Materials for Teaching Software Inspections

James E. Tomayko
SEI MSE Project

James S. Murphy
School of Computer Science
Carnegie Mellon University

Approved for public release.
Distribution unlimited.

This document was prepared for the

SEI Joint Program Office
ESC/ENS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This document has been reviewed and is approved for publication.

FOR THE COMMANDER

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.
This work was funded by the U.S. Department of Defense.

Copyright © 1993 Carnegie Mellon University

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scienti and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agen personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, At FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Services. For information on ordering, plea contact NTIS directly: National Technical Information Services, U.S. Department of Commerce, Springfield, VA 22161.

Copies of this document are also available from Research Access, Inc., 3400 Forbes Avenue, Suite 302, Pittsburgh, PA 15213.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

Preface

This educational materials package was developed for instructors of software verification techniques in graduate and undergraduate software engineering courses, and for those who teach industrial continuing education courses on the meaning and methods of software inspections.

Software inspections are a low-tech, highly effective verification technique. Research has consistently shown that the defect detection rate of inspections is higher than that of many traditional testing techniques. This package includes materials for demonstrating how to perform an inspection and also for "selling" students on the effectiveness of inspections. It complements EM-5, *Scenes from Software Inspections*, providing additional background material and exercises for using that set of educational materials.

Materials for Teaching Software Inspections contains the following:

1. Introductory essay on the history and results of software inspections
2. Annotated bibliography
3. Teaching suggestions for the instructor
4. Inspection materials: code, report forms, and actual results
5. Video: *Software Inspections: Utility or Futility*, a report on inspection results on an actual project
6. Video: *Candid Inspection*, which shows portions of an actual inspection

Note: Both videos are on the same tape cartridge, separated by titles. The inspection materials and videos (items 4, 5, and 6) can be ordered from the SEI. An order form is provided at the end of this document.

Software Inspections: History, Technique, and Results

Inspections are one of the most effective, yet lowest technology, quality assurance techniques that can be applied to software development at all stages of the life cycle. In conventional manufacturing, inspections by quality assurance specialists are an accepted practice. These inspections take place at selected points on an assembly line and are used to certify that parts and assemblies are correctly built to the specifications. Even with the advent of advanced tools such as X-ray and sonic devices coupled to expert systems, the most common form of inspection remains a human being making an experienced judgment.

Michael E. Fagan of IBM is credited with introducing the use of inspections in software development. Though many programmers use informal peer reviews of their code, Fagan made the formal inspection an integral part of the development process [Fagan 76]. Inspections have the obvious benefit of locating errors in code or other documentation. Fagan also viewed them as a contributor to disciplined development. By requiring inspections at various points in the development life cycle, software engineers not only improved the quality of the work products involved but also gained valuable data on defect injection and resolution.

The completeness of a software product is most often determined by testing. Inspections can also contribute to the determination of when a product is ready for shipment. In Fagan's original data, design and code inspections located 82 percent of all errors in a specific product. Acceptance test and actual use by a customer for six months revealed zero defects.

Inspections are used to inculcate quality throughout the development process, not just at the implementation stage. Even though most of the examples and data given in the literature refer to code inspections, successful project teams use inspections for all deliverables, including requirements and design documentation, as well as user manuals. Following the principle that the earlier a defect is found, the easier and cheaper it is to fix, the utility of inspections for work products other than code is apparent.

If an organization maintains records of inspection results and the results of all other defect identification methods, it can determine the average percentage of errors located and thus indicate when a product is ready to move on to the next step in development.

The Formal Inspection Process

Inspections are a team activity. Most inspections can be accomplished by four people: the producer of the item to be inspected (such as code, design, or user manual), a moderator to facilitate the process, and two technically competent inspectors. One person also acts as a recorder. The inspection is preceded by a period of preparation by each member of the group. Except for the moderator sometimes, team members usually need at least an hour to prepare. The inspection itself is usually limited to two hours because longer durations tend to reduce the efficiency of the team. After the inspection meeting is a follow-up period, beginning with a report and ending with the closure of open items such as the disposition of major defects.

Preparation includes two possible activities: a group overview and solo study. The first time an inspection team has to deal with the components of a particular product, a lead designer or someone with similar knowledge of the software product gives an overview of the requirements and design. Each inspector spends time individually studying the document or code prior to its inspection.

The inspection meeting begins with team members reporting the time each spent in preparation, a valuable metric. Then one of the inspectors acts as a reader, going through the code or design one line or item at a time. Each member of the team has an opportunity to ask for clarification or point out a defect in the current item. The recorder writes down the defects, which are later classified as "minor" or "major" (a minor defect could be a syntax error such as a missing semicolon in code; a major defect could be a failure to implement a requirement either through logic error or omission).

After the inspection is completed, the recorder prepares a report listing metrics such as preparation time, elapsed time of the inspection meeting itself, and the major and minor defects (sample report forms are in [Fagan 76]. The minor defects are usually turned over to the original producer of the inspected material for rectification. The major defects may require the attention of the configuration control board or other change control mechanism. Defect repair is accompanied by any necessary changes to documentation prior to closure. Records of defect type and location in the product can be used for causal analysis and continuous process improvement.

Pitfalls of Inspections

One of the greatest dangers of inspections is the inability of producers and inspectors to differentiate the product from the person creating the product. Software engineers are sometimes embarrassed by the inspection process when their carelessness or bad judgment is revealed in a "public" setting. When a particular product has many errors, inspectors may sometimes get caught up in a "feeding frenzy," attacking the producer. The moderator is charged with the responsibility of keeping the inspection focused on the product and also maintaining a professional tone during the meeting. Under no circumstances should the results of inspections be used as part of performance appraisals.

Another pitfall is attempting to use inspections without adequately budgeting time for preparation and follow-up. Insufficient preparation reduces the number of lines or items that can be inspected in a particular meeting because time is spent in trying to understand the code. Insufficient follow-up often means that defects remain, defects that may not be found by later testing. Since the results of the inspections in locating defects are so outstanding, it is much cheaper to spend time at these early stages in product development than to find and repair defects later.

Results of Inspections

The results of using formal inspections are most marked in the decreasing cost of rework and in the side effect of improving individual software engineering skills. Fagan's early data indicated that 82 percent of all errors in applications software development could be found with inspections. A later report of a 6,000-line business application indicates that inspections found 93 percent of all defects [Ackerman 89]. Since inspections can be conducted even prior to unit testing, they are inexpensive compared to finding errors in integration or acceptance testing phases. The Jet Propulsion Laboratory estimates it saves $25,000 in each inspection [Bush 90]. However, the process does not lend itself to saving more money through acceleration: Russell reports that defects found per thousand lines of code dropped from 50 to 15 when the pace of inspections increased from 150 to 450 lines per hour [Russell 91]. Finally, although it has not been quantified, software engineers report that their own programming skills improve as a result of participating in inspections. This is not so surprising since people are taught to be better writers by reading good writing and by receiving critiques. The same principle can apply to programming.

Using the Materials

This educational materials package, together with EM-5, *Scenes from Software Inspections*, provides the instructor with a variety of materials to use in teaching the techniques of software inspection.

In a recent course on software verification techniques, the authors used the following assignment and activity sequence:

1. Read [Fagan76], [Ackerman89], and [Russell91], and view the video *Software Inspections: Utility or Futility*. Then write an essay on the following: What are the potential advantages and disadvantages of inspection technology in your personal software development field? In what ways can advances in information technology be utilized to improve the inspections process?
2. Attend software inspections training, which uses the *Scenes from Software Inspections* video and the *Candid Inspection* video as a basis for demonstration and discussion.
3. Participate in an inspection, including all preparation and follow-up work.
4. Write an individual evaluation of the inspection you participated in, commenting on its effectiveness at defect identification and on its process.

Attachment A contains the design overview, code, and sample results of the assignments specified here so that instructors can see what might be expected from students who do these assignments. Attachment E is the hard-copy version of the slides from the *Software Inspections: Utility or Futility* videotape, and Exhibit F is the design overview and code inspected in *Candid Inspection*.

Another sequence of assignments and activities in a course with a lab component could be the following:

1. The instructor lectures on the origins of inspections and their effectiveness. Students prepare by reading [Fagan76], and [Russell91] prior to attending the lecture; the instructor uses *Software Inspections: Utility or Futility* during the lecture as additional material.
2. The instructor lectures on how to conduct an inspection, reviewing the roles and method. *Scenes from Software Inspections* and *Candid Inspection* are used as examples.

3. Students are split into teams and conduct an inspection during a laboratory period. The instructor, hopefully with some help, listens in to the inspection teams to ensure that they are performing the inspection correctly.
4. Attachments C and D contain design documentation, pre-inspected code, inspection reports, and post-inspection code for two different modules of software that is being used in a robot to maintain the Space Shuttle thermal protection system. Either of these may be used for the exercise. Instructors should distribute Attachment B, which contains the coding standards (violations of coding standards are considered defects) and system header files for the example modules, along with one of the pre-inspected code listings.
5. After the inspection, distribute the report and resulting repaired code to compare with the results of the in-class inspection. Alternatively, one of the complete exhibits could be used for an in-class walkthrough and the other for an actual inspection.

Annotated Bibliography

[Ackerman 89] Ackerman, Frank A.; Buchwald, Lynne S.; and Lewski, Frank H. "Software Inspections: An Effective Verification Process." *IEEE Software* (May 1989): 31-36.

The authors recount experiences with inspections at AT&T. The article contains a useful chart of a requirements inspection checklist. The section on experiences is a good survey of industry practice and results.

[Bush 90] Bush, Marilyn. "Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory," 196-198. *Proceedings of the 12th International Conference on Software Engineering*, IEEE Computer Society Press, 1990.

A "work in progress" paper describing the Jet Propulsion Laboratory's initial uses of software inspections.

[Fagan 76] Fagan, Michael E. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal 15*, 3 (1976): 182-211.

The original paper describing the software inspection technique. Fagan gives detailed specifications for the roles of participants in an inspection and the content of reports. He also gives an overview of early results of the use of inspections in IBM. His emphasis in the paper is how inspections are just a part of the overall process control of the development of software.

[Fagan 86] Fagan, Michael E. "Advances with Inspections." *IEEE Transactions on Software Engineering* (July 1986): 744-751.

This paper is more a ten-year update than a report on spectacular advances. Fagan has considerably more results to survey.

[Russell 91] Russell, G. W. "Experience with Inspection in Ultralarge-Scale Developments." *IEEE Software* (January 1991): 25-31.

This paper is an exceptionally good report of results on large projects. Russell does a data analysis that reveals such metrics as the rate of defect detection as a function of the speed at which an inspection is conducted. A very convincing case for the use of inspections on big projects.

Attachment A

Example Inspection Exercise

Attachment A

Contents

Attachment B

Coding Standards and System Headers for Exercises

Attachment B

Contents

Attachment C

Design and Code for State Sensor Inspection Exercise

Attachment C

Contents

Attachment D

Design and Code for Master Sequencer Inspection Exercise

Attachment D

Contents

Attachment E

Slide Set for Video *Formal Inspections: Utility or Futility?*

Attachment F

Design and Code for Video *Candid Inspection*

Attachment F

Contents

Section 1

MAPS Software Overview

Section 2

State Sensor Producer's Overview

Section 3

state_sensor Program

Section 4

Module 2 - Software Inspections

Section 5

Lessons Learned from the State Sensor Inspection

Section 6

An Essay on Software Inspections

Section 7

Software Inspections

Section 1

MAPS Coding Standards

Section 2

maps.h

Section 1

MAPS Design - State Sensor

Section 2

state.before

Section 3

state.resolve

Section 1

MAPS Design - Master Sequencer

Section 2

master.before

Section 3

master.resolve

Section 4

master.after

Section 1

MAPS Design - Joystick Manager

Section 2

MAPS Document - Joystick Manager

Section 3

joystick.c.lined