A Guide on Inspections and Other Review Techniques

Karol Frühauf

Peer Reviews in Software: A Practical Guide
by Karl E. Wiegers, Addison-Wesley Professional, 2002, 256 pp., ISBN 0-201-73485-0, US$39.99

I was eager to read *Peer Reviews in Software* for three reasons: first, to see if it confirmed my experience; second, to learn something new; and third, to steal ideas for better presentation of the topic in my teaching. But judging the book on the basis of my expectations would be unfair. It deserves to be measured against Karl Wiegers' goal. In his own words, "the principal goal of this book is to help you effectively perform appropriate reviews of deliverables that people in your organization create." It's primarily intended for people who want to participate in reviews and people who should encourage reviews in their organizations.

The book completely satisfies the expectations it creates. It is well organized, sticks to the topic, and provides all the information the reader needs, all in easy-to-understand language. The book isn't only about software quality, it's also a piece of quality work.

It's not surprising, then, that Chapter 1 contains, besides justification of reviews, the section "A Personal Commitment to Quality." This contains a quote that should be every software-producing organization's motto: "If you don't have time to do it right, when will you have time to do it over?" Wiegers introduces *peer review* as the general term covering any technique that people use to examine software development work products to find defects. Reviews, inspections, and walkthroughs are examples of specific peer review techniques. I like "peer review" better than the term "static testing," which people often use to distinguish reviews from "dynamic testing," which involves actually executing programs to detect defects.

In Chapter 2, Wiegers discusses the cultural prerequisites for peer reviews in an organization and how reviews benefit the different roles in a development effort. He points out clearly why it's a mistake to consider a review a milestone and not a task before a milestone; let's hope many managers eventually grasp this idea. The "Peer Review Sophistication Scale" section lays the groundwork for Chapter 3, which introduces different techniques in order of formality. Inspections represent the most formal end of the spectrum, followed by team review, walkthrough, pair programming (including agile programming), peer deskcheck, passaround, and ad hoc review. This chapter concludes with suggestions for selecting the appropriate technique depending on the situation.

Chapter 4 presents an overview of the inspection process. Chapters 5–9 provide all the necessary material for training on that process, starting with planning, via preparation and meeting, to closure, and eventually to inspection data analysis. These chapters also describe the tasks of the different roles in the particular stages of the process, as well as the results of these tasks. The presentation is clear and thorough enough for novices to learn the process and contribute to its success. (An inspection moderator would need additional formal training. This is always the case, because only born moderators can learn to moderate by reading a book.) Of course, trainers will still need to prepare exercises, but such exercises will likely be more effective than book-based exercises because they'll use material from real projects in the company.

The next two chapters help establish peer reviews in an organization. Chapter 10 is about the organizational matters involved in installing a peer review program and setting up a review process. Chapter 11 highlights the critical success factors for such a program and provides hints on how to avoid the usual traps that people encounter when installing reviews.

The final chapter, "Special Review Challenges," covers conditions that make the inspection process more difficult, such as large work products, geographical or time separation, generated and nonprocedural code, too many participants, and lack of qualified reviewers. This chapter provides clear evidence that Wiegers has dealt personally with the review process and knows how to solve real-life problems concerning reviews.

Appendix A explains the role of reviews in Capability Maturity Models and in ISO 9000-3, which helps to relate their requirements to the book's material. Appendix B contains supplemental material.

Objectively measured, the book is a win for the intended readers. Software engineers should read the whole book to understand all facets of reviews. Managers hoping to learn something about reviews could skip Chapters 5–9 and still find enough solid material to make the right decisions regarding whether and how to install reviews. I can't imagine that any manager would decide not to implement reviews after reading this book, so the main task is to get it into managers' hands.

Now for my subjective measurement. First, the book confirms my experience very much indeed, except on one point. It's not always the program's author who decides which defects—if not all—to remove. For example, after a review of software design document, the project manager usually needs to decide this on the basis of the actual situation's risks. Second, I did learn some tricks in applying inspections. Third, I won't use the author's forms (my own forms seem less clumsy because I'm used to them), but I will steal some checklist entries and some criteria for my catalog of criteria for selecting review techniques. Overall, the book was a win for me.

Even in these agile times, reviews have their value. I don't mean pair programming, which is a rather extreme peer review technique; I mean inspections or other techniques that are a bit less rigorous. I would go so far as to claim that only organizations with mature review processes in place can successfully introduce agile methods. Agile methods rely on face-to-face communication and common ownership of the software. Applying reviews is the first step in this direction.

Reviews pull developers out from their dark corners so they have to put their work products into the public light and under the scrutiny of their peers, and have to learn to listen to an appraisal of their work without blaming. Additionally, review programs have a positive side effect: team members' values converge, and a consensus grows about what is and isn't good for achieving the common goal. Without these prerequisites, agile methods are unlikely to prosper. *Peer Reviews in Software* is a good roadmap for creating these prerequisites and for harvesting cost benefits along the way.

Karol Frühauf is a consultant and chairman of *INFOGEM AG*. Contact him at karol.fruehauf@infogem.ch.