AD-A273 690

IDA PAPER P-2876

SOFTWARE INSPECTION INSERTION EFFORTS FOR THE BALLISTIC MISSILE DEFENSE ORGANIZATION

Bill Brykczynski, *Task Leader*

Reginald N. Meeson
David A. Wheeler

September 1993

Prepared for
Ballistic Missile Defense Organization

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

The work reported in this document was conducted under contract MDA 903 89 C 0003 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 1993	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Software Inspection Insertion Efforts for the Ballistic Missile Defense Organization	5. FUNDING NUMBERS MDA 903 89 C 0003 Task Order T-R2-597.21
6. AUTHOR(S) Bill Brykczynski, Reginald N. Meeson, David A. Wheeler	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Defense Analyses (IDA) 1801 N. Beauregard St. Alexandria, VA 22311-1772	8. PERFORMING ORGANIZATION REPORT NUMBER IDA Paper P-2876
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) BMDO/GSI The Pentagon, Room 1E149 Washington, D.C. 20301-7100	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, unlimited distribution: October 22, 1993.	12b. DISTRIBUTION CODE 2A

13. ABSTRACT (Maximum 200 words)

Software inspection is an industry-proven process for improving quality and reducing cost and schedule risks during software development. Software inspections are highly effective at detecting and eliminating defects, especially during the early phases of software development. Unfortunately, the software inspection process is not widely used by DoD contractors. The Ballistic Missile Division Organization (BMDO) tasked IDA to promote awareness of this defect detection process to several BMD program offices. This report summarizes information collected, developed, and distributed to those program offices. IDA efforts to insert the process into BMDO policies, standards, and the Brilliant Eyes software development program are described.

14. SUBJECT TERMS Software Inspections; BMDO; Software Development; Software Defects.			15. NUMBER OF PAGES 108
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

IDA PAPER P-2876

SOFTWARE INSPECTION INSERTION EFFORTS FOR THE BALLISTIC MISSILE DEFENSE ORGANIZATION

Bill Brykczynski, *Task Leader*

Reginald N. Meeson
David A. Wheeler

September 1993

IDA
INSTITUTE FOR DEFENSE ANALYSES
Contract MDA 903 89 C 0003
Task T-R2-597.21

PREFACE

This paper was prepared by the Institute for Defense Analyses (IDA) for the Ballistic Missile Defense Organization (BMDO), under the task titled "Software Testing of Strategic Defense Systems." The objective of the task was to assist the BMDO in planning and monitoring software testing research, development, and practice, with particular emphasis on the transition and application of software inspection methods.

This paper was reviewed by the following members of IDA: Dr. Dennis W. Fife, Dr. Michael Frame, Ms. Audrey A. Hook, Dr. Richard J. Ivanetich, Ms. Beth Springsteen, and Mr. Glen White.

Table of Contents

Table of Contents

List of Figures

List of Tables

SUMMARY

The software inspection process is one of the most impressive examples of "best practice" in the commercial software industry. Software inspection has proved to be highly effective at improving quality, reducing schedule risks, eliminating defects and reducing development costs in complex systems. Up to 80 percent of all software defects can be identified and eliminated early during software development by using inspections. When inspections are combined with normal testing practices (e.g., dynamic analysis, regression testing), defects in fielded software can be reduced by a factor of 10. By decreasing the amount of rework typically experienced in development, inspection increases productivity and reduces costs and delivery time. Cost and schedule reductions for typical applications are on the order of 30 percent.

The Cost of Defects in Software Development

Figure S-1 reflects the high cost of testing in software development.[1] It is commonly held that at least 40 percent of software development is spent on unit, integration, and system testing. Upon closer examination, however, a large part of the testing time is not actually used for testing. The extra testing time is taken up by correcting defects that are found by tests, as shown in Figure S-2. As much as 40 to 50 percent of typical software

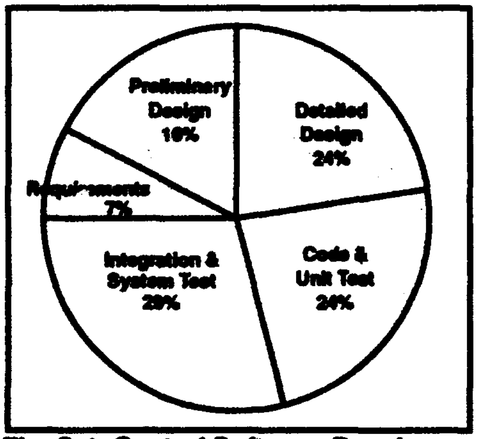


Fig. S-1. Cost of Software Development

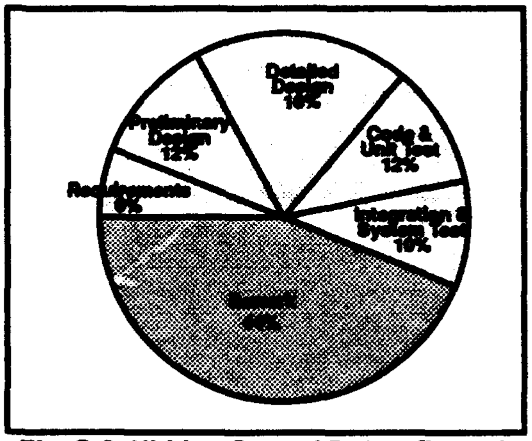


Fig. S-2. Hidden Cost of Defect Rework

[1] The data in Figures S-1 and S-2 are from Boehm, Barry W., "Improving Software Productivity," *IEEE Computer*, Vol. 20, No. 9, September 1987, pp. 43-57.

development effort is devoted to correcting defects. Such a high level of rework does not reflect positively on the process and, hence, is not often openly reported.

Many defects found in testing are directly traceable to requirements and design flaws that could have been detected earlier. Defects detected soon after they are introduced are relatively easy and inexpensive to correct. When not detected until later in development, the cost of correcting these same defects is compounded by having to undo work based on the incorrect foundations. Finding in the last stages of testing that a requirement was not correctly understood can easily lead to cost and schedule overruns. The cost of correcting the same problem in the requirements phase is usually negligible. The most effective process known for finding defects across all stages of software development is inspection.

The Software Inspection Process

Inspections are detailed examinations of work-in-progress. The objective of inspections is to identify defects. Co-workers study work products independently and then meet to review the work in detail. No time is spent during an inspection meeting discussing how to correct a defect. Corrections are left for the author to make later. Work products are small but complete chunks of work—on the order of 200 to 250 lines of code. Requirements, designs, and other work products are inspected in similar-sized chunks (e.g., four to six pages of text). Work products are considered work-in-progress until the inspection and any necessary corrections are completed. Inspection teams are formed by four to five coworkers. Each inspector will typically spend one to four hours reviewing the work product and related information before an inspection, depending on the level of familiarity with the material.

Responsibilities for several roles are assigned to inspection team members. The most important role is that of the moderator who runs the inspection meeting and is responsible for keeping the inspection on track. The reader paraphrases the work product while the author and other inspectors read along and comment on discrepancies. The recorder records the location and a brief description of all defects encountered. Other roles may exist depending upon the type of work product.

Inspection meetings are generally limited to a maximum of two hours. It has been observed that after two hours, the number of defects detected drops off significantly.

The two principal outputs from an inspection are a list of defects for the author to correct and an inspection summary for management that describes what was inspected, who the inspectors were, and the number and severity of defects found. In addition, any systemic defects that are identified are reported for consideration in general process improvement.

Inspections in the Department of Defense and Ballistic Missile Defense Organization

Although the benefits of the inspection process have been widely published, the process is not often used by Ballistic Missile Defense Organization (BMDO) and Department of Defense (DoD) contractors. There are several likely reasons for this state of practice. Contractors and program offices may be unaware of the inspection process and its benefits. Contractors may believe they already perform inspections but actually employ less rigorous and effective walkthrough methods. Contractors and program offices may overreact to the up-front cost of inspections and ignore the potential value of substantial downstream cost and schedule reductions. Contractors may also view including the cost of inspections in their bids as reducing competitive advantage, since competitors may submit higher-risk but lower-cost bids without inspections. If the program office has not included a requirement or evaluation points for the use of inspections in its Request for Proposal, it may not have grounds to reject higher-risk proposals.

IDA Inspection Activities

During the course of this study, the activities of the Institute for Defense Analyses (IDA) were concentrated in three areas. First, a comprehensive analysis of the inspection process was performed. This analysis was based on a review of a wide range of published information on the inspection process, discussions with DoD and commercial industry contractors on their software review practices, and conversations with people who provide inspection training. Second, IDA reviewed BMDO software policies, standards, and guidelines and suggested improvements regarding their treatment of the inspection process. Third, IDA assisted in an effort to insert inspections into the software development process of the Brilliant Eyes program.

Recommendations

1. Senior BMD executives should strongly encourage the use of the inspection process for all BMD software development. The BMDO General Manager and Service Program Executive Officers can influence element program managers to consider implementing the inspection process. Without senior-level leadership and commitment it will be much more difficult to convince program managers and their staff to implement the inspection process. Promoting advanced technology, especially an industry-proven process such as inspections, is an appropriate function for BMDO senior executives.

2. Future BMD RFP's should provide explicit incentives for contractors to bid the inspection process. Evaluation of the proposed inspection process should be a specific technical criterion in making the source selection decision. By incentivizing inspections in the RFP, contractors can then reflect the upfront costs of inspection in their bids. It should be noted, however, that inspection benefits are unlikely if BMDO relies primarily on contract wording to obtain inspection usage. Active interest by BMDO and the program offices to communicate the importance of the inspection process to the contractor is also necessary.

3. BMDO should provide funding for initial inspection training to all BMD element program offices. By specifically allocating training funds, BMDO emphasizes the importance placed on inspections. This funding should be allocated for two types of training: full training for all program office staff involved with software development, including program manager, software leads, and acquisition officials, and initial training for existing contractor personnel. The program office should be encouraged to provide subsequent training to existing contractors, or to allow the contractor to directly charge inspection training costs against the contract. For maximum benefit, all contractor development personnel, and their managers, should receive inspection training.

4. Summary inspection data and results should be collected from software developers by the program offices. This data is needed to assess the effectiveness of inspections and to manage the software development process. Benefits from inspections will be available immediately, quantifiable in terms of defects detected and rework avoided. Results from "success stories" can be used immediately to increase awareness and adoption of inspections BMD-wide.

1. INTRODUCTION

1.1 PURPOSE AND SCOPE

The process of software inspection is a highly effective technique for improving quality and reducing cost and schedule risks during software development. This paper presents the results of an effort by the Institute for Defense Analyses (IDA) during fiscal year 1993 to increase awareness of the software inspection process within the Ballistic Missile Defense (BMD) program. This paper provides information on the software inspection process that is useful from several perspectives:

a. *Program managers* will find a concise overview of software inspection that examines the problems addressed by inspections, the industry benefits reported from use of the process, and the costs involved in implementing the process.

b. *Program office staff* will find a detailed examination of Department of Defense (DoD) and BMD standards, guidelines, and policies with regard to inspection, and a general approach used by IDA to promote the use of inspections for a BMD program office. Commonly asked questions about inspections and guidance on contracting for the inspection process will also be of interest to program office staff.

c. *Government contractors* will find the comprehensive bibliography useful in exploring published industry experience and data on the inspection process, including methods for inserting the process into an organization. The discussion on differences between the inspection process and less rigorous review processes (e.g., walkthroughs) can help contractors determine whether or not they in fact are using inspections.

Recommendations are provided for increasing the use of the inspection process in BMD software development.

1.2 BACKGROUND

In the late 1980's, the Strategic Defense Initiative Organization (SDIO) tasked IDA to examine the technology needed for testing the software for a Strategic Defense System (SDS). An assessment of the state of the art in software testing techniques and their potential applicability to SDS software was performed [Youngblut 1989]. In 1991, IDA produced a paper that described an SDIO initiative designed to develop and deploy needed software testing technology to ensure the development of reliable and cost-effective software for the Strategic Defense Initiative (SDI) program [Brykczynski 1992]. The SDIO then asked IDA to identify high-payoff software testing techniques that should be focused upon and emphasized by SDI program offices. From the previous analysis of existing software testing techniques, the process of software inspection was found to be, by far the most effective at detecting defects throughout the software development lifecycle.

Even though the benefits of inspections have been verified and documented, and the process is used by commercial software industry leaders, inspections are not commonly practiced by DoD contractors. Recognizing the current state of DoD inspection practice, the Ballistic Missile Defense Organization (BMDO) tasked IDA in fiscal year 1993 to increase the exposure of the software inspection process to BMD program offices.

2. OVERVIEW OF SOFTWARE INSPECTIONS

In this section we describe what software inspections are and explain how they contribute to eliminating defects and improving the software development process. We start out with a review of the costs of software development, focusing on the costs of testing and rework. Next, we describe the inspection process and show how it contributes to defect, cost, and schedule reduction. We conclude this section with examples of the success of inspections reported in the literature.

2.1 COST OF SOFTWARE TESTING

Figure 1 shows a pie chart that reflects the high cost of testing in software development [Boehm 1987]. Between 45 and 50 percent of the effort is taken up by requirements

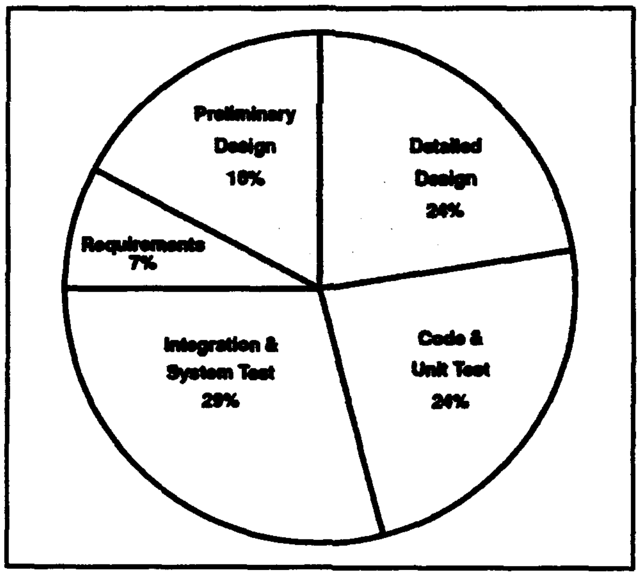


Figure 1. Cost of Software Development

and design. Other sources indicate coding consumes 10 to 15 percent [Jones 1991]. Roughly 40 percent of software development, therefore, is spent on unit, integration, and system testing. Upon further investigation, however, we found that a large part of the testing time is not actually used for testing. The extra testing time is taken up by correcting defects that are found by tests. This is shown in Figure 2 [Boehm 1987].

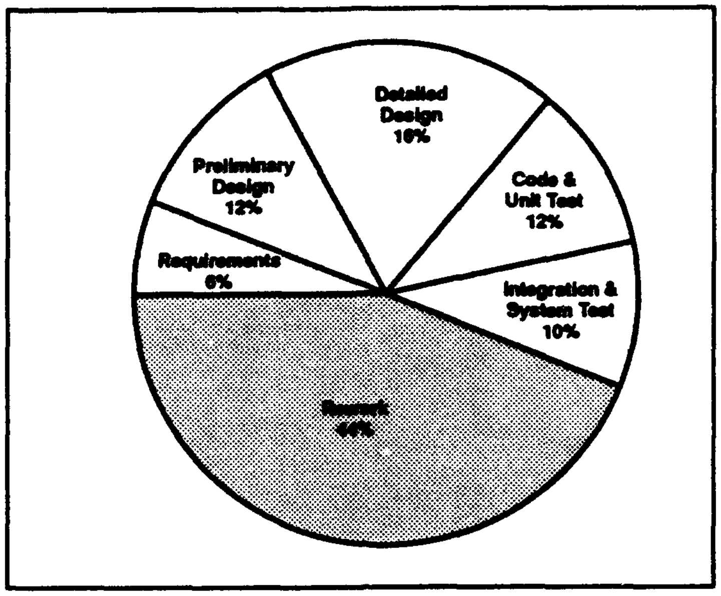


Figure 2. Hidden Cost of Defect Rework

At least 40 to 50 percent of typical software development effort is devoted to correcting defects. Such a high level of rework does not reflect positively on the process and, hence, is not often openly reported. Figure 3 shows how the time spent correcting defects

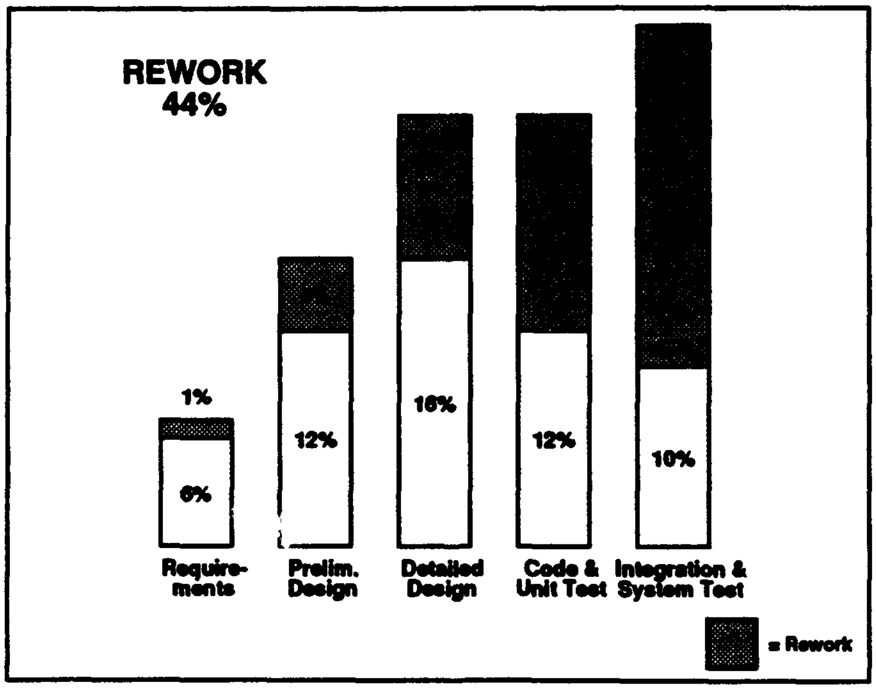


Figure 3. Distribution of Rework Cost

is distributed over project phases [Boehm 1987]. Rework grows steadily and by the final integration and test phase typically consumes two-thirds of the development effort.

Many defects found in testing are directly traceable to requirements and design flaws that could have been detected earlier. Defects that are detected soon after they are introduced are relatively easy and inexpensive to correct. When they are not detected until later in development, the costs for correcting these same defects are compounded by having to undo work that was based on the incorrect foundations. Finding that a requirement was not correctly understood in the last stages of testing can easily lead to cost and schedule overruns. The cost of correcting the same problem in the requirements phase is usually negligible. The most effective process known for finding defects across all stages of software development is inspection.

2.2 SOFTWARE INSPECTION PROCESS

Inspections are detailed examinations of work in progress. The objective of inspections is to identify defects. Co-workers study work products independently and then meet to review the work in detail. No time is spent during an inspection meeting discussing how to correct a defect. Corrections are left for the author to make later.

The inspection process can be applied to many different types of work products found in software development, such as requirements, design, code, test cases, etc. Work products are small but complete chunks of work—on the order of 10-20 pages of requirements, 200 to 250 lines of code, etc. Work products are considered work in progress until the inspection and any necessary corrections are completed.

Inspection teams are formed by 4 to 5 coworkers. Each inspector will typically spend 1 to 4 hours reviewing the work product and related information before an inspection, depending on the level of familiarity with the material.

Responsibilities for several roles are assigned to inspection team members. The most important is the role of moderator, who runs the inspection meeting and is responsible for keeping the inspection on track. The reader paraphrases the work product while the author and other inspectors read along and comment on discrepancies. The recorder, also called the scribe, records the location and a brief description of all defects encountered. Other roles may exist depending upon the type of work product.

Inspection meetings are generally limited to a maximum of two hours in length. It has been observed that, after two hours, the number of defects found drops off significantly [Fagan 1976a].

The two principal outputs from an inspection are a list of defects for the author to correct and an inspection summary for management that describes what was inspected, who the inspectors were, and the number and severity of defects found. In addition, any systemic defects that are identified are reported for consideration in general process improvement.

2.3 BASIC INSPECTION STEPS

As shown in Table 1, the inspection process is composed of six basic steps: planning, overview, preparation, examination (the group meeting), rework, and follow-up. These basic steps are derived from several published descriptions of the inspection process [IEEE 1028-1988, Fagan 1986, Gilb 1987]. Variations in the process do exist. For example, Fagan uses the term "inspection" and Gilb uses the term "meeting" instead of the word examination. Gilb also adds two additional steps: a "third hour" step for ideas suppressed at the meeting, and a "causal analysis" step in which meeting statistics are analyzed.

Table 1. Basic Inspection Steps

Inspection Step	Actions Taken
Planning	The author assembles the materials for the moderator. The moderator assures that the entry criteria are met, and assures that the inspection team is selected, scheduled, and that materials are distributed.
Overview	The author presents an overview of the product to educate the other inspectors (and possibly other project personnel who could benefit from such a presentation). This step is optional.
Preparation	The inspectors individually become familiar with the software element and use the inspection checklist appropriate to the product type.
Examination	The inspectors meet as a group to find defects. The moderator asks for preparation times and manages the meeting. The reader, who is not the author, presents the material (generally paraphrasing). A defect list is created and reviewed. The team determines if the material is to be (1) accepted, (2) reworked but only the moderator needs to verify the rework, or (3) reworked and re-inspected.
Rework	The author revises the materials, addressing all items on the inspection defect list.
Follow-up	Verify that the rework has been correctly performed. This may be a review with the moderator, or may involve re-inspection.

2.4 MANAGEMENT PARTICIPATION

When managers participate in inspections, the process tends to identify only superficial defects. The inspectors may become worried that identifying a defect in the product negatively impacts the author's professional evaluation, and may fear later retaliation in kind. Thus they may choose to only find superficial defects. By finding superficial defects, inspectors report respectable numbers of defects and authors are not embarrassed by serious defects in their work. The result is that the inspections are ineffective and serious defects remain in work products. To avoid this phenomenon, managers should not participate directly in inspections. Instead, they should monitor inspection summary reports to ensure that inspections are being conducted effectively and measure the overall quality of work by the number of defects that remain in finished products.

2.5 OTHER TYPES OF REVIEWS

There are several other types of reviews that can easily be confused with inspections. First are the major MIL-STD-1521B program milestone reviews, such as System Requirements Review (SRR), Preliminary Design Review (PDR), Critical Design Review (CDR), and Test Readiness Review (TRR) [MIL-STD-1521B]. Although defects may be exposed in these reviews, milestone products, which represent completion of major stages of work, are usually much too large to review in detail. The purpose and format of inspections is quite different from these reviews. Inspections are not intended to substitute for these reviews.

Walkthroughs and informal peer reviews are similar in many ways to inspections. Walkthroughs and informal peer reviews are less rigorous and, for example, may skip the preparation time, eliminate roles, permit the product author to lead the review, eliminate the follow-up on corrections, and eliminate data collection for measuring effectiveness and process improvement. These deviations weaken the process, making walkthroughs and informal peer reviews significantly less effective at finding defects than inspections. More detail on how to distinguish between walkthroughs, informal peer reviews, and inspections is presented in Section 3.

2.6 BENEFITS FROM SOFTWARE INSPECTIONS

Figure 4 shows the conventional sequence of software development phases. The numbers located by the boxes show the numbers of defects that are passed on from one development phase to the next. The 50 defects per thousand lines of code (KLOC) at the completion of unit testing and the 10 per KLOC delivered to the field are well-documented industry averages [Jones 1986, Jones 1991]. Anecdotal data from individual projects was

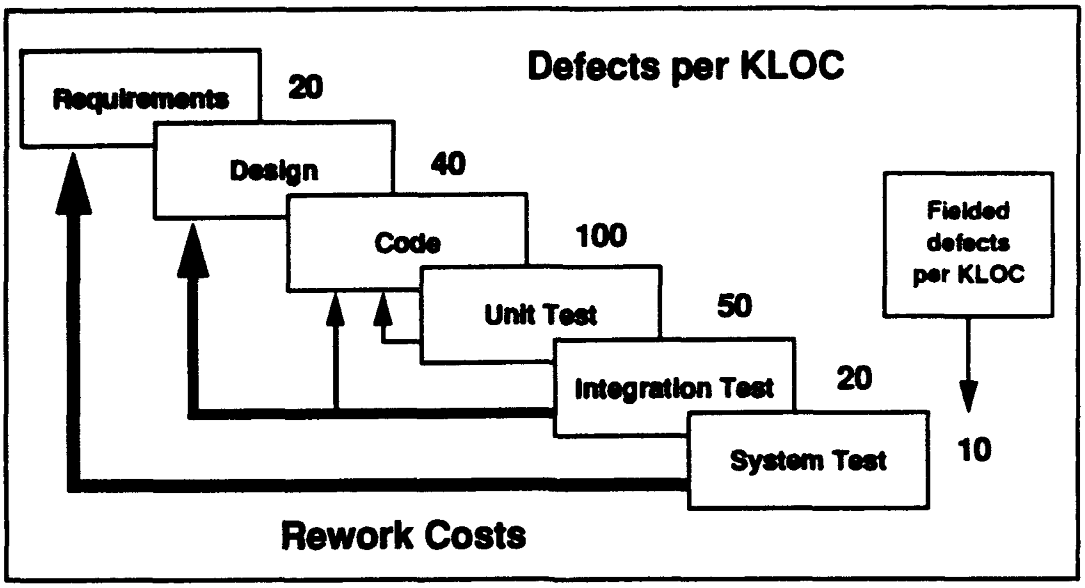


Figure 4. Typical Software Defect Profile

used to fill in round numbers for defects in other phases

The width of the arrows in Figure 4 that point back from the testing phases to earlier construction phases suggest the relative costs of correcting defects that were not detected earlier. For example, a misunderstood requirement that is not recognized until the final system testing phase will typically have the highest cost to correct. It may also delay system delivery.

Figure 5 depicts the introduction of inspections for requirements, design, and code into the development process. The objective of these inspections is to find all the defects at each phase and to proceed to the next phase with a completely correct basis. Even though the ideal of completely eliminating defects is rarely achieved, success rates of 80 percent are consistently reported [Fagan 1986, Freedman 1982]. Therefore, the number of defects passed on to succeeding phases is reduced significantly. Although not shown in Figure 5, inspections can also be used to ensure that other products, such as test procedures and data, are correct.

The numbers of defects passed from phase to phase when inspections are used are dramatically reduced. (The numbers in parentheses in Figure Defect Profile with Inspections on page 9 show the numbers of defects when inspections are not used.) The cumula-

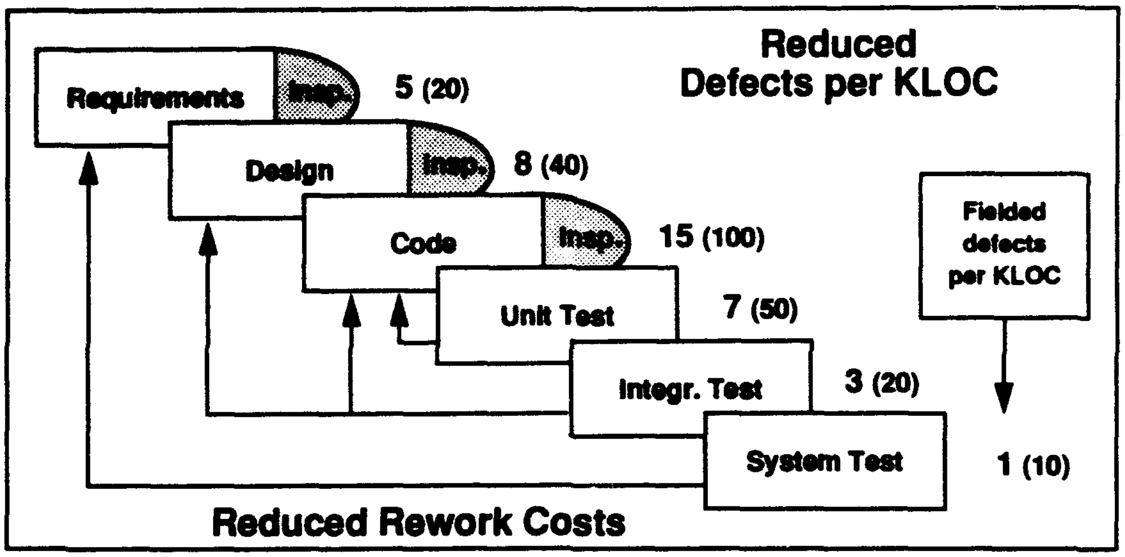


Figure 5. Defect Profile with Inspections

tive effect of requirements, design, and code inspections along with normal testing, is an order of magnitude reduction in the number of defects in fielded products. In addition to this gain in quality, there is a corresponding gain in productivity because the amount of rework needed to correct defects during testing is significantly reduced. This reduction in rework is illustrated in Figure 5 by the reduced width, in comparison with Figure 4, of the arrows pointing back from testing phases

Inspections reduce the number of defects in work products throughout the development process. More defects are found earlier, where they are easier and much less expensive to correct. Inspections are also able to uncover defects that may not be discovered by testing. Examples of this include identifying special cases or unusual conditions where an algorithm would produce incorrect results. In addition to finding defects, inspections serve as a training process where inspectors (who are also authors of similar work products) learn to avoid introducing certain types of defects.

In describing software process improvement activities at Raytheon, Dion uses the graph shown in Figure 6 to illustrate that they have been able to reduce the cost of software rework by a factor of four [Dion 1993]. The "Cost of rework" curve in this figure shows a steady decrease since the start of their process improvement initiative. He attributes the reduction in rework primarily to software inspections this way: "In our case, the cost of design and coding rose slightly because formal inspections replaced informal reviews. However, it was this very change that enabled us to achieve rework savings in uncovering source-code problems before software integration and eliminating unnecessary retesting." Although more time is now spent fixing defects during design and coding, these small increases are completely overshadowed by the savings achieved by not having to fix them later during integration and testing. The cost of fixing coding defects during integration, for example, has been reduced by a factor of five.

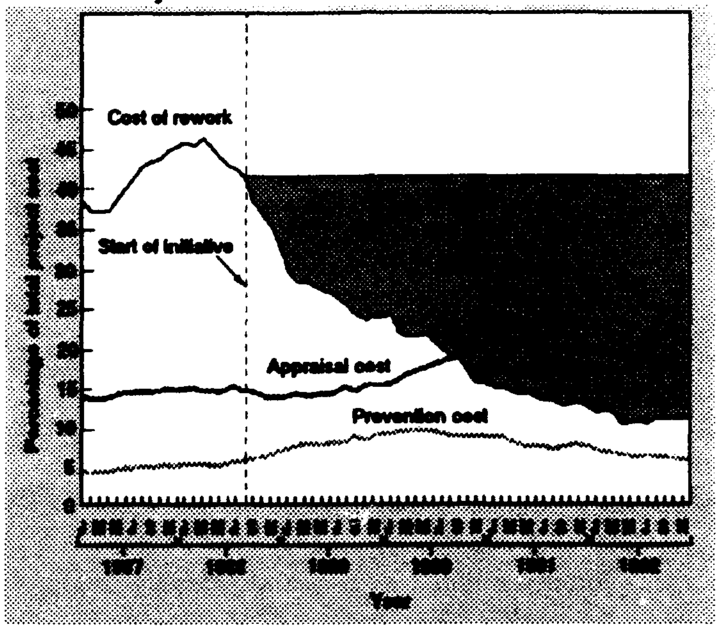


Figure 6. Raytheon Savings from Reduced Rework

2.7 COST OF INSPECTION

Inspections require an up-front investment of approximately 15 percent of total software development costs. This investment pays off in a 25 to 35 percent overall increase in productivity. This productivity increase, as demonstrated by Fagan in industry studies, can be translated into a 25 to 35 percent schedule reduction [Fagan 1986].

Figure 7 shows typical spending-rate profiles for development projects with and without inspections. The taller curve shows increased spending early in the project, which reflects the time devoted to inspections [Fagan 1986]. This curve then drops quickly through the testing phases. The broader curve, for projects that do not use inspections, shows lower initial spending but much higher spending through testing, which reflects the 44 percent rework being done [Boehm 1987]. The area under the inspection curve, which represents total development cost, is approximately 30 percent less than the area under the non-inspection curve.

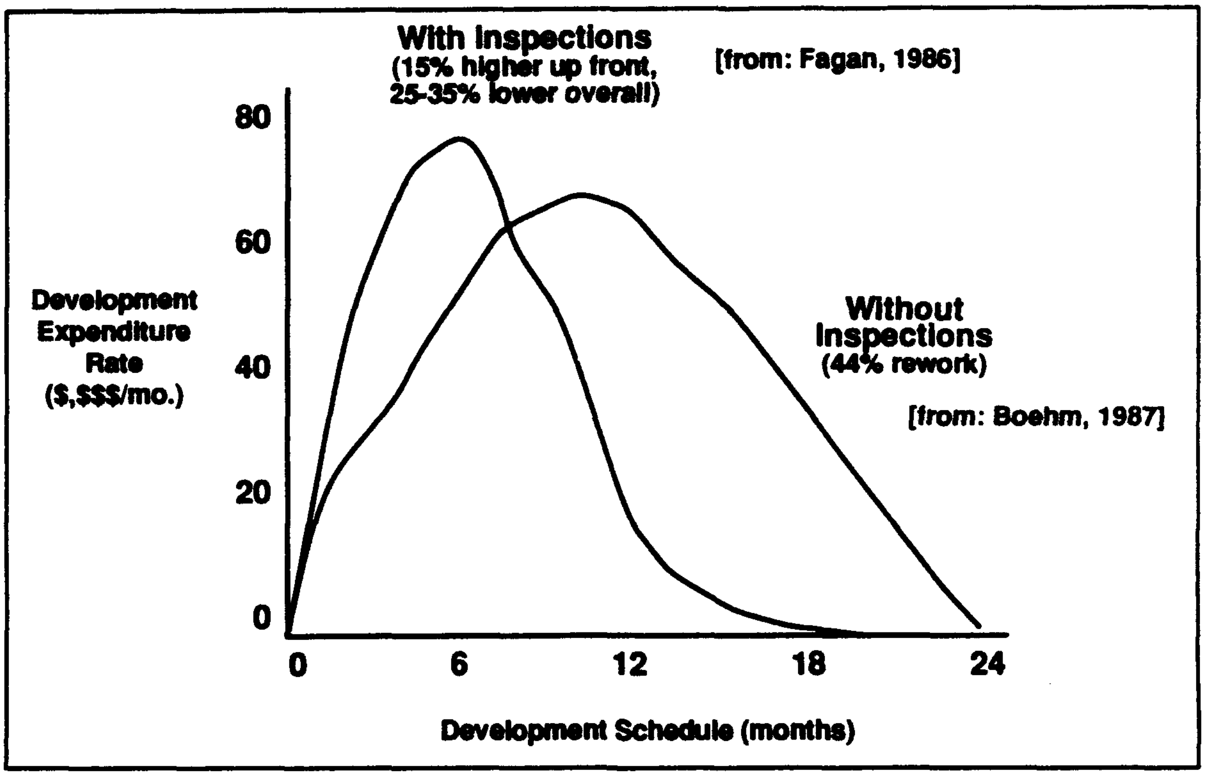


Figure 7. Software Development Spending Profiles

2.8 RECENT EXPERIENCE REPORTS FROM INDUSTRY

Russell gives the following example of the cost effectiveness of inspections on a project at Bell Northern Research (BNR) that produced 2.5 million lines of code [Russell 1991]. It took approximately one staff hour of inspection time for each defect found by inspection. It took 2 to 4 staff hours of testing time for each defect found by testing. Inspections, therefore, are 2 to 4 times more efficient than testing. Later they found that it took, on average, 33 staff hours to correct each defect found after the product was released to customers. Inspections reduced the number of these defects by a factor of 10. For commercial software developers, all of these costs resulting from defects are paid out of profits.

Inspections are widely used in the commercial software industry where quality and productivity are critical to a company's survival. There are many published reports on the use and benefits of inspections from companies such as International Business Machines (IBM), American Telephone and Telegraph (AT&T), and Bull. The National Aeronautics and Space Administration (NASA) Space Shuttle Program and Jet Propulsion Laboratory have also published positive experiences using inspections [Myers 1988, Kelly 1992]. A host of other organizations have reported positive inspection experiences: American Express, AETNA Life and Casualty, and the Standard Bank of South Africa [Fagan 1986], Shell Research [Doolan 1992], and Hewlett-Packard [Blakely 1991].

2.9 INDUSTRY INSPECTION USAGE IS INCREASING

The interest in software inspection and reports of successful results have been increasing since the development of the process. One measure of this interest is the number of published papers relating to inspections, many of which include positive results from industry use. Figure 8 shows the number of papers on inspections that have been published each year, based on the set of 77 papers identified in Section D.1 of Appendix D. Since there is a great deal of fluctuation in the number of papers published each year, a moving average is also shown that smooths out some of the fluctuations and makes the trend easier to see. For each year, the moving average is the average of that year and the two prior years. Since only a portion of 1993 data was available, the total amount of papers for 1993 was linearly estimated (based on 5 papers being published by August 1993). As can be seen by the moving average, there was initially sporadic publication on inspections, then followed by an increasing number of publications.

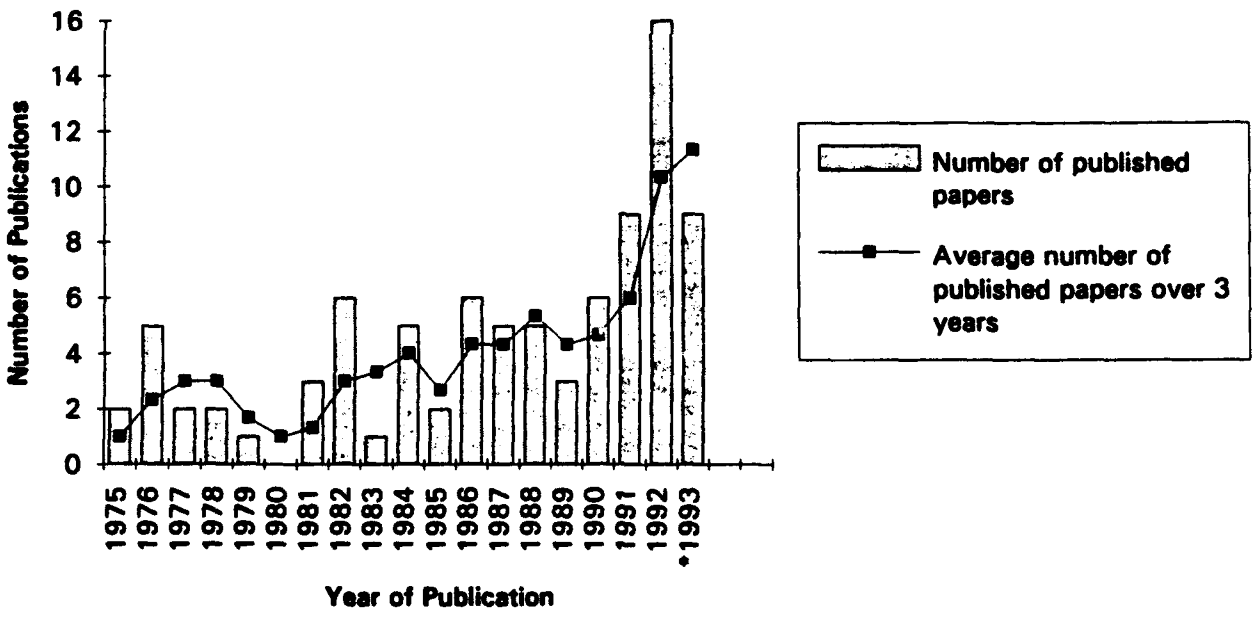


Figure 8. Number of Inspection-Related Papers

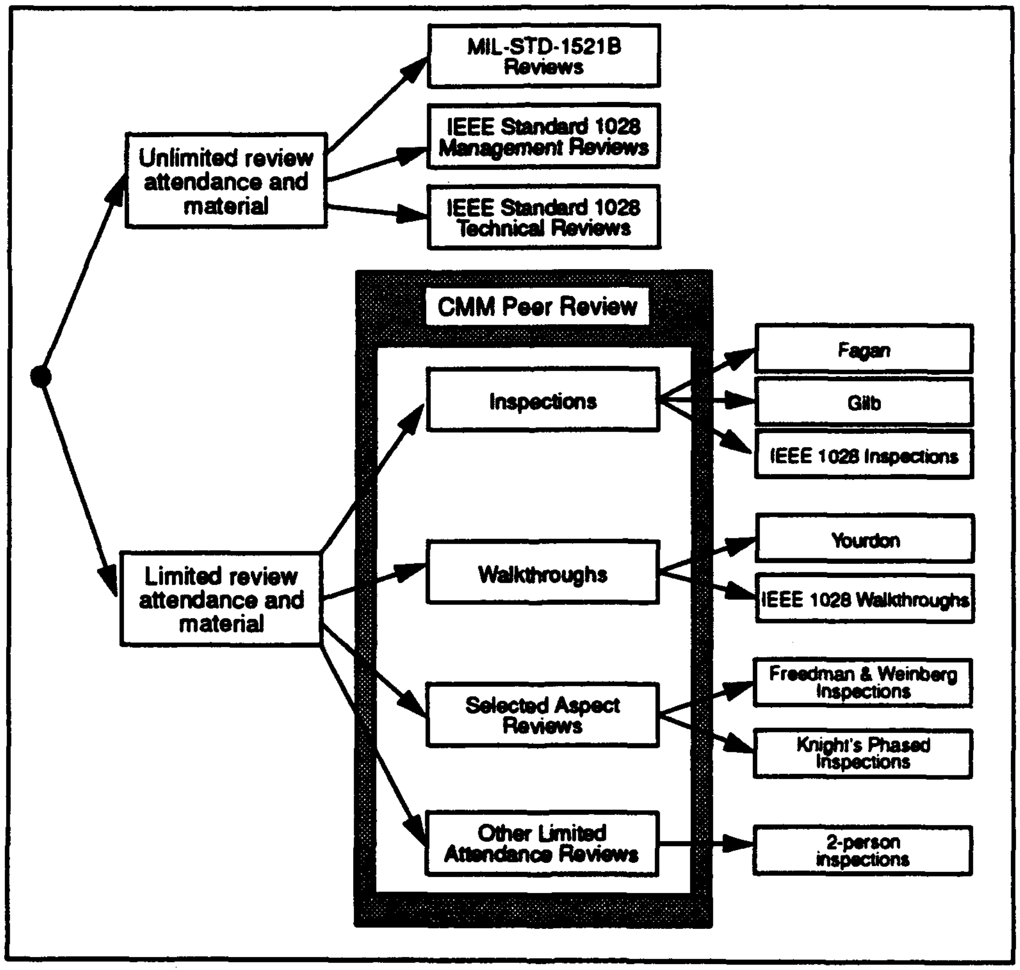
There are other indicators of increasing interest and use of inspections. In 1988, the Institute for Electrical and Electronics Engineers (IEEE) published a standard that defined an inspection process. Development of such a standard indicates that there was significant industry interest in the inspection process [IEEE 1028-1988]. A professional organization of people interested in inspections, the Software Inspection and Review Organization (SIRO), was recently established. Although the SIRO's influence is not clear at this time, the simple establishment of such a group indicates an increasing interest in the subject. Appendix C provides information on SIRO.

3. COMPARISON OF REVIEW APPROACHES

There are a number of different software review approaches, each of which has specific advantages and disadvantages. This section provides an overview and comparison of many of these review approaches.

3.1 TAXONOMY OF REVIEW APPROACHES

Figure 9 shows a taxonomy of various review approaches that reads from left to right. Review approaches may be divided into two basic classes: (1) those that have no



MIL-STD-1521B Reviews
IEEE Standard 1028 Management Reviews
IEEE Standard 1028 Technical Reviews
Unlimited review attendance and material
CMM Peer Review
Inspections
Fagan
Gilb
IEEE 1028 Inspections
Walkthroughs
Yourdon
IEEE 1028 Walkthroughs
Limited review attendance and material
Selected Aspect Reviews
Freedman & Weinberg Inspections
Knight's Phased Inspections
Other Limited Attendance Reviews
2-person inspections

Figure 9. Taxonomy of Review Approaches

limit on the number of the personnel that participate in the review nor on the amount of the material, and (2) those that permit only a limited number of reviewers and also limit the amount of the material reviewed. We consider three types of reviews that do not limit the number of personnel who attend: those defined in [MIL-STD-1521B] (therefore, required by [DoD-STD-2167A]), and the two types defined in [IEEE 1028-1988], management reviews and technical reviews. We have grouped the reviews that limit the number of reviewers into four categories: inspections, walkthroughs, selected aspect reviews, and other reviews that limit the number of personnel attending the review. In addition, these reviews can also be grouped together under the term "peer reviews," as described in the Software Engineering Institute's (SEI's) Capability Maturity Model (CMM). The CMM peer review will be examined in Section 3.4.

Table 2 shows some of the various differences between these types of reviews. Table 3 shows more detailed characteristics of the four different review approaches defined in [IEEE 1028-1988].

Unfortunately, terms for reviews are not universally agreed upon, and this has created a great deal of confusion. We have used the terms defined in IEEE Standard 1028 as a baseline, and added terms to create this taxonomy. An organization may, however, conduct a review that they term a "walkthrough" but which would fit under the category of an IEEE Standard 1028 technical review, or inspection, or some other category. Note also that Freedman and Weinberg define a review process that they call an "inspection," but it is very different from the inspection process defined in IEEE Standard 1028 [Freedman 1982]. Thus even if a review process is called an inspection, it may not be one as defined by IEEE Standard 1028.

3.2 REVIEWS WITH UNLIMITED ATTENDANCE

MIL-STD-1521B, IEEE Standard 1028 management reviews, and IEEE Standard 1028 technical reviews are all reviews with unlimited attendance and no constraints on the amount of material covered. There may be fifty, one hundred, or more reviewers, and the products reviewed may be hundreds or thousands of pages. These reviews tend to examine a large amount of product in a relatively short amount of time.

Table 2. Review Characteristics

	UNLIMITED ATTENDANCE REVIEW			LIMITED ATTENDANCE REVIEW			
	MIL-STD-1521B Review	**IEEE 1028 Management Review**	**IEEE 1028 Technical Review**	**Inspections**	**Walkthroughs**	**Selected Aspects Reviews**	**Other Reviews**
NUMBER OF REVIEW MEMBERS	Unlimited.	Unlimited (at least two).	Unlimited (at least three).	Three to seven (depending on technique).	Two to seven (depending on technique).	Two to seven (depending on technique).	Two to seven (depending on technique).
MATERIAL SIZE	Moderate to high.			Relatively low.			
OBJECTIVE	Evaluate progress and technical adequacy of pro*duced products* to determine if they form a satisfactory basis for proceeding to the next activity.	Ensure progress. Recommend corrective action. *Ensure proper* allocation of resources.	Evaluate conformance to specifications and plans. *Ensure* change integrity.	Detect defects. Verify resolution.	Detect defects. Examine alternatives. Forum for learning.	Detect defects and possibly other objectives.	
LEADERSHIP	By contractor, with contracting agency as co-chair.	Usually responsible manager.	Usually lead engineer.	Trained moderator who is not the product author.	Usually the product author.	Varies, depending on technique.	
IS MANAGEMENT PRESENT?	Yes, as is the contracting agency.	Yes.	Usually not.	Usually not.			
OUTPUT REPORTS	Conference agenda, conference minutes, action items.	Management review reports.	Technical review reports.	Defect list and summary inspection report.	Walkthrough report.	Defect list.	Varies.

Table 3. Review Approaches Defined by IEEE Standard 1028[a]

Category and Attributes	Management Review	Technical Review	Software Inspection	Walkthrough
Objective	Ensure progress. Recommend corrective action. Ensure allocation of resources.	Evaluate conformance to specifications and plans. Ensure change integrity.	Detect and identify defects. Verify resolution.	Detect defects. Examine alternatives. Forum for learning.
Delegated Controls				
Decision making	Management team charts course of action. Decisions are made at the meeting or as a result of recommendations.	Review team petitions management or technical leadership to act on recommendations.	Team chooses from predefined product dispositions. Defects must be removed.	All decisions made by producer. Change is the prerogative of the producer.
Change verification	Change verification left to other project controls.	Leader verifies as part of review report.	Moderator verifies rework.	Change verification left to other project controls.
Group Dynamics				
Recommended size	Two or more persons.	Three or more persons.	Three to six persons.	Two to seven persons.
Attendance	Management, technical leadership, and peer mix.	Technical leadership and peer mix.	College of peers meet with documented attendance.	Technical leadership and peer mix.
Leadership	Usually the responsible manager.	Usually the lead engineer.	Trained moderator.	Usually producer.
Procedures				
Material volume	Moderate to high, depending on the specific *statement of objectives* for the meeting.	Moderate to high, depending on the specific *statement of objectives* for the meeting.	Relatively low.	Relatively low.
Presenter	Project representative.	Software element representative.	Presentation by *reader* other than producer.	Usually the producer.
Data collection	As required by applicable policies, standards, or plans.	Not a formal project requirement. May be done locally.	Formally required.	Not a formal project requirement. May be done locally.
Outputs				
Reports	Management review report.	Technical review reports.	Defect list and summary. Inspection report.	Walkthrough report.
Database entries	Any schedule changes must be entered into the project tracking database.	No formal database required.	Defect counts, characteristics, severity, and meeting attributes are kept.	No formal database requirement.

a. (Reprinted from IEEE Std 1028-1988, IEEE Standard for Software Reviews and Audits, copyright ©1988 by the Institute for Electronics and Electronics Engineers, Inc. This information was written within the context of IEEE Std 1028-1988. The IEEE takes no responsibility or liability for and will assume no liability for damages resulting from the placement and context in this publication. Information is reproduced with the permission of the IEEE.)

3.2.1 MIL-STD-1521B Reviews

MIL-STD-1521B defines the requirements for a set of technical reviews and audits in DoD acquisition. The specific requirements vary depending on the specific technical review or audit, but in general their purpose is to permit the acquisition agent to evaluate progress and technical adequacy of products and to determine if they form a satisfactory basis for proceeding to the next activity. The reviews represent major DoD-STD-2167A milestones.

Compared to IEEE Standard 1028, many of these reviews are a combination of management reviews and technical reviews. Technical issues are certainly important in these reviews, but managerial issues such as schedule and cost are also an important part of the reviews.

The specific reviews and audits defined in MIL-STD-1521B are:

- System Requirements Review (SRR). The objective of this review is to ascertain the adequacy of the contractor's efforts in defining system requirements. It is to be conducted when a significant portion of the system functional requirements have been established.
- System Design Review (SDR). The objective of this review is to evaluate the optimization, correlation, completeness, and risks associated with the allocated technical requirements.
- Software Specification Review (SSR). A review of the finalized Computer Software Configuration Item (CSCI) requirements and operational concept. The SSR is conducted when CSCI requirements have been sufficiently defined to evaluate the contractor's responsiveness to and interpretation of the system, segment, or prime item level requirements.
- Preliminary Design Review (PDR). This review is conducted for each configuration item or aggregate of configuration items to (1) evaluate the progress, technical adequacy, and risk resolution (on a technical, cost, and schedule basis) of the selected design approach, (2) determine its compatibility with performance and engineering speciality requirements of the Hardware Configuration

Item (HWCI) development specification, (3) evaluate the degree of definition and assess the technical risk associated with the selected manufacturing methods/processes, and (4) establish the existence and compatibility of the physical and functional interfaces among the configuration item and other items of equipment, facilities, computer software, and personnel.

- Critical Design Review (CDR). This review is conducted for each configuration item when detail design is essentially complete. The purpose of this review is to (1) determine that the detail design of the configuration item under review satisfies the performance and engineering specialty requirements of the HWCI development specification, (2) establish the detail design compatibility among the configuration item and other items of equipment, facilities, computer software and personnel, (3) assess configuration item risk areas (on a technical, cost, and schedule basis), (4) assess the results of the producibilty analyses conducted on system hardware, and (5) review the preliminary hardware product specifications.
- Test Readiness Review (TRR). A review conducted for each CSCI to determine whether the software test procedures are complete and to assure that the contractor is prepared for formal CSCI testing. Software test procedures are evaluated for compliance with software test plans and descriptions, and for adequacy in accomplishing test requirements.
- Functional Configuration Audit (FCA). A formal audit to validate that the development of a configuration item has been completed satisfactorily and that the configuration item has achieved the performance and functional characteristics specified in the functional or allocated configuration identification.
- Physical Configuration Audit (PCA). A technical examination of a designated configuration item to verify that the configuration item "As Built" conforms to the technical documentation that defines the configuration item.
- Formal Qualification Review (FQR). The test, inspection, or analytical process by which a group of configuration items comprising the system are verified to have met specific contracting agency contractual performance requirements (specifications or equivalent). This review does not apply to hardware or software requirements verified at FCA for the individual configuration item.

- Production Readiness Review (PRR). This review is intended to determine the status of completion of the specific actions that must be satisfactorily accomplished prior to executing a production go-ahead decision. The review is accomplished in an incremental fashion during the Full-Scale Development phase, usually two initial reviews and one final review to assess the risk in exercising the production go-ahead decision.

3.2.2 IEEE Standard 1028 Management Reviews

The objective of management reviews as defined by IEEE Standard 1028 is to provide recommendations for (1) making activities progress according to plan, based on an evaluation of product development status, (2) changing project direction or to identify the need for alternative planning, and (3) maintaining global control of the project through adequate allocation of resources. Project management personnel attend these meetings.

3.2.3 IEEE Standard 1028 Technical Reviews

The objective of technical reviews as defined by IEEE Standard 1028 is to evaluate a specific software element and provide management with evidence that (1) the software elements conform to its specifications, (2) the development (or maintenance) of the software element(s) is being done according to plans, standards, and guidelines applicable for the project, and (3) changes to the software element(s) are properly implemented, and affect only those systems areas identified by the change specification. Project management generally do not attend these meetings.

3.3 REVIEWS WITH LIMITED ATTENDANCE

There are a number of different kinds of reviews that limit the number of personnel in attendance, keeping the reviewer group small. The typical number of attendees are three through seven. The amount of material to be reviewed is also intensionally kept small. Many review approaches do not specify the amount of review material, but ranges from five to fifty pages, depending on the approach, are typical. Many of these approaches also limit meeting time, typically two hours or less, to keep the participants fresh and alert.

We have grouped these reviews into four categories: inspections, walkthroughs, selected aspect reviews, and other reviews that have limited attendance.

3.3.1 Inspections

Inspections were originally developed by Michael Fagan in 1972 at IBM Kingston, New York [Fagan 1976a].[1] Since then, a number of variations on Fagan's work have appeared, including IEEE Standard 1028 inspections, and Tom Gilb's inspection process [Gilb 1988].

Fagan defines his inspection process in [Fagan 1976a] and [Fagan 1986]. He describes six steps: planning, overview, preparation, inspection (meeting), rework, and follow-up. He defines the roles of author, reader (who paraphrases the design or code as if they will implement it), tester (who views the product from the testing standpoint), and a trained moderator. Fagan limits code inspections to about 125 non-comment lines of code per hour and the meetings should be at most 2 hours, so approximately 250 non-comment lines of code are the maximum to be permitted in an inspection. In Fagan's approach, the moderator records all defects.

IEEE Standard 1028 defines minimum requirements for an inspection process and permits a number of variations from Fagan's original process [IEEE 1028-1988]. The IEEE process requires that someone have the role of recorder (who records defects found), but the recorder may or may not be the moderator. IEEE Standard 1028 does not give any specific quantitative limits on the amount of review material, preparation rate, or inspection meeting rate, other than saying that the material volume is relatively low and should be whatever can be comfortably handled with 1.5 hours of preparation and a 2 hour inspection meeting.

Gilb's inspection approach makes minor additions to Fagan's inspection approach [Gilb 1988]. Gilb adds a "third hour" problem solving step that may be performed immediately after the main inspection meeting or soon afterward. During the main inspection meeting, participants are not allowed to discuss solutions or improvements, as with Fagan's method, so that the participants can concentrate on finding defects. Gilb does not want good solution ideas to be lost, so he adds some additional meeting time (the "third hour") where these other ideas can be aired. Gilb also specifically includes a process improvement task, termed a causal analysis meeting.

[1] Weinberg and Freedman [Weinberg 1984, pp. 403] deny that any specific review approach was invented by a specific person or company. It is true that reviews of some kind are as old as software itself, but the ideas of creating a written review process, measuring results to improve the review process itself, and having a reader other than the author paraphrase the work appear to begin with Fagan.

3.3.2 Walkthroughs

Walkthroughs are similar to inspections in the sense that they are also reviews of a product by a small group of people used to detect defects. Usually, however, the person leading the review meeting and presenting the product is the producer of that product. In addition, most walkthrough approaches are less formal, since data collection is often not required and change verification is the prerogative of the producer (i.e., an independent check might not be made). Walkthroughs also permit suggestions for general improvement to be made during the main meeting, while inspections concentrate (during the main meeting) on finding defects. Two walkthrough approaches are described below as representative. First is Edward Yourdon's, and the second is IEEE Standard 1028 walkthroughs.

Edward Yourdon has defined a specific walkthrough approach under the title "structured walkthroughs" [Yourdon 1989]. Yourdon defines walkthrough as "a group peer review of a product." The reviewers are to find defects, but suggestions that do not involve defects are permitted.

The roles in Yourdon's version of walkthroughs include presenter (usually the producer), coordinator, secretary (who records comments and is not the same as the coordinator), maintenance, standards-bearer, and user (representative). The preparation time for a walkthrough should average 1 hour. A walkthrough should average 30-60 minutes, with 2 hours the absolute maximum and less time (e.g., 15 minutes) perfectly acceptable. The product size should be 50-100 lines of code, 1-3 structure charts, or 1-2 data flow diagrams plus a data dictionary. No more than two walkthroughs should be held back-to-back. A walkthrough begins with a review of any old actions from previous walkthroughs.

A walkthrough can be scheduled at a number of different points for a particular product—for example, code could be reviewed before compilation, after successful compilation, or after test. Yourdon recommends that walkthroughs be held after successful automatic syntax checking (for example, by a compiler or CASE tool) but before any kind of execution testing.

Management is then given a verdict on the product's disposition that is signed by all reviewers: accepted as-is, revise but no further walkthrough needed, or revise and conduct another walkthrough. Management is not given a count of defects or their type, and is urged not to use this information for performance evaluation (except for the largest outliers). Managers should in general not be present, as they can affect the utility of a walkthrough.

IEEE Standard 1028 also defines minimum requirements for a walkthrough [IEEE 1028-1988]. It defines the objective of a walkthrough to evaluate a software element, mainly "to find defects, omissions, and contradictions, to improve the software element, and to consider alternative implementations." Other important objectives include "exchange of techniques and style variations, and education of the participants. A walkthrough may point out efficiency and readability problems in the code, modularity in the design or untestable design specifications." Note that this permits many different subjects to be raised that are important, but can also reduce the amount of time available for finding defects. The presenter (usually the author) then "walks through" the software element in detail, while defects, suggested changes, and improvements are noted and written. The roles defined here are moderator, recorder, and author (who is also the presenter). The author may also be the moderator.

3.3.3 Selected Aspect Reviews

A selected aspect review confines the review of material to a pre-selected set of aspects to review. Reviewers look *only* at these pre-selected aspects of the product, one at a time. These aspects are usually further broken down into a specific set of items to look for, called a checklist. Even if a detailed checklist is not written, the selected aspects are essentially a higher-level checklist of aspects to cover in the review. Selected aspect reviews bear a number of similarities to the inspection and walkthrough process defined in IEEE Standard 1028. Inspections also often use checklists, and selected aspect reviews may even be called inspections by the creators of the review approach. The key difference of selected aspect reviews from inspections and walkthroughs is that the reviewers look *only* at the pre-selected aspects, one at a time, while inspections and walkthroughs look for all defects in a product. This distinction has a number of corollaries: defects that are not among the pre-selected aspects are unlikely to be found, and the fixed agenda means that the role of moderator does not require much technical ability. Selected aspect reviews do not require paraphrasing of the product, as an inspection generally would.

Freedman and Weinberg define a selected aspect review approach that they call an inspection [Freedman 1982]. Note that what they term an inspection is very different from the definition of inspections in IEEE Standard 1028. They note that larger amounts of material are generally covered than in other reviews, and while sessions may be limited in time (1-2 hours) the entire review may take weeks.

John Knight has also developed a selected-aspect review technique called "phased inspections" [Knight 1991]. A phased inspection[2] consists of a series of phases, where each phase addresses one or a small set of related properties (i.e., aspects) that it is deemed desirable for the product to have. Phases are conducted in series, with each depending on the properties established in the preceding phases. Each phase is carried out by staff whose training and experience is appropriate for the phase—some phases may only require rudimentary knowledge. Each inspector is required to sign a statement after the phase that the product possesses the prescribed property to the best of the his knowledge.

There are two basic types of phases in Knight's approach: single-inspector phases and multiple-inspector phases. In single-inspector phases one person uses a list of unambiguous checks from a checklist or executes a computer program that checks those selected aspects. In multiple-inspector phases personnel examine the documentation for completeness, examine the product thoroughly in isolation preparing a defect list, and then a reconciliation step in which the lists are compared (the lists are supposed to be identical).

Multiple-inspector phases bear some similarity to inspections as defined in IEEE Standard 1028, but there are a number of key differences: there is no verbal overview (all information must be available in documentation), individual time is spent creating an complete defect list instead of simply preparing for understanding, and the lists created are supposed to be identical before the meeting. Knight does not attempt to use the group's combined intelligence to find defects missed by the individuals (termed the "phantom inspector" by Fagan [Fagan 1986]), but instead depends on reconciliation, which provides the inspectors a challenge and incentive for thoroughness. One important factor in Knight's phased inspections is that they were designed to be supportable by computers, and prototype tools have been developed.

Unfortunately, as Knight himself notes, there is very little experimental evidence of the utility of phased inspections. They are an interesting approach to reviews and may be beneficial, and Knight has performed a very limited experiment with a few graduate students at the University of Virginia [Knight 1991]. Unfortunately, we are unaware of any larger-scale experiments that demonstrate the utility of this approach in an industrial setting.

2 John Knight's terminology is different than IEEE Standard 1028 and ours. Knight uses the term "Fagan inspection" where we use the term "inspection," and uses the term "inspection" where we use the term "selected aspects review."

3.3.4 Other Reviews that Limit Attendance

There are other review approaches that do not fit into the categories above. One is Bisant & Lyle's "two-person inspection" approach [Bisant 1989]. This is a review approach for projects that wish to use an inspection-like process but for which larger groups are not available. This approach is similar to Fagan inspections except that it only requires two people (including the author) and there is no moderator.

"Desk-checking" reviews take the limit on attendance to the extreme: a single person examines the code in isolation. This approach can certainly find defects, but is totally dependent on the person's experience and self-discipline.

The defect detection effectiveness of "two-person inspection" and "desk-checking" is highly dependent upon the personnel experience and skill level.

3.4 CMM PEER REVIEW KEY PROCESS AREA

The Software Engineering Institute has developed a Capability Maturity Model (CMM) that can be used to assess or evaluate the maturity of a contractor's software development process [Paulk 1993a, Paulk 1993b]. The model characterizes an organization in terms of a maturity level. There are five levels, each comprising a set of process goals that, when satisfied, stabilize an important component of the software process. Except for Level 1, each maturity level is decomposed into several Key Process Areas (KPA's) that indicate the areas an organization should focus on to improve its software process. KPA's identify the issues that must be addressed to achieve a maturity level. Figure 10 shows the five maturity levels and their respective KPA's.

Within maturity level 3 there is a KPA named "Peer Reviews" that closely relates to the inspection process. Figure 11 presents a summary of the attributes that indicate whether the implementation and institutionalization of the Peer Review KPA is effective, repeatable, and lasting. The CMM provides additional detail for each of these attributes. For example, Figures 12 and 13 describe the resource requirements and the procedure description of the Peer Review KPA.

A question arises as to whether inspections, structured walkthroughs, etc. satisfy the criteria of the Peer Review KPA. The current description of the CMM states that:

> "The purpose of Peer Reviews is to remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of the defects that can be prevented. The peer review is an important and effective engineering

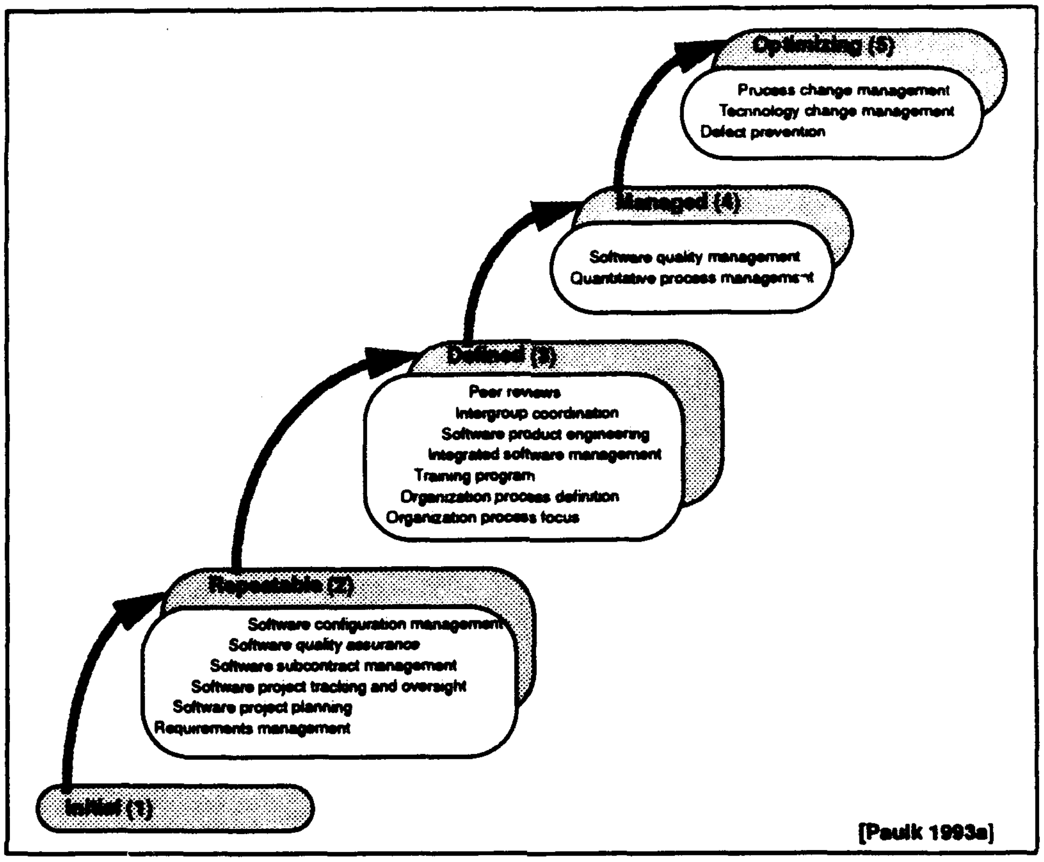

Optimizing (5)
Process change management
Technology change management
Defect prevention
Managed (4)
Software quality management
Quantitative process management
Defined (3)
Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus
Repeatable (2)
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Initial (1)
[Paulk 1993a]

Figure 10. Key Process Areas by Maturity Level

method that is called out in Software Product Engineering and that can be implemented via Fagan-style inspections [Fagan86], structured walk-throughs, or a number of other collegial review methods [Freedman90]." [Paulk 1993a, pp. 35-36]

Informal or ad hoc forms of "collegial review methods" do not satisfy the criteria set forth by the Peer Review KPA. Implementations of structured walkthroughs vary widely across industry, so it is impossible to make a blanket statement like "structured walk-throughs satisfy the Peer Review criteria." It is certainly possible to implement a rigorous structured walkthrough process that satisfies the Peer Review criteria. A rigorous structured walkthrough process that focuses on defect detection would look very similar to the inspection process. Figure 14 suggests a number of factors that can be used to distinguish between the two.

Goals

- Peer review activities are planned.
- Defects in the software work products are identified and removed.

Commitment to perform

- The project follows a written organizational policy for performing peer reviews.

Ability to perform

- Adequate resources and funding are provided for performing peer reviews on each software work product to be reviewed.
- Peer review leaders receive required training in how to lead peer reviews.
- Reviewers who participate in peer reviews receive required training in the objectives, principles, and methods of peer reviews.

Activities performed

- Peer reviews are planned, and the plans are documented.
- Peer reviews are performed according to a documented procedure.
- Data on the conduct and results of the peer reviews are recorded.

Measurement and analysis

- Measurements are made and used to determine the status of the peer review activities.

Verifying implementation

- The software quality assurance group reviews and/or audits the activities and work products for peer reviews and reports the results.

Figure 11. Summary of the CMM Peer Review KPA

Adequate resources and funding are provided for performing peer reviews on each software work product to be reviewed.

Resources and funding are provided to:

- Prepare and distribute the peer review materials.
- Lead the peer review.
- Review the materials.
- Participate in the peer review and any follow-up reviews required based on the defects identified in the peer review.
- Monitor the rework of the software work product based on the defects identified in the peer review.
- Collect and report the data resulting from the peer reviews.

Figure 12. Peer Review Resource Requirements

Peer reviews are performed according to a documented procedure.

This procedure typically specifies that:

- Peer reviews are planned and led by trained peer review leaders.
- Review materials are distributed to the reviewers in advance so they can adequately prepare for the peer review.
- Reviewers have assigned roles in peer reviews.
- Readiness and completion criteria for the peer reviews are specified and enforced.
  -- *Issues in satisfying these criteria are reported to the appropriate managers.*
- Checklists are used to identify criteria for the review of the software work products in a consistent manner.
  -- The checklists are tailored to the specific type of work product and peer review.
  -- The checklists are reviewed by the checklist developers' peers and potential users.
- Actions identified in the peer reviews are tracked until they are resolved.
- The successful completion of peer reviews, including the rework to address the items identified in the peer reviews, is used as a completion criterion for the associated task.

Figure 13. Peer Review Procedure Description

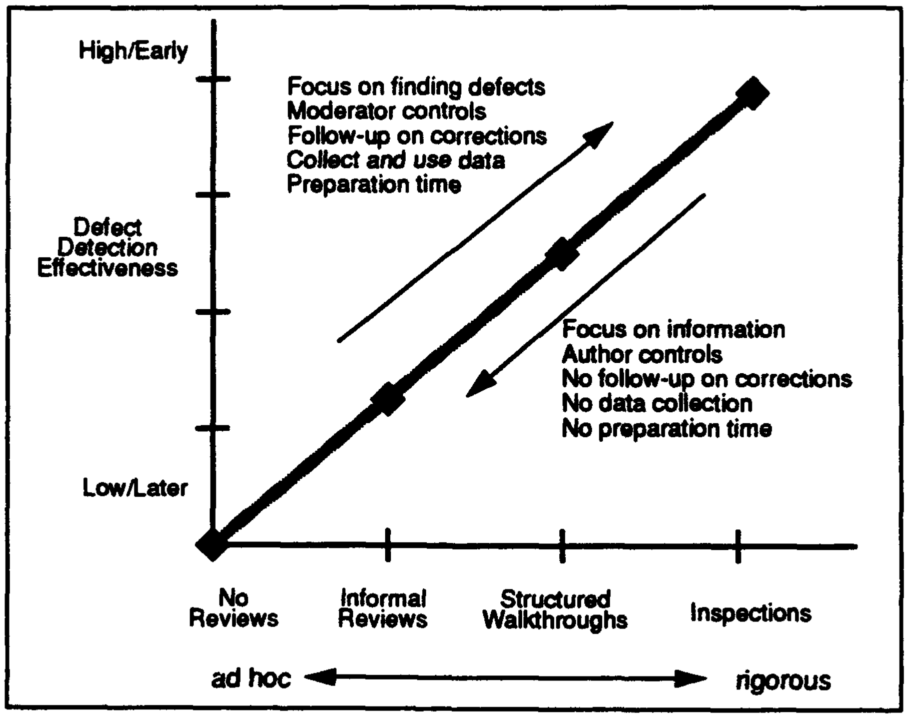


Figure 14. Review Effectiveness Factors

Most inspections, including the Fagan inspection process specifically mentioned, closely map to the Peer Review criteria. Figure 15 illustrates defect detection efficiency in terms of informal reviews, structured walkthroughs and inspections. The Peer Review criteria can be satisfied by a rigorous structured walkthrough process or by the inspection process.

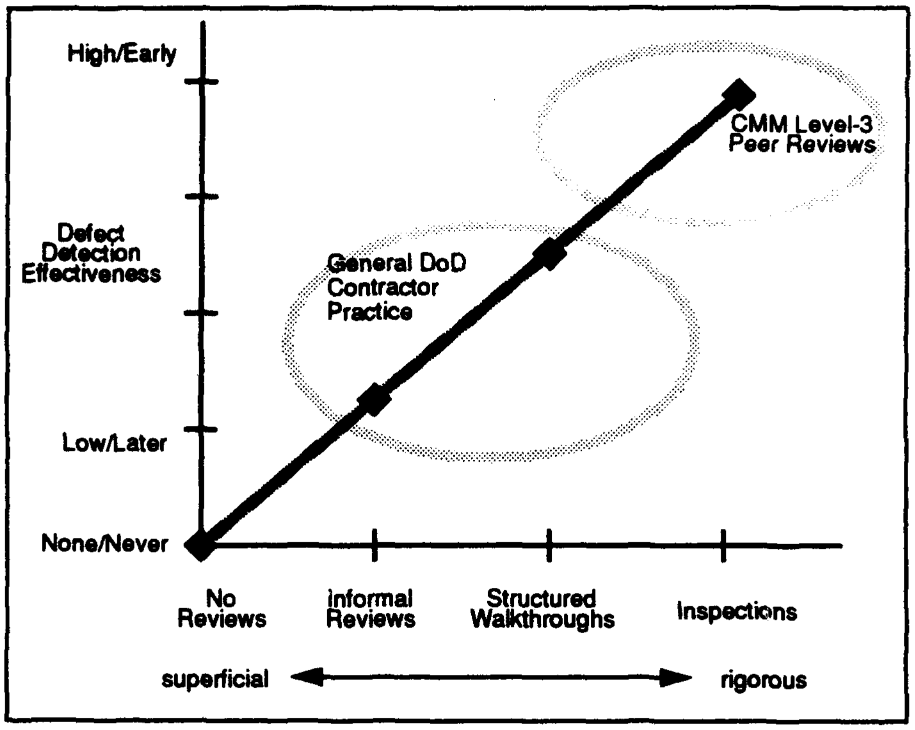


Figure 15. Effectiveness of Software Reviews

4. INSPECTIONS WITHIN THE BMD PROGRAM

This section of the document provides an analysis of DoD and BMDO policies, guidelines, and standards with regard to the inspection process. In addition, IDA efforts to insert inspections into the software development process of the Brilliant Eyes program is described.

4.1 INSPECTIONS IN BMDO AND DOD STANDARDS

Current DoD and BMDO software policies and guidelines include inspections as a recommended or at least acceptable approach. There are no policies or guidelines that directly impede the use of inspections. However, in general, they also allow less effective methods such as walkthroughs and informal peer reviews to be used instead. Only the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) requires, through the Level 3 Key Process Area (KPA), more formal peer reviews that approach inspections. There is DoD standard, DOD-STD-1703 (also known as [NSAM 83-3]), that requires inspections. However, DOD-STD-1703 is not required for use in most BMDO programs.

Figure 16, adapted from [SDIO 1992], shows the various DoD and BMDO policies, standards, and guidelines that address inspections. Each rectangle denotes a policy, standard, or guideline document. SDIO directive 3405 also requires the use of the SEI's CMM. The relation of the CMM to inspections is covered in the next section.

The following paragraphs discuss how each policy, standard, or guideline addresses software inspections The purpose of this section is to show that there are no policies or guidelines that directly impede the use of inspections, and to show how 'IDA has suggested changes to some standards to promote inspection usage. It is important to note that many of these documents are currently under revision. Some of the anticipated changes, based on current draft documents, are noted to show how near-term revisions are likely to relate to software inspections. IDA has proposed changes to many BMDO documents to encourage the use of inspections.

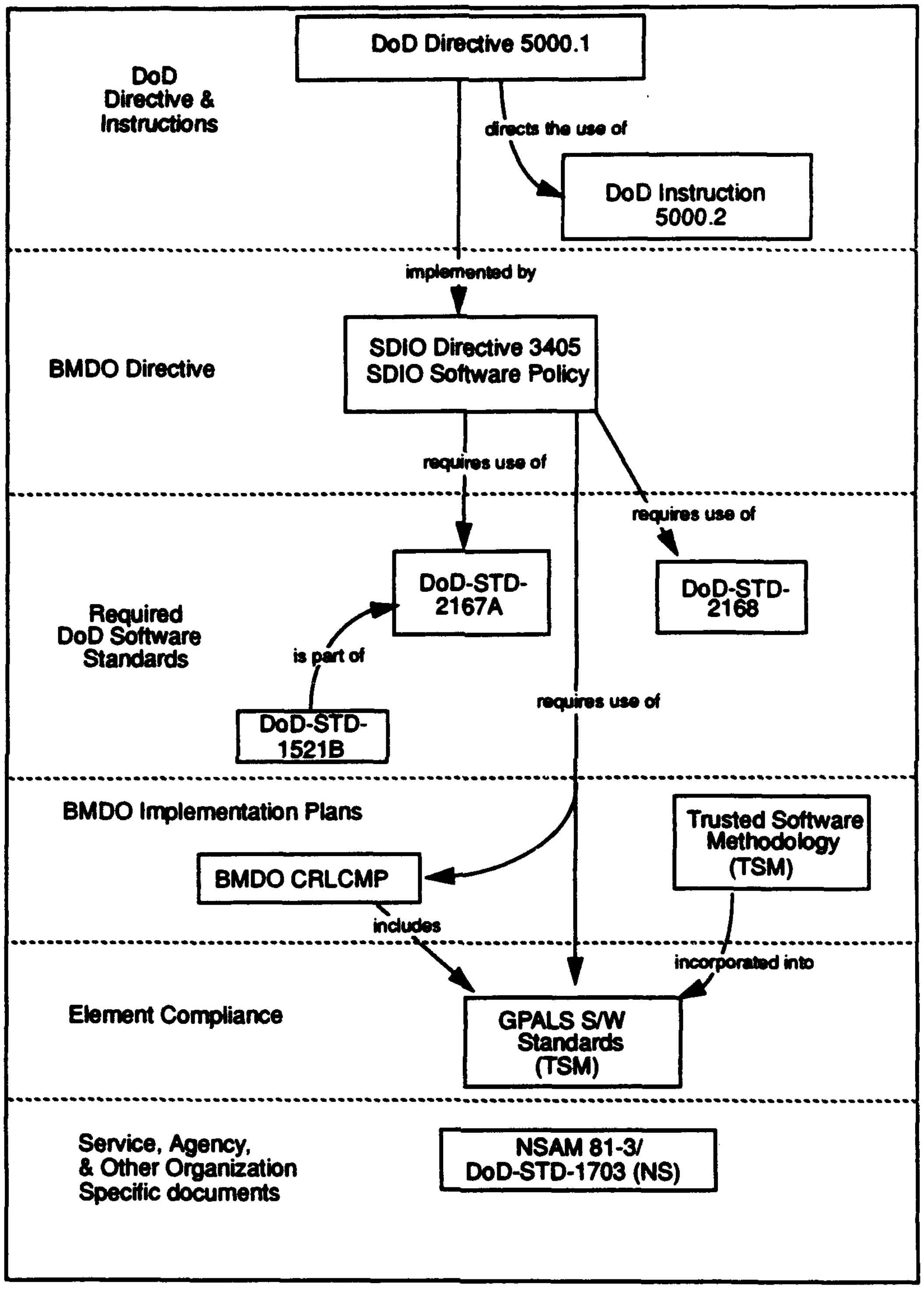

DoD Directive & Instructions
DoD Directive 5000.1
directs the use of
DoD Instruction 5000.2
implemented by
BMDO Directive
SDIO Directive 3405 SDIO Software Policy
requires use of
requires use of
Required DoD Software Standards
DoD-STD-2167A
DoD-STD-2168
is part of
DoD-STD-1521B
requires use of
BMDO Implementation Plans
Trusted Software Methodology (TSM)
BMDO CRLCMP
includes
incorporated into
Element Compliance
GPALS S/W Standards (TSM)
Service, Agency, & Other Organization Specific documents
NSAM 81-3/ DoD-STD-1703 (NS)

Figure 16. Policies, Standards, and Guidelines Relating to Inspections

4.1.1 DoD Directives and Instructions

Inspections are encouraged by the highest-level policies for DoD acquisition programs, but other less effective methods are also encouraged to the same degree and none are required. DoD directive 5000.1 is the primary DoD directive for providing policies and procedures for managing acquisition programs, except where statutory requirements override and where guidance pertaining to contracting is provided [DoDD 5000.1]. DoD directive 5000.1 does not specifically mention inspections, but it does establish DoD Instruction 5000.2 as a secondary document to providing such policies and procedures [DoDI 5000.1]. DoD Instruction 5000.2, part 6D attachment 1, states that "Specific practices that should be used are ... (3) walkthroughs, inspections, or reviews of requirements documents, design, and code." It also recommends that the "contractor establish[es] a uniform software error data collection and analysis capability to provide insights into reliability, quality, safety, cost, and schedule problems. The contractor should use management information to foster continuous improvements in the software development process."

4.1.2 BMDO Policy

BMDO's software policy, currently SDIO Directive 3405, revision 1 (13 March 1992), does not specifically mention software inspections. This directive's point 15, testing, does state that "developers shall conduct rigorous software testing... throughout all phases of the software life cycle." It requires that the software be developed in accordance with the "Trusted Software Development Methodology" (TSM) as contained in the Global Protection Against Limited Strikes (GPALS) Software Standards in point 10, and the use of the SEI's CMM in point 3. Both the TSM and CMM refer to peer reviews of products. It also requires adherence to DoD-STD-2167A and DoD-STD-2168. BMDO Policy 3405 is under revision, and current drafts of the new version specifically recommend, but do not require, software inspections as defined in [IEEE 1028-1988]. IDA provided recommended changes for this policy to encourage the use of inspections.

4.1.3 Required DoD Standards

DoD standards require various reviews and evaluations during software development, but do not specifically require inspections. These include standards DoD-STD-2167A, MIL-STD-1521B, and DoD-STD-2168.

DoD-STD-2167A establishes requirements to be applied during the acquisition, development, or support of software systems. DoD-STD-2167A requires software product evaluations of nearly every development product, but does not specify how those products are to be evaluated—instead, it provides some minimum evaluation criteria. The product types to undergo software product evaluations are essentially every product developed, including: the software requirements specification (SRS) for each computer software configuration item (CSCI), the software design document (SDD) for each CSCI, the computer software component (CSC) test procedures, the source code for each computer software unit (CSU), and CSU test procedures and test results. The minimum criteria include internal consistency, understandability, traceability, consistency with specified other documents, and appropriate analysis, design or coding techniques being used. This information is spread across the standard (it is in sections 4.4, 5.1.4, 5.2.4, 5.3.4, 5.4.4, 5.5.4, 5.6.4, 5.7.4, figures 4-10, and appendix D). DoD-STD-2167A does not, however, specify how these evaluations are to be performed, and thus permits any approach to be used.

DoD-STD-2167A also requires formal reviews during software development and references [MIL-STD-1521B]. MIL-STD-1521B, however, only defines the requirements for the formal (milestone) reviews (using the terminology of IEEE 1028-1988). In particular, MIL-STD-1521B only defines the requirements for Systems Requirements Review (SRR), System Design Review (SDR), Critical Design Review (CDR), Test Readiness Review (TRR), Functional Configuration Audit (FCA), Formal Qualification Review (FQR), and Production Readiness Review (PRR). Inspections may be used in addition to these reviews, but these reviews are not inspections.

The purpose of DoD-STD-2168 is to establish requirements for a software quality program to be applied during the acquisition, development, and support of software systems [DoD-STD-2168]. Its section 4.6 requires on-going evaluations of the processes used in software development and the resulting software and associated documentation, but does not specify how these evaluations should take place. DoD-STD-2168 does not impede the use of inspections and requires the use of some evaluation process, but it does not require any particular approach.

DoD-STD-2167A is in the process of being updated [SDD 1992]. This new standard has been given the working description of MIL-STD-SDD, Software Development and Documentation, and is to be an update of the two primary DoD software development standards, DoD-STD-2167A (used for mission critical systems) and DoD-STD-7935A (used for automated information systems), as well as DoD-STD-1703(NS) (used by the National Security Agency (NSA)). The December 1992 draft's appendix D describes software product evaluations in a manner similar to DoD-STD-2167A. The draft requires a large number of different products to be evaluated and provides some minimum evaluation criteria. It does not, however, require any particular evaluation approach.

4.1.4 BMDO Implementation Plans and Compliance Documents

The purpose of the BMDO's Computer Resources Life Cycle Management Plan (CRLCMP) is to establish the management and support framework for the development, deployment, operation, control, and integration of all computer resources within the BMD system [CRLCMP 1992]. One of the CRLCMP's annexes is the GPALS Software Standard [SDIO 1992]. The 29 May 1992 version of this document requires the use of walkthroughs, but the way the term "walkthrough" is defined would also permit the use of inspections. This text is inconsistent with the text of the GPALS Software Standard, since the standard requires peer reviews, not walkthroughs. Specifically, the CRLCMP states: "Walkthroughs are internal working-level reviews of design and/or code to detect errors and to ensure completeness, accuracy, and adherence to standards; they shall be scheduled regularly to ensure that all software design decisions are reviewed as they are made. Walkthroughs shall be accompanied by evidence and documentation as required by the GPALS Software Standards (Annex B)" [CRLCMP, pp. 8-13—8-14]. IDA has provided BMDO with suggested changes to remove this inconsistency in the CRLCMP [Wheeler 1993].

The main text of the 1992 GPALS Software Standard has some basic text about "peer reviews," which it defines as either walkthroughs or inspections. Unfortunately, its definition for inspections is not consistent with IEEE 1028-1988, Fagan's, or any other documented definition for inspections, and this can cause confusion for developers. Instead, the GPALS Software Standard defines inspections as "a walkthrough... [but] a formal event with additional formal participants from within the developer's organization such as the quality assurance and test departments. Results of inspections are always reported to the developer's internal program management while walkthroughs only require recording in SDFs and SEFs." It requires a role termed the "monitor," who leads the review. It recommends that the material be limited to that which can be adequately covered in four hours, twice the length of time recommended by typical inspection processes. On the positive side, it does require that participants be provided adequate time (preferably 3-7 working days) to review the material. The document notes that members of management normally do not participate in walkthroughs, but no guidance on this point is given for inspections. [SDIO 1992, 4-18—4-20].

IDA has provided recommended changes to the GPALS Software Standard, which may be included in its update [Brykczynski 1993]. These changes recommend but do not require inspections. This suggested text requires peer reviews of all analyses, requirements, designs, code, test data and procedures, and documentation, and defines "peer review" as either a walkthrough or an inspection as defined in IEEE Standard 1028-1988. These changes also provide guidelines that recommend inspections over walkthroughs, especially for critical software and early phases of development, and note that training is necessary to achieve full effectiveness. The changes also note that the material covered in an inspection must be limited (e.g., roughly 200-250 statements for code so it can be addressed in a two-hour meeting).

4.1.5 Trusted Software Methodology

BMDO's software policy requires the use of the Trusted Software Development Methodology, now termed the Trusted Software Methodology (TSM). TSM consists of a set of trust principles that are aimed at increasing requirements compliance by reducing the number of software flaws introduced during software development and maintenance by malicious or nonmalicious causes. The trust principles are organized into a hierarchy of trust classes named T1 through T5 that represent increasing amounts of rigor of the techniques required for reducing the number of software flaws.

At the time of this writing the TSM is being updated. Both the version of the TSM currently being used and the proposed new version give requirements for reviews that are necessary for inspections but do not specifically require inspections. Either also permits walkthroughs and other review techniques.

The set of trust principles that are required for use by BMDO software development projects are defined in an appendix in the GPALS Software Standard. The September 4, 1992 version of the GPALS Software Standard includes a version of the TSM defining four review principles that relate to inspections: requirements analysis review, design review, source code review, and test review. For any trust level (T1 or higher) this version of the TSM requires a mechanism or procedure to be in place for a large number of products, including all software requirements, the entire design, all source code, CSU test cases, and CSU test results, and all tests. Trust level T1 also requires documented review guidelines, checklists, criteria for determining the frequency of reviews, criteria to ensure that there has been adequate preparation time, guidelines to ensure completeness, consistency, and correctness, and that all resulting actions be documented and revisited. Trust level T2 adds a requirement for checking traceability and that these reviews should be used to enforce source code standards. Trust level T3 adds requirements to define the review process techniques (such as paraphrasing), that limitations be defined on review rates (pages/hour), that review members be given roles (including recorder and moderator). A few T3 requirements specifically mention "inspection," implying that inspections were the intended process for T3.

The required number of people present varies depending on the trust level. At T1 there must be at least 2 people, the author and one other person qualified in software development. At T2 there must be at least 3 people, the author and two other people qualified in software development. persons qualified in software development. At T3 there must be at least 4 people: those required for T2 and an independent reviewer. There is no maximum number of reviewers.

On April 1, 1993, a revised version of the TSM was published [SDIO 1993]. This version consolidated the trust principles, changing the total number of principles from 44 to 25. The authors' intent is to incorporate this version of the TSM into an update of the GPALS Software Standard, though the TSM may be modified further before incorporation. In this new TSM version there is a single review principle, termed the "Peer Review Principle." There is little change in the actual requirements; the scope of products to review is widened slightly, and some of the requirements are moved to higher trust levels.

For all trust levels above T1, the current version of the TSM requires a mechanism or procedure to be in place for a large number of products, including software requirements and supporting analysis, all components of the design, all source code listings, all CSU, CSC, and CSCI test plans, test procedures, cases, results, and reports, verification conditions, and all operation and support documents. Note that this is a larger list than the older TSM, since it includes such products as supporting analysis, operation, and support documents. Trust level T2 requires documented peer review guidelines to be in place that define checklists, the frequency of reviews, criteria to ensure adequate preparation time, criteria to ensure completeness, consistency, and correctness of the items under review. Trust level T2 also requires that a record of each review be kept; note that some of these requirements were formerly in trust level T1. Trust level T3 adds requirements to define the review process techniques (such as paraphrasing), that limitations be defined on review rates, that review members be given roles (including recorder), and that the moderators be trained. Note that the requirement for moderator training is new. The number of reviewers required varies by trust level in the same way as the older version. The references include a reference to IEEE 1028-1988.

For both versions of TSM, trust level T3 comes closest to requiring an inspection process, but it does not require inspections. Even at T3, there is no specific limit to material size, review rate, or preparation rate (simply a requirement to document a maximum), no requirement for methods that completely cover the material such as paraphrasing (simply a requirement to document the method), and the author may do all the presenting.

4.1.6 Other Policies and Standards

There are many different documents specific to the military services, agencies, and other organizations, and reviewing all of them would require an extensive effort. One document that is particularly interesting with regard to inspections is NSAM 81-3, also known as DoD-STD-1703(NS). This document is required for use in many National Security Agency (NSA) projects [NSAM 81-3]. DoD-STD-1703(NS) essentially requires a slightly weakened version of Fagan inspections. Inspections are specifically required for software detailed designs and code, and [Fagan 1976a] and [Fagan 1976b] are specifically referenced. DoD-STD-1703(NS) does not require inspections of software requirements specifications, test plans, test cases, or the user's manual, but suggests that doing so may be valuable. The purpose of inspection meetings are "to find errors." DoD-STD-1703(NS) specifies that inspections are normally composed of a moderator, the author, and one or more inspectors, and a reader and recorder must be appointed. The inspection process in this document has six steps: planning, overview, preparation, inspection meeting, rework, and follow-up. Preparation rates are given for detailed design (about 100 lines of Program Design Language (PDL)/hour) and code (about 125 lines of code/hour). The author's role in the inspection meeting is usually limited to answering technical questions, although this is not specifically required by the document. DoD-STD-1703(NS) recommends that the reader paraphrase, but it allows the reader to read verbatim (which is weaker than Fagan's approach). Inspection meetings are limited to two hours. Inspection meeting rates are to be about 130 lines of PDL/hour for detailed design and about 150 lines of code/hour for code. Sample problem list codes and checklists are provided. The document also notes that data from inspections should never be used to evaluate software developers [NSAM 81-3 1987, pp. 4-53—4-63]. This document is to be superceded by MIL-STD-SDD [SDD 1992]. This is of some concern since MIL-STD-SDD will probably not require the discipline of MIL-STD-1703(NS) [Taylor 1993].

4.2 INSPECTIONS IN BRILLIANT EYES

The BMDO funds the development of a number of weapon systems. One of these systems, the Brilliant Eyes (BE) satellite, will provide early warning and tracking of ballistic missiles. IDA's inspection insertion efforts during fiscal year 1993 focused on the BE program. The BE program is a useful candidate for software inspection insertion for two reasons. First, the BE program is in the demonstration/validation (DEM/VAL) acquisition phase with two contractors competing for the eventual Engineering and Manufacturing Development (EMD) contract. Software production has been determined to be a critical success factor for BE, and one objective of the DEM/VAL phase is to understand critical processes. Thus, the knowledge and experience gained by software inspection insertion can be applied during the later EMD phase.

The BE program is also a useful candidate for introducing inspections because the program manager has emphasized the importance of software to his program. He strongly advocated the use of Software Capability Evaluations (SCE's) as a source selection criteria for the two DEM/VAL contractors. In addition, a future SCE is expected to help down-select to one BE contractor. Thus, software process improvement, especially in the area of inspections, is likely to be of interest to the two BE prime contractors.

We discussed the inspection process with several other BMD program offices and provided them with inspection information, such as technical papers and briefings. However, due to available funds, we limited our primary insertion efforts to the BE program. This section of the paper provides a general description of the approach used to encourage inspection usage in the BE program. Because of the competition-sensitive nature of the current BE contract, specific details of the inspection insertion effort are not included.

4.2.1 Approach

Two approaches were initially considered for inserting inspections into BE software development. The first approach involved mandating the use of inspection, either via the RFP/contract or by later agreement. We rejected this approach for two reasons: First, we began the insertion effort after the contracts were let. Thus, it would require much effort to renegotiate the contracts to mandate the use of inspections. Second, mandating inspections does not guarantee that the process will be used appropriately. We were concerned that a contractor could "go through the motions" of software inspection without gaining the benefits.

Instead, we sought voluntary adoption of the inspection process by the BE contractors. We felt that the benefits that result from a rigorous inspection process were sufficient to convince a large development organization to experiment with the technology. In addition, given the future BE down-select, we felt that software process improvement, such as inserting inspections into practice, would be of interest to both contractors.

4.2.2 Establishing Commitment

Senior level management commitment is critical to successfully inserting an inspection process. For DoD systems, a commitment is needed from both the government and contractor organizations. We developed a briefing to describe the problem that inspections address, as well as the cost and benefits reported from industry inspection experience reports. Much of the information found in Section 2 was derived from this briefing. We first gave the briefing to senior level BMDO officials. We felt it was important for these officials to have an understanding of the cost impacts of inspections (see Section 2.7). The briefing was positively received and we were encouraged to discuss the insertion effort with the BE program office.

We next briefed the BE program manager. The objective of this briefing was to provide him with sufficient information to make an informed decision to advocate the use of inspections by the BE prime contractors. This objective was achieved, as he directed us to give the same briefing to the program managers for each of the BE prime contractors.

We subsequently met with both contractors and discussed their voluntary use of the inspection process during BE software development. We examined their current software review procedures and identified areas that would need to be modified to meet the spirit of the inspection process. A hypothetical example of an area needing modification, but one that is often encountered in the software review procedures of DoD contractors, involves the objective of the review. Often, review procedures have multiple stated objectives: ensure compliance with project standards, promote product understanding, reduce defects, etc. In contrast, all aspects of the inspection process are focused primarily on efficiently finding defects. Thus, many organizations must develop "tighter" versions of existing procedures in order to implement an inspection process.

4.2.3 Insertion Considerations

The BE inspection insertion effort was based on encouraging the contractors to voluntarily adopt the inspection process. We did not want to dictate how inspections were inserted, but we did provide insertion advice and guidance. Due to the competition sensitive nature of these discussions, we cannot describe specific suggestions provided to the contractors. We can, however, describe a number of general issues that should be considered in any inspection insertion activity:

Method of insertion. Some organizations develop pilot or demonstration programs to introduce the inspection process, while others choose to implement inspections project-wide. A pilot program can be used to:

a. Verify the benefits of inspection to the organization through actual use.

b. Gain a better understanding of how to implement inspections across the organization.

c. Encourage inspection adoption by the rest of the organization if the pilot program is performed by people whose processes tend to be emulated.

Pilot programs can be implemented on existing work products that are being maintained or on products that are separate development items, such as a CSCI. Inspections can also be initially inserted project-wide. The benefits of this insertion method are that work product defects are addressed early and project-wide. A pilot program would focus on a subset of the project's workproducts, the remaining of which would exhibit higher latent defect rates. Another benefit is that higher insertion leverage may be obtained if senior level organizational commitment to program-wide inspections is pursued.

Measurement data. There are a number of published references available that describe the data to collect during the inspection process [Grady 1991, Ackerman 1989]. Some even provide sample forms that can be used to collect the data. An organization that plans to introduce inspections must decide:

a. What data to collect. Typical data includes the number, type and severity of defects, amount of effort applied to inspection (e.g., preparation, inspection meeting, and rework), and post-inspection defect data to evaluate the effectiveness of the inspection process.

b. How the data will be used. It is critical to examine the questions that will be addressed with inspection data. For example, project or software managers should examine overall inspection data to identify potential methods for reducing common defect occurrences. Inspection effort rates should be monitored to ensure that inspectors devote sufficient time to the inspection process. Higher-than-average defect rates for properly inspected work products may suggest the scheduling of additional dynamic testing for that module.

c. The process for storing inspection data. This includes developing procedures for data entry and determining who has access to raw and composite data.

Training. It is essential that everyone who participates in an inspection be trained in the inspection process. Attempts at inserting inspections can fail if proper training is not provided to all inspection participants. In addition, the managers of inspection participants must also undergo training to sensitize them in their role (i.e., how not to obstruct the process) as well as to describe the benefits/costs of inspections and how inspection data can be used to better control the software development process.

Training can be obtained from outside sources or provided by in-house staff. Outside sources are often experienced in the potential pitfalls in inserting inspections and can offer support in reducing these risks. In-house staff, if properly trained, may provide a more flexible training capability.

5. CONCLUSION AND RECOMMENDATIONS

In this paper, we have presented a range of information on software inspections. Inspections have been proven in industry to reduce software development costs and schedules and, at the same time, substantially improve product quality and reliability. The inspection process represents a sound basis from which software engineering and management process improvement can be instilled in software development. Published experience reports clearly indicate that inspections are a commercial industry "best practice." Yet, inspections are not routinely practiced by BMDO or other DoD contractors. Unless there is a concerted effort by BMDO, it is unlikely that inspections will be used widely in BMD software development. The following recommendations identify the key activities that must be performed to implement inspections BMD-wide.

1. Senior BMD executives should strongly encourage the use of the inspection process for all BMD software development. The BMDO General Manager and Service Program Executive Officers can influence element program managers to consider implementing the inspection process. Without senior-level leadership and commitment it will be much more difficult to convince program managers and their staff to implement the inspection process. Promoting advanced technology, especially an industry-proven process such as inspections, is an appropriate function for BMDO senior executives.

2. Future BMD RFP's should provide explicit incentives for contractors to bid the inspection process. Evaluation of the proposed inspection process should be a specific technical criterion in making the source selection decision. By incentivizing inspections in the RFP, contractors can then reflect the upfront costs of inspection in their bids. It should be noted, however, that inspection benefits are unlikely if BMDO relies primarily on contract wording to obtain inspection usage. Active interest by BMDO and the program offices to communicate the importance of the inspection process to the contractor is also necessary.

3. BMDO should provide funding for initial inspection training to all BMD element program offices. By specifically allocating training funds, BMDO emphasizes the importance placed on inspections. This funding should be allocated for two types of training: full training for all program office staff involved with software development, including program manager, software leads, and acquisition officials, and initial training for existing contractor personnel. The program office should be encouraged to provide subsequent training to existing contractors, or to allow the contractor to directly charge inspection training costs against the contract. For maximum benefit, all contractor development personnel, and their managers, should receive inspection training.

4. Summary inspection data and results should be collected from software developers by the program offices. This data is needed to assess the effectiveness of inspections and to manage the software development process. Benefits from inspections will be available immediately, quantifiable in terms of defects detected and rework avoided. Results from "success stories" can be used immediately to increase awareness and adoption of inspections BMD-wide.

APPENDIX A. COMMONLY ASKED INSPECTION QUESTIONS

This appendix provides answers to commonly asked questions about software inspections.

What prevents contractors from using inspections?

There are a number of reasons DoD contractors are not using inspections. Contractors may be unaware of inspections or their benefits. Contractors mistakenly believe they have an inspection process in place, but what they are actually doing is the less formal/rigorous walkthrough. *Unless walkthroughs are focused on defect detection, they do not produce* dramatic defect detection rates. The inspection process is not called out in the contract, so there may be less motivation to implement such a process. Bids that include inspection costs have difficulty claiming rework cost savings and, therefore, may appear less competitive. The up-front costs (e.g., training and front-loading contract spending) associated with implementing an inspection process may be difficult to "sell" to the contractor program manager. Another inhibitor may be the perception of lower profitability resulting from an effective inspection process: contractors often profit from rework.

Hardware inspection processes have existed for many years. They are used to ensure that defective products are not delivered to a customer. Hardware inspections are not the most effective method for improving product quality because they often occur late in the product lifecycle. How can we avoid the traps we run into with hardware inspections?

The use of the term "inspection" is perhaps unfortunate, but the method is very *closely akin to the philosophy of Total Quality Management (TQM) espoused by Deming* and Juran [Deming 1986, Juran 1986]. Software inspections are not merely a check on product quality. They provide timely feedback for both corrective action and process improvement. The inspection process works well with products, from all phases of software development: requirements, designs, test cases, documentation and code. Defects in these products are identified very early, allowing a higher quality product to proceed. Inspections are sometimes called "in-process inspections" to emphasize their early use.

What type of work products can be inspected? Are all work products of a particular type inspected or a subset? How much material is inspected at one time?

All types of software development work products (requirements, designs, code, test plans, documentation) can be inspected. Inspections are particularly valuable in early stages of development because defects that are not found are costly to fix later, and because few other techniques for finding defects are available.

Unlike statistical sampling techniques used in manufacturing, software inspections are usually applied to all work products. Until software development methods reduce human error to significantly lower levels, techniques to find and remove defects in all products are necessary.

An inspection should cover material equivalent to approximately 250 lines of high-level source code. This may correspond roughly to 20 pages of requirements expressed in English, or 15 data flow or structure diagrams of design. Attempting to inspect at much higher rates reduces a team's defect-detection effectiveness.

What data is collected during the inspection process? How is this data used?

A variety of data should be collected during the inspection process. Examples include defect data (number, type, and severity), inspector preparation and meeting times, work product size, and estimated correction effort. Detailed reporting of defects by employee should be avoided for the same reasons that managers are excluded from inspection meetings. Summary data averaged over the development staff enable managers to judge the effectiveness of the inspection process. This data can also provide valuable insight into other parts of the software development process which can be used for process improvement.

What is management going to do with data resulting from the inspection process?

Management uses inspection data to ensure that the process is healthy and effective. Management should examine what types of defects are being made so that the right part of the process can be improved on, and check to see if implemented process "improvements" are making things better or worse.

An example question management might pose is: "I see that memory leakage errors are occurring frequently—can we buy a tool that will help find these for us?" Or, "We now have that tool, but we make just as many errors. Why is that?" Perhaps training has not been provided in the tool, or the tool is not useful. In any case, managers can use inspection data to improve upon the software development process.

Managers are going to be interested in who makes errors. How do you keep inspection defect data anonymous?

Collecting data is an important part of the inspection process. Examining the type, severity and number of defects during the development process can provide managers with valuable feedback for process control. Some organizations summarize inspection defect data to mask defect counts for specific work products. Other organizations provide managers with specific training in appropriate uses of inspection data. Others have managers and their subordinates co-sign a document where the manager promises not to use inspection data in this way. It is important for managers to understand that abuse of defect data will reduce inspection results.

Co-workers participating in inspections routinely help each other learn to avoid common mistakes.

What happens when defects are discovered during an inspection meeting? What happens to defects afterwards?

The responsibility of the recorder or scribe is to make a list of all defects found during an inspection. In addition to recording the location and a brief description of each defect, many inspection practitioners also record the severity, type, and estimated effort to correct each defect.

After the inspection meeting, the author is responsible for correcting all defects. Other inspection participants may take action items to help resolve issues that cannot be answered immediately such as unclear specifications or questions about interfaces to other work products. When major revisions are necessary, the inspection team may decide to re-review the work product. Otherwise, after making the necessary corrections, the author reviews the changes with the inspection moderator to verify that all identified defects have been eliminated. This final verification step completes the inspection process for that work product.

How many people participate in an inspection? Do managers participate?

Inspections should involve 4 to 6 people. Fewer people find too few defects, and more people do not find many more. Inspections are labor intensive and must make the best use of peoples' time. A key aspect of inspections is that it is a review by peers, without managers. Having managers present during inspection meetings distorts the technical team's defect detection focus. As a defense mechanism, inspectors may report more superficial defects so that the inspection appears to be effective, but are less critical of the work product and report fewer serious defects so that the author does not look bad.

Why do you need inspection training? It sounds pretty straightforward.

Introducing inspections represents a pretty significant process change for most software developers. Training provides motivation and indoctrination, as well as explanation of all the steps in the new process. Training also provides an effective forum for communicating information on how to handle potential interpersonal conflicts that may arise during the inspection process.

Inspections require participants to fill several important roles. Effective training is necessary to prepare participants for the responsibilities of these roles. Because it is useful to understand each of the roles and to rotate roles in different inspections, all roles should be covered in a general training program. Also, because inspections are often mistaken for less formal walkthrough methods, training should cover the differences that make inspections more effective. Inspection moderators typically need additional training to learn how to guide and control inspections. Managers should also receive training so they understand the benefits of inspections, what the technical staff does during inspections, and why managers should not attend inspection meetings.

How can inspections be inserted into on-going projects? There are few new starts where you can do it right from the beginning.

Inspecting code based on uninspected requirements and designs invariably produces long lists of "issues" questioning the meaning of requirements and the completeness and consistency of designs. This can seriously reduce the effectiveness of inspections because inspectors cannot directly determine whether or not the code is correct. So this is not the best approach for introducing inspections into on-going projects.

A more effective approach for the initial introduction of inspections in mid-development is to start with a subsystem or major component where the requirements and design can be inspected before the coding phase is completed. Even though the requirements and design are "mature", expect to find defects in them when you inspect them. This is not as effective as starting from the very beginning, but much of the time required to inspect and correct the requirements and design will be recouped by the savings in later rework. Revise the initial coding, along with any initial test plans and test data, then proceed with inspection of these work products.

For mature legacy systems, inspections can be introduced most easily when extensions or enhancements are made. Inspection of all new material can be done "correctly" from the beginning. Interfaces with the existing system will be the difficult part. To correct latent defects and to sufficiently understand how the new code must interact with existing code, you will need to inspect the requirements, design, and code surrounding the interfaces with the existing system. For older systems, this may amount to reverse engineering to figure out what the system does. Without a full understanding of the existing system's requirements and design, it will be impossible to assure that new code will interact with it correctly.

What can this approach do if your project is in a turmoil of system requirements changes?

Unstable system requirements present serious problems for all software development methods. We can't claim inspections are immune to such disruptions. The effort invested in inspecting software requirements, designs, and code that have to be scrapped because of system requirements changes is lost along with the affected work products. Losses are likely to be higher for inspected work products than uninspected work, because changes are more frequently made earlier in a program, before the savings in rework are realized. Losses will also be higher for the DoD when a program has to be cancelled before the savings in rework are realized.

Assume that inspections add 15 percent to the cost of development and produce overall cost and schedule reductions of 30 percent, as described by Fagan. System requirements changes would then have to completely obliterate all software requirements, design, and code *twice* at the worst possible times before non-inspections would break even on cost with inspections. The first time we lose 15 percent to inspections. The second time we lose another 15 percent. The third time—no changes this time, please!—we make it all back by saving 30 percent. In addition, we still finish the third system well before the non-inspected approach. This scenario must represent a situation much more severe than mere "turmoil."

The potential for losses due to inspections does not support an argument for developing lower-quality products at lower initial costs. The quality and reliability achieved by inspections cannot be added on later to completed systems at any cost.

APPENDIX B. CONTRACTING FOR INSPECTIONS

Direct reference to the inspection process in the Request for Proposals (RFP) will allow offerors to reflect the impact inspections may have to their proposal. For example, training costs may need to be included and the upfront schedule ramifications must be recognized. By addressing inspections in the RFP, the Government also indicates the importance of the process.

DoD Standard 2167A, Defense System Software Development, establishes requirements that provide the basis for Government insight into a contractor's software development, testing, and evaluation efforts [DoD-STD-2167A]. The standard does not provide explicit support for reporting data commonly collected during the inspection process. However, a Data Item Description (DID) can be written by the Government to require that the contractor implement an inspection process and report inspection-related data.

This appendix provides suggestions on how to incorporate inspections into an RFP. Guidance is given for the type of inspection-related data that should be reported via a DID. Additionally, an actual example of Government requirements for the inspection process is given.

B.1 INSPECTIONS IN THE RFP

The Government should encourage or require a contractor to implement processes that provide for early, efficient, and measurable defect detection and correction. The process of software inspection is currently the best known technique available to address this objective. DoD-STD-2167A provides only general software process requirements (e.g., configuration management, formal qualification testing). In order to promote early and efficient defect detection, the Government should consider augmenting the RFP to include direct reference to the inspection process. The RFP should reflect the following considerations in the appropriate sections:

a. A statement of the Government's desire to have the contractor use inspections during the software development process should be included in the RFP. A reference to the inspection process described in IEEE Standard 1028, IEEE Standard for Software Reviews and Audits, would ensure that the contractor understands that a rigorous inspection process is desired, as opposed to the less rigorous structured walkthrough or informal review [IEEE 1028-1988].

b. A DID should be created to specify the in-process inspection data to be produced by the contractor. The DID should also specify where the inspection data will be located (e.g., the Software Development Folder (SDF)). This data is described in the next section.

c. The Software Development Plan (SDP) should be tailored to have the contractor describe the proposed inspection process, the types of workproducts that will undergo inspection (e.g., requirements, design code, test cases) and identify where the schedule for the inspections will be located (e.g., the SDF).

After contract award and during the development process, the Government should periodically check to ensure that the inspection process is "healthy" by examining the data being collected by the contractor (e.g., scheduled inspections are consistently being performed, reasonable rates are being logged for preparation and defect detection, the defect data is being used for process improvement, etc.). However, the Government should not micro-manage the process. The contractor will hopefully understand the value of the inspection process and take "ownership" of that process.

B.2 DATA ITEM DESCRIPTION FOR INSPECTIONS

Many of the examples of successful industry inspection usage describe a common set of data collected during the inspection process. This data is used by the contractor to monitor the progress of development and the quality of the product, and to improve the development process via immediate process improvement feedback. A DID should be developed to require that the contractor collect and record the following data per inspection:

a. Type of inspection

b. Number, type and severity of defects detected

c. Resources

    1. Number of inspection participants

2. Number of hours required for the overview, preparation, meeting, and rework phases

Further detail or refinement of this data is possible (e.g., defect severity can be classified in different ways), but this activity should be left to the contractor. The Government should leave room for the contractor to define and implement the finer details of the information to be produced.

B.3 EXAMPLE NSA INSPECTION TEXT

The National Security Agency uses NSA Manual 81-2 and 81-3 for managing the acquisition of software [NSAM 81-2, NSAM 81-3]. NSA 81-3 is also a military standard, DoD-STD-1703(NS). These manuals have been used on hundreds of projects and explicitly require the use of design and code inspections. These manual are applicable to both contractual and in-house software developments. Figure B-1 provides the inspection-specific text of NSA Manual 81-2. This text is an excellent example of the Government specifying the use of inspection during software development.

4.3 SOFTWARE DESIGN AND CODE INSPECTIONS

POLICY

Software developers shall conduct unit design and code inspections to facilitate the early detection of errors. These inspections shall be accomplished by having the software unit design and code reviewed by a team of individuals (normally between 3 and 7 people) who have sufficient technical abilities to review the material. Inspection procedures shall be documented in a Software Standards and Practices Manual.

REQUIREMENTS.

1. As a minimum, design and code inspections shall be conducted at the unit level as the design and code of each unit are completed. When the inspections of a software unit have been completed, the date of completion for each inspection shall be entered on the cover sheet of the Unit Development Folder for that unit.
2. The inspection team shall be composed of a moderator, the author or developer, and one or more inspectors. The moderator manages the inspection process and chooses the inspectors on the basis of special skills or knowledge they can bring to the inspection. At a unit design inspection, the individuals responsible for programming and testing the unit should be members of the inspection team. At a unit code inspection, the individual who designed the unit and the people responsible for testing the unit should be members of the inspection team.
3. As a minimum, unit design inspections shall include checks for the following:
   (a) Responsiveness of design to requirements;
   (b) Design completeness and consistency;
   (c) Flow of data through input/output interfaces;
   (d) Testability;
   (e) Compliance with design standards identified in the Software Standards and Practices Manual;
   (f) Other appropriate criteria, such as exception handling and error recovery procedures, modularity, and simplicity.
4. As a minimum, unit code inspections shall include checks for the following:
   (a) Ensure that the code matches the design;
   (b) Ensure the correctness of the code;
   (c) Compliance with programming standards identified in the Software Standards and Practices Manual;
   (d) Maintainability and Testability;
   (e) Other appropriate criteria, such as external linkages, logic, data area usage, and register usage.
5. The technique used for unit design and code inspections shall consist of six formal steps: Planning, Overview, Preparation, Inspection, Rework, and Follow-up.

RESPONSIBILITIES

Members of organizations acquiring and developing software are responsible for implementing these policies unless specific policies have been waived by the Acquisition Decision Authority. Members of other organizations participating in the software acquisition must also comply with them. Figure 3 depicts the single point of management responsibility for each activity, review and decisions made during the software acquisition phase. Roles and responsibilities identified in Figure 3 may be satisfied by designated people within a project management structure rather than by a single Software Acquisition Manager or Software Development Manager.

Figure B-1 Actual Text for Inspection Requirements

APPENDIX C. SOFTWARE INSPECTION AND REVIEW ORGANIZATION

The Software Inspection and Review Organization (SIRO) is an informal professional organization that promotes the exchange of information on group-based software examination methods. SIRO began as an informal group of industry practitioners interested in discussing experiences and techniques for software review techniques such as inspections, structured walkthroughs, peer review, etc. SIRO does not advocate any particular review method, nor does it directly support or endorse any workshops, tutorials or training. Formal SIRO meetings began in 1992 and are often held in conjunction with conferences such as the annual Software Testing and Review (STAR) conference and the Applications of Software Measurement (ASM) conference.

SIRO publishes a newsletter about two times a year. An on-line archive service is available for SIRO members to examine a variety of software inspection and review information. Contact Mike Carny, mike@cray.com, 612-683-5635, for more information on this service. For SIRO membership information, contact the SIRO Membership Secretary, PO Box 61015 Sunnyvale, CA 94088-1015. At the present time, there are no membership fees.

APPENDIX D. BIBLIOGRAPHY WITH ABSTRACTS

This appendix provides a bibliography of inspection-related references. Abstracts are provided verbatim for many of the references. The bibliography is separated into three sections. Section D.1 contains references focusing on the subject of software inspection. Section D.2 provides additional references relating to software reviews and walkthroughs. Section D.3 identifies software engineering textbooks that have chapters discussing inspection techniques.

D.1 INSPECTION REFERENCES

[Ackerman 1982]

Ackerman, A. Frank, Amy S. Ackerman, and Robert G. Ebenau. "A Software Inspections Training Program," *COMPSAC '82: 1982 Computer Software and Applications Conference*, Chicago, IL Nov. 8-12, pp. 443-444. IEEE Computer Society Press.

[Ackerman 1984]

Ackerman, A. Frank, Priscilla J. Fowler, and Robert G. Ebenau. 1984. "Software Inspections and the Industrial Production of Software," *Software Validation*, H.L. Hausen, ed., pp. 13-40, Elsevier, Amsterdam.

Abstract: Software inspections were first defined by M.E. Fagan in 1976. Since that time they have been used within IBM and other organizations. This paper provides a description of software inspections as they are being utilized within Bell Laboratories and the technology transfer program that is being used for their effective implementation. It also describes the placement of software inspections within the overall development process, and discusses their use in conjunction with other verification and validation techniques.

[Ackerman 1989]

Ackerman, A. Frank, Lynne S. Buchwald, and Frank H. Lewski. "Software Inspections: An Effective Verification Process," *IEEE Software*, Vol. 6, No. 3, May 1989, pp. 31-36.

Abbreviated Introduction: 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.

[Ascoly 1976]
Ascoly, Joseph, Michael J. Cafferty, Stephen J. Gruen, and O. Robert Kohli. "Code Inspection Specification," IBM Corp., Kingston, NY, Technical Report TR 21.630, 1976.

Abstract: Examination of computer programs by people other than the code is recognized as a tangible method for improving quality in programming. This report is intended for use as a specification for conducting inspections of program code. Inspections are considered to be a more rigorous form of examination than walk-throughs. They stress participant preparation, error detection versus solution hunting and education, and accountability for resolution of problems detected. Inspections are applicable in both systems and application programming environments.

[Bisant 1989]
Bisant, David B., and James R. Lyle. "A Two-Person Inspection Method to Improve Programming Productivity," *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, Oct. 1989, pp. 1294-1304.

Abstract: This paper reviews current research and investigates the effect of a two-person inspection method on programmer productivity. This method is similar to the current larger team method in stressing fault detection, but does not use a moderator.

The experiment used a Pretest-Posttest Control Group design. An experimental and control group of novices each completed two programming assignments. The amount of time taken to complete each program. (Time1, Time2) was recorded for each subject. The subjects of the experimental group did either a design inspection, a code inspection, or both during the development of the second program. An analysis of variance was performed and the relationship between Time1 and Time2 was modeled for both groups. A comparison of the models revealed the experimental group improved significantly in programming speed as a result of using the two-person inspection. It also appeared as though this method was more effective at improving the performance of the slower programmers.

This two-person method could have its application in those environments where access to larger team resources is not available. If further research establishes consistency with this method then it might be useful as a transition to the larger team method.

[Blakely 1991]
Blakely, Frank W. and Mark E. Boles. "A Case Study of Code Inspections," *Hewlett-Packard Journal*, Vol. 42, No. 4, Oct. 1991, pp. 58-63.

Abstract: Code inspections have become an integral part of the software development life cycle in many organizations. Because it takes some project time and because engineers initially feel intimidated by the process, code inspections have not always been readily accepted. Additionally, there has not always been enough evidence (metrics) to provide that for the time and effort invested, the process has any value in reducing defects and improving overall software quality. Since the early days, the process has become better understood and documented, and recent articles have provided concrete metrics and other evidence to justify the value of the process.

This paper describes our experiences in bringing the code inspection process to HP's Application Support Division (ASD). We describe both the positive and negative findings related to using code inspections. Although we only have metrics for one project, out main goal here is to present how we implemented the inspection process and to illustrate the type of data to collect and what might be done with the data.

[Bollinger 1992]

Bollinger, Donald E., Frank P. Lemmon, and Dawn L. Yamine. "Providing HP-UX Kernel Functionality on a New PA-RISC Architecture," *Hewlett-Packard Journal*, Vol. 43, No. 3, Jun. 1992, pp. 11-15.

Abstract: Hewlett-Packard Co's HP 9000 Series 700 workstation development goals required that the HP-UX kernel laboratory change the normal software development process, the number of product features and the management structure. The laboratory wanted to change or add the minimum number of HP-UX kernel functions that meet customer needs and its own performance goals while also adapting to a new I/O system. The resulting HP-UX kernel code is called minimum core functionality (MCF). The management structure was changed to allow small teams of individual developers and first-level managers to make important program decisions quickly and directly. The performance team included members from hardware, kernel, languages, graphics and performance measurement groups; the team's goal was to maximize system performance in computation, graphics and I/O. The quality control plan, certification process, design and code reviews, branch and source management and test setup process are described.

[Britcher 1988]

Britcher, Robert N. "Using Inspections to Investigate Program Correctness," *IEEE Computer*, Vol. 21, No. 11, Nov. 1988, pp. 38-44.

Conclusion: As we develop better tools for recording and compiling software designs and code, those who think about and practice programming will take greater interest in the more obscure aspects of a program: its intent, meaning, resilience, and developmental history. Although the problem of writing correct programs, especially those embedded within large systems or products, remains largely unsolved in practice, the situation is improving. We can use inspections to further the investigation into how correct programs are constructed. Several such inspections will be carried out to determine their usefulness and refine their practice. The purpose of incorporating correctness arguments into inspections is not to improve inspections, but to improve programming. This is not a modest objective. Steps will necessarily be small.

[Brothers 1990]

Brothers, L., V. Sembugamoorthy, and M. Muller. "ICICLE: Groupware for Code Inspection," *CSCW 90: Proceedings of the ACM Conference on Computer Supported Cooperative Work*, Oct. 1990, pp. 169-181.

[Brykczynski 1993]

Brykczynski, Bill, and David A. Wheeler. "An Annotated Bibliography on Software Inspections," *ACM Software Engineering Notes*, Jan. 1993, Vol. 18, No. 1, pp 81-88.

[Buck 1981]

Buck, F.O. "Indicators of Quality Inspections," IBM Corp., Technical Report TR21.802, Sep. 1981.

Abstract: Management of a software development effort using the formal inspection process requires constant monitoring of the quality of those inspections. The number of errors found during an inspection is not an adequate indicator of a quality inspection. The number of errors found is just as much a function of the quality of the materials being inspected as it is a function of the quality of the inspection itself. This report presents an analysis of the results of many code inspections on the same materials. With constant quality materials, the alternative inspection indicators could be more accurately evaluated.

[Buck 1984]

Buck, Robert D. and James H. Dobbins. 1984. "Application of Software Inspection Methodology in Design and Code," *Software Validation*, H.L. Hausen, ed., Elsevier, Amsterdam, pp. 41-56.

[Bush 1990]

Bush, Marilyn. "Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory," *12th International Conference on Software Engineering*, 1990, pp. 196-199, IEEE Computer Society Press.

Abstract: Finding and fixing defects early in the software development life cycle is much cheaper than finding and fixing the same defects later on. After surveying detection practices in the best of industry, JPL Software Product Assurance decided that the most cost-effective early defect detection technique was the "Fagan inspection" procedure. This paper will describe this technique, how it was introduced to JPL, some of the difficulties involved in "transferring technology" and the first provisional set of results.

[Chaar 1992]

Chaar, J.K., M.J. Halliday, I.S. Bhandari, and R. Chillarege. "In-process Metrics for Software Inspection and Test Evaluations," IBM Corp., Technical Report 80725, 1992.

[Christel 1991]

Christel, M.G. "A Comparative Study of Digital Video Interactive Interfaces in the Delivery of a Code Inspection Course," Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, Jun. 1991.

Abstract: Past research into interactive video educational software has focused primarily on comparing the instructional effectiveness of an interactive video course with more traditional media, such as classroom lecture. Typical effectiveness measures include recall performance and attitude shifts. While such research generally finds in favor of the interactive video course, few formal examinations of the course exist to explain these results, including studies into the contributions of the interface design. An interactive digital video code inspection course was used to investigate whether the capabilities of digital video interfaces provide any advantages in an educational computer course.

Two by two factorial experiments were conducted to determine the effects of a computer course which included motion video versus one which contained no such video, and the effects of navigating through a series of related still images (surrogate travel) versus clicking a mouse on predefined areas of a single still image. The effects under study were recall performance, and shifts in meaning, measured with semantic scales, toward code inspection-related terms and educational media terms.

The code inspection course, developed at the Software Engineering Institute on a Digital Video Interactive platform, was used by seventy-two college seniors and master's students. Each student used one of these four treatments of the course in isolation for up to three hours.

The findings suggest that the presence of motion video in interfaces can lead to better recall performance than if no motion video exists in the interface. Material containing some motion video will be recalled better than if the same material is presented as audio with still images.

There were also significant differences in the shifts in meaning, calculated by subtracting a pretest score from a post test score, produced by the motion video and navigation independent variables. After the course, the surrogate travel navigation subjects rated code inspection-related terms as more powerful and "classroom instruction" as less powerful than the single still navigation subjects. Subjects receiving motion video shifted their views of code inspection concepts toward more active than did the subjects receiving no motion video.

[Christel 1992a]

Christel, Michael. "Virtual Reality Today on a PC," *Instruction Delivery Systems*, Jul./Aug. 1992, pp. 6-9.

Abstract: A digital video course on software technical reviews illustrates how it is done and offers some lessons learned.

[Christel 1992b]

Christel, Michael G., and Scott M. Stevens. Rule Base and Digital Video Technologies Applied to Training Simulations. SEI Technical Review '92, Software Engineering Institute, Pittsburgh, PA, 1992.

Abstract: The Advanced Learning Technologies Project developed a digital video course on code inspections from 1987 to 1990. The essence of this course is an environment in which a student participates in a code inspection as a contributing reviewer of the code.

The student chooses an inspection role, and later assumes all the responsibilities of that role while performing in a code inspection simulation. The student is an active participant in the code inspection, and his or her contributions affect the course of the inspection dialogue and ultimately the success of the inspection. In addition, the role the student takes in the inspection is not predetermined but is selected by the student.

To participate effectively in the code inspection, the student needs to recognize and react to the other reviewer's comments and their emotional states. The importance of

group process issues necessitates that the inspection simulation be presented as realistically as possible while still preserving the flexibility of dynamic role selection and active participation. The code inspection course makes use of digital video for dynamic scene creation in addressing this requirement.

These techniques are applicable beyond the code inspection course to other instructional simulations. The synergistic effects of using digital modeling and dynamic scene creation can significantly improve the utility of low-cost simulators and part-task trainers.

[Christel 1992c]

Christel, Michael G. "Experiences with an Interactive Video Code Inspection Laboratory." *Lecture Notes in Computer Science 640: Software Engineering Education SEI Conference Proceedings*, Oct. 1992 in San Diego, CA, C. Sledge, editor. Berlin: Springer-Verlag, 1992, pp. 395-411.

Abstract: Software engineers need practical training in addition to classroom lectures in order to obtain the knowledge and skills necessary to succeed in industry. This training is provided by laboratories in other engineering disciplines. Such laboratories have been implemented as computer-based interactive video courses in the past, with numerous advantages. Based on this success, an interactive video course was created for use as a "code inspection laboratory", in which the skills of preparing for and participating in code inspections are learned and practiced. This paper summarizes the anecdotal feedback and usage data from 120 students who used the course over the past two years. Lessons learned from these experiences are discussed, with implications for the development of future interactive video software engineering laboratories.

[Christenson 1987]

Christenson, Dennis A. and Steel T. Huang. "Code Inspection Management Using Statistical Control Limits," *National Communications Forum*, Vol. 41, No. 2, Chicago, IL, 1987, pp. 1095-1100.

[Christenson 1988]

Christenson, Dennis A. and Steel T. Huang. "A Code Inspection Model for Software Quality Management and Prediction," *GLOBECOM '88. IEEE Global Telecommunications Conference and Exhibition*, Hollywood, FL, 1988, pp. 468-472.

[Christenson 1990]

Christenson, Dennis A., Steel. T. Huang, and Alfred J. Lamperez. "Statistical Quality Control Applied to Code Inspections," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 2, Feb. 1990, pp. 196-200.

Abstract: Code inspections have been used on the 5ESS Switch project since 1983. Beginning wit ha training program for all the developers involved in the project, code inspections have improved with each new 5ESS Switch generic. The improvement in code inspections has been the result of hard work and innovation on the part of the 5ESS Switch software developers, and the use of some "Statistical Quality Control" (SQC) techniques.

Variations on a standard SQC technique, the control chart, have been used to track the metrics indicative of the effectiveness of code inspections. Parameters used in the computation of these metrics include the preparation effort, inspection time, number of inspectors, the size of the inspected unit of code, and the number of errors found at the inspection. The exact form that these "control charts" have taken has evolved and improved with experience.

[Collofello 1987]

Collofello, James S. "Teaching Technical Reviews in a One-Semester Software Engineering Course," *ACM SIGCSE Bulletin*, Vol. 19, No. 1, Feb. 1987, pp. 222-227.

Abstract: Software technical reviews are essential to the development and maintenance of high quality software. These review processes are complex group activities for which there exist an abundance of basic concepts evolved over years of practical experience. In a typical one-semester software engineering course very little of this information is adequately conveyed to students. Texts supporting this course are also very weak in this area. This paper provides a practical approach for teaching about software technical reviews in a one-semester software engineering course. The contents for two to three lectures on this topic are described as well as suggested exercises and an approach for integrating technical reviews with the usual team project. An extensive annotated bibliography is also provided to assist instructors and students.

[Collofello 1988]

Collofello, James S. "The Software Technical Review Process," Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, CMU/SEI-CM-3-1.5, Jun. 1988.

Capsule Description: This module consists of a comprehensive examination of the technical review process in the software development and maintenance life cycle. Formal review methodologies are analyzed in detail from the perspective of the review participants, project management and software quality assurance. Sample review agendas are also presented for common types of reviews. The objective of the module is to provide the student with the information necessary to plan and execute highly efficient and cost effective technical reviews.

[Cross 1988]

Cross, John A., ed. "Support Materials for the Software Technical Review Process," Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, CMU/SEI-SM-3-1.0, Apr. 1988.

[Crossman 1977]

Crossman, Trevor D. "Some Experiences in the Use of Inspection Teams," *15th Annual ACM Computer Personnel Research Conference*, Aug. 1977, p. 143.

[Crossman 1979]

Crossman, Trevor D. "Some Experiences in the Use of Inspection Teams in Application Development," *Applications Development Symposium*, Monterey, CA, Oct. 1979. pp. 163-168.

[Crossman 1982]

Crossman, Trevor D. "Inspection Teams, Are They Worth It?" *Proceedings 2nd National Symposium on EDP Quality Assurance*, Chicago, IL, Mar. 24-26, 1982.

[Deimel 1991]

Deimel, L.E. "Scenes of Software Inspections. Video Dramatizations for the Classroom," Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, CMU/SEI-91-EM-5, May 1991.

Abstract: This report describes the videotape "Scenes of Software Inspections," which contains brief dramatizations that demonstrate appropriate and inappropriate conduct of software inspections. The tape also includes scenes that show other kinds of group interactions. Any of these scenes can be incorporated into lectures, self-study materials, or other educational delivery mechanisms, to illustrate how to perform inspections, an important software engineering technique.

[Dichter 1992]

Dichter, C.R. "Two Sets of Eyes: How Code Inspections Improve Software Quality and Save Money," *Unix Review*, Vol. 10, No. 2, Jan. 1992, pp. 18-23.

Abstract: Programmers can detect a large percentage of software bugs by inspecting code to supplement testing. Testing alone will not determine if code will work on different platforms, if it is written efficiently and whether it adheres to particular coding guidelines or standards. Inspections and walkthroughs are two kinds of software reviews. Programmers perform inspections by sequential reading of code to search for bugs by using an inspection checklist. In walkthroughs, inspectors play the role of the computer by searching the code for logical errors. Programmers can start improving code by using advanced linter tools and then inspecting code for errors the linter will not catch. Code-counting tools are also helpful. Programmers may find that inspections on code of more than 1,000 lines will help find bugs that testing would not turn up and which would be more expensive to correct later.

[Dobbins 1987]

Dobbins, J.H. 1987. "Inspections as an Up-Front Quality Technique," *Handbook of Software Quality Assurance*, G.G. Schulmeyer and J.I. McManus, eds., pp. 137-177, NY: Van Nostrand Reinhold.

[Doolan 1992]

Doolan, E. P. "Experience with Fagan's Inspection Method," *Software—Practice and Experience*, Vol. 22, No. 2, Feb. 1992, pp. 173-182.

Abstract: Fagan's inspection method was used by a software development group to validate requirements specifications for software functions. The experiences of that group are described in this paper. In general, they have proved to be favourable. Because the costs of fixing errors in software were known, the payback for every hour invested in inspection was shown to be a factor 30. There are also other benefits that are much more difficult to quantify directly but whose effect is significant in terms of the overall quality of the software.

Some pointers are given at the end of this paper for those who want to introduce Fagan's inspection method into their own development environment.

[Drake 1992]

Drake, Janet, Vahid Mashaykhi, John Riedl, and Wei-Tek Tsai. "Support for Collaborative Software Inspection in a Distributed Environment: Design, Implementation, and Pilot Study," University of Minnesota Technical Report, TR 92-33, Jun. 1992.

[Ebenau 1981]

Ebenau, R.G. "Inspecting for Software Quality," *Second National Symposium in EDP Quality Assurance*, 1981. DPMA Educational Foundation, U.S. Professional Development Institute, Inc., 12611 Davon Drive, Silver Spring, MD 20904.

[Fagan 1976a]

Fagan, Michael E. "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182-211.

Abstract: Substantial net improvements in programming quality and productivity have been obtained through the use of formal inspections of design and of code. Improvements are made possible by a systematic and efficient design and code verification process, with well-defined roles for inspection participants. The manner in which inspection data is categorized and made suitable for process analysis is an important factor in attaining the improvements. It is shown that by using inspection results, a mechanism for initial error reduction followed by ever-improving error rates can be achieved.

[Fagan 1976b]

Fagan, M.E. "Design and Code Inspections and Process Control in the Development of Programs," IBM Corp., Poughkeepsie, NY, Technical Report TR 00.2763, Jun. 10, 1976. This report is a revision of "Design and Code Inspections and Process Control in the Development of Programs," IBM Corp., Kingston, NY, Technical Report TR 21.572, Dec. 17, 1974.

Abstract: Substantial net improvements in programming quality and productivity have been obtained through the use of formal inspections of design and code. Improvements are made possible by a systematic and efficient design and code verification process, with well defined roles for inspection participants. The manner in which inspection data is categorized and made suitable for process analysis is an important factor in attaining the improvements. Using inspection results, a mechanism for initial error reduction followed by ever improving error rates (down to minimum process average levels) can be achieved.

[Fagan 1977]

Fagan, Michael E. "Inspecting Software Design and Code," *Datamation*, Oct. 1977, pp. 133-144.

Introduction: Successful management of any process requires planning, measurement, and control. In program development, these requirements translate into defining the programming process in terms of a series of operations, each having its own exit crite-

ria. Next there must be some means of measuring completeness of the product at any point of its development by inspections or testing. And finally, the measured data must be used for controlling the process.

Design and code inspections have been applied successfully in several programming projects, both large and small, and including systems and applications programs. They have not been found to "get in the way" of programming, but instead enabled higher predictability than other means and improved productivity and product quality.

A process may be described as a set of operations occurring in a definite sequence that operates on a given input and converts it to some desired output. A general statement of this kind is sufficient to convey the notion of the process. In a practical application, however, it is necessary to describe the input, output, internal processing, and processing times of a process in very specific terms if the process is to be executed and we are to get practical output.

[Fagan 1986]

Fagan, Michael E. "Advances In Software Inspections," *IEEE Transactions on Software Engineering*, Vol. 12, No. 7, Jul. 1986, pp. 744-751.

Abstract: This paper presents new studies and experiences that enhance the use of the inspection process and improve its contribution to development of defect-free software on time and at lower costs. Examples of benefits are cited followed by descriptions of the process and some methods of obtaining the enhanced results.

Software inspection is a method of static testing to verify that software meets its requirements. It engages the developers and others in a formal process of investigation that usually detects more defects in the product—at and lower cost—than does machine testing. Users of the method report very significant improvements in quality that are accompanied by lower development costs and greatly reduced maintenance efforts. Excellent results have been obtained by small and large organizations in all aspects of new development as well as in maintenance. There is some evidence that developers who participate in the inspection of their own product actually create fewer defects in future work. Because inspections formalize the development process, productivity and quality enhancing tools can be adopted more easily and rapidly.

[Fowler 1986]

Fowler, Priscilla J. "In-Process Inspections of Workproducts at AT&T," *AT&T Technical Journal*, Vol. 65, No. 2, Mar./Apr. 1986, pp. 102-112.

Abstract: In-process inspections are examination meetings held to find defects in design and development work products, including intermediate versions of the product or system in requirements and design documents. Because these inspections delimit the phases of design and development processes, they can prevent the passage of defects from one phase to the next and significantly reduce the number of defects released to customers. Software development projects within AT&T's research and development community have been using in-process inspections effectively for several years to reduce defects, and hardware projects began using them one and a half years ago. In

addition, the experience of installing in-process inspections in project organizations has yielded a wealth of information on technology transfer. This article defines inspections, describes the installation process, and discussed some uses for inspection data.

[Freedman 1982]

Freedman, Daniel P. and Gerald M. Weinberg. 1982. *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*. 3rd ed. Boston, MA: Little, Brown and Company.

[Gilb 1991]

Gilb, Tom. "Advanced Defect Prevention Using Inspection, Testing, and Field Data as a Base," *American Programmer*, May 1991, pp. 38-45.

[Gintell 1993]

Gintell, John, John E. Arnold, Michael Houde, Jacek Kruszelnicki, Roland McKenney, and Gerard Memmi. "Scrutiny: A Collaborative Inspection and Review System," to be appear in *Fourth European Software Engineering Conference*, Garwisch - Partenkirchen, Germany, September 13-17, 1993.

Abstract: This paper describes a Bull US Applied Research Laboratory project to build a collaborative inspection and review system called Scrutiny using Conversation-Builder from the University of Illinois at Urbana-Champaign. The project has several distinct aspects: technology oriented research, prototype building, experimentation, and tool deployment/technology transfer. Described are the design of the current operational version of Scrutiny for inspection-only, the evolutionary design of Scrutiny to handle various forms of review, and some initial thoughts on integration with other CASE frameworks and tools. The problem domain selected, the development environment, lessons learned thus far, some ideas from related work, and the problems anticipated are discussed here.

[Glass 1993]

Glass, Robert L. "Error Detection: Which is Better, Reviews or Testing?" *Journal of Systems and Software*, Editors Corner, Vol. 22, No. 1-2, pp. 1-2.

[Graden 1986]

Graden, Mark E. and Palma S. Horsley. "The Effects of Software Inspections on a Major Telecommunications Project," *AT&T Technical Journal*, Vol. 65, No. 3, May/Jun. 1986, pp. 32-40.

Introduction: Software inspections are a highly formalized and rigorous technique used for the identification and removal of errors in software products. Faithfully applied, they have beneficial impact on the productivity and quality of a project. As a result, software inspections were selected as a critical ingredient in the overall Software Quality Assurance Plan to guide the development and evolution of a major, real time telecommunications software project.

This paper describes how the results of software inspections have been used to explain differences in end-product quality and identifies useful techniques for applying the results of software inspections to manage the software development process.

[Hale 1978]

Hale, R. M. "Inspections in Application Development—Introduction and Implementation Guidelines," IBM Corp., Form GC20-2000-0 (Jul. 1977) updated by TNL GN20-3814 (Aug. 1978).

[Hollocker 1990]

Hollocker, Charles P. 1990. *Software Reviews and Audits Handbook*, NY: John Wiley & Sons.

[IBM 1976]

IBM. "Code Reading, Structured Walkthroughs, and Inspections," IBM Corp., Report GE-19-5200, Zoetermeer, Netherlands, 1976.

[IEEE 1988]

"IEEE Standard for Software Reviews and Audits," ANSI/IEEE STD 1028-1988, IEEE Computer Society, Jun. 30, 1989.

> **Scope:** The purpose of this standard is to provide definitions and uniform requirements for review and audit processes. It does not establish the need to conduct specific reviews or audits; that need is defined by local policy. Where specific reviews and audits are required, standard procedures for their execution must be defined.
>
> This standard provides such definition for review and audit processes that are applicable to products and processes throughout the software life cycle. Each organization shall specify where and when this standard applies and any intended deviations from this standard.

[Johnson 1993]

Johnson, Philip M., and Danu Tjahjono. "Improving Software Quality through Computer Supported Collaborative Review," *Third European Conference on Computer Supported Cooperative Work, Milan*, Italy, Sep. 1993.

> **Abstract:** Formal technical review (FTR) is a cornerstone of software quality assurance. However, the labor-intensive and manual nature of review, along with basic unresolved questions about its process and products, means that review is typically underutilized or inefficiently applied within the software development process. This paper introduces CSRS, a computer-supported cooperative work environment for software review that improves the efficiency of review activities and supports empirical investigation of the appropriate parameters for review. The paper presents a typical scenario of CSRS in review, its data and process model, application to process maturation, relationship to other research, current status, and future directions.

[Jones 1985]

Jones, C.L. "A Process-integrated Approach to Defect Prevention," *IBM Systems Journal*, Vol. 24, No. 2, 1985, pp. 150-167.

> **Abstract:** Recent efforts to improve quality in software have concentrated on defect detection. This paper presents a programming process methodology for using causal analysis and feedback as a means for achieving quality improvements and ultimately

defect prevention. The methodology emphasizes effective utilization of all error data to prevent the recurrence of defects.

[Kelly 1992]

Kelly, John C., Joseph S. Sherif, and Jonathan Hops. "An Analysis of Defect Densities Found During Software Inspections," *Journal of Systems and Software*, Vol. 17, No. 2, Feb. 1992, pp. 111-117.

Abstract: Software inspection is a technical evaluation process for finding and removing defects in requirements, design, code, and tests. The Jet Propulsion Laboratory (JPL), California Institute of Technology, tailored Fagan's original process of software inspections to conform to its software development environment in 1987. Detailed data collected from 203 inspections during the first three years of experience at JPL included averages of staff time expended, pages covered, major and minor defects found, and inspection team size. The data were tested for homogeneity. Randomized samples belonging to the various phases or treatments were analyzed using the completely randomized block design analysis of variance (a = 0.05). The results showed a significantly higher density of defects during requirements inspections. The number of defect densities decreased exponentially as the work products approached the coding phase because defects were fixed when detected and did not migrate to subsequent phases. This resulted in a relatively flat profile for cost to fix. Increasing the pace of the inspection meeting decreased the density of defects found. This relationship held for major and minor defect densities, although it was more pronounced for minor defects.

[Kindl 1992]

Kindl, Mark R. "Software Quality and Testing: What DoD Can Learn from Commercial Practices," US Army Institute for Research in Management Information, Communications, and Computer Sciences, ASQG-GI-92-012, 31 August 1992.

Abstract: With regard to software testing in DoD, we can summarize our conclusions in two fundamental ideas. First, DoD knows how to produce quality software at low cost. This is because organizations such as DoD STEP, Army STEP, and Software Engineering Institute have already researched and documented policies for DoD. A few commercial software developers practice many of the DoD policies and directives now, and produce quality software (for example, IBM FSC Houston). Second, quality cannot be tested into software. Only a well-defined, well-disciplined process with a continuous improvement cycle can ensure software quality. However, testing cannot be underestimated. Systematic testing activities that detect error earliest in the life cycle are necessary to drive process improvement and optimize the development of quality software. Such testing methods as formal inspection find defects early. This enables cost-effective error resolution, identification and removal of defect causes, and thus, prevention of future defect insertion. If practiced with discipline, such methods can evolve a self-correcting software development process that is stable, modeled, measured, and therefore, predictable. This development process engineers quality software faster at reduced cost.

[Kitchenham 1986]

Kitchenham, B.A., Kitchenham, A.P., and J.P. Fellows. "Effects of Inspections on Software Quality and Productivity," *ICL Technical Journal*, Vol. 5, No. 1, May 1986, pop. 112-122.

[Knight 1991]

Knight, John C. and Ethella Ann Myers. "Phased Inspections and their Implementation," University of Virginia, Computer Science Report No. TR-91-10. May 12, 1991. Also published in *ACM Software Engineering Notes*, Vol. 16, No. 3, Jul. 1991, pp. 29-35.

Abstract: Since the 1970s, non-mechanical review methods have become very popular as verification tools for software products. Examples of existing review methods are formal reviews, walkthroughs, and inspections. Another example is Fagan Inspections, developed in 1976 by Michael Fagan in an effort to improve software quality and increase programmer productivity. Fagan Inspections and other existing methods have been empirically shown to benefit the software development process, mainly by lowering the number of defects in software early in the development process. Despite this success, existing methods are limited. They are not rigorous, therefore, they are not dependable. A product that has been reviewed with an existing method has no quantitative qualities that are ensured by the method used.

This thesis presents a new review method, Phased Inspection, that was developed to be rigorous, reliable, tailorable, heavily computer supported, and cost effective. Phased Inspection consists of a series of partial inspections termed phases. Each phase is intended to ensure a single or small set of related properties. Phases are designed to be as rigorous as possible so that compliance with associated properties is ensured, at least informally, with a high degree of confidence.

A detailed description of Phased Inspection, and evaluation framework and preliminary evaluation, and a prototype toolset for support of Phased Inspection is presented.

[Kohli 1975]

Kohli, O. Robert "High-Level Design Inspection Specification," IBM Corp., Kingston, NY, Technical Report TR 21.601, Jul. 21, 1975.

Abstract: This report is written to be used as a specification for the inspection of high level design materials. This inspection (called I_0) together with the inspections of low level (detailed) design (I_1) and code (I_2) constitute an efficient process for detecting and removing programming errors prior to any machine testing. The report describes in detail the process of inspection high level design materials against specific exit criteria. Satisfaction of the exit criteria constitutes meeting the high level design complete checkpoint. Thus, I_0 provides a checkpoint for management to enable better control of the programming process. Inspections are applicable in both systems and application programming environments.

[Kohli 1976]

Kohli, O. Robert and Ronald A. Radice. "Low-Level Design Inspection Specification," IBM Corp., Kingston, NY, Technical Report TR 21.629, Apr. 1976.

Abstract: Examination of program design by people other than the designer is recognized as a tangible method for improving quality in programming. This report is intended for use as a specification for conducting an inspection of detailed (low level) design. This inspection (called I_1), together with the inspection of high level design (I_0) which precedes it and cod (I_2) which follows it, constitute an efficient process for detecting and removing programming errors prior to any machine testing. Inspections are applicable in both systems and application programming environments.

[Koontz 1986]

Koontz, W.L.G. "Experience with Software Inspections in the Development of Firmware for a Digital Loop Carrier System," *IEEE International Conference on Communications*, 1986 Conference Record, pp. 1188-1189.

[Larson 1975]

Larson, Rodney R. "Test Plan and Test Case Inspection Specification," IBM Corp., Kingston, NY, Technical Report TR21.585, Apr. 4, 1975.

Abstract: Inspections of design and code have proven to be a valuable part of the development cycle of a software component. Similar benefits can be derived by applying inspection techniques to the functional verification test plan and test cases. This report addresses how to apply an inspection process to the functional verification test plan and test cases.

[Letovsky 1987]

Letovsky, S., J. Pinto, R. Lampert, and E. Soloway. "A Cognitive Analysis of a Code Inspection," In *Empirical Studies of Programming*, G. Olson, S. Sheppard, and E. Soloway, Eds, Ablex Publishers, Norwood, N.J. 1987, pp. 231-247.

[Martin 1990]

Martin, Johnny, and W.T. Tsai. "N-fold Inspection: A Requirements Analysis Technique," *Communications of the ACM*, Vol. 33, No. 2, Feb. 1990, pp. 225-232.

Abstract: N-fold inspection used traditional inspections of the user requirements document (URD) but replicates the inspection activities using N independent teams. A pilot study was conducted to explore the usefulness of N-fold inspection during requirements analysis. A comparison of N-fold inspection with other development techniques reveals that N-fold inspection is a cost-effective method for finding faults in the URD and may be a valid technique in the development of mission-critical software systems.

[McCormick 1981]

McCormick, K.K. "The Results of Using a Structured Methodology, Software Inspections, and a New Hardware/Software Configuration on Application Systems," *Second National Symposium in EDP Quality Assurance*, 1981. DPMA Educational Foundation, U.S. Professional Development Institute, Inc., 12611 Davon Drive, Silver Spring, MD 20904.

[McKissick 1984]

McKissick, John Jr., Mark J. Somers, and Wilhelmina Marsh. "Software Design Inspection for Preliminary Design," *COMPSAC '84: 1984 Computer Software and Applications Conference*, Las Vegas, NV, Jul. 1984, pp. 273-281.

Abstract: The continuing need for improved computer software demands improved software development techniques. A technique for the inspection of preliminary software designs is described in this paper. Experience and results from the application of this technique are presented.

[Morakabati 1993]

Morakabati, Reza. "PCTE-based Inspection Tool —Design and Implementation," Bull USARL Research Report, RAD/USARL/93018, 1993.

[Myers 1978]

Myers, Glenford J. "A Controlled Experiment in Program Testing and Code Walkthroughs-Inspections," *Communications of the ACM*, Vol. 21, No. 9, Sep. 1978, pp. 760-768.

Abstract: This paper describes an experiment in program testing, employing 59 highly experienced data processing professionals using seven methods to test a small PL/1 program. The results show that the popular code walkthrough/inspection method was as effective as other computer-based methods in finding errors and that the most effective methods (in terms of errors found and cost) employed pairs of subjects who tested the program independently and then pooled their findings. The study also shows that there is a tremendous amount of variability among subjects and that the ability to detect certain types of errors varies from method to method.

[NASA 1983]

National Aeronautics and Space Administration. "Guidelines for Software Inspections," NASA Contractor Report 166521, 1983.

Scope: This document describes the application of Software Inspections as a means of Software Quality Assurance for use on software development projects at NASA/AMES Research Center, Moffett Field, California. These procedures have been adapted from original procedures and samples published by IBM in 1976. The procedures and materials were researched, adapted, tested, and documented for NASA/Ames by Informatics Professional Services.

[NSAM 1986]

National Security Agency/Central Security Service Software Acquisition Manual, NSAM 81-2, Fort George G. Meade, Maryland, May 15, 1986.

Abbreviated Foreward: When NSA Manual 81-2 was published in 1978, it established Agency policies and procedures for managing the acquisition of software. Since then, the management methodology described in the Manual has been used on hundreds of projects to improve the manner in which we acquire software systems.

This second edition of the Manual continues our efforts to develop better software products. It is not a radical departure from the original Manual, but it does bring the policies up-to-date. During the past seven years, we have received many recommendations and suggestions for improvement, and we have incorporated many of them in this second edition.

The policies in this Manual define the roles and responsibilities of Software Acquisition Managers and Software Development Managers. They require planning prior to development, specification of software before coding, reviews to assess progress, testing, and formal acceptance of software end products. Most of all, the Manual requires a disciplined, structured approach to software acquisition management.

[NSAM 1987]

National Security Agency/Central Security Service Software Product Standards Manual, NSAM 81-3/DOD-STD-1703(NS), Fort George G. Meade, Maryland, April 15, 1987.

Abbreviated Foreward: This manual is a companion volume to NSA Manual 81-2, the NSA/CSS Software Acquisition Manual. For in-house software development, it is called NSA Manual 81-3, NSA/CSS Software Product Standards Manual. For contracted software acquisition, the document is called DOD-STD-1703(NS), Software Product Standards. It provides outlines of required documents, programming standards, and descriptions of recommended software design methodologies. It also identifies Data Item Descriptions recommended for use in contracted software acquisition.

[O'Neill 1991]

O'Neill, Don. "What is the Standard of Excellence?" *IEEE Software*, May 1991, pp. 109-111.

[Peele 1982a]

Peele, R. "Code Inspections at First Union Corporation," *COMPSAC '82: 1982 Computer Software and Applications Conference*, Chicago, IL Nov. 8-12, 1982, pp. 445-446, IEEE Computer Society Press.

Abbreviated Introduction: During 1980, a task force was formed within the Systems Development Division of First Computer Services to examine the coding and testing functions and to recommend ways to increase productivity and improve the quality of these functions while maintaining high staff morale. The task force evaluated the Design and Code Inspection process developed by Mike Fagan of IBM and concluded that this approach offered [several] potential quality assurance benefits.

[Peele 1982b]

Peele, R. "Code Inspection Pilot Project Evaluation," *Second National Symposium in EDP Quality Assurance*, DPMA Educational Foundation, U.S. Professional Development Institute, Inc., 12611 Davon Dr., Silver Spring, MD 20904.

Abstract: At First Computer, a code inspection is conducted after the coding of a program or module is complete as indicated by a clean compilation of the program and prior to unit testing of the program. The completed program specifications and a clean compilation are the entry criteria for the inspection process. An inspection team at First

Computer consists of four members: one moderator and three inspectors. The moderator is the key person in the process with the responsibility to ensure the best possible review of the program. The moderator approves the team members for the inspection and makes the necessary decisions related to scheduling and conducting the sessions. The moderator is the facilitator of the inspection meetings but is also an active participant charged with finding defects. The moderator must log all defects found during the sessions, ensure that all defects found are corrected by the author, and decide whether or not to reinspect the code.

[Reeve 1991]

Reeve, J.T. "Applying the Fagan Inspection Technique," *Quality Forum*, Vol. 17, No. 1, Mar. 1991, pp. 40-47.

Abstract: This paper asks and briefly explains what Fagan inspection is, and how it differs from more established techniques. It proposes how the technique may be used as an integral part of the product appraisal process from initial proposal to release to customer. A proven plan of action for establishment of the technique is also proposed, together with evidence of its success.

[Remus 1984]

Remus, Horst. 1984. "Integrated Software Validation in the View of Inspections/Reviews," *Software Validation*, H.L. Hausen, ed., pp. 57-64, Elsevier, Amsterdam.

Abstract: The Software Development Process is being looked at as to the specific contribution of inspections/reviews to the discovery of wrong design directions or implementations. The benefits are evaluated under the aspects of quality/productivity improvement and/or cost savings.

[Runge 1982]

Runge, B. "The Inspection Method Applied to Small Projects," *6th International Conference on Software Engineering*, 1982, pp. 416-417.

Abstract: The Inspection Method is a quality-control for written material. It is used on large projects and takes 3 to 8 persons for correct use. This excludes small projects with less than three persons from proper inspection. This paper shows how the personnel restriction may be circumvented in small projects. An example of inspection in a small project (writing a report) is given.

[Russell 1991]

Russell, Glen W. "Experience with Inspection in Ultralarge-Scale Developments," *IEEE Software*, Vol. 8, No. 1, Jan. 1991, pp. 25-31.

Abbreviated Introduction: ... inspections can be very cost-effective and highly beneficial, even when scaled up for ultralarge projects. Here I present quantitative results based on a 1988 study of inspection of 2.5 million lines of high-level code at Bell-Northern Research.

The data represent one of the largest published studies in the industry and confirm that code inspection is still one of the most efficient ways to remove software defects. In the

box on pp. 28-29, I describe how to successfully introduce inspections in large-scale production environments.

[Schneider 1992]

Schneider, G. Michael, Johnny Martin, and W.T. Tsai "An Experimental Study of Fault Detection in User Requirements Documents," *ACM Transactions on Software Engineering and Methodology,* Vol. 1, No. 2, Apr. 1992, pp. 188-204.

Abstract: This paper describes a software engineering experiment designed to confirm results from an earlier project which measured fault detection rates in *user requirements documents* (URD). The experiment described in this paper involves the creation of a standardized URD with a known number of injected faults of specific type. Nine independent inspection teams were given this URD with instructions to locate as many faults as possible using the N-fold requirements inspection technique developed by the authors. Results obtained from this experiment confirm earlier conclusions about the low rate of fault detection in requirements documents using formal inspections and the advantages to be gained using the N-fold inspection method. The experiment also provides new results concerning variability in inspection team performance and the relative difficult of locating different classes of URD faults.

[Sherif 1992]

Sherif, Joseph S. and John C. Kelly. "Improving Software Quality Through Formal Inspections," *Microelectronics and Reliability,* Vol. 32, No. 3, Mar. 1992, pp. 423-431.

Abstract: The software inspection process was created for the dual purpose of improving software quality and increasing programmers' productivity. This paper puts forward formal inspections as an alternative to and a better method than technical walkthroughs in the software lifecycle reviewing process. Examples of benefits gained in the development of defect-free software by utilizing formal inspections are cited.

[Shirey 1992]

Shirey, Glen C. "How Inspections Fail," *9th International Conference on Testing Computer Software,* Jun 15-18, 1992, Washington DC, pp. 151-159.

Abstract: This paper discusses an experience in the application of inspections in software development and how a concentration on the mechanics of the technology rather than acting on the information it provides failed to improve product quality. In this paper practitioners will find a comprehensive inspection model used to audit the inspection practices of the group studied. Managers will find examples of how to integrate the information provided by inspections into their Software Development Process.

[Stevens 1989]

Stevens, Scott M. "Intelligent Interactive Video Simulation of a Code Inspection," *Communications of the ACM,* Jul. 1989, Vol. 21, No. 7, pp. 832-843.

Abstract: The need for technical solutions to learning, in the software engineering field is increasing. The Advanced Learning Technologies Project (ALT) has developed a highly interactive, high-fidelity simulation of group process communication. The first

course demonstrating these techniques is on the formal technical review known as code inspection.

[Tomayko 1993]

Tomayko, James E., and James S. Murphy, "Materials for Teaching Software Inspections," Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, CMU/SEI-93-EM-7, Feb. 1993.

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, and 6) Video: *Candid Inspection,* which shows portions of an actual inspection.

[Tripp 1991]

Tripp, Leonard L., William F. Struck, and Bryan K. Pflug, "The Application of Multiple Team Inspections on a Safety-Critical Software Standard," *4th Software Engineering Standards Application Workshop,* May 20-24, 1991, San Diego, CA, pp. 106-111.

Abbreviated Introduction: This paper discusses the application of multiple team inspections to improve the technical review and quality of a safety-critical software standard. Inspections provide a structured and manageable discipline to an often ad-hoc technical review process.

The discussion includes material on the background of the standard and its revision, the inspection process, the application of a multiple team approach to inspections and results, lessons learned, and conclusions.

[van Emden 1992]

van Emden, Maarten H. "Structured Inspections of Code," *Software Testing, Verification and Reliability,* Sep. 1992, Vol 2, No. 3, pp. 133-153.

Abstract: Cleanroom programming and code inspections independently provide evidence that it is more efficient to postpone the testing of code to a later stage than is usually done. This paper argues that an additional gain in quality and efficiency of development can be obtained by *structuring* inspections by means of an inspection *pro-*

tocol. The written part of such a protocol is prepared by the programmer before the inspection. It is modelled on 'Floyd's method for the verification of flowcharts. However, the protocol differs from Floyd's method in being applicable in practice. Structured inspections gain this advantage by not attempting to be a proof; they are no more than an articulation of existing forms of inspection. With the usual method of structured programming it may be difficult to prepare the inspection protocol. On the other hand, 'assertion-driven programming' (of which an example is included in this paper) not only facilitates protocol preparation, but also the coding itself.

[Weinberg 1984]

Weinberg, Gerald M. and Daniel P. Freedman. "Reviews, Walkthroughs, and Inspections," *IEEE Transactions on Software Engineering*, Vol. 12, No. 1, Jan. 1984, pp. 68-72.

Abstract: Formal technical reviews supply the quality measurement to the "cost effectiveness" equation in a project management system. There are several unique formal technical review procedures, each applicable to particular types of technical material and to the particular mix of the Review Committee. All formal technical reviews produce reports on the overall quality for project management, and specific technical information for the producers. These reports also serve as an historic account of the systems development process. Historic origins and future trends of formal and informal technical reviews are discussed.

[Weller 1992a]

Weller, Edward F. "Experiences with Inspections at Bull HN Information Systems," *4th Annual Software Quality Workshop*, Aug. 2-6, 1992, Alexandria Bay, NY.

Abstract: Bull's experiences with the inspection process over the last two years will be discussed by using four case studies. Several successes as well as one "failure" are included. Data for requirements, design, and code inspections, and how it has been used outside the inspection process, are also presented.

[Weller 1992b]

Weller, Edward F. "Lessons Learned from Two Years of Inspection Data," *3rd International Conference on Applications of Software Measurement*, Nov. 15-19, 1992, La Jolla, CA, pp. 2.57-2.69. Also published in *Crosstalk: The Journal of Defense Software Engineering*, No. 39, Dec. 1992, pp. 23-28.

Abstract: Bull HN Information System's Major Systems Division in Phoenix initiated an inspection program in April 1990. Data collection was crucial to early buy-in to the inspection process. During the last 2 years, this data has been used to highlight potential direction for continuing process improvement. The data is also the basis for continuing development staff and management commitment to the program. Various metrics and the conclusions we have drawn from them will be discussed. A "case study" approach will highlight both the "good" and "bad" uses of inspection data for software process management.

[Wenneson 1985]

Wenneson, G. "Quality Assurance Software Inspections at NASA Ames: Metrics for Feedback and Modification," *Tenth Annual Software Engineering Workshop*, Dec.10, 1985, Goddard Space Flight Center.

[Youngblood 1991]

Youngblood, P.A. "Naval Ocean System Center (NOSC) Software Formal Inspection Process, Version 1.0," Technical Document 2246, December 1991.

Abbreviated Introduction: This document is a detailed presentation of the Naval Ocean Systems Center (NOSC) software formal inspection process. The inspection process identifies defects in requirements, design, test plans, software, and user documentation early in the software development process.

The main text defines each element of the inspection process and discusses an implementation strategy for the process at NOSC. Appendix A lists references and applicable DoD standards. Appendix B presents detailed formal inspection procedures. Appendix C gives sample formal inspection checklists. Appendix D gives sample formal inspection forms. Appendix E outlines the formal inspection metrics (FIM) database maintained by the chief moderator, who verifies that projects are implementing the process correctly and identifies areas for improvement of the inspection process. Appendix F outlines a suggested project database. Appendix G discussed metric analysis of inspection data. Appendix H outlines an inspection plan for the inspection process.

D.2 REVIEW/WALKTHROUGH-RELATED REFERENCES

[Bias 1991]

Bias, Randolph. "Walkthroughs: Efficient Collaborative Testing," *IEEE Software*, Vol. 8, No. 5, Sep. 1991, pp. 94-95.

[Hart 1982]

Hart, J. "The Effectiveness of Design and Code Walkthroughs," *COMPSAC '82: 1982 Computer Software and Applications Conference*, Chicago, IL Nov. 8-12, 1982, pp. 515-522, IEEE Computer Society Press.

[Lehman 1976]

Lehman, John H. "Software Engineering Techniques in Computer Systems Development," Department of the Air Force, Report No.: SM-ALC/ACD-76-04. 15 Dec. 1976.

[Lemos 1979]

Lemos, Ronald S. "An Implementation of Structured Walkthroughs in Teaching COBOL Programming," CACA, Vol. 22, No. 6, Jun. 1979, pp. 335-340.

[MILS 1985]

Military Standard for Technical Reviews and Audits for Systems, Equipments, and Computer Software. United States Department of Defense. 1985 MIL-STD-1521B.

[Myers 1988]

Myers, Ware. "Shuttle Code Achieves Very Low Error Rate," *IEEE Software*, Vol. 5, No. 5, Sep. 1988, pp. 93-95.

[Parnas 1987]

Parnas, David L. and David M. Weiss. "Active Design Reviews: Principles and Practices," *Journal of Systems and Software*, No. 7, 1987, pp. 259-265.

[Remus 1979]

Remus, Horst, and S. Zilles. "Prediction and Management of Program Quality," *4th International Conference on Software Engineering*, Sep. 1979, pp. 341-350, IEEE Computer Society Press.

Abstract: Techniques such as design reviews, code inspections, and system testing are commonly being used to remove defects from programs as early as possible in the development process. The objective of the authors is to demonstrate that predictors can be devised which tell us how well defects are being removed during the defect removal process.

[Shelly 1982]

Shelly, Gary B. and Thomas J. Cashman. 1982. "Implementation of Structured Walkthroughs in the Classroom," Section 12 of *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*, 3rd ed., pp. 425-434. Boston, MA: Little, Brown and Company.

[Waldstein 1976]

Waldstein, N.S. "The Walk-Thru—A Method of Specification, Design, and Review," IBM Corp., Poughkeepsie, NY, Technical Report TR 00.2436, 1976.

[Weinberg 1971]

Weinberg, Gerald M. 1971. *The Psychology of Computer Programming*. NY: Van Nostrand Reinhold.

D.3 SOFTWARE ENGINEERING TEXTBOOKS DISCUSSING REVIEWS, WALKTHROUGHS AND INSPECTIONS

[Dunn 1984]

Dunn, Robert H. 1984. *Software Defect Removal.* NY: McGraw-Hill Book Co. pp. 102-125.

[Dyer 1992]

Dyer, Michael. 1992. *The Cleanroom Approach to Quality Software Development.* NY: John Wiley & Sons. pp. 96-99.

[Gilb 1988]

Gilb, Tom. 1988. *Principles of Software Engineering Management.* Reading, MA: Addison-Wesley Publishing Co. pp 205-226, pp. 403-422.

[Grady 1992]

Grady, Robert B. *1992. Practical Software Metrics for Project Management and Process Improvement.* Prentice Hall, Englewood Cliffs, NJ.

[Humphrey 1989]

Humphrey, Watts. 1989. *Managing the Software Process.* Reading, MA: Addison-Wesley Publishing Co. pp. 463-486.

[Jones 1986]

Jones, Capers. 1986. *Programming Productivity.* NY: McGraw-Hill Book Co.

[Jones 1991]

Jones, Capers. 1991. *Applied Software Measurement.* NY: McGraw-Hill Book Co.

[Myers 1976]

Myers, Glenford J. 1976. *Software Reliability: Principles and Practices,* NY: John Wiley & Sons. pp. 17-25.

[Pressman 1992]

Pressman, Roger S. 1992. *Software Engineering: A Practitioner's Approach.* 3rd Edition. NY: McGraw-Hill Book Co. pp. 558-570.

[Yourdon 1989]

Yourdon, Edward. 1989. *Structured Walkthroughs,* 4th Edition. Englewood Cliffs, NJ: Yourdon Press.

LIST OF REFERENCES

[Ackerman 1989]

Ackerman, A. Frank, Lynne S. Buchwald, and Frank H. Lewski. "Software Inspections: An Effective Verification Process," *IEEE Software*, Vol. 6, No. 3, May 1989, pp. 31-36.

[Bisant 1989]

Bisant, David B., and James R. Lyle. "A Two-Person Inspection Method to Improve Programming Productivity," *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, Oct. 1989, pp. 1294-1304.

[Blakely 1991]

Blakely, Frank W. and Mark E. Boles. "A Case Study of Code Inspections," *Hewlett-Packard Journal*, Vol. 42, No. 4, Oct. 1991, pp. 58-63.

[Boehm 1987]

Boehm, Barry W. "Improving Software Productivity," *IEEE Computer*, Vol. 20, No. 9, September 1987, pp. 43-57.

[Brykczynski 1992]

Brykczynski, Bill, Reginald N. Meeson, Christine Youngblut, and David A. Wheeler. "Software Testing Initiative for Strategic Defense Systems," Institute for Defense Analyses, IDA Paper P-2701, March 1992.

[Brykczynski 1993]

Brykczynski, Bill, Reginald Meeson, and David A. Wheeler. May 4, 1993. "Memorandum for David Harris, subject, GPALS Software Standard."

[CRLCMP 1992]

Department of Defense (DoD) Strategic Defense Initiative Organization (SDIO). May 29, 1992. "GPALS Computer Resources Life Cycle Management Plan (GPALS/CRL-CMP)." SDI-S-SD-91-000001-01. Washington, DC: DoD SDIO.

[Deming 1986]

Deming, W. Edwards. Out of the Crisis, MIT Press, 1986.

[Dion 1993]

Dion, Ray. "Process Improvement and the Corporate Balance Sheet," *IEEE Software*, Vol. 10, No. 4, July 1993, pp. 28-35.

[DoDD 5000.1]

Department of Defense Directive 5000.1., "Defense Acquisition," 23 Feb 1991.

[DoDI 5000.2]

Department of Defense Instruction 5000.2, "Defense Acquisition Management Policies and Procedures," 23 Feb 1991.

[DoD-STD-2167A]

Department of Defense Standard 2167A, "Defense System Software Development," 29 February 1988.

[DoD-STD-2168]

Department of Defense Standard 2168, "Defense System Software Quality Program," 29 April 1988.

[DoD-STD-7935A]

Department of Defense Standard 7935A, "DoD Automated Information Systems (AIS) Documentation Standards," 31 October 1988.

[Doolan 1992]

Doolan, E. P. "Experience with Fagan's Inspection Method," *Software—Practice and Experience*, Vol. 22, No. 2, Feb. 1992, pp. 173-182.

[Fagan 1976a]

Fagan, Michael E. "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182-211.

[Fagan 1976b]

Fagan, Michael E. "Design and Code Inspections and Process Control in the Development of Programs," IBM Corp., Poughkeepsie, NY, Technical Report TR 00.2763, Jun. 10, 1976.

[Fagan 1986]

Fagan, Michael E. "Advances in Software Inspections," *IEEE Transactions on Software Engineering*, Vol. 12, No. 7, July 1986, pp. 744-751.

[Freedman 1982]

Freedman, Daniel P. and Gerald M. Weinberg. 1982. Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products. 3rd ed. Boston, MA: Little, Brown and Company.

[Gilb 1987]

Gilb, Tom. 1987. Principles of Software Engineering Management, Reading, MA: Addison-Wesley Publishing Co.

[Gilb 1991]

Gilb, Tom. "Advanced Defect Prevention Using Inspection, Testing, and Field Data as a Base," *American Programmer*, May 1991, pp. 38-45.

[Grady 1992]

Grady, Robert B. 1991. Practical Software Metrics for Project Management and Process Improvement. NJ: Prentice Hall.

[IEEE 1028-1988]

IEEE Standard for Software Reviews and Audits, ANSI/IEEE STD 1028-1988, IEEE Computer Society, Jun. 30, 1989.

[Jones 1986]

Jones, Capers. 1986. Programming Productivity. NY: McGraw-Hill Book Co.

[Jones 1991]

Jones, Capers. 1991. Applied Software Measurement. NY: McGraw-Hill Book Co.

[Juran 1986]

Juran, J.M. Leadership for Quality, 1986.

[Kelly 1992]

Kelly, John C., Joseph S. Sherif, and Jonathan Hops. "An Analysis of Defect Densities Found During Software Inspections," *Journal of Systems and Software*, Vol. 17, No. 2, Feb. 1992, pp. 111-117.

[Knight 1991]

Knight, John C. and Ethella Ann Myers. "Phased Inspections and their Implementation," University of Virginia, Computer Science Report No. TR-91-10. May 12, 1991. Also published in *ACM Software Engineering Notes*, Vol. 16, No. 3, Jul. 1991, pp. 29-35.

[MIL-STD-1521B]

Department of Defense Military Standard 1521B, "Technical Reviews and Audits for Systems, Equipments, and Computer Software," 4 June 1985.

[Myers 1988]

Myers, Ware. "Shuttle Code Achieves Very Low Error Rate," *IEEE Software*, Vol. 5, No. 5, Sep. 1988, pp. 93-95.

[NSAM 81-2]

National Security Agency Manual 81-2, "NSA/CSS Software Acquisition Manual," 15 May 1986, Fort George G. Meade, Maryland: NSA.

[NSAM 81-3]

National Security Agency Manual 81-3/DoD-STD 1703(NS), 15 April 1987. Two titles; the National Security Agency (NSA) refers to this for in-house development as "NSA Manual 81-3, NSA/CSS Software Product Standards Manual." For contracted software acquisition the document is called "DoD-STD-1703(NS), Software Product Standards."

[Paulk 1993a]

M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber, "Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, CMU/SEI-93-TR-24, February 1993.

[Paulk 1993b]

M.C. Paulk, C.V. Weber, S. Garcia, M.B. Chrissis, and M. Bush, "Key Practices of the Capability Maturity Model, Version 1.1," Software Engineering Institute, CMU/SEI-93-TR-25, February 1993.

[Russell 1991]

Russell, Glen W. "Experience with Inspection in Ultralarge-Scale Developments," *IEEE Software*, Vol. 8, No. 1, Jan. 1991, pp. 25-31.

[SDD 1992]

Department of Defense Standard SDD, "Draft Military Standard: Software Development and Documentation," 22 December 1992.

[SDIO 1992]

Strategic Defense Initiative Organization. December 17, 1992 (change pages) and September 4, 1992. GPALS Software Standards.

[SDIO 1993]

Strategic Defense Initiative Organization. April 1, 1993. Revised Software Trust Principles.

[Taylor 1993]

Taylor, Carol. August 5, 1993. Personal communication between David A. Wheeler with Carol Taylor (National Security Agency).

[Weinberg 1984]

Weinberg, Gerald M. and Daniel P. Freedman. "Reviews, Walkthroughs, and Inspections," *IEEE Transactions on Software Engineering*," Vol. 12, No. 1, Jan. 1984, pp. 68-72.

[Weller 1991]

Weller, Edward F. "Lessons Learned from Two Years of Inspection Data," *3rd International Conference on Applications of Software Measurement*, November 15-19, 1992, La Jolla, CA, pp. 2.57-2.69.

[Wheeler 1993]

Wheeler, David A. August 6, 1993. "Memorandum for Tom Barrett, Subject, GPALS CRLCMP Walk-through Section."

[Yourdon 1989]

Yourdon, Edward. 1989. Structured Walkthroughs, 4th Edition. Englewood Cliffs, NJ: Yourdon Press.

[Youngblut 1989]

Youngblut, Christine, Bill R. Brykczynski, John Salasin, Karen D. Gordon, and Reginald N. Meeson. SDS Software Testing and Evaluation: A Review of the State-of-the-Art in Software Testing and Evaluation with Recommended R&D Tasks. Institute for Defense Analyses, IDA Paper P-2132, February 1989.

LIST OF ACRONYMS

ASM	Applications of Software Measurement (conference)
AT&T	American Telephone & Telegraph
BE	Brilliant Eyes
BMD	Ballistic Missile Defense
BMDO	Ballistic Missile Defense Organization
BNR	Bell Northern Research
CDR	Critical Design Review
CMM	Capability Maturity Model
CRLCMP	Computer Resources Life Cycle Management Plan
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DEM/VAL	Demonstration/Validation
DID	Data Item Description
DoD	Department of Defense
EMD	Engineering and Manufacturing Development
FCA	Functional Configuration Audit
FQR	Formal Qualification Review
GPALS	Global Protection Against Limited Strikes
HWCI	Hardware Configuration Item
IBM	International Business Machines
IDA	Institute for Defense Analyses
IEEE	Institute for Electrical and Electronics Engineers
KLOC	K (thousand) Lines of Code

KPA	Key Process Area
NASA	National Aeronautics and Space Administration
NSA	National Security Agency
NSAM	National Security Agency Manual
PCA	Physical Configuration Audit
PDL	Program Design Language
PDR	Preliminary Design Review
PRR	Production Readiness Review
RFP	Request for Proposals
SCE	Software Capability Evaluations
SDD	Software Development and Documentation, or Software Design Document
SDF	Software Development Folder
SDI	Strategic Defense Initiative
SDIO	Strategic Defense Initiative Organization
SDP	Software Development Plan
SDR	System Design Review
SDS	Strategic Defense System
SEI	Software Engineering Institute
SIRO	Software Inspection and Review Organization
SRR	System Requirements Review
SRS	Software Requirements Specification
SSR	Software Specification Review
STAR	Software Testing and Review (conference)
TQM	Total Quality Management
TRR	Test Readiness Review
TSM	Trusted Software Methodology