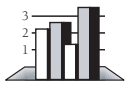
21.3 Formal Inspections

Further Reading If you want to read the original article on inspections, see "Design and Code Inspections to Reduce Errors in Program Development" (Fagan 1976).

An inspection is a specific kind of review that has been shown to be extremely effective in detecting defects and to be relatively economical compared to testing. Inspections were developed by Michael Fagan and used at IBM for several years before Fagan published the paper that made them public. Although any review involves reading designs or code, an inspection differs from a run-of-the-mill review in several key ways:

- Checklists focus the reviewers' attention on areas that have been problems in the past.
- The inspection focuses on defect detection, not correction.
- Reviewers prepare for the inspection meeting beforehand and arrive with a list of the problems they've discovered.
- Distinct roles are assigned to all participants.
- The moderator of the inspection isn't the author of the work product under inspection.
- The moderator has received specific training in moderating inspections.
- The inspection meeting is held only if all participants have adequately prepared.
- Data is collected at each inspection and is fed into future inspections to improve them.
- General management doesn't attend the inspection meeting unless you're inspecting a project plan or other management materials. Technical leaders might attend.

What Results Can You Expect from Inspections?



HARD DATA

Individual inspections typically catch about 60 percent of defects, which is higher than other techniques except prototyping and high-volume beta testing. These results have been confirmed numerous times at various organizations, including Harris BCSD, National Software Quality Experiment, Software Engineering Institute, Hewlett Packard, and so on (Shull et al 2002).

The combination of design and code inspections usually removes 70–85 percent or more of the defects in a product (Jones 1996). Inspections identify error-prone classes early, and Capers Jones reports that they result in 20–30 percent fewer defects per 1000 lines of code than less formal review practices. Designers and coders learn to improve their work through participating in inspections, and inspections increase productivity by about 20 percent (Fagan 1976, Humphrey 1989, Gilb and Graham 1993, Wiegers 2002). On a project that uses inspections for design and code, the inspections will take up about 10–15 percent of project budget and will typically reduce overall project cost.

Inspections can also be used for assessing progress, but it's the technical progress that is assessed. That usually means answering two questions: Is the technical work being done? And is the technical work being done *well*? The answers to both questions are byproducts of formal inspections.

Roles During an Inspection

One key characteristic of an inspection is that each person involved has a distinct role to play. Here are the roles:

Moderator The moderator is responsible for keeping the inspection moving at a rate that's fast enough to be productive but slow enough to find the most errors possible. The moderator must be technically competent—not necessarily an expert in the particular design or code under inspection, but capable of understanding relevant details. This person manages other aspects of the inspection, such as distributing the design or code to be reviewed, distributing the inspection checklist, setting up a meeting room, reporting inspection results, and following up on the action items assigned at the inspection meeting.

Author The person who wrote the design or code plays a relatively minor role in the inspection. Part of the goal of an inspection is to be sure that the design or code speaks for itself. If the design or code under inspection turns out to be unclear, the author will be assigned the job of making it clearer. Otherwise, the author's duties are to explain parts of the design or code that are unclear and, occasionally, to explain why things that seem like errors are actually acceptable. If the project is unfamiliar to the reviewers, the author might also present an overview of the project in preparation for the inspection meeting.

Reviewer A reviewer is anyone who has a direct interest in the design or code but who is not the author. A reviewer of a design might be the programmer who will implement the design. A tester or higher-level architect might also be involved. The role of the reviewers is to find defects. They usually find defects during preparation, and, as the design or code is discussed at the inspection meeting, the group should find considerably more defects.

Scribe The scribe records errors that are detected and the assignments of action items during the inspection meeting. Neither the author nor the moderator should be the scribe.

Management Including management in inspections is not usually a good idea. The point of a software inspection is that it is a purely technical review. Management's presence changes the interactions: people feel that they, instead of the review materials, are under evaluation, which changes the focus from technical to political. However, management has a right to know the results of an inspection, and an inspection report is prepared to keep management informed.

Similarly, under no circumstances should inspection results be used for performance appraisals. Don't kill the goose that lays the golden eggs. Code examined in an inspection is still under development. Evaluation of performance should be based on final products, not on work that isn't finished.

Overall, an inspection should have no fewer than three participants. It's not possible to have a separate moderator, author, and reviewer with fewer than three people, and those roles shouldn't be combined. Traditional advice is to limit an inspection to about six people because, with any more, the group becomes too large to manage. Researchers have generally found that having more than two to three reviewers doesn't appear to increase the number of defects found (Bush and Kelly 1989, Porter and Votta 1997). However, these general findings are not unanimous, and results appear to vary depending on the kind of material being inspected (Wiegers 2002). Pay attention to your experience, and adjust your approach accordingly.

General Procedure for an Inspection

An inspection consists of several distinct stages:

Planning The author gives the design or code to the moderator. The moderator decides who will review the material and when and where the inspection meeting will occur; the moderator then distributes the design or code and a checklist that focuses the attention of the inspectors. Materials should be printed with line numbers to speed up error identification during the meeting.

Overview When the reviewers aren't familiar with the project they are reviewing, the author can spend up to an hour or so describing the technical environment within which the design or code has been created. Having an overview tends to be a dangerous practice because it can lead to a glossing over of unclear points in the design or code under inspection. The design or code should speak for itself; the overview shouldn't speak for it.

Cross-Reference For a list of checklists you can use to improve code quality, see page xxix.

Preparation Each reviewer works alone to scrutinize the design or code for errors. The reviewers use the checklist to stimulate and direct their examination of the review materials.

For a review of application code written in a high-level language, reviewers can prepare at about 500 lines of code per hour. For a review of system code written in a high-level language, reviewers can prepare at only about 125 lines of code per hour (Humphrey 1989). The most effective rate of review varies a great deal, so keep records of preparation rates in your organization to determine the rate that's most effective in your environment.

Some organizations have found that inspections are more effective when each reviewer is assigned a specific perspective. A reviewer might be asked to prepare for the inspection from the point of view of the maintenance programmer, the customer,

or the designer, for example. Research on perspective-based reviews has not been comprehensive, but it suggests that perspective-based reviews might uncover more errors than general reviews.

An additional variation in inspection preparation is to assign each reviewer one or more scenarios to check. Scenarios can involve specific questions that a reviewer is assigned to answer, such as "Are there any requirements that are not satisfied by this design?" A scenario might also involve a specific task that a reviewer is assigned to perform, such as listing the specific requirements that a particular design element satisfies. You can also assign some reviewers to read the material front to back, back to front, or inside out.

Inspection Meeting The moderator chooses someone other than the author to paraphrase the design or read the code (Wiegers 2003). All logic is explained, including each branch of each logical structure. During this presentation, the scribe records errors as they are detected, but discussion of an error stops as soon as it's recognized as an error. The scribe notes the type and the severity of the error, and the inspection moves on. If you have problems keeping the discussions focused, the moderator might ring a bell to get the group's attention and put the discussion back on track.

The rate at which the design or the code is considered should be neither too slow nor too fast. If it's too slow, attention can lag and the meeting won't be productive. If it's too fast, the group can overlook errors it would otherwise catch. Optimal inspection rates vary from environment to environment, just as preparation rates do. Keep records so that over time you can determine the optimal rate for your environment. Other organizations have found that for system code, an inspection rate of 90 lines of code per hour is optimal. For applications code, the inspection rate can be as rapid as 500 lines of code per hour (Humphrey 1989). An average of about 150–200 nonblank, noncomment source statements per hour is a good place to start (Wiegers 2002).

Don't discuss solutions during the meeting. The group should stay focused on identifying defects. Some inspection groups don't even allow discussion about whether a defect is really a defect. They assume that if someone is confused enough to think it's a defect, the design, code, or documentation needs to be clarified.

The meeting generally should not last more than two hours. This doesn't mean that you have to fake a fire alarm to get everyone out at the two-hour mark, but experience at IBM and other companies has been that reviewers can't concentrate for much more than about two hours at a time. For the same reason, it's unwise to schedule more than one inspection on the same day.

Inspection Report Within a day of the inspection meeting, the moderator produces an inspection report (e-mail or equivalent) that lists each defect, including its type and severity. The inspection report helps to ensure that all defects will be corrected, and

it's used to develop a checklist that emphasizes problems specific to the organization. If you collect data on the time spent and the number of errors found over time, you can respond to challenges about inspection's efficacy with hard data. Otherwise, you'll be limited to saying that inspections seem better. That won't be as convincing to someone who thinks testing seems better. You'll also be able to tell if inspections aren't working in your environment and modify or abandon them, as appropriate. Data collection is also important because any new methodology needs to justify its existence.

Rework The moderator assigns defects to someone, usually the author, for repair. The assignee resolves each defect on the list.

Follow-Up The moderator is responsible for seeing that all rework assigned during the inspection is carried out. Depending on the number of errors found and the severity of those errors, you might follow up by having the reviewers reinspect the entire work product, having the reviewers reinspect only the fixes, or allowing the author to complete the fixes without any follow-up.

Third-Hour Meeting Even though during the inspection participants aren't allowed to discuss solutions to the problems raised, some might still want to. You can hold an informal, third-hour meeting to allow interested parties to discuss solutions after the official inspection is over.

Fine-Tuning the Inspection

Once you become skilled at performing inspections "by the book," you can usually find several ways to improve them. Don't introduce changes willy-nilly, though. "Instrument" the inspection process so that you know whether your changes are beneficial.

Companies have often found that removing or combining any of the stages costs more than is saved (Fagan 1986). If you're tempted to change the inspection process without measuring the effect of the change, don't. If you have measured the process and you know that your changed process works better than the one described here, go right ahead.

As you do inspections, you'll notice that certain kinds of errors occur more frequently than other kinds. Create a checklist that calls attention to those kinds of errors so that reviewers will focus on them. Over time, you'll find kinds of errors that aren't on the checklist; add those to it. You might find that some errors on the initial checklist cease to occur; remove those. After a few inspections, your organization will have a checklist for inspections customized to its needs, and it might also have some clues about trouble areas in which its programmers need more training or support. Limit your checklist to one page or less. Longer ones are hard to use at the level of detail needed in an inspection.

Egos in Inspections

Further Reading For a discussion of egoless programming, see *The Psychology of Computer Programming*, 2d ed. (Weinberg 1998).

The point of the inspection itself is to discover defects in the design or code. It is not to explore alternatives or to debate about who is right and who is wrong. The point is most certainly *not* to criticize the author of the design or code. The experience should be a positive one for the author in which it's obvious that group participation improves the program and is a learning experience for all involved. It should not convince the author that some people in the group are jerks or that it's time to look for a new job. Comments like "Anyone who knows Java knows that it's more efficient to loop from *0* to *num-1*, not *1* to *num*" are totally inappropriate, and if they occur, the moderator should make their inappropriateness unmistakably clear.

Because the design or code is being criticized and the author probably feels somewhat attached to it, the author will naturally feel some of the heat directed at the code. The author should anticipate hearing criticisms of several defects that aren't really defects and several more that seem debatable. In spite of that, the author should acknowledge each alleged defect and move on. Acknowledging a criticism doesn't imply that the author agrees with the content of the criticism. The author should not try to defend the work under review. After the review, the author can think about each point in private and decide whether it's valid.

Reviewers must remember that the author has the ultimate responsibility for deciding what to do about a defect. It's fine to enjoy finding defects (and outside the review, to enjoy proposing solutions), but each reviewer must respect the author's ultimate right to decide how to resolve an error.

Inspections and *Code Complete*

I had a personal experience using inspections on the second edition of *Code Complete*. For the first edition of this book I initially wrote a rough draft. After letting the rough draft of each chapter sit in a drawer for a week or two, I reread the chapter cold and corrected the errors I found. I then circulated the revised chapter to about a dozen peers for review, several of whom reviewed it quite thoroughly. I corrected the errors they found. After a few more weeks, I reviewed it again myself and corrected more errors. Finally, I submitted the manuscript to the publisher, where it was reviewed by a copy editor, technical editor, and proofreader. The book was in print for more than 10 years, and readers sent in about 200 corrections during that time.

You might think there wouldn't be many errors left in the book that had gone through all that review activity. But that wasn't the case. To create the second edition, I used formal inspections of the first edition to identify issues that needed to be addressed in the second edition. Teams of three to four reviewers prepared according to the guidelines described in this chapter. Somewhat to my surprise, our formal inspections found several hundred errors in the first edition text that had not previously been detected through any of the numerous review activities.

If I had any doubts about the value of formal inspections, my experience in creating the second edition of *Code Complete* eliminated them.

Inspection Summary

Inspection checklists encourage focused concentration. The inspection process is systematic because of its standard checklists and standard roles. It is also self-optimizing because it uses a formal feedback loop to improve the checklists and to monitor preparation and inspection rates. With this control over the process and continuing optimization, inspection quickly becomes a powerful technique almost no matter how it begins.

Further Reading For more details on the SEI's concept of developmental maturity, see *Managing the Software Process* (Humphrey 1989).

The Software Engineering Institute (SEI) has defined a Capability Maturity Model (CMM) that measures the effectiveness of an organization's software-development process (SEI 1995). The inspection process demonstrates what the highest level is like. The process is systematic and repeatable and uses measured feedback to improve itself. You can apply the same ideas to many of the techniques described in this book. When generalized to an entire development organization, these ideas are, in a nutshell, what it takes to move the organization to the highest possible level of quality and productivity.

cc2e.com/2199

CHECKLIST: Effective Inspections

- ❑ Do you have checklists that focus reviewer attention on areas that have been problems in the past?
- ❑ Have you focused the inspection on defect detection rather than correction?
- ❑ Have you considered assigning perspectives or scenarios to help reviewers focus their preparation work?
- ❑ Are reviewersrs given enough time to prepare before the inspection meeting, and is each one prepared?
- ❑ Does each participant have a distinct role to play—moderator, reviewer, scribe, and so on?
- ❑ Does the meeting move at a productive rate?
- ❑ Is the meeting limited to two hours?
- ❑ Have all inspection participants received specific training in conducting inspections, and has the moderator received special training in moderation skills?
- ❑ Is data about error types collected at each inspection so that you can tailor future checklists to your organization?

- ❑ Is data about preparation and inspection rates collected so that you can optimize future preparation and inspections?
- ❑ Are the action items assigned at each inspection followed up, either personally by the moderator or with a reinspection?
- ❑ Does management understand that it should not attend inspection meetings?
- ❑ Is there a follow-up plan to assure that fixes are made correctly?

21.4 Other Kinds of Collaborative Development Practices

Other kinds of collaboration haven't accumulated the body of empirical support that inspections or pair programming have, so they're covered in less depth here. The collaborations covered in this section includes walk-throughs, code reading, and dog-and-pony shows.

Walk-Throughs

A walk-through is a popular kind of review. The term is loosely defined, and at least some of its popularity can be attributed to the fact that people can call virtually any kind of review a "walk-through."

Because the term is so loosely defined, it's hard to say exactly what a walk-through is. Certainly, a walk-through involves two or more people discussing a design or code. It might be as informal as an impromptu bull session around a whiteboard; it might be as formal as a scheduled meeting with an overhead presentation prepared by the art department and a formal summary sent to management. In one sense, "where two or three are gathered together," there is a walk-through. Proponents of walk-throughs like the looseness of such a definition, so I'll just point out a few things that all walk-throughs have in common and leave the rest of the details to you:

- The walk-through is usually hosted and moderated by the author of the design or code under review.
- The walk-through focuses on technical issues—it's a working meeting.
- All participants prepare for the walk-through by reading the design or code and looking for errors.
- The walk-through is a chance for senior programmers to pass on experience and corporate culture to junior programmers. It's also a chance for junior programmers to present new methodologies and to challenge timeworn, possibly obsolete, assumptions.
- A walk-through usually lasts 30 to 60 minutes.
- The emphasis is on error detection, not correction.