then there are the excuses. Resistance often appears as NAH (not applicable here) syndrome (Jalote 2000). People who don't want to do reviews will expend considerable energy trying to explain why reviews don't fit their culture, needs, or time constraints. One excuse is the arrogant 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?" "It's not my problem if Jack has bugs in his code." Other developers imagine that their software prowess has moved them beyond needing peer reviews. "Inspections have been around for 25 years; they're obsolete. Our high-tech group uses only leading-edge technologies."

Protesting that the inspection process is too rigid for a go-with-the-flow development approach signals resistance to a practice that is perceived to add bureaucratic overhead. Indeed, the mere existence of a go-with-the-flow development process implies that long-term quality isn't a priority for the organization. Such a culture might have difficulty adopting formal peer reviews, although informal reviews might be palatable.

Overcoming Resistance to Reviews

To establish a successful review program, you must address existing barriers in the categories of knowledge, culture, and resistance to change. Lack of knowledge is easy to correct if people are willing to learn. My colleague Corinne found that the most vehement protesters in her organization were already doing informal reviews. They just didn't realize that a peer deskcheck is one type of peer review (see Chapter 3). Corinne discussed the benefits of formalizing some of these informal reviews and trying some inspections. A one-day class that includes a practice inspection gives team members a common understanding about peer reviews. Managers who also attend the class send powerful signals 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 you to understand your team's culture and how best to steer the team members toward improved software engineering practices

(Bouldin 1989; Caputo 1998; Weinberg 1997; Wiegers 1996a). What values do they hold in common? Do they share an understanding of—and a 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* (Constantine 1993). 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. A management decree that projects will conduct reviews might succeed in a closed culture, but not in other types of organizations.

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 prefer review meetings that include discussions of proposed solutions rather than inspections, which emphasize 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 eases implementation of an inspection program.

Entrepreneurial, fast-growing, and leading-edge companies often develop a random culture populated by autonomous individuals who like to go their own ways. In random organizations, individuals who have performed peer reviews in the past might continue to hold them. The other team members might not have the patience for reviews, although they could change their minds if quality problems from chaotic projects burn them badly enough.

However you describe your culture, people will want to know what benefits a new process will provide to them personally. A better way to react to a proposed process change is to ask, "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.

Table 2–1 identifies some benefits various project team members might reap from reviewing major life-cycle deliverables. Of course, the customers also come out ahead. They receive a timely product that is more robust and reliable, better meets their needs, and increases their productivity. Higher customer satisfaction leads to business rewards all around.

Table 2–1. Benefits from Peer Reviews for Project Roles

Project Role	***Possible Benefits from Peer Reviews***
• Developer	• Less time spent performing rework • Increased programming productivity • Confidence that the right requirements are being implemented • 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 the overall system with other team members
• Development Manager	• Shortened product development cycle time • Reduced field service and customer support costs • Reduced lifetime maintenance costs, freeing resources for new development projects • Improved teamwork, collaboration, and development effectiveness • 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 • More maintainable and better 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
• Project Manager	• Increased likelihood that product will ship on schedule • Earlier visibility of quality issues • Reduced impact from staff turnover through cross-training of team members
• Quality Assurance Manager	• Ability to judge the 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

Arrogant developers who believe reviews are beneath them might enjoy getting praise and respect from coworkers as they display their superior work during reviews. If influential resisters come to appreciate the value of peer reviews, they 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. Because 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.

In another case, a newly hired system architect who had experienced the benefits of inspections in his previous organization was able to overcome resistance from members of his new team. The data this group collected from their inspections backed up the architect's conviction that they were well worth doing.

Peer Review Sophistication Scale

Figure 2–2 depicts a scale of an organization's sophistication in its practice of software peer reviews (for a similar scale, see Grady 1997). The value added to the organization is greater at the higher stages. Use this scale to calibrate your organization's current review performance and see what it would take to enhance it.

In the worst case (stage 0), no one in the organization performs reviews. Stage 1 is a small advance, with team members holding impromptu "over-the-shoulder" reviews. Perhaps the team members haven't heard about peer reviews or don't think they have time to do them. They might have rejected reviews as inappropriate for their project for some reason.

At stage 2, team members periodically hold unstructured reviews. The participants might not realize there are different types of peer reviews. They have not adopted a common review vocabulary or process. They might hold a walkthrough or other informal review and call it an inspection, even though it did not conform to an actual inspection process. The team's review objectives include both finding defects and exchanging knowledge.

By stage 3, the project team holds reviews routinely. They are built into the project schedule and the team understands how to perform structured, formal reviews such as inspections. The organization has adopted a peer review process that incorporates multiple review methods, using standard forms and defect checklists. You don't need to start at the bottom of the scale, so aim to begin at least at stage 3, incorporating inspections into routine practice. This will keep you from getting stuck at a lower level and missing out on the full benefits of formal peer reviews.

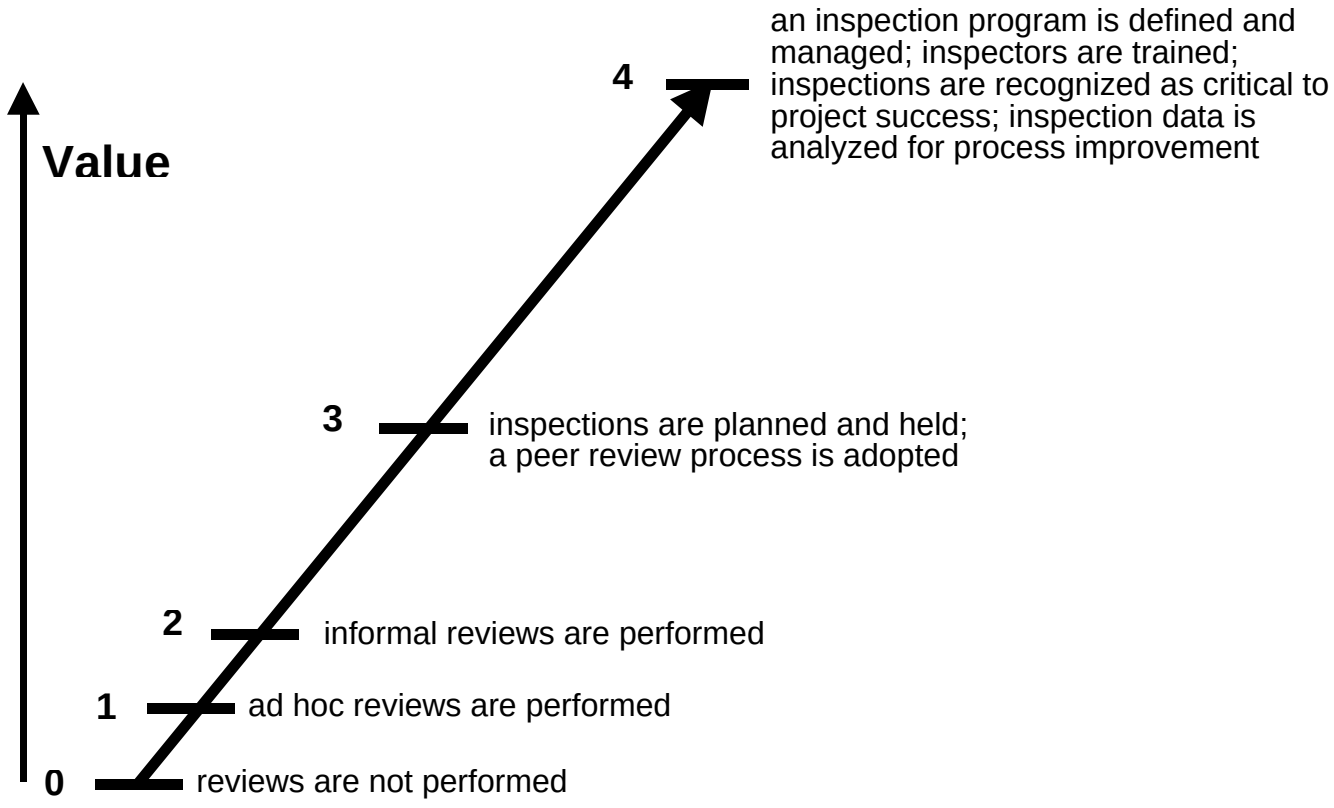

Figure 2-2. Peer review sophistication scale.

The most successful organizations reach stage 4, which represents a paradigm shift to a new way for the organization to create software products. Stage 4 organizations have established an official inspection program, staffed with a peer review coordinator and managed by a peer review process owner (see Chapter 10). They have identified the various kinds of work products that will be inspected. All participants and managers are trained in inspection. The review coordinator verifies that inspections are conducted as scheduled and collects data from them. These data are analyzed for product quality assessment, process improvement, and defect prevention. Inspections are recognized as critical contributors to project success, and the team members would not be comfortable working in an environment where peer review was not a standard practice.

Planning for Reviews

If you don't plan reviews as project tasks and allocate resources to them, they can appear to slow the project down, as does any unanticipated work. Your team can hold informal, ad hoc reviews whenever someone desires constructive input from coworkers. However, frequent unplanned reviews will drain time from the reviewers, who might be less likely to request or participate in these informal reviews.

Incorporate formal reviews into the project's schedule or work breakdown structure. A well-defined software development life cycle itemizes specific exit criteria for key phase deliverables, including passing an appropriate peer review. Figure 1–1 illustrated the major project checkpoints at which you should schedule reviews. Some teams use planning checklists of the tasks required for common project activities, such as implementing a module or an object class. Include reviews on such checklists. Also conduct interim reviews of major deliverables prior to completion to ensure they are meeting their quality goals. Informal reviews can let you judge whether a deliverable is ready for inspection and can serve as quick quality filters from an outside viewpoint.

The effort you devote to peer reviews might seem like extra overhead, but it is not really additional time in your project schedule. Think of it as a reallocation of effort you would otherwise spend on testing and the pervasive rework of debugging, patching in missed requirements, and so on. Keeping records of reviews and their benefits will help you judge the appropriate level of investment needed to meet your project's quality goals.

Project planners sometimes treat reviews as milestones, as shown in Figure 2–3. However, in project-planning terms, a milestone is a state, not an activity. Milestones have a duration of zero time and consume no resources, so treating reviews as tasks in your plan, as Figure 2–4 illustrates, is more accurate. The milestone is reached when you deem that the deliverable has passed the review. If you treat reviews as milestones, the project schedule can appear to slip when you perform reviews, because the effort they require was not anticipated. Depending on the kind of reviews you hold, review tasks will include effort for individual preparation, review meetings, or both. You should also plan to perform rework after every quality control activity, but your team will spend less time on rework as its development practices improve.

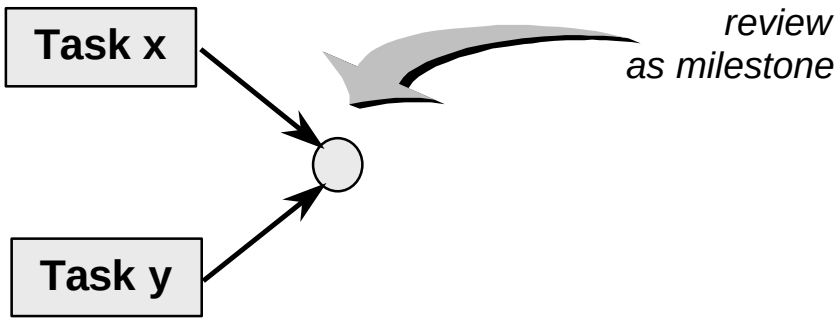

Figure 2-3. WRONG: Review treated as a milestone.

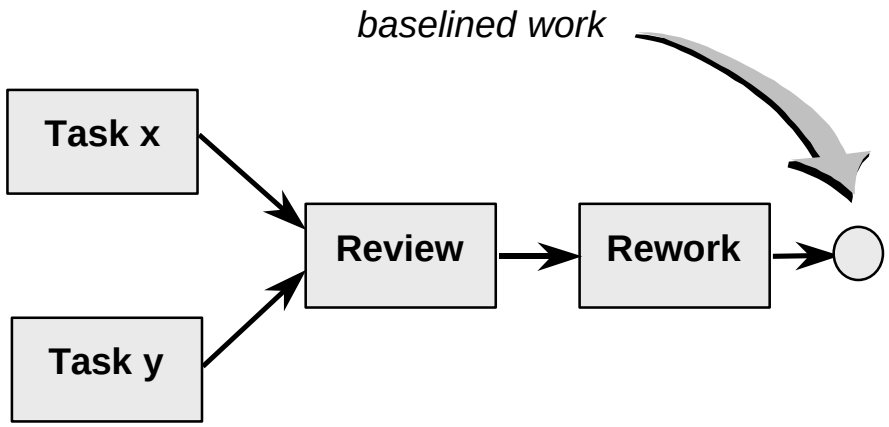

Figure 2-4. RIGHT: Review and rework treated as tasks.

How can you estimate how much time to plan for preparation, review meetings, and rework? If you keep even simple records of the time your team members actually spend on these activities, you can make better estimates for future reviews. Data on the most effective rates of coverage of material during preparation and in review meetings will help you judge the time needed for these review stages (see Chapter 5). Without such data, you are never estimating—you're guessing.

Guiding Principles for Reviews

All peer reviews should follow several basic principles to make them powerful contributors to product quality and team culture (Wiegers 1996a). First, ***check your egos at the door***. There are two aspects to being egoless. As an author, keep an open mind and be receptive to suggestions for improvements. Avoid the temptation to argue every point raised, defend your decisions to the death, or explain away errors. As a reviewer, remember that you're not trying to show how much smarter you are than the author.

Another useful guideline is to ***keep the review team small***, generally between three and seven participants. Larger teams make the review more expensive without adding proportionate value. They slow the rate at which the group can cover material in a review meeting. Large groups are prone to distracting side conversations and can easily go off on time-wasting tangents. Chapter 12 suggests what to do if a lot of people wish to participate in a review.

The prime objective of a peer review is defect detection, so strive to ***find problems during reviews, but don't try to solve them***. Technical people like to tackle challenging problems; that's why we're in the software business. However, the author should fix the identified problems *after* the review meeting, working with selected reviewers on specific issues to reap the benefits of another's experience. Formal reviews, such as inspections, include a moderator or review leader role. As described in Chapter 5, the inspection moderator is responsible for keeping the meeting focused on finding defects and for limiting problem-solving discussions to just a minute or two. If

you don't use a moderator, the participants will have to monitor themselves so the meeting doesn't derail into an extended brainstorming session on the first bug found.

Another guiding principle is to ***limit review meetings to about two hours***. My friend Matt once pointed out to me that "the mind cannot absorb what the body cannot endure." When you are distracted by physical discomfort or exhaustion, you're no longer an effective reviewer. Take a short break halfway through a long review meeting, and come back tomorrow to finish if you didn't properly cover all the material in the first meeting. Scope the work into logical chunks that the team can examine in one to two hours, based on your organization's historical rates for reviewing or inspecting different types of work products (see Chapter 5).

To use the team's time as efficiently as possible, ***require advance preparation*** for formal reviews. During informal review meetings, participants come in cold and listen to the author describe his work. Serious defect-hunting efforts, such as inspection, demand that the meeting participants have already examined the product on their own to understand it and find issues to raise during the meeting. The participants need to receive the review materials several days prior to the meeting to give them time to prepare for the inspection meeting.

Being sensitive to the human and cultural issues of peer reviews and following these basic guidelines will maximize the contribution reviews make to your development and maintenance projects.