A 20-year-old inspection technique has served developers well in the quest for software quality improvement. But radical changes are on the horizon that may seriously steer the future of formal review.

Reengineering Inspection

Despite many advances in automated verification and validation, human review of software artifacts is still a uniquely important method for software quality improvement. Unlike automated methods such as testing, review does not require an executable or formally specified artifact, enabling quality improvement on upstream work products such as initial requirements documents. This has significant economic implications, since studies show that defects can be one to two orders of magnitude less costly to remove from initial requirements documents than from implemented systems after distribution to the customer. Moreover, review is effective for discovering certain soft, but nevertheless costly, defects such as logically correct but poorly structured code. Indeed, review has a unique educational capability: The process of analyzing and critiquing software artifacts produced by others is a potent method for learning about languages, design techniques, application domains, and so forth.

Software review encompasses a broad spectrum of activities, from informal, individual desk checking to review using computer-mediated groupware environments with built-in process models. Formal technical review (FTR) is an umbrella term for review methods involving a structured encounter where a group of technical personnel analyzes an artifact in order to improve both the quality of the product and the review process.

Starting in 1976, when Michael Fagan published his seminal article [2], the Fagan Inspection method (including such close variants as Tom Gilb's Inspection [3]) has come to dominate both the theory and practice of review. Indeed, many researchers and practitioners equate the terms "inspection" and "formal technical review." Inspection and its variants have not enjoyed unwarranted popularity. In addition to Fagan's reports of cost-effective quality improvement at IBM, inspection-based techniques at Hewlett-Packard were calculated to yield a cost savings of $21 million [4]. Others have also reported favorable cost-benefit analyses for inspection.

Despite a consistent stream of positive findings over 20 years, industry adoption of inspection appears to remain quite low, although no definitive data exists. For example, an informal USENET survey we conducted found that 80% of 90 respondents practiced inspection irregularly or not at all. Some industry practitioners have found adoption and practice of inspection-based review to be difficult, costly, ineffective, and/or excessively time consuming, despite the prospect of quality improvement.

Although low usage and adoption difficulty provide only circumstantial evidence of problems with inspection-based methods, more direct evidence arises from recent research by Porter [7] and Votta [8]. Among other findings, their research indicates that meeting-based methods such as inspection can impose a hidden but significant cost by increasing development interval time, and that alternative approaches can overcome this problem without impacting upon defect detection effectiveness.

This article does not attempt to negate two decades of research and practice showing that inspection-based methods can improve an organization's ability to detect defects, particularly when inspection is adopted by an
organization that previously lacked any FTR. Rather, this article combats a myopia that has developed within the review community, where inspection is often viewed as not only effective, but optimal. This myopia prevents those afflicted from seeing the inspection glass ceiling that circumscribes the potential application and effectiveness of FTR to software development. It can also stifle innovative experimentation in software review.

Inspection is a product of the mid-1970s—a precursor to the Internet, computer-supported cooperative work technologies, the World-Wide Web, geographically distributed development groups, and virtual organizations. Even the most comprehensive material on inspection defines “inspection tools” as word processors and spreadsheets [3].

To shatter the ceiling restricting FTR use and effectiveness, we must reengineer inspection—in other words, radically redesign the process. This article supports such reengineering by articulating seven guidelines for next generation, breakthrough approaches to FTR.

Seven recommendations for the future of formal technical review

Provide tighter integration between FTR and the development method.

Minimize meetings and maximize asynchronicity in FTR.

Shift the focus from defect removal to improved developer quality.

Build organizational knowledge bases on review.

Outsource review and insource review knowledge.

Investigate computer-mediated review technology.

Break the boundaries on review group size.

The Future of Formal Technical Review

A major reason why the future of FTR will bear little resemblance to its past is the increasing ubiquity of local- and wide-area networks within the software development industry, and the organizational transformations co-occurring with this technology. Networks provide essential infrastructure for wide-area development (WAD), where software development effort is geographically distributed across a town, county, or the globe. Such dispersion may be motivated by the desire to obtain diverse cultural viewpoints and presence during the development of software, by the economic advantages of so-called “offshore development,” by participation in virtual enterprises, or by the desire to exploit the 24-hour clock present when development groups are widely separated.

Inspection and its allies, however, are firmly rooted within a paradigm whose focal point is a manual, face-to-face group process. They cannot adapt easily to geographically and chronologically distributed organizations, any more than the I/O paradigm embodied in punch cards can adapt easily to modern graphical user interfaces. Rather than port inspection, FTR researchers and practitioners must reengineer it. The following seven guidelines provide starting points for this process.

Software development method-specific FTR: With the exception of Cleanroom Development [1], current review techniques are remarkably decoupled from the software development method. Most FTR methods have a one-size-fits-all mentality: the same process is assumed to work equally well regardless of whether development follows a traditional waterfall lifecycle, or evolutionary development, or rapid prototyping, or a risk-driven spiral model. Customizing methods to the development context is limited to such peripheral issues as the contents of checklists. The parochial nature of such process improvement is revealed by the fact that metric data is almost never applied to evaluating if some other FTR method might be preferable to the one in place!

In the future, more effective forms of FTR can result from exploiting the characteristics of the surrounding development method, just as Cleanroom’s Verification-based Inspection does today. For example, a spiral model FTR method may take into account the importance of risk assessment and the number of previous turns around the spiral when designing not only checklists, but also the roles of participants and the analysis techniques chosen.

Minimal meeting, asynchronous FTR. The current dominance of inspection-based approaches leads to the prevailing notion that face-to-face meetings are central to FTR. However, asynchronous mechanisms for review artifact analysis and discussion is possible and, in many cases, preferable. First, asynchronous review accommodates the divergent schedules of geographically and organizationally distributed review groups. There is no longer a need for the group to meet in the same place at the same time. Second, asynchronous review can support the review of larger artifacts, since the rate of review is impacted negatively by the number of participants meeting simultaneously. Third, asynchronous review can ameliorate the interval time problem identified by
Porter and Votta for manual, meeting-based review methods. Fourth, asynchronous review reduces cost by eliminating air-time fragmentation—the idle time incurred by individuals during most group meetings. Finally, although meeting proponents often justify their need based upon the presence of group synergy, such synergy can also be observed during asynchronous review, and Porter and Votta's research questions the true contribution of this synergy anyway. Future FTR methods should view face-to-face meetings as the phase of last resort, when other, lower cost alternatives have already been employed.

Beyond defect removal to improved developer quality. Although software quality improvement is the stated goal of all FTR methods, most operationalize this improvement quite narrowly via a single metric—the number of defects removed per unit time. The focus leads directly to a well-known heuristic for successful review meetings: "Raise issues, don't resolve them." In other words, never attempt to discuss the solutions to a problem during review—simply note the issue and move on.

Unfortunately, by focusing on this single metric, FTR methods suppress or eliminate the ability of the group to improve software quality in other ways. As one example, suppose that improving the software development skills of the participants was a first-class goal of a review method. In other words, it was measured and used to assess the effectiveness of review just like defect removal. A strong argument can be made that overall software quality is affected far more profoundly by improvements to developer skills, which reduces future defect creation, than by simply removing defects from current individual documents.

If such developer quality improvement was a first-class, empirically measured goal of review, then the "raise issues, don't resolve them" heuristic would lose much of its vigor because focused issue resolution discussions are a high quality, efficient, and effective means for learning about alternative design/implementation strategies and their advantages and disadvantages.

An even more striking change resulting from this new review goal is the utility of reviewing documents of known high quality. If defect removal is the only measured goal of review, then review will tend to focus on documents predicted to have large numbers of defects. It would be pointless to review a document known to have only a small number of defects (or perhaps none at all). However, learning is enhanced by analyzing positive examples as well as negative ones. If developer quality improvement was a first-class goal of review, then occasional review of high-quality documents would help produce the desired improvements in the metrics associated with review. Future FTR methods should focus on the producer as much as on the products of review.

Beyond defect removal to organizational guideline knowledge bases. Another casualty of the current fixation on defect removal is the potential of FTR to incrementally generate and maintain an organizational knowledge base of indicators of high-quality software development, as manifested within review artifacts. The development of such a knowledge base is crucial for an organization that desires quality insights gained during one review to be generalized and made available in a useful manner to other reviewers and developers. Without such a repository, organizational learning about software quality is greatly impaired.

Support for organizational knowledge bases in current FTR methods consists of checklists, or review guidelines, that should be developed and maintained as a result of review. However, current methods treat these guidelines as a second-class citizen in two important ways, thus crippling their effectiveness. First, development and use of guidelines is not measured as part of review effectiveness. For example, consider a review session that finds only one defect in the current document, but spends time to generate 10 new guidelines that are subsequently used to discover 100 defects in later review sessions. While this review was certainly effective in improving the organization's software quality, current FTR metrics do not measure either the number of guidelines resulting from a review meeting, nor the downstream effectiveness of these guidelines. Based upon the metrics, this review session would be a failure, since a very low number of defects were discovered in the original document relative to the time invested.

The second way in which guideline development is discriminated against in current FTR methods is by its placement as a separate, third-hour activity. The simple fact that generation and maintenance of guidelines occurs after the real measured work is accomplished virtually guarantees it will receive less effort than that devoted to defect detection.

In the future, FTR methods should accord to guideline knowledge-base generation the first-class status it deserves. They will measure effort spent on guideline development and fold that into the calculation of review effectiveness. In addition, the development of guidelines will no longer be an appendage to review, but an integral and integrated component.

Outsourcing review and insourcing review knowledge. Another reengineering trend for the future of FTR is outsourcing. There already exists at least one Internet-based software review service, where clients send review artifacts concerning the Microsoft Windows User Interface to the company, who sends back a Word document containing review information.
In the future, outsourcing review should become a viable software quality improvement technique. For example, a company might hire an external consultant with specialized knowledge not present in the company to participate in FTR. The goal of the consultant’s participation will be not only to discover defects in the current document, but also to help educate company staff and to enhance the organization’s guideline knowledge base.

In addition to outsourcing review, companies could insource review knowledge by “buying, not building” their review guideline databases. For example, a guideline database providing high-quality insights into Java software development would be of great commercial value today.

Computer-mediation. A clear trend for the future of FTR is computer-mediation of review process and products. Prior research demonstrates a spectrum of benefits possible from computer-mediated FTR. First, a computer-mediated review environment can reduce clerical overhead, increase the accuracy of recorded review commentary, and allow automated collection of metrics data and their subsequent analysis [5]. In contrast, attempts to simply graft computer support onto a manual inspection-based process have led to decreased data accuracy and increased clerical overhead [9]. In addition, computer-mediation supports the reengineering trends identified here: Computer-mediation can tightly integrate the review method with other components of the specific software development method, it supports asynchronous review naturally, and it facilitates both alternative metrics collection and knowledge-base generation. Finally, computer-mediation allows review consultants to telecommute to review, and supports standardized structures for shrink-wrapped guideline databases.

Organizations will not adopt computer-mediated review simply because of such potential advantages over manual techniques. They will move toward it because WAD and virtual enterprises require computer-mediated review, if the software quality assurance process is to adhere to the underlying distributed organizational model.

Review mega-groups. Current FTR methods vary in their recommendations on the best group size, but there is widespread consensus that the group should never exceed 6-9 members. This upward bound results from inspection-based methods, in which the overhead and group process issues for meetings involving larger groups tends to outweigh the potential benefits.

Sometimes, however, review by larger numbers of people may be useful. In fact, in certain circumstances, it may be required. For example, at Jet Propulsion Laboratory, certain software maintenance documents could require formal review by 20 to 30 different people. Simply scheduling these review meetings, much less holding them effectively presented enormous problems. To resolve them, JPL instituted “Electronic Design Reviews,” a computer-mediated process involving distribution and review of these documents using electronic mail [6]. Future FTR techniques should exploit computer-mediation to dramatically increase the number of participants who can efficiently and effectively participate in review.

Next Steps
Software development organizations and their underlying technologies are far different than they were 20 years ago, and with these changes come new opportunities. Now is the time to rethink some of our implicit assumptions about software improvement through formal technical review, and explore radical FTR redesigns.

To further this process, we established the Internet Formal Technical Review archive at: http://www.ics.hawaii.edu/~johnson/FTR/. This Web site provides information on review materials, tools, organizations, consultants, and an online bibliography of over 200 references on software review, many including links to the full text. The FTR Archive also provides a link to the home page of the Software Inspections and Review Organization (SIRO), an industry-centered organization to promote research and practice of software review. We invite your participation in designing the future of formal technical review.

References

Philip M. Johnson (johnson@hawaii.edu) is an associate professor of information and computer sciences at the University of Hawaii in Honolulu.