Software Inspections: An Effective Verification Process

A. Frank Ackerman, *Institute for Zero-Defect Software*
Lynne S. Buchwald, *Qualitech*
Frank H. Lewski, *AT&T Bell Laboratories*

Properly applied, software inspections rival testing in defect detection and correction but cost less time and effort. This article explains what inspections are and how to perform them.

A fundamental of a good verification and validation program is that verification steps must be performed throughout the software-development effort. Traditionally, reviews and walkthroughs at different points in the development have been used to do so. But, in the past 13 years, several development organizations have replaced or supplemented traditional reviews and walkthroughs with software inspections. For V&V purposes — the detection and correction of defects (faults, failures, and other deficiencies resulting from nonconformance to standards or requirements) early in the development process — software inspections have been found to be superior to reviews and walkthroughs.

The first published description of the inspection process appeared in a 1976 paper[1] by its inventor, Michael E. Fagan. Since the publication of this initial paper, the Fagan inspection process (or variations of it) has been presented in many papers, books, and reports.[2,3] Without exception, all these reports claim that the use of software inspections improves product quality, as well as the productivity and manageability of the development process.

We have been involved with software inspections since 1981. On the whole, our experience has been consistent with the results reported in the literature. Still, we find few papers on the topic and much misunderstanding about the process and how to apply it. This article is an attempt to clarify what software inspections are, to explain how you can use them to improve both your process and your product, and to summarize what is known about their effectiveness.

What inspections are

The software-inspection process as it was defined by Fagan involves the interplay of five elements:

- six well-defined inspection steps

0740-7459/89/0500/0031/$01.00 © 1989 IEEE

(planning, overview, preparation, meeting, rework, and follow-up),

- four well-defined inspection roles (moderator, recorder, reader, and producer),
- the formal collection of process and product data,
- the product being inspected, and
- a supporting infrastructure.

In this article, "product" means the intermediate, development product, not the final product delivered to the customer. This development product is also known as the work product.

Inspection steps. Although most of the time and energy of an inspection is focused on the inspection meeting, the other five steps are essential for effective defect detection and correction. Two are often underused: overview and preparation.

The purpose of the overview is to bring all the inspectors to the point where they can easily read and analyze the inspection materials. A half-hour or hour presentation by the producer can often save the other inspectors two or three hours each in achieving the desired level of understanding of the inspection materials.

The preparation step lets the inspectors thoroughly understand the product so each can participate fully in the meeting. In the preparation step, the inspectors should first review the checklist for that type of inspection. They should also study the ranked distribution of defect types found by recent inspections. Inspectors should note how much time they spend in preparation; the moderator will collect this data and sum it to provide the total preparation time that went into each meeting. You use this information to assess the overall effectiveness of inspection in an organization.

Inspection roles. Creating software is a highly individualistic intellectual activity. Each software component is typically the work of one person or of a team of two or three people. In a sense, each piece of software is its own unique world. To effectively inspect a software product for defects, the inspection team must penetrate this world and discover its secrets. This is most easily done by those who are creating similar worlds. Thus, the selection of inspectors for a software product is often constrained to a handful of other developers working on similar or interfacing components.

The reader role is the least understood and the one most likely to be missing in an organization's inspection process. The purpose of the reader is to focus the inspection team's attention sharply on the details of the materials being inspected. To do this, the reader paraphrases and summarizes each section of the material. Where design or code logic is especially complex, the reader should be prepared to lead the group in acting out the logic with selected test cases. It may also be appropriate for the reader to prepare special exhibits to help the team keep all the relevant information together.

In a sense, each piece of software is its own unique world. To effectively inspect a software product for defects, the inspection team must penetrate this world and discover its secrets. This is most easily done by those creating similar worlds.

One of the most common questions asked by those who have not experienced a software inspection is "Why must the producer be present?" The answer is that if the producer is not present, the inspection team may spend too much effort trying to understand the product. Also, the presence of the producer enhances the communication from the other inspectors to the producer about the defects that must be fixed. When training inspectors, we always include a practice inspection. Rarely is the producer present at this practice inspection and, inevitably, at its conclusion the need for the producer's presence is obvious.

In our training courses, we find that the role of the reader is the hardest to explain, so we aren't surprised to find that some organizations use a process that does not include this role, that combines it with the moderator, or that assigns it to the producer. However, we often hear of organizations who later change their process by defining the reader as a separate role.

Data collection. The collection and analysis of data lies at the heart of the inspection process. The primary uses of this data are immediate process control and long-term process improvement. Although many organizations, for practical reasons, do not collect all of the data listed below, there is general agreement that these data should be collected for each inspection:

- the date the product was distributed for inspection,
- the date of the meeting,
- the date the rework was complete,
- whether this was an initial inspection or a reinspection, and the type of inspection it was,
- the identity of the product inspected,
- the size of the material inspected,
- the name of the moderator,
- the number of inspectors,
- whether an overview was held,
- the preparation time,
- the meeting duration,
- the number of defects of each type and severity found by the inspection, and
- the disposition (pass, rework, or reinspect) decided on by the inspection team.

If the staff hours used for rework is also collected, you can estimate the effectiveness of inspections versus testing.

Product. One characteristic that sets inspections apart from less formal examinations is the use of explicit entry criteria, which define when an inspection may begin. An inspection does not continue beyond the planning stage until the moderator verifies that the specified entry criteria have been met. Examples of entry criteria are:

- The existence of inspected predecessor products. For example, a detailed design document should not be inspected until an inspected (and reworked) high-

Completeness

1. Are all sources of input identified?
2. What is the total input space?
3. Are there any "don't care" sets in the input space?
4. Is there a need for robustness across the entire input space?
5. Are there any timing constraints on the inputs?
6. Are all types of outputs identified?
7. What is the total output space?
8. Are there any timing constraints on the outputs?
9. Are there sets that are in both the input space and the output space? Is there any state information?
10. What are all the types of runs?
11. What is the input space and the output space of each type of run?
12. For each type of run, is an output value specified for each input value?
13. Is the invocation mechanism for each run type defined? Are initial states defined?
14. Are all environmental constraints described?
15. Are sample input/output examples provided where appropriate?
16. Are all necessary performance requirements described?
17. Should reliability requirements be specified? Is an operational profile specified?

Ambiguity

1. Are all special terms clearly defined?
2. Does each sentence have a single interpretation in the problem domain?
3. Is the input-to-output mapping clearly defined for each type of run?

Consistency

1. Do any of the designated requirements conflict with the descriptive material?
2. Are there any input states that are mapped to more than one output state?
3. Is a consistent set of quantitative units used? Are all numeric quantities consistent?

Figure 1. Sample defect types for a requirements-inspection checklist.

level design is available.

• Conformance to pro forma standards. It can be useful to require that products be specially formatted for inspection. For example, you might require that the document to be inspected be run off with line numbers.

• Satisfaction of automated checks. Examples include spelling and grammar checks for text documents and error-free-compilation and complexity checks for code.

Supporting infrastructure. Inspections cannot occur spontaneously. They must be planned or supported by an individual or organization with the responsibility for them. We call this job the "software inspections coordinator." Its tasks include

• learning about inspections and convincing the project to try them,

• determining where inspections should be used on the project,

• creating and documenting the project-specific inspection procedures in an inspections manual,

• organizing training in the inspection process and keeping documentation and training up-to-date,

• collecting inspection data for the project's inspection database,

• issuing reports based on the information in the database, and

• analyzing the data in the database, comparing project results with those reported elsewhere, and making recommendations for process improvements.

You may apply inspections to more than one type of product. Typical products are requirements specifications, design documents, code listings, test plans, and test-case specifications. You may also apply inspections to hardware specifications and designs.[4] For each of these types of inspections, the project-inspection documentation must specify the inspection entry criteria, the job functions that should be represented at the inspection meeting, the recommended preparation rate, the recommended inspection rate, the type of defects to be inspected for and how they can be found, and the inspection exit criteria.

The definition of the types of defects to be sought in each type of inspection is an important part of the inspection infrastructure. These definitions focus the inspection meeting sharply on defect detection and keep it from wandering into questions of style or alternative designs. Also, the inspection-preparation checklist should be keyed to the defined defects and should provide suggestions on how and where to find defects of each type. Figure 1 gives some examples of defect types for a requirements inspection.

In addition to typing defects, you should also note their severity. Different organizations use different severity definitions, but two levels — major and minor — are usually sufficient. One important use of severity is to distinguish defects that will not affect execution from those that will. This distinction is important when you compare the effectiveness of inspections with testing, since only those defects that offset execution will be detected during testing.

Fagan also requires that defects be classified as missing, wrong, or extra. This classification provides additional information that you can use for process improvement, but it also adds additional complexity to the process. We do not know of any organization that has used missing, wrong, or extra data to quantitatively demonstrate an improvement in its development process.

Based on our experience, most organizations use an inspection process that is very similar to the one originally defined by Fagan. This similarity probably exists because most of the successful programs we know about are part of IBM or were initiated by people who had some direct contact with the IBM procedures.

Essential characteristics. In practice, each organization that implements software inspection varies the process. Whatever the variation, we believe the essential characteristics of software inspections — as opposed to other types of reviews — are a set of technical examinations

• performed by knowledgeable peers,

• of an explicit, completed product,

• at which the producer is present and participates,

• whose primary purpose is to find defects in a completed product so they may be corrected before they contaminate later activities,

• used routinely and according to some plan,

• supported by infrastructure elements,

• carried out with specific roles and following a specific set of steps,

• conducted by at least three people,

one of whom (usually called the moderator) is responsible for the effectiveness of the examination, and

- that produce consistent data that can be used for project management, quality control, and process improvement.

We require *all* of these elements for a review to qualify as an inspection.

Inspection metrics. Once you've implemented an inspection process, how do you judge its effectiveness? Useful generic metrics for evaluating and comparing inspection programs include

- the average preparation effort per unit of material,
- the average preparation rate per unit of material,
- the average examination effort per unit of material,
- the average explanation rate per unit of material,
- the average number of defects found per unit of material,
- the average staff-hours spent per defect,
- the average number of major defects found per unit of material, and
- average staff-hours spent per major defect.

The most commonly used units for inspected materials are thousand lines of source code and thousand lines of noncomment source code. For requirements and design inspections, page counts and line counts are also used. You can convert these units to equivalent lines of code when a project ends by relating the total number of pages or lines produced to the total number of lines of code in the final product.

What are typical values for these metrics? Is there any consistency in these values across organizations and across projects? The following is a sample of values from the literature and from private reports:

- The development group for a small warehouse-inventory system used inspections on detailed design and code. The results for detailed design were 3.6 hours of individual preparation per thousand lines, 3.6 hours of meeting time per thousand lines, 33.7 defects found per thousand lines, 1.0 hours per defect found, 7.1 major defects found per thousand lines, and 4.8 hours per major defect found. The results for code were 7.9 hours of preparation per thousand lines, 4.4 hours of meetings per thousand lines, 42.2 defects found per thousand lines, and 1.2 hours per defect found.
- A major government-systems developer[5] reported the following average results from inspections on more than 562,000 lines of detailed design: 5.76 hours of individual preparation per thousand lines, 4.54 hours of meetings per thousand lines, 17.88 defects found per thousand lines, and 0.58 hours per defect found. The same developer reported the following average results from inspection on 249,000 lines of code: 4.91 hours of individual preparation per thousand lines, 3.32 hours of meetings per thousand lines, 12.25 defects found per thousand lines, and 0.67 hours per defect found.

A major government contractor reported that their code-inspection data showed a surprising consistency over different applications and languages. Indeed, there is surprising consistency industry-wide.

- Two quality engineers from a major government-systems contractor[6] reported that their code-inspection data showed a surprising consistency over different applications and programming languages: eight to 12 defects found by inspections per thousand lines of new code and three to five staff-hours per major defect detected by inspections.

Given the great variety of applications and development environments in these and other efforts, there is a surprising consistency in the results. From detailed microcode to high-level business applications, from small engineering groups to large data-processing shops, there is general agreement that the use of inspections improves the quality of the final product.

The data also shows that, while inspections do not eliminate testing, they can significantly reduce the testing effort because inspections are from two to 10 times more efficient at defect removal than testing. Furthermore, regardless of the application or the language, you can expect inspections to find from seven to 20 major defects per thousand noncomment lines of source code and to find major defects at a cost of one to five staff-hours.

Inspection experience

To be effective, you must apply inspections to a substantial portion of a product. The best strategy is to apply inspections to all documents of a given type (such as all design descriptions or all new or modified code). Inspections can therefore take up a noticeable part of the project's time. Estimates range from 4 to 15 percent. How can you justify this expense? There are two answers.

1. Inspections improve quality. If everything else in the development process remains the same, the use of inspections will result in a higher quality product. This is especially true if you use inspections at various development stages, such as at system definition and system implementation. Using inspections at various stages will help you catch defects in later stages that have slipped through inspections at earlier stages.

The literature contains many claims for improved product quality based on the use of inspections, but these claims are only rarely quantified. Sometimes, relative quality-improvement figures are given. Two claims of improvement are:

- After the introduction of inspections, a large computer manufacturer reported that the number of defects found in released code was reduced by two thirds.[7]
- A major aerospace contractor that has a rigorous and comprehensive inspection program reported an after-release defect rate of less than 0.11 defects per thousand lines of source code.[8]

2. Inspections improve productivity. Along with improved quality, substantial productivity gains have also been reported. Such gains are possible for two reasons.

First, the longer a defect remains in a

product, the more costly it is to remove it. One large telecommunications firm has said that each software fault that must be corrected after the product is delivered costs $10,000. Such a sum can pay for a lot of inspection hours.

Second, except for reviews and walkthroughs, the only other widely applicable technique for detecting and eliminating software defects is testing. If inspections can detect and eliminate faults more cheaply than testing, they can be used to improve productivity and to shorten development schedules.

Inspection versus testing. Can inspections in fact detect and eliminate faults more cheaply than testing? For most development organizations, the answer is yes.

Our experience and that reported in the literature supports this:

- On a 6,000-line IBM business application, inspections found 93 percent of all defects discovered.[7]
- Rework at the design and coding levels is 10 to 100 times less expensive than if it is done in the last half of the development process.[1]

The basic trade-off behind these claims is the average amount of effort expended to detect and eliminate a defect during inspection versus the average effort during testing. Although it is easy to collect data during inspection that will yield an average for the number of hours required to detect a major defect, it is harder — and less precise — to collect data on the rework effort. The usual procedure is to collect this data during follow-up by asking the producer to estimate the effort spent in eliminating the defects uncovered by the inspection.

A further problem is that inspections and testing each detect different types of defects with different efficiencies. One way to address this problem is to try to classify the defects found at an inspection into minor defects and major defects. Major defects should be caught by testing. With this distinction in mind, consider the following reports on inspection defect detection and elimination efficiency:

- A banking computer-services firm that tried out inspections kept statistics on unit testing of some of the programs inspected and found that it took 4.5 hours to eliminate a defect by unit testing compared to 2.2 hours by inspection, according to data the firm supplied us.
- An operating-system development organization for a large mainframe manufacturer reported that the average effort involved in finding a design defect by inspections is 1.4 staff-hours compared to 8.5 staff-hours of effort to find a defect by testing, according to data the organization supplied us.
- A large switching-system project found that defects found in inspections cost an average of 10 times less to eliminate than defects found during development but outside inspections and reviews.[4]

Inspection has certain attributes that might account for these reports. At an inspection meeting, the product under examination is being considered in toto

Can you conclude that inspections can replace testing? In general, the answer is no. They can reduce the cost and schedule of testing, but they do not eliminate it.

— the entire input space, the entire output space, and the relationships between them are under continual scrutiny. While the inspection team may occasionally consider an isolated point in the input space and step through a program or detailed design to check that this input point results in the required output, it does so only to gain an understanding of the whole input space or some large subset.

In this sense, an inspection is like mathematics: It deals abstractly with entire classes rather than concretely with individual elements, as testing does. While an inspection requires preparation and meeting time, testing also requires preparation time to design and prepare test cases and to set up and take down test configurations.

When deciding whether to use code inspections, it is important that you consider this test-preparation time and the flexibility and speed with which tests can be selected, executed, and analyzed. Another consideration is the amount of effort involved in isolating and correcting the faults underlying the failures uncovered by testing. In testing, faults tend to be corrected one by one. In inspection, all the defects discovered and listed at the meeting can be corrected in a single rework pass.

Can you therefore conclude that inspections can replace testing? In general, the answer is no. Inspections can replace — and have replaced — testing at the unit level, but beyond that their effect is to reduce testing cost and schedule, not eliminate testing.

Other benefits. The regularity of inspection, combined with the fact that both process and product quality data is collected from each inspection, means that inspections give project management more definite and more dependable milestones than less formal review processes. The data collected by inspections can also be a key ingredient in making judgements about quality-improvement proposals. Christenson and Huang[9] provide an excellent example of how inspection data can be used to diagnose and cure quality problems.

Survey. Because Fagan's original paper on inspections and the later reports in the literature are almost unanimously agreed that inspections are effective, it would be useful to know to what extent software inspections are used both nationally and internationally, how much inspection processes vary and what effects these variations have, and whether there is a consistent set of organizational, managerial, and technical factors that accompany successful and unsuccessful inspection programs.

A comprehensive, scientific study of these questions is obviously beyond the scope of a few people acting privately, but we thought we might conduct a preliminary survey. By soliciting the cooperation of our clients and by tapping into our professional networks, we tried to contact all the English-speaking software-development organizations we could find that were using or had used a peer-review process that fell within our definition of a inspection.

We prepared an extensive interview form and an accompanying organization/project data sheet. Although we expected that many of those surveyed would not be able to give us any quantitative results, we were disappointed to find that they would tell us very little, if anything, unless they had published something on their use of inspections. We were therefore left only with the clients we had worked with or with contacts they referred us to.

We were sensitive to the proprietary issues involved and were willing to take whatever steps required to ensure confidentiality, but we usually met a blank wall when we stepped outside our immediate circle of clients.

Still, we were able to obtain some information on the use of inspections in 15 organizations. The data we obtained is generally consistent with the results cited above, but its narrow scope disappointingly limited our ability to answer the questions we posed.

Most organizations that have used inspections have found them to have significant benefit. In fact, one of our interviewees told us that she did not want to discuss her organization's use of inspections because they were a competitive advantage in bidding for software contracts! Of the organizations that have established rigorous and routine inspection processes, none reported that it later dropped inspections because they were ineffective or inefficient. In fact, several organizations were planning to extend inspections.

Inspections are a very useful tool. If more could be discovered and published about how to initiate and use inspections, the whole software industry could benefit. We suggest that a government agency like the US Commerce Dept.'s National Institute of Standards and Technology sponsor a comprehensive study. With the proper sponsorship, a cooperative approach could be taken that could benefit everyone. ❖

Acknowledgment

We gratefully acknowledge the help of John C. Kelly of the Jet Propulsion Laboratory in pointing out additional sources of information on inspections.

References

1. M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.*, Vol. 15, No. 3, 1976, pp. 182-211.
2. A.F. Ackerman, P.J. Fowler, and R.G. Ebenau, "Software Inspections and the Industrial Production of Software," in *Software Validation*, H.L. Hausen, ed., Elsevier, Amsterdam, 1984, pp. 13-40.
3. T. Gilb, *The Principles of Software-Engineering Management*, Addison-Wesley, Reading, Mass., 1987.
4. P.J. Fowler, "In-Process Inspections of Products at AT&T," *AT&T Technical J.*, March/April 1986, pp. 102-112.
5. J.H. Dobbins, "Inspections as an Up-Front Quality Technique," in *Handbook of Software Quality Assurance*, G.G. Schulmeyer and J.I. McManus, eds., Van Nostrand Reinhold, New York, 1987, pp. 137-177.
6. R.D. Buck and J.H. Dobbins, "Application of Inspection Methodology in Design and Code," in *Software Validation*, H.L. Hausen, ed., Elsevier, Amsterdam, 1984, pp. 41-56.
7. M.E. Fagan, "Advances in Inspections," *IEEE Trans. Software Eng.*, July 1986, pp. 744-751.
8. "Application for the National Aeronautics and Space Administration Excellence Award for Quality and Productivity," tech. report, IBM Federal Systems Div., Houston, 1986.
9. D.A. Christenson and S.T. Huang, "Code-Inspection Management Using Statistical Control Limits," *Proc. Nat'l Comm. Forum*, Nat'l Comm. Forum, Chicago, 1987, pp. 1095-1101.

A. Frank Ackerman is a coauthor of another article in this issue. His biography appears on p. 27.

Lynne S. Buchwald is president of Qualitech, a Brooklyn, N.Y., consulting firm specializing in software engineering and quality assurance. She has more than 12 years of experience in software development and software-engineering technology transfer and training, most recently concentrating in the areas of development methodologies and CASE tools.

Buchwald received a BA in linguistics from Trinity College and an MA in ancient near-Eastern languages from the University of Pennsylvania. She is a member of ACM and IEEE.

Frank H. Lewski is a member of the technical staff of the Quality Methods Group at AT&T Bell Laboratories. Since 1986, he has provided training and consulting in software engineering, including inspections, to more than 25 internal development projects.

Lewski received a BS in computer science from Pennsylvania State University and did graduate work in computer engineering at Syracuse University. He is a member of IEEE.

Address questions about this article to Ackerman at Institute for Zero-Defect Software, 200 Runnymede Pkwy., New Providence, NJ 07974.