Phased Inspections and Their Implementation

E. Ann Myers and John C. Knight

Computer Science Report No. TR-92-01
January 7, 1992

PHASED INSPECTIONS AND THEIR IMPLEMENTATION†

E. Ann Myers *John C. Knight*

Department of Computer Science
University of Virginia
Thornton Hall
Charlottesville, VA, 22903

(804) 924-7605

ABSTRACT

Inspection of software work products is common practice and has been shown to be a valuable tool for the software engineer. However, we believe that the technology is not being exploited as fully as possible. We define an enhanced inspection technique called *Phased Inspection* that addresses the deficiencies of existing inspection techniques. This technique is designed to permit the inspection process to be rigorous, tailorable, efficient in its use of resources, and heavily computer supported. The Phased Inspection process is designed to permit the engineer to trust the results of a specific inspection and to ensure that inspection results are repeatable. Phased Inspections inspect the work product in a series of small inspections termed phases each of which is designed to ascertain whether the work product possesses some desirable property. The skills of the staff performing a phase are tailored to the goals of the phase, and the checking that is performed during a given phase is defined precisely and computer supported. An important goal of Phased Inspection is computer support and we present details of a toolset that supports Phased Inspection by providing the inspector with extensive computer assistance and by checking for compliance with the required process. A preliminary evaluation of Phased Inspection is also presented.

Keywords and Phrases: Software inspections, reviews, walkthroughs.

† Sponsored in part by NASA grant NAG-1-1073, in part by SAIC Inc., and in part by the MITRE Corporation.

1. INTRODUCTION

Software reviews are not a new idea, they have been around almost as long as software. One of the most natural ways to check if something is correct is to look at it. Babbage and von Neumann both regularly asked colleagues to examine their programs [5]. In the 1950's and 1960's, large software projects often included some sort of software reviews, and in the 1970's these review mechanisms began appearing in publications. By the mid to late 1970's, various review methods had emerged with different names: software reviews, technical reviews, formal reviews, walkthroughs, structured walkthroughs, and code inspections. Each review method had different forms to fill out, different review team sizes and makeup, etc., but none suggested any approach for reviewing the software or other work product other than just looking at it and discussing it.

One might wonder why reviews are used at all since most software is tested anyway. There are several reason for doing something other than testing, including the the expense of testing and its insufficiency. Linger et al [8] state "It is well known that a software system cannot be made reliable by testing." Similarly, in support of inspections in engineering, Petroski states in his text *To Engineer Is Human* [11, p52]:

> "Engineers today, like Galileo three and a half centuries ago, are not superhuman. They make mistakes in their assumptions, in their calculations, in their conclusions. That they make mistakes is forgivable; that they catch them is imperative. Thus it is the essence of modern engineering not only to be able to check one's own work, but also to have one's work checked and to be able to check the work of others."

An interesting observation about inspections made by Fagan is that although a program cannot be tested completely, it can be *inspected* completely [3]. Since independent inspections are routine in many other disciplines, for example, financial accounting and building construction, it is surprising that inspection is not a significant element of all software development.

Empirical evidence has emerged showing that review methods based on human examination of a paper version of a product can have considerable benefit, usually by lowering the number of faults in the software. Freedman and Weinberg [4, 5] report that in large systems, reviews have reduced the number of errors reaching the testing stages by a factor of ten. This reduction cut testing costs by 50 - 80% including review costs. Fagan, referring to results compiled by Russell [12], states that "65 - 90% of operational defects are detected by inspection at 1/4 to 2/3 the cost of testing and removed at 1/7 - 1/2 the cost." [3].

Despite their success, existing review methods have major limitations. For example, existing review methods are not rigorous and are far too dependent on human effort. The lack of rigor means that, although existing review methods are cost effective statistically and generally beneficial to software development, they do not ensure that a *particular* product has any clear-cut quality after review. A project manager cannot say anything definite about a specific product after it has been reviewed but can usually say that reviews improve the quality of his organization's products overall. This is a rather serious limitation for the manager.

The general dependence of review methods on human effort is unnecessary. It is possible to supplement the review process considerably with computer resources. This permits far more efficient use of human time and more complete coverage of items that have to be reviewed.

In this paper we describe an enhanced technique for the inspection of software work products called *Phased Inspections*. This technique is designed to permit the inspection process

to be rigorous, tailorable, efficient in its use of resources, and heavily computer supported. Phased Inspections inspect the work product in a series of small inspections termed phases each of which is designed to ascertain whether the work product possesses some desirable property. The skills of the staff performing a phase are tailored to the goals of the phase, and the checking that is performed during a given phase is defined precisely and computer supported.

As well as describing an enhanced review process for software engineers to follow, we also present details of a comprehensive toolset to support Phased Inspections. The toolset contains extensive facilities that assist the inspector thereby allowing inspections to proceed rapidly. The toolset also supports enforcement of the process thus ensuring that inspections are carried out as required.

Since it is not sufficient merely to claim benefits for a new process, we also present a framework for evaluation of Phased Inspections and the results of a preliminary experimental evaluation.

2. EXISTING REVIEW METHODS

In the 1950's and 1960's many large software projects included some form of software review in the development process, but it was not until the work of Weinberg appeared in 1971 [14] that the review of programs in all stages of development was advocated and a method proposed. Since that time, review methods have appeared frequently in the literature. These review methods can be placed into one of three categories characterized by the strategy that drives the review process:

(1) *Formal Reviews*
In a formal review, the author of the software or one of the reviewers familiar with the software introduces the software to the rest of the reviewers. The flow of the review is driven by the presentation and issues raised by the reviewers.

(2) *Walkthroughs*
Walkthroughs are usually used to examine source code as opposed to design and requirements documents. The participants do a step-by-step, line-by-line simulation of the code. The author of the code is usually present to answer any questions the other participants might have.

(3) *Inspections*
In an inspection, a list of criteria that the software must satisfy determines the flow of the review. While walkthroughs and formal reviews are generally biased towards error detection, inspections are often used to establish other properties such as portability and adherence to standards [5]. A reviewer may be supplied with a checklist of items, or he may only be informed of the desired property. Inspections are also used to check for particular coding errors that have been prevalent in the past.

One of the most popular review methods was developed by Fagan [1, 2]. Fagan wanted to create a new review process that would improve software quality and increase programmer productivity. His method, informally known as Fagan Inspections, is a combination of a formal review, an inspection, and a walkthrough. This combination of review methods has made Fagan Inspections somewhat more formal and therefore more effective than previous methods.

Fagan's inspection method consists of five steps: overview, preparation, inspection, rework, and follow-up. In the overview, the author of the software explains the design and the logic of the software to the inspectors. During preparation, the inspectors study the software and any design documentation to prepare for the inspection. The inspection is controlled by a moderator, who in turn chooses a reader. The reader guides the inspectors through the work product in a detailed examination searching for errors. Every line of code is examined. A report of the inspection is prepared and given to the author who corrects the defects that were identified. The follow-up step checks that the defects were corrected.

Active Design Reviews are an important advance in review methods introduced by Parnas and Weiss [10]. The approach taken is to conduct several different, brief reviews rather than one large review thereby avoiding many of the difficulties of conventional reviews that Parnas and Weiss cite.

3. DEFICIENCIES OF EXISTING METHODS

Although existing methods are very successful, careful examination of their application in practice reveals various deficiencies. Clearly, no single method suffers from all of the deficiencies we identify. We note specifically that Active Design Reviews [10] suffer from relatively few. The following is an accumulation of deficiencies from various techniques:

- While all kinds of faults can be found by existing review methods, existing methods focus on defect detection. Defect detection is important, but correctness is not the only desirable characteristic of software products. Maintainability, portability, and reusability are examples of other characteristics with which a review method might be concerned. These other characteristics are important since, for example, a software product might have no errors but its value might be drastically reduced if it is not maintainable.

- In general, existing review methods are not dependable. As noted above, although they are beneficial in a statistical sense, existing methods do not ensure that a *particular* product has any specific quality after review. A project manager can usually say only that reviews improve the general quality of his organization's products. As noted above, this is a rather serious limitation for the manager. Managers should be able to make assumptions about qualities held by a particular product after review. In order to make reviews dependable, it must be possible to assert, either with certainty or with high probability, that a product which has been reviewed has certain properties. This means that the review process must be rigorous. Rigor permits conclusions to be drawn about a property of a product, and allows these same conclusions to be drawn about every product that is inspected. Equally important, rigor also allows the same conclusions to be drawn about a product irrespective of who is performing the review.

- Existing methods make ineffective use of human resources. It is not uncommon for highly paid software engineers participating in a review to debate spelling, comment conventions, and like trivia. Also, reviews are group activities and as such are susceptible to dominance by a single strong-willed individual. Others may have useful comments but are inhibited in such situations. In addition, a group activity in which there is no detailed, required, active participation by each member permits individuals who failed to prepare to sit quietly, not contribute, and for this to go largely unnoticed.

- There are many different types of defect that a software product might have. For example, there might be defects in the logic, the computations, or the tasking structure; there might be unacceptable inefficiencies; or there might be defects in the form of omitted functionality. In an inspection that follows traditional practice, the product is examined once, and it is expected that defects of all types will be checked for during this single examination. Although the participants in a traditional inspection might be experts in appropriate different areas, the inspectors are required to be checking for all the different types of defect simultaneously. It is unlikely that they will be able to meet this intellectual challenge.

- Existing review methods target paper products for examination and perform examinations typically in a meeting. Little to no computer support is used thereby exposing the process to risks of incompleteness and making poor use of human resources.

- Certain elements of existing methods are inappropriate. The overview step included in many review methods, for example, is quite inappropriate. It suggests strongly that the documentation of the product being reviewed is deficient in some way, perhaps even missing. If the documentation is complete *and properly presented*, an overview should not be necessary.

By addressing these deficiencies, we aim to improve inspection technology.

4. PHASED INSPECTIONS

We believe the benefits of inspections to be so great that they should be a required component of the creation of every work product in the software lifecycle. Further, we believe that for inspections to achieve their maximum cost effectiveness (and thereby productivity), they must be rigorous. Inspection should be a precisely defined activity that achieves a prescribed set of results. These results, once achieved, should be completely dependable thereby permitting other parts of the software lifecycle, such as testing, other forms of verification, and maintenance to be simplified, reduced, or streamlined.

Phased Inspection is an enhanced review method that is designed to deal with the deficiencies noted in the previous section and to provide the benefits just outlined. The most important goal for Phased Inspections is that they be rigorous. Rigor in a review method must be supported in at least two areas. First, the review method must be defined by a rigorous process. It must be possible to know exactly what actions will take place during an inspection so that inspectors know exactly what is required of them and so that the inspection process is repeatable. The second important support area is compliance checking. Just as inspections are required to check the work of others, so the work of the inspector must be checked. It must be possible to show that the rigorous process defined for inspections has actually been followed in practice.

Other goals for Phased Inspections are that they be tailorable so that they can serve functions other than defect detection, heavily computer supported so that human resources are used only where necessary, and efficient so that maximum use is made of available resources [9].

A Phased Inspection consists of a series of coordinated partial inspections termed *phases*. Each phase is intended to ensure that the product being inspected possesses either a single specific property or small set of closely related properties. The property checked during a given phase is chosen to be intellectually manageable so that comprehensive checking is a reasonable

expectation. If this is not possible, the property is split so that multiple phases can be used. The properties examined are ordered so that each phase can assume the existence of properties checked in preceding phases. The inspectors performing a given phase are held responsible for assuring that the properties defined for that phases have been fully checked. Taken together, the set of phases constitute a single phased inspection, and the total set of properties checked in all the phases ensure that the inspected product possesses some desirable general characteristic.

The concept of Phased Inspection has benefited from the work on Active Design Reviews [10]. Active Design Reviews focus on error detection in designs whereas Phased Inspections are intended to be used on any work product including requirements, designs, and source code. In addition, the phases of a phased inspection are orthogonal to the reviews of which an active review would be composed. A phase examines an entire product for compliance with a specific property whereas a review in an active design review examines part of the product. There are other differences between the two techniques.

Phased Inspections are tailorable so that they can be used to check for a wide range of desirable characteristics. They are *not* intended solely for finding errors. For example, they can be used to ensure that a particular work product has certain important properties related to portability, reusability, or maintainability. The present level of understanding of what is required to make software truly portable, for example, requires that the software comply with an extensive set of design rules. Inspection for compliance is a significant undertaking over and above what might be needed to inspect for errors and warrants a separate inspection in its own right.

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. To achieve this, two different types of phase have been defined. The first phase type, referred to as a *single-inspector* phase, is a rigidly formatted process driven by a list of unambiguous checks. For each check, the software either complies or it does not. The software cannot complete this type of phase until it complies with all of the checks in the list. As the name implies, the intent is that the checks will be performed by a single inspector working alone.

Single-inspector phases are used to establish properties such as compliance with important programming practices. Many subtle faults have been attributed in software to simple mistakes such as the omission of a break statement from within a switch statement in 'C', failure to include a return statement in some path through a function in Ada, and the incorrect combination of a record field and tag value in Pascal. Such errors can be avoided in many cases by following guidelines on language use, and assurance that the guidelines have been followed is a fairly simple inspection process.

Clearly, many simple qualities of this type can be established with a static analyzer. Our goal is to provide an inspection technology for those situations where static analysis is beyond the state of the art or a suitable analyzer does not exist.

The staff used in single-inspector phases can be chosen so that their qualifications meet the needs of the phase. A single-inspector phase checking compliance with internal documentation standards might be undertaken by a technical writer whereas a phase checking programming practices might be performed by a junior software engineer.

The second type of phase, referred to as a *multiple-inspector* phase, is designed to check for those properties of the software that cannot be captured in a precise yes/no statement. Typically,

such properties include completeness or correctness issues for requirements or functional correctness concerns of implementations. In a multiple-inspector phase, several inspectors examine the product *independently* and in a controlled way. They are provided at the outset with the necessary reference documentation for the product and begin with an examination of the reference documentation. The inevitable questions of clarification that they generate serve to improve that documentation. Note that in contrast to the overview of a traditional inspection, the inspectors are not provided with information that is not generally available. After examining the reference documents, they proceed with separate, independent inspections of the work product with the goal of establishing that the work product has the property defined for the phase. The separate inspections are followed by a *reconciliation* in which the individual inspectors compare their findings. In theory, the findings should be identical but in practice they will only be similar. The intent of this procedure is to avoid the personnel difficulties found to occur in typical group inspections.

We are presently experimenting with different approaches to improving the rigor of multiple-inspector phases. Our goal is to drive them with the same type of checklists used in single-inspector phases so as to achieve a similar degree of repeatability and rigor.

As an example of a simple phased inspection, consider the goal of checking 'C' source code for elementary desirable characteristics that might be considered important in production software. The phases, the goals of each phase, and the likely personnel involved are shown in Table 1. Phase 1 ensures compliance with required internal documentation checking format, placement, spelling, and grammar at the same time. Phase 2 examines the source code layout for compliance with required format. This phase might be obviated by a formatting tool that enforces local standards. Phase 3 checks the source code for readability in areas such as meaningful identifiers, use of abbreviations, and compliance with local naming standards. Production software written in 'C' has been known to use single-character identifiers thereby making the maintenance task much harder. Checking for compliance with good programming practices is done in phase 4. Checks in this phase might include freedom from unnecessary goto statements and absence of assignment in Boolean expressions. The checks performed in phase 5 are assurance of the correct use of various programming constructs such as updating the variables controlling while statements and explicitly closing files that are successfully opened. Finally, phase 6 is a multiple-inspector phase aimed at checking functional correctness.

Phase	Type	Goal	Staff
1	Single	Internal Documentation	Technical Editor
2	Single	Source Code Layout	Technical Editor
3	Single	Source Code Readability	Junior Software Engineer
4	Single	Programming Practices	Software Engineer
5	Single	Local Semantics	Software Engineer
6	Multiple	Functionality	Senior Software Engineers

Table 1 - Example Phased Inspection

5. COMPUTER SUPPORT

Phased Inspections are well suited to computer support. A prototype of a toolset for supporting Phased Inspections has been developed. This toolset, called **InspeQ** (Inspecting software in phases to ensure Quality) is implemented on Sun 3, Sun 4, and IBM RS/6000 computers running various forms of Unix. It uses the X-window display system and the OSF/Motif widget set.

5.1. Specific Facilities

Computer support features that have been implemented in **InspeQ** include navigation facilities for displaying the work product, documentation display facilities, facilities for the inspector to record his comments, and facilities to enforce the Phased Inspection process. Each of these features is described below.

- *Work Product Display*

The display window of **InspeQ** is a general tool for looking at the work product. It is the primary facility that the inspector uses during an inspection. The tool permits display, scrolling, repositioning, and searching the text. Multiple instances of the display window can be used to permit inspection of related but separate areas.

The work product display window is at the top of a hierarchy of window interfaces to tools that facilitate the inspection process. The other tools are the checklist display, the standards display, the highlight display, and the comments display. These tools are made available from a menu bar on the display window.

- *Checklist Display*

The checklist window displays the checklist associated with the current inspection phase for the product. The intent is to ensure that the inspector is informed of exactly what checks are involved in a given phase. The display also accepts input from the inspector indicating the status of the various required checks. This permits **InspeQ** to check compliance. The inspector can indicate for each checklist item either that the product *complies*, that the product *does not comply*, that the item was *not checked*, or that the item was *not applicable*.

- *Standards Display*

The standards window displays a complete definition of the standards that the checklists are designed to check. Examples that comply with the check are included for illustrative purposes also. Each item in the standards display is in a one-to-one correspondence with the checklist items in the checklist display and this is reflected in numbering within the two displays.

- *Highlight Display*

The highlighting facility helps the inspector find and isolate specific product features quickly. As the name implies, the highlighter allows the inspector to highlight certain syntactic features of interest by menu selection. In addition, the highlighter allows features to be extracted and displayed in a separate window. Multiple instances of a particular feature can be isolated in the highlighting window one at a time.

Isolating features in a separate window allows the inspector to concentrate on narrow sections of the product if desired, avoiding distraction by the feature's surroundings. If the inspector is checking a switch statement in a 'C' program, for example, he does not need to check how the control variable for the switch statement is used before or after the switch statement.

The highlight facility is useful in a number of ways. For example, a checklist item might require the inspector to check that all while statements terminate. The highlight facility allows the inspector to highlight all the while statements in the product, and sequentially check each one until he has checked them all. Without this facility, the inspector would have to locate the statements of interest either manually or using some general-purpose editor, and would have to monitor compliance manually.

The highlight facility does not support all desired syntactic elements of all possible work products. It requires syntactic information about the product produced by a syntax analyzer. General syntax analyzers for **InspeQ** have yet to be implemented. Presently, limited support is provided for 'C' source code.

- *Comments Display*

An inspector needs to record anything in the product with which he is not satisfied. The file comments window provides an editable text display for the inspector to do this. The commands controlling the display of the comments window are roughly equivalent to Emacs text editor commands. The inspector can use the Emacs commands, the mouse, or various special-purpose buttons to navigate about the file.

In order to provide context for the inspector's typed comments, sections of text or just the line numbers associated with a piece of text from any text display can be pasted into the comments using the mouse and various buttons. Pasting text from the work product being examined can be very useful when it is hard to explain a problem but easy to show by example. The inspector can paste a copy of the faulty code and then edit his comments. Another useful technique is to paste two copies of the faulty code and edit one to show how a correction can be made. This is sometimes an easy way of explaining a complex idea to the author. Where the text is overkill, the line numbers can be pasted instead to help the author identify the region to which an inspector's comments apply. **InspeQ** formats the inspector's comments in a file for submission to the author.

5.2. Enforcement

Enforcement is an integral part of the Phased Inspection process because it is essential to achieving the goal of rigor. **InspeQ** currently provides enforcement in two ways. First, it makes sure the inspector marks all of the items of the checklist associated with the assigned phase. If the inspector elects to pass the product on the current phase, **InspeQ** confirms that the inspector has marked all of the checklist items as compliant or not applicable, refusing to pass the work product if they are not. The inspector is believed if he marks a checklist item.

The second way in which enforcement is supported is to ensure that the phases of a Phased Inspection process are performed in the required order. Since phases can be defined to depend on one another, order is important.

Future versions of **InspeQ** will make it possible to associate specific types of product features with checklist items, and the inspector will not be able to mark a checklist item until **InspeQ** has verified that he has examined every feature of the types associated with the checklist item. For example, if a checklist item requires an inspector to check that every while statement in a program terminates, **InspeQ** will ensure that every while statement was at least examined in the highlight display.

6. PRELIMINARY EVALUATION

Phased Inspections were developed to create a rigorous and reliable review method for software work products. We expect Phased Inspections to reduce the cost and effort of some other stages of development also. For example, both system testing effort and maintenance effort might be reduced by Phased Inspections of requirements, designs, and code. It is not sufficient, however, to claim these benefits based purely on the insight (fantasy?) of the developers of the method; a systematic evaluation is required to determine whether Phased Inspections fulfill these expectations. Fundamentally, an evaluation has to answer the most important question: "Are Phased Inspections cost effective?" No matter how reliable or rigorous Phased Inspections are, if they are not cost effective, they will not be used.

In this section, we outline an evaluation framework for Phased Inspections and report the results of a small evaluation experiment. The evaluation framework and the experiment are described in detail elsewhere [9]. The framework breaks the problem of evaluation down into five areas; feasibility, performance, resources used, consistency achieved, and utility of computer support. The concerns in these five areas are as follows:

Feasibility:

(1) Are Phased Inspections feasible in the sense of being a workable process?

(2) Can checklists be developed routinely that have the desired characteristics?

(3) Does the computer support toolset actually assist inspectors in performing Phased Inspections?

Performance:

(1) Does the performance achieved depend on the particular type of work product? For example, are Phased Inspections more effective on source code than test plans or requirements specifications?

(2) Does the performance achieved depend on the application domain? For example, are Phased Inspections more effective on work products developed for real-time systems than database applications?

(3) Does the notation in which the work product is written affect performance? For example, are Phased Inspections of source programs more useful on programs written in 'C' than those written in 'Ada'? One might expect so given the difference in philosophy of the two languages.

(4) Does the performance achieved depend on the experience and specific skills of the inspectors?

Resources:

(1) Can inspectors with lessor skills be used successfully in phases involving only simple checks, and does this produce the expected savings in resources?

(2) What resources are required to build comprehensive checklists?

(3) How long do inspections take and what is the variance in inspection times? Does the time taken depend on inspectors skills and background?

(4) What level of effort is needed to manage Phased Inspections?

(5) Does Phased Inspection of one work product save resources in the preparation of other work products?

Consistency:

(1) Do different groups of inspectors implementing the same instantiation of Phased Inspection on the same work products consistently achieve the same results?

(2) Can phases depend on previous phases?

(3) Does a Phased Inspection permit useful conclusions to be drawn about a specific work product after inspection as desired?

Computer Support:

(1) Does computer support reduce the resources needed to perform an inspection?

(2) Does computer support improve the rigor or quality of the inspection process or the workproducts being inspected?

(3) To what extent can compliance checking be made complete and does it help?

(4) What are the right tools and interfaces for a computer toolset?

Each of these questions needs to be answered in order to be able to assess the concept of Phased Inspections. For the most part, the questions can only be answered by experimentation, and, in order for the conclusions drawn to be statistically meaningful, there need to controls for the experiments and sufficient replicates to permit estimation of the variance of the quantities measured.

Unfortunately, the full-scale experimentation required is unlikely to be supported by any organization unless there is good reason to believe that the outcome will be favorable. Such experimentation is not feasible in an academic environment. This does not mean, however, that experiments with Phased Inspections should not be conducted. Quite the contrary, constrained experiments might not produce conclusive results, but they might provide good indications of the relative utility of Phased Inspections. The purpose of developing the evaluation framework was to define the way in which the long-term process of experimentation in which the authors are engaged might proceed so as to evaluate Phased Inspections thoroughly.

A limited experimental evaluation of Phased Inspections has been performed using graduate students in the Computer Science Department of the University of Virginia and a faculty member from the College of William and Mary. The Phased Inspection used in the experiment was a general instantiation for software source code written in 'C'. The phases are summarized above in Table 1. The target of the inspection was part of the source code of the prototype toolset, **InspeQ**. The evaluation experiment was conducted in two parts. The first part evaluated single-inspector phases; the second evaluated multiple-inspector phases.

Four inspectors working individually performed phases 1 through 4. Three of the inspectors had previous industrial software-development experience and experience programming

in 'C'. One inspector did not know 'C' and had no industrial experience, and one inspector had previous experience with software review methods. The inspectors' backgrounds, their allocation to phases, and the times taken for the individual phases are summarized in Table 2.

Phases 1 through 4 were applied to the **InspeQ** source code that implements the work product display of the toolset. This code was chosen because of its relative simplicity. It is 643 lines long including comments and implements relatively simple algorithms. Comments are included in the number of lines inspected since comments are not excluded from inspection.

Two groups each consisting of 2 inspectors participated in the evaluation of the multiple-inspector phase. Both groups inspected the *same* product. The purpose of replication was to try to get an indication of the degree of variance in performance. In Group 1, both of the inspectors had extensive experience with 'C', and one of the inspectors had prior experience in industry and with inspections. In Group 2, neither of the inspectors had much experience with 'C'. One of the inspectors had industrial experience and previous experience with inspections. The inspectors' backgrounds, their allocation to phases, and the times taken for the individual phases are summarized in Table 3.

In this part of the experiment, phase 6 was applied to the toolset source code that implements file operations. This source code is 1015 lines long including comments. Since phase 6 is a multiple-inspector phase, it includes an examination of the reference documentation prior to the parallel individual inspections performed by each member of the group, and it includes a reconciliation step following the individual inspections. The documentation inspection for each group took about 1 hour to complete. The average time for individual inspection was 2 hours, and the reconciliations for both groups took 1 hour.

The first conclusion that we can draw from this experiment is that the whole process is feasible. The reactions of the inspectors was very favorable and the time expended was reasonable. For the most part, inspectors performing the single-inspector phases were able to decide whether the product complied with the checklist items of a phase. The inspectors felt, in general, that **InspeQ** was understandable and easy to use, and that the computer support provided by **InspeQ** was useful. None of the inspectors required more than 2 hours to complete their

	Inspector			
	A	B	C	D
'C' Experience	None	Extensive	Extensive	Extensive
Industrial Experience	None	Extensive	Extensive	Extensive
Inspection Experience	None	None	None	Some
Phase Performed	Phase 1	Phase 2	Phase 3	Phase 4
Time Required	1.5 hrs	1.45 hrs	1 hr	1.5 hr

Table 2 - Single-Inspector Experiment

	Inspector			
	Group 1		Group2	
	E	F	G	H
'C' Experience	Extensive	Extensive	Some	None
Industrial Experience	Extensive	None	None	Extensive
Inspection Experience	Extensive	None	None	Extensive
Time Required	1.5 hrs	1.25 hrs	1 hr	2 hr

Table 3 - Multiple-Inspector Experiment

assigned phase. Inspector B also noted that as he became more familiar with the checklist items associated with the phase he was conducting, his rate of progress improved rapidly.

All of the inspectors who participated in the single-inspector phase experiment indicated the highlight and file comments features were convenient and useful. Inspectors A and B were of the opinion that the checklist display did not provide adequate options for marking checklist items and this has subsequently been improved.

In the single-inspector phase experiment, inspector A, who had very little knowledge of 'C', had a more difficult time inspecting the product than the inspectors who had experience with 'C'. Inspector A felt that his inspection could have been more effective if the checklist items had been more explicit. An inspector with previous experience with the programming language or inspections might be able to augment his understanding of poorly worded checklist items with that experience. The more explicit, understandable, and unambiguous the checklist items, the more likely that the results of a phase are the same regardless of the skills and background of the inspector conducting the inspection.

Our second conclusion from this experiment is that the checklists for this particular inspection instantiation were developed very easily. The sources for the phases and the checklists were published texts on programming style [6, 7, 8, 13], and the personal experience of the authors with the types of errors typically found in 'C' source code. On the other hand, an instantiation of Phased Inspections being produced to check for reusability of Ada parts is proving more difficult to develop. An extensive literature search failed to produce more than a few specific checks and programming practices that would promote reusability in Ada source code. Indicating, for example, that information hiding and abstract data types should be used in reusable software is not a sufficient basis for checklist items in a Phased Inspection.

Our third set of conclusions is in the context of multiple-inspector phases. There was a lack of computer support for the multiple-inspector phase part of the experiment. The only substantial facilities available in the toolset for use with a multiple-inspector phase are the work product display and the comments display. The product reference documentation had to be distributed to the inspectors in printed form because there were no facilities for displaying it in the toolset. The inspectors felt that it was difficult and time consuming not to have access to a display of the

reference documentation and such a facility will be added.

A knowledge of the programming language affected results in both parts of the experiment, but more so in the multiple-inspector phase evaluation. The inspectors in Group 1 with similar and extensive backgrounds in 'C' detected similar errors. Where they did not detect similar errors, the inspectors quickly came to an agreement over whether there was actually an error during the reconciliation. In Group 2, the inspectors had little experience with 'C' and produced greatly dissimilar error lists. They were less successful than Group 1 at detecting defects in the source code, and tended to focus on detecting defects in the design of the product rather than the implementation.

Past experience, either with inspections or with the programming language affected performance. This indicates a need for more direction in the multiple-inspector phases, but it is also possible that inspectors who do not have sufficient experience should not be included in multiple-inspector phases.

Finally, we note that this evaluation was too limited to assess conclusively the contribution of **InspeQ** to the Phased Inspections process. It did, however, uncover shortcomings of the first version of **InspeQ** that have subsequently been corrected. The shortcomings included a poor interface to the search mechanism in the work product display, limited checklist marking facilities, and restricted ability to copy material from one **InspeQ** display to another.

7. CONCLUSION

We believe that inspection is one of the most valuable tools that the software engineer has available but that the technology is not being exploited to its full potential. We have defined an enhanced inspection technique called Phased Inspection that addresses the deficiencies of existing inspection techniques. The most important goal of Phased Inspection is rigor so that engineers can trust the results of a specific inspection and so that inspection results are repeatable. We have also presented details of a toolset that supports Phased Inspection by providing the inspector with as much computer assistance as possible and by checking for compliance with the required process of Phased Inspection. A preliminary evaluation of Phased Inspection was very encouraging, and an evaluation of its utility in determining reusability of Ada software parts is presently underway.

8. ACKNOWLEDGEMENTS

It is a pleasure to acknowledge Ran Atkinson, Darrell Kienzle, James Leatherby, Keith Miller, Steven Santos, Kristian Simsarian, Edward Smierciak, and Sudhir Srinivasan, who volunteered for the evaluation experiment and spent many hours of their own time learning about Phased Inspections, the toolset, and performing the inspections while being monitored. This work was funded in part by NASA under grant number NAG-1-1073, in part by SAIC Inc., and in part by the MITRE Corporation.

REFERENCES

(1) Fagan, M.E., "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, Vol. 15, No. 3, 1976.

(2) Fagan, M.E., "Advances in Software Inspections", *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7, July, 1986.

(3) Fagan, M.E. and J.C. Knight, "Testing is Not the Best Means of Defect Detection and Removal", Achieving Quality Software - A National Debate, Society for Software Quality, San Diego, CA, January, 1991.

(4) Freedman, D.P. and G.M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Little, Brown and Company, Boston, Toronto, 1982.

(5) Freedman, D.P. and G.M. Weinberg, "Reviews, Walkthroughs, and Inspections", *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 1, January, 1984.

(6) Kernighan, B.W. and P.J. Plauger, *The Elements of Programming Style*, Second Edition, McGraw Hill, New York, 1978.

(7) Kernighan, B.W. and D. M. Ritchie, *The C Programming Language*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

(8) Linger, R.C., H.D. Mills, and B.I. Witt, *Structured Programming : Theory and Practice*, Addison-Wesley, Reading, MA., 1979.

(9) Myers, E.A., *Phased Inspections and Their Implementation*, Thesis of Univ. of VA, May 1991.

(10) Parnas, D.W., D.M. Weiss, "Active Design Reviews: Principles and Practices", Proceedings of *Eighth International Conference on Software Engineering*, London, England, August, 1985.

(11) Petroski, H., *To Engineer Is Human: The Role of Failure in Successful Design*, St. Martin's Press, New York, 1985.

(12) Russell, G.W., "Experience with Inspection in Ultralarge-Scale Developments", *IEEE Software*, Vol. 8, No. 1, January 1991.

(13) Software Productivity Consortium, *Ada Quality and Style: Guidelines For Professional Programmers*, Van Vostrand Reinhold, New York, 1989.

(14) Weinberg, G.M., *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York, 1971.