Software Inspection

Tom Gilb, Dorothy Graham; Addison-Wesley, 1993, reprinted 1994, 471 pages
ISBN 0-201-63181-4

Book report
Jim McDonough, May 5, 2009

Review

This book praises and expands upon the work of the software inspection process begun in 1972 at IBM by Michael E. Fagan, which is described in the Fagan paper Design and code inspections to reduce errors in program development, published in IBM Systems Journal, Volume 15, Number 3, 1976.

This book, Software Inspection, is predicated on a procedure which involves multiple "inspectors" to review a "product document" using relevant "source documents" following a prescribed "process" having specific entry and exit criteria.

- An Inspector is not necessarily a peer – indeed, an inspector can play the part of a typical user who would be expected to understand how to use a new software feature, and the authors encourage involvement in a typical inspection of those who are new to the process. Inspectors perform various roles during the inspection process with no two inspectors checking primarily for the same things (pp. 70; 76-77).

- A Product Document can be any type of document which contributes to the software – a design, a requirements specification, a code listing, a test plan, a contract, a rule set (p. 45).

- A Source Document is a primary document against which the product document is to be checked for complicity, such as ideas, memos, contracts, policies, requirements and plans (p. 46).

- The Process prescribed is formal (p. 41). When a document is presented for inspection, an Inspection Leader selects the inspection participants and their respective duties (moderator, author, inspector, scribe). Inspectors engage in solitary inspection activities as per their assigned role, after which there is a joint logging meeting with all inspection participants to discuss what has been identified and to determine as a group whether there are any other concerns (p. 81). The result of the meeting is a lis of defects to be addressed. Defects are identified not only for the product document but also for source documents. Metrics are collected and recorded in a central database to enable an evaluation of the cost-effectiveness of the inspection process (p. 109).

It should be noted that the original Fagan process begun at IBM was concerned mainly with the code and not so much with its associated documentation, where the code used with the original IBM study was a piece of the design of a large operating system component, designed by three programmers to be coded in PL/S (a proprietary IBM language replacement for its own system assembly language) by 13 programmers. This book seems to take the position also of a large development project, but does not restrict the inspection process to only the computer source code (p. 45). Furthermore, it recognizes and recommends constantly reevaluating and improving the inspection process, rules, standards and other contributing documents to get the most improvement in quality (pp. 37; 114).

The basic principle embodied here is that the sooner a defect can be identified the cheaper it is to make the correction. This seems to be the crux of inspections – to identify defects before they can reveal themselves later in source code and tests. However, the authors fully acknowledge that whichever method is first used to discover software problems – inspection, coding, testing – will be where the most defects will be identified, simply because it is being used first.

This book goes a long way impressing upon the reader of the need for training in the various roles of inspectors. In many organizations, there is an unfounded assurance on the part of management that inspectors naturally know how to perform software inspections, and as a consequence, there is no training to participants in improving the skills necessary to perform inspections properly.

Much has changed in the software development industry in the 15 years since this book was reprinted, and some of the recommended techniques may be obsolete, such as providing inspectors with printed copies of source code listings. Also, years ago there was the concern with the cost of computer time versus the cost of programmer time. This concern nearly has been reversed, where now the cost of programmer time has risen dramatically and the cost of computer time has dropped to negligible. In this context, it is now more cost-effective for a language compiler or automated source code inspector to identify certain defects which otherwise would be left to manual inspection, a less effective and higher cost option.

Pros

- Establishes a comprehensive inspection philosophy which considers any contributing document as a potential area of inspection.
- Endorses use of the Shewhart PDSA Cycle (Plan, Do, Study, Act) to improve the inspection process itself.
- Encourages the continual update of standards, rules, checklists, etc.
- Clearly establishes and describes the roles of those involved in the inspection process.
- Clearly outlines a process to follow during an inspection.
- Identifies the distinction between defect detection and defect prevention.
- Recognizes the need for training for participants in the inspection process.
- Recognizes the need for a Process Change Management Team which can facilitate improvements to the software development process.

Cons

- Recommends a procedure which involves many inspection personnel for a single document; the time involved by all participants easily could exceed the cost of no inspection and leaving defects to be found later.
- Stops short of suggesting how to use collected metrics to improve the inspection process and determine whether the process is in statistical control.
- Is 15 years behind the current state of the art in software design.

Notable citations within

“In the software engineering world, there is now quite a lot of Inspection going on. But there is strong evidence to suggest that much of this Inspection is not correctly done, and many practitioners do not realize it.” (p. ix)

“Whenever any user of a procedure has any problems with it or has any constructive suggestion, there should be an immediate and natural path of communication to the owner of the process which the procedure describes. If not, many useful insights will be lost. People will begin to ignore procedure and productivity will decline.” (p. 55)

“In principle each checklist has an owner, named on the checklist. Only this owner makes official changes to the checklist. Everything else is a suggestion to the owner.” (p. 59)

“The metrics are the most important component to Inspection. They provide the means for actually seeing what is happening in the whole software development process. Without metrics, you are working ‘blind’ (as many review techniques do). Inspection (and other) metrics are used to monitor not only the

Inspection process, but also the effect of introducing changes to the software development process. It is essential that test and field metrics are linked to Inspection metrics, so that the benefits from the Inspection process can be properly traced." (p. 93)

"Inspection metrics are far more important than most people think. Inspection metrics are, if anything, one of the most important differences between Inspection and other similar processes such as reviews and walkthroughs." (p. 109)

"It is essential that test and field metrics are also collected and collated with the Inspection metrics, so that the effects of Inspection and other techniques can be assessed across the whole of the development processes. If Inspection metrics only are kept, and not compared to test and field data, then you will not know how effective the Inspections really are. The real payback comes in the savings in testing and field operation." (p. 111)

"Product Inspection is primarily a *defect detection* and removal method. The other aspect of Inspection is primarily a *defect prevention* method, and is ultimately far more powerful and effective than product-based Inspection alone." (p. 114)

"Process improvement is based on the fundamental concepts of 'process control', first developed by Shewhart and later by Juran and Deming. The nature of a process is dynamic. Events happened in a time sequence, and 'flow' through various stages." (p. 116)

"The brainstormed causes and process improvements will be logged to the QA database and from there will be followed up by the Process Change Management Team (PCMT). If anyone is particularly interested in working on one particular improvement idea, they will be formally encouraged to accept an assignment to take deeper analytical and corrective action by the Process Change Management Team." (p. 123)

"When Process Change Management Teams do not yet exist, we make it a practice to assign personal improvement items to those who show enthusiasm and insight at the process brainstorming meeting." (p. 132)

"It sometimes seems that an organization actively works to prevent improvement to its own processes, through bureaucracy, apathy or simply 'we've always done things this way'. When the possibility of real improvement is perceived by software development professionals, their enthusiasm and idealism can only benefit the organization if the changes can be implemented. It is a great shame if real grass-roots positive initiatives are squashed inadvertently by organizational mass". (p. 135)

"If you are part of a large company, then a frustration bureaucracy probably exists. It is not evil in intent. But few bureaucracies (except perhaps Inspection!) are intentionally designed to encourage and permit change. Inspection requires change. So you have to expect to deal with your bureaucracy to get Inspection installed and working well." (p. 209)

"How to be Sure Inspections Fail

Technical issues (incomplete list)

- Concentrate on trivia
- Don't keep metrics, don't look at metrics, or never tell anyone what they are
- Inspect everything regardless of initial quality
- Don't ever upgrade your entry or exit criteria or checklists, rules and procedures
- Don't allow any changes in source documents or standards
- Don't use checklists or roles; have everyone cover the same ground

Organizational issues (incomplete list)

- Make costs visible, keep benefits hidden
- Use defect metrics for individual personnel appraisal
- Don't provide training, especially for Inspection leaders

- Don't provide ongoing support to improve the Inspection process
- Never allow any change to the inspection process

Implementation issues (incomplete list)

- Don't bother with management commitment, start bottom-up
- Don't take notice of existing practices or previous review experience
- Don't check whether anticipated benefits had happened
- Don't keep track of any savings due to Inspection"

(pp. 253-254)

"You can expect the following situation regarding rules:

(1) There are existing software engineering standards. Nobody uses them. They are not in useful shape. Too voluminous, too much trivia.
(2) There are several areas needing rules for which there are no standards whatsoever."

(pp. 259)

"There are usually no checklists, at least no really useful and significant ones available to begin with. Do not get somebody to draft checklists all at once. This is invariably wasted effort, because:

- Too much trivia gets on the lists;
- The lists are too voluminous;
- Such lists have no credibility;
- There tends to be no way to get rid of them, except to ignore them"

(p. 261)

"When there is no clear and strong leadership, the quality improvement process will not survive because it will not be done properly. It will therefore not have significant results, and will not be perceived as a solution to current needs. Conventional wisdom says that 'quality' initiative must come from the top to succeed. Certainly, they must have serious support high up. But they need very active and capable 'champions' – probably the type of person who has read the book this far. Yes, *you*" (p. 263)

"Appendix A: A one-page Inspection Handbook"
This Handbook shall never be printed on more than a page after updates. This note will always be included" (p. 361)

"Standards should be:
Attainable … Economic … Applicable … Consistent … All-inclusive … Understandable … Stable … Maintainable … Legitimate … and Equitable
He explains these concepts in more depth. J. M. Juran in Management Breakthrough, p. 235."
(p. 423)