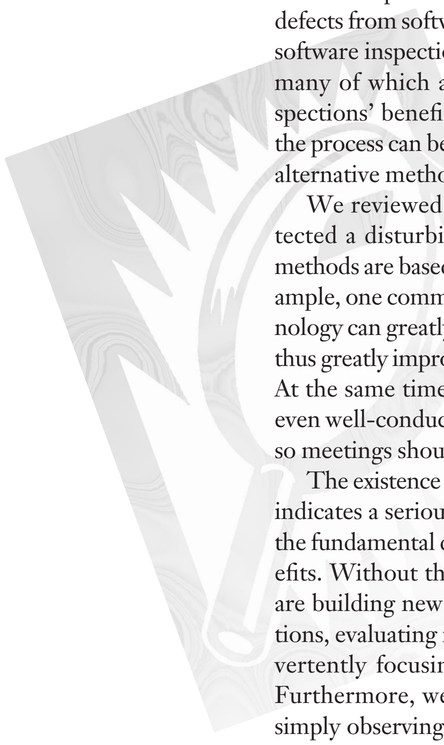
adam porter
university of maryland
lawrence votta
lucent technologies

What Makes Inspections Work?

New views of mature ideas on software quality and productivity.

Recent reports in the literature contain many empirical evaluations of software inspections. Here, Adam Porter and Larry Votta summarize their findings, suggesting results that are ready to be put into practice as well as an agenda for further research. —Shari Lawrence Pfleeger



SOFTWARE INSPECTIONS ARE CONSIDered a cheap and effective way to detect and remove defects from software. One popular bibliography on software inspections contains well over 200 entries, many of which are case studies documenting inspections' benefits. Still, many people believe that the process can be improved and have thus proposed alternative methods.

We reviewed many of these proposals and detected a disturbing pattern: too often competing methods are based on conflicting arguments. For example, one community argues that groupware technology can greatly improve inspection meetings and thus greatly improve overall inspection effectiveness. At the same time, another community argues that even well-conducted meetings discover few defects, so meetings should be discarded altogether.

The existence of these and other competing views indicates a serious problem: we have yet to identify the fundamental drivers of inspection costs and benefits. Without this information, we can't tell if we are building new methods based on faulty assumptions, evaluating new methods improperly, or inadvertently focusing on low-payoff improvements. Furthermore, we will not get this information by simply observing and comparing the performances of different methods. Instead, we must get inside the methods by using the scientific method to isolate the behavior of different drivers.

To do this, we first built a taxonomy of the potential drivers of inspection costs and benefits. Then we conducted a family of experiments to evaluate their effects. We chose effort and interval, measured as calendar time to completion, as our primary measures of cost. Defects discovered per thousand lines of code served as our primary measure of inspection benefits.

COST-BENEFIT DRIVERS. Many organizations use a three-step inspection process: individual analysis, team analysis, and repair. We suspect that the costs and benefits of this process may be driven by several mechanisms, both internal and external to itself:

- ♦ structure (how the steps of the inspection are organized into a process);
- ♦ techniques (how each step is carried out);
- ♦ inputs (reviewer ability and code quality);
- ♦ context (interactions with other inspections, project schedule, personal calendars); and
- ♦ technology (tool support).

Competing methods show we have yet to identify the fundamental drivers of inspection costs and benefits.

EXPERIMENTS. We conducted a family of experiments to understand how various factors affect inspection costs and benefits.

Process structure. This study looked at the effect of process structure. Prior to the study, we reviewed and identified key differences in the structure of several inspection methods, including Fagan inspections, active design reviews, *N*-fold inspections, phased inspections, and two-person inspections. The main structural differences between these methods are the size of the review team, the number of teams, and the strategy used to coordinate multiple teams. One of our null hypotheses was that none of these factors drives inspection effectiveness.

We ran an experiment (Adam Porter, Harvey Siy, Carol Toman, and Lawrence G. Votta, "An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development," *IEEE Transactions on Software Engineering*, Vol. 23, No. 6, 1997) at Lucent Technologies on a project that was developing a compiler and environment to support developers of the Lucent 5ESS telephone switching system. The finished system contains about 75K lines of

Editor:
Shari Lawrence Pfleeger
Systems/Software
4519 Davenport St. NW
Washington, DC
20016-4415
s.pfleeger@ieee.org

C++ code. The data collection phase ran for 18 months, during which time we observed 88 inspections. We selected the treatment used during a given inspection by randomly assigning the team size (one, two, or four reviewers), the number of inspection teams (one or two), and—for two-team inspections—the coordination strategy: either independent inspections, or two sequential inspections with repair in between.

Inspection effort appears to be driven solely by the number of reviewers who participate in the inspection.

Surprisingly, none of the independent variables had a significant effect on effectiveness. Consequently, we suspect that simply restructuring the inspection process (the approach taken by most research in this area) will not significantly increase effectiveness. We also found that many of the assumptions underlying contemporary methods did not hold in practice. While inspections with one reviewer were less effective than those with two, inspections with two reviewers were no less effective than those with four (an argument for smaller teams). Also, teams that reviewed the same code unit found few common defects (an argument against multiple-team, sequential inspections).

Inspection effort appears to be driven solely by the total number of reviewers participating in the inspection. None of these independent variables had a significant effect on interval. However, the premeeting interval (time from start of inspection to the meeting) of two-team, two-person inspections with repair in-between was about twice as long (four weeks versus two weeks) as all other premeeting intervals. Further investigation showed that the time needed to schedule the second inspection was responsible for the delay. These findings suggest that even simple types of coordination may substantially increase a process's interval.

Process inputs. The inspection performances we've described show considerable variation. This suggests that something other than process structure strongly affects inspection effectiveness. One obvious possibility is the differences in process inputs. We investigated this by modeling variation in the data as a function of process inputs and process structure (Adam Porter, Harvey Siy, Audris Mockus, and Larry Votta, "Sources of Variation in Software Inspections," *ACM Transactions on Software Engineering Methodology*, January 1998). Our goal was to determine the relative effect of process structure and process input on inspection effectiveness. We found that 50 percent of the variation could be explained by the code's size, its functionality, and the reviewers who inspected it; the process structure explained only 3 percent. We also found that even when the variation due to these inputs was factored out, process structure did not significantly affect effectiveness. We interpret this to mean that how code is constructed and analyzed has far more influence on effectiveness than does how the process is structured.

For interval data, we found that the code's author, and the presence of certain reviewers, explained 36 percent of the variation, while process structure explained only 3 percent. The model for premeeting interval was similar, but included the structure variable, Repair. That is, by factoring out the variation due to process inputs, we discovered an effect due to process structure (namely Repair). Our interpretation is that process inputs (mostly the code unit's author) have a greater influence on interval than process structure does. Nevertheless, we found that inspection processes involving multiple, sequential inspections, significantly lengthen interval.

These results reinforce our previous findings—that recently proposed changes to inspection process structure are largely ineffective in improving effectiveness and, in some cases, can require substantially more effort and interval.

Inspection techniques. The two previous studies suggest that better techniques for analyzing documents may do more to improve effectiveness than will better inspection methods. There are two contexts in which a review team analyzes a document: individually and cooperatively. Historically, team analysis has been the focus of inspection research. Some recent studies suggest that, in practice, team analysis is not essential (Lawrence G. Votta, "Does Every Inspection Need a Meeting?" *Proceedings ACM SIGSOFT '93 Symposium on Foundations of Software Engineering*, ACM, New York, 1993).

Team analysis. Because meetings are expensive, it is important to determine exactly how they contribute to inspections and whether superior alternatives exist. From the viewpoint of effectiveness, meetings are essential if

- many faults are found during the meetings and
- because of these meetings, more faults are found than would be otherwise.

To help answer these questions, we examined three approaches to inspecting software. The first two involve meetings, the third does not.

- Preparation-Inspection (PI). Each reviewer individually analyzes the document to become familiar with it. Afterward, the team holds an inspection meeting to find faults.
- Detection-Collection (DC). Each reviewer individually analyzes the document to detect faults. The team then meets to collect the defects found earlier and, if possible, find more.
- Detection-Detection (DD). Each reviewer individually analyzes the document to detect faults. Later, each reviewer conducts fault detection a second time, again individually.

We hypothesize that inspection methods such as DD, which eliminate meetings, are at least as cost-effective as the PI and DC methods, which rely heavily on them. We expect this result because the benefit of additional individual analysis, as provided by the DD method, should be greater than holding inspection meetings.

To evaluate this hypothesis, we designed and conducted a controlled experiment involving 21 graduate students in computer science and 27 professional software developers. (Adam Porter and Philip

Johnson, "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies," *IEEE Transactions on Software Engineering*, Vol. 23, No. 3). We found that the meetingless inspections found more defects than did those with meetings, but that there was no significant difference between the two ways of having inspections with meetings. We also found no significant difference between the DD and DC method in the first phase of the inspection. Rather, the DD method performed the best because it found more new faults in the second phase. Finally, we found no evidence that inspections with meetings found specific faults with greater frequency than did the meetingless inspections.

These results suggest that even well-prepared-for meetings may find fewer defects than do individuals working alone. Although they cannot be justified on the basis of defect discovery only, meetings do have value in that they suppress false positives and let groups make decisions and coordinate activities.

Individual analysis. Preparation is the first step of the inspection process. Methods for conducting this step have three components: techniques, responsibilities, and a coordination policy.

♦ Detection techniques range in prescriptiveness from intuitive, nonsystematic procedures such as ad hoc or checklist techniques to explicit and highly systematic procedures such as correctness proofs.

♦ A reviewer's individual responsibility may be general, to identify as many defects as possible, or specific, to focus on a limited set of issues, such as ensuring appropriate use of hardware interfaces, identifying untestable requirements, or checking conformity to coding standards.

♦ Individual responsibilities may or may not be coordinated among review team members. When they are not coordinated, all reviewers have identical responsibilities. In contrast, reviewers in coordinated teams have distinct responsibilities.

Reviewers frequently use ad hoc or checklist defect detection methods. Ad hoc reviewers use nonsystematic techniques and are assigned the same general responsibilities. Checklist reviewers receive a list of items to search for, usually derived from important lessons learned during previous inspections within a specific environment or domain.

We performed an experiment to test the following hypothesis: that nonsystematic techniques with general and identical reviewer responsibilities lead to overlap and gaps in coverage, thereby lowering the overall inspection effectiveness, but that systematic approaches with specific, distinct responsibilities reduce gaps and improve coverage, thereby increasing the inspection's overall effectiveness. To explore this hypothesis we prototyped a set of defect-specific techniques, which we called scenarios: collections of procedures for detecting particular classes of defects. Each reviewer executes a single scenario and all reviewers coordinate to achieve broad coverage of the document. We then conducted a controlled experiment (Adam Porter, Lawrence G. Votta, and Vic Basili, "Comparing Detection Methods for Software Requirement Inspections: A Replicated Experiment," *IEEE Transactions on Software Engineering*, Vol. 21, No. 6, 1995) using as subjects 48 graduate students in computer science and 21 professional software developers.

We divided the participants into 23 three-member teams. Each team inspected two software requirement specifications using some combination of ad hoc, checklist, and scenario methods. The experimental results showed that

♦ the scenario method had a higher defect detection rate than either the ad hoc or checklist methods;

♦ scenario reviewers were more effective at detecting the defects their scenarios were designed to uncover and were no less effective at detecting other defects; and

♦ checklist reviewers were no more effective than ad hoc reviewers.

These results suggest that improved defect detection techniques may indeed improve overall inspection effectiveness.

Process environment. Initially, we found that structure (high degrees of coordination) and environment (workload, priorities, and deadlines) can significantly affect interval. However, much of the variation in interval was still unaccounted for. Through direct observation and surveys we found that developers often choose which of their many activities to perform at any given time. We conjectured that these are not just random choices, but are influenced by the process environment: the logistic, organizational, and execution context in which the inspection process operates. To test this conjecture, we modeled the effect of process environment on inspection interval (Adam Porter, Harvey Siy, and Lawrence Votta, "Understanding the Effects of Developer Activities on Inspection Interval," *Proceedings 19th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, Calif., 1997). The factors we examined include measures of workload, life cycle phase during which the code is inspected, and the presence of deadlines. We found that process environment explains more variation than does process structure and process inputs combined. However, we are still far from explaining the majority of variation in inspection interval. Nevertheless, this analysis has several implications.

In particular, it is instructive to compare the pre- and postmeeting models. We find that the premeeting interval is not significantly affected by the workload of the author nor that of the inspection team. The author's coding load is significant but it is a negative contributor. This suggests that inspections progress despite increases in the number of code units on which the author is working. These observations imply that authors and reviewers give a higher priority to premeeting inspection tasks than they do to pending coding assignments. On the

Results suggest that even well-prepared-for meetings may find fewer defects than do individuals working alone.

other hand, the postmeeting interval is significantly affected by pending inspections and rework, which implies that rework has a low enough priority that authors defer it to complete coding tasks. Our interpretation of these results is that developer workload affects interval, developers prioritize their work, and that deadlines alter priorities.

Technology. Many companies are developing software using multiple, geographically separated teams. When this happens, dependencies between tools, processes, and people can substantially increase development interval. In situations where development teams are geographically separated, it can be very expensive to hold inspection meetings. Our research suggests that inspection meetings may not always be necessary. Thus, we used this situation as an opportunity to field test our experiment results. (J. Perpich et al., "Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development," *Proceedings of the Nineteenth International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, Calif., 1997). That is, if the results are reasonably general, causal, and actionable, then they should enable us to change the inspection process to reduce its interval without sacrificing effectiveness. To validate our theories, we, along with Jim Perpich, Dewayne Perry, and Michael Wade of Lucent Technologies, identified several contributors to inspection delays. The most interesting of these is blocking due to synchronization and sequencing of inspection subtasks. To reduce these delays, we considered three strategies: reduce paper, automatically generate necessary reports, and reduce synchronization and coordination. The first two strategies are straightforward, but the best way to reduce synchronization was less obvious. We considered three approaches.

♦ *Share results.* In the manual process, reviewers perform individual analysis privately. That is, each reviewer's findings are unknown to the other reviewers until the inspection meeting occurs. Our approach was to make each reviewer's findings public in nearly real time.

♦ *Eliminate inspection meetings.* Our previous research suggests that meetings significantly lengthen inspection interval, but contribute little to effectiveness. Therefore, we eliminated the inspection meeting.

♦ *Overlap preparation and repair.* Because we eliminated the meeting, the process has only two major phases, preparation and repair. Although these two phases are normally performed sequentially, we let them overlap. That is, the author may begin repairs as soon as defects are found.

Web-based workflow system. To enforce these process changes and to collect performance data, we designed a Web-based workflow system called HyperCode. HyperCode is currently being used by several development groups at Lucent. Our main experiment involves two development teams: one in Naperville, Ill., and the other in Whippany, N.J. We hypothesize that the HyperCode process will have a smaller interval than the manual one, but be no less effective. The experiment is currently running and initial results suggest that HyperCode reduces inspection interval by about 25 percent, with no apparent reduction in effectiveness. We must, however, continue running the experiment to better support these findings.

All empirical research must evolve beyond simply measuring toward developing general theories.

Generally, we believe that tool development should be based on the kind of scientific studies we describe here. When tools are developed, we would like to see greater attention paid to capturing the exact principles that the tools exploit.

ASSESSMENT. Given our experiments' results, we conclude that the techniques and technology supporting individual performance with respect to defect detection methods have more influence on effectiveness than do the nontechnical factors on which current research focuses extensively. We determined this by combining the following findings:

♦ process structure had no significant effect on effectiveness,

♦ process inputs explained more variation than process structure did,

♦ individual analysis was more effective than team analysis, and

♦ improved individual analysis techniques significantly improved performance.

We conclude that *effort* is driven primarily by the number of reviewers participating in the inspection, and that the *interval* for a given process is driven mostly by nontechnical factors outside the control of the process itself.

Our work suggests that general and identifiable mechanisms drive the costs and benefits of inspections. However, we lack a comprehensive theory that brings these principles together. Thus, the highest priority for the inspection community should be to develop such a theory; we are currently working with other researchers who have proposed one. More generally, we believe that all empirical research must evolve beyond simply measuring and comparing performances toward developing validated theories that are general, causal, and suggestive of control strategies. ♦

Adam A. Porter is an assistant professor in the Department of Computer Science and the Institute for Advanced Computer Studies at the University of Maryland. His research interests include empirical methods for identifying and eliminating bottlenecks in industrial development processes, experimental evaluation of fundamental software engineering hypotheses, and development of tools that demonstrably improve the software development process. He can be reached at aporter@cs.umd.edu.

Lawrence G. Votta is a member of the technical staff in the Software Production Research Department of Lucent Technologies in Naperville, Illinois. His research interest is to understand how to measure, model, and do credible empirical studies with large and complex software developments. He can be reached at votta@bell-labs.com.