



NORTH-HOLLAND

Evolving and Packaging Reading Technologies

Victor R. Basili

*Department of Computer Science and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD*

Reading is a fundamental technology for achieving quality software. This paper provides a motivation for reading as a quality improvement technology, based upon experiences in the Software Engineering Laboratory at NASA Goddard Space Flight Center, and shows the evolution of our study of reading via a series of experiments. The experiments range from early reading vs. testing experiments to various Cleanroom experiments that employed reading to the development of new reading technologies currently under study.
© 1997 by Elsevier Science Inc.

1. INTRODUCTION

Reading is a fundamental technology for achieving quality software. It is the only analysis technology we can use throughout the entire life cycle of the software development and maintenance processes. And yet, very little attention has been paid to the technologies that underlie the reading of software documents. For example, where is software reading taught? What technologies have been developed for software reading? In fact, what is software reading?

During most of our lives, we learned to read before we learned to write. Reading formed a model for writing. This was true from our first learning of a language (reading precedes writing and provides simple models for writing) to our study of the great literature (reading provides us with models of how to write well). Yet, in the software domain, we never learned to read, e.g., we learn to write programs in a programming language, but never how to read them.

We have not developed reading-based models for writing. For example, we are not conscious of our audience when we write a requirements document. How will they read it? What is the difference between reading a requirements document and reading a code document? We all know that one reads a novel differently than one reads a text book. We

know that we review a technical paper differently than we review a newspaper article. But how do we read a requirements document, a code document, or a test plan? There are many factors that affect the way we read.

Let us define some terms so that we understand what we mean by reading. We differentiate a technique from a method, from a life cycle model. A technique is the most primitive. It is an algorithm, a series of steps producing the desired effect, and requires skill. A method is a management procedure for applying techniques, organized by a set of rules stating how and when to apply and when to stop applying the technique (entry and exit criteria), when the technique is appropriate, and how to evaluate it. We will define a technology as a collection of techniques and methods. A life cycle model is a set of methods that covers the entire life cycle of a software product.

For example, reading by step-wise abstraction (Linger, et al. 1979) is a technique for assessing code. Reading by step-wise abstraction requires the development of personal skills; one gets better with practice. A code inspection is a method that is defined around a reading technique, which has a well defined set of entry and exit criteria and a set of management supports specifying how and when to use the technique. Reading by stepwise abstraction and code inspections together form a technology. Inspections are embedded in a life cycle model, such as the Cleanroom development approach, which is highly dependent on reading techniques and methods. That is, reading technology is fundamental to Cleanroom development.

In what follows, we will discuss the evolution and packaging of reading as a technology in the Software Engineering Laboratory (SEL) (Basili, et al. 1992; Basili, et al. 1994) via a series of experiments from some early reading vs. testing technique experiments, to various Cleanroom experiments, to the development of new reading techniques currently under study.

Address correspondence to Victor R. Basili, Department of Computer Science / Institute for Advanced Computer Studies, University of Maryland, AV Williams Building 115, College Park, MD 20742

In the SEL, we have been working with a set of experimental learning approaches: the Quality Improvement Paradigm, the Goal Question Metric Paradigm, the Experience Factory Organization, and various experimental frameworks to evolve our knowledge and the effectiveness of various life cycle models, methods, techniques, and tools (Basili, 1985; Basili and Weiss 1984; Basili and Rombach 1988; Basili 1989). All of these approaches have been applied to the series of experiments we've conducted at the University of Maryland and at NASA to learn about, evaluate, and evolve reading as a technology.

2. READING STUDIES

Figure 1 provides a characterization of various types of experiments we have run in the SEL. They define different scopes of evaluation representing different levels of confidence in the results. They are characterized by the number of teams replicating each project and the number of different projects analyzed yielding four different experimental treatments: blocked subject-project, replicated project, multi-project variation, and single project case study.

The approaches vary in cost, level of confidence in the results, insights gained, and the balance between quantitative and qualitative research methods. Clearly, an analysis of several replicated projects costs more money but provides a better basis for quantitative analysis and can generate stronger statistical confidence in the conclusions. Unfortunately, since a blocked subject-project experiment is so expensive, the projects studied tend to be small. To increase the size of the projects, keep the costs reasonable, and allow us to better simulate the effects of the treatment variables in a realistic environment, we can study very large single project case studies and even multi-project studies if the right environment can be found. These larger projects

tend to involve more qualitative analysis along with some more primitive quantitative analysis.

Because of the desire for statistical confidence in the results, the problems with scale up, and the need to test in a realistic environment, one approach to experimentation is to choose one of the multiple team treatments (a controlled experiment) to demonstrate feasibility (statistical significance) in the small project, and then to try a case study or multi-project variation to analyze whether the results scale up in a realistic environment—a major problem in studying the effects of techniques, methods and life cycle models.

2.1 Reading by Step-wise Abstraction

In order to improve the quality of our software products at NASA, we have studied various approaches. One area of interest was to understand the relationship between reading and testing in our environment. Early experiments showed very little difference between reading and testing (Hetzel 1972; Myers 1978). But reading in these studies was simply reading, without a technological base. Thus we attempted to study the differences between various specific technology based approaches. Our goal was to analyze *code reading*, *functional testing* and *structural testing* to evaluate and compare them with respect to their *effect on fault detection effectiveness*, *fault detection cost* and *classes of faults detected* from the viewpoint of the researchers (Basili and Selby 1987). The study was conducted in the SEL, using three different programs: a text formatter, a plotter, and a small database. The programs were seeded with software faults, (9, 6, and 12 faults respectively), and ranged in size from 145 to 365 LOC. The experimental design was a blocked subject-project, using a fractional factorial design. There were 32 subjects.

Scopes of Evaluation

		# Projects	
		One	More than one
# of Teams per Project	One	Single Project	Multi-Project Variation
	More than one	Replicated Project	Blocked Subject-Project

Figure 1. Classes of studies.

Specific techniques were used for each of the three approaches studied. Code reading was done by step-wise abstraction, i.e., reading a sequence of statements and abstracting the function they compute and repeating the process until the function of the entire program has been abstracted and can be compared with the specification. Functional testing was performed using boundary value, equivalence partition testing, i.e., dividing the requirements into valid and invalid equivalence classes and making up tests that check the boundaries of the classes. Structural testing was performed to achieve 100% statement coverage, i.e., making up a set of tests to guarantee that 100% of the statements in the program have been executed.

As a blocked subject-project study, each subject used each technique and tested each program. The results were that code reading found more faults than functional testing, and functional testing found more faults than structural testing. Also, code reading found more faults per unit of time spent than either of the other two techniques. Different techniques seemed to be more effective for different classes of faults. For example, code reading was more effective for interface faults and functional testing more effective for control flow faults.

A second set of conclusions, based upon the perception of the readers and testers, was that code readers were better able to assess the actual quality of the code that they analyzed than the testers. And in fact, the structural testers were better able to assess the actual quality of the code they analyzed than the functional testers. That is, the code readers felt they only found about half the faults (and they were right), where the functional testers felt that had found about all the faults (and they were wrong). Also, after the completion of the study, over 90% of the participants thought functional testing worked best. This was a case where perception or intuition was clearly wrong.

Based upon this study, reading was implemented as part of the SEL development process. However, much to our surprise, reading appeared to have very little effect on reducing defects. It should be noted that the SEL keeps baselines of defect rates for project sets. This leads us to two possible hypotheses:

Hypothesis 1: People did not read as well as they should have because they believed that testing would make up for their mistakes.

To test this first hypothesis, we ran an experiment that showed that if a developer reads and cannot

test they do a more effective job of reading than if they read and know they can test later. This supported hypothesis 1.

Hypothesis 2: There is a confusion between reading as a technique and the method in which it is embedded, e.g., inspections.

This addresses the concern that we often use a reading method (e.g., inspections or walk-through) but do not often have a reading technique (e.g., reading by step-wise abstraction) sufficiently defined within the method. To some extent, this might explain the success of reading in this experiment (Basili and Selby 1987) over the studies by Hetzel (Hetzel 1972) and Myers (Myers 1978).

Thus we derived the following conclusions from the studies described thus far:

- Reading using a particular technique is more effective and more cost effective than specific testing techniques, i.e., the reading technique is important. However, different approaches may be effective for different types of defects.
- Readers need to be motivated to read better, i.e., the ability to read a document effectively seems to be related to the readers' belief that their reading of the document is important.
- We may need to better support the reading process, i.e., the reading technique may be different from the reading method.

2.2 The Cleanroom Approach

The Cleanroom approach, as proposed by Harlan Mills (Currit, et al., 1986) addressed the above issues by providing a particular reading technique (step-wise abstraction) and a motivation for reading (the developer cannot test). To study the effects of the approach and reduce the risk of applying it in the SEL, we ran a controlled experiment at the University of Maryland.

The goal of this study was to analyze the *Cleanroom process* in order to *evaluate and compare* it to a non-Cleanroom process with respect to the *effects on the process, product and developers* from the point of view of the *researchers* (Selby, et al., 1987). This study was conducted using upper division and graduate students at the University of Maryland. The problem studied was an electronic message system of about 1500 LOC. The experimental design was a replicated project using 15 three-person teams (10 used Cleanroom). They were allowed 3 to 5 test submissions to an independent tester. We collected

data on the participants' background, attitudes, on-line activities, and testing results.

The major results were:

- With regard to process, the Cleanroom developers (1) felt they more effectively applied off-line review techniques, while others focused on functional testing, (2) spent less time on-line and used fewer computer resources, and (3) tended to make all their scheduled deliveries.
- With regard to the delivered product, the Cleanroom products tended to have the following static properties: less dense complexity, higher percentage of assignment statements, more global data, more comments; and the following operational properties: the products more completely met the requirements and a higher percentage of test cases succeeded.
- With regard to the effect on the developers, most Cleanroom developers missed program execution, modified their development style, but said they would use the Cleanroom approach again.

2.3 Cleanroom in the SEL

Based upon this success, we decided to try the Cleanroom approach in the SEL (Basili and Green, 1994). The study goal was to analyze the *Cleanroom process* in order to *evaluate and compare it to the standard SEL development process* with respect to the *effects on the effort distribution, cost, and reliability* from the point of view of the *SEL organization*. This was the basis for a single-project case study in which Cleanroom was applied to a 40 KLOC ground

support system. To evaluate and integrate Cleanroom into the SEL, we used the Quality Improvement Paradigm to set up our learning process. We define the six steps of the QIP as they apply to the introduction of Cleanroom into the SEL:

Characterize: Describe the product and its environment. For example, what are the relevant models, baselines and measures, what are the existing processes, what is the standard cost, relative effort for activities, reliability, what are the high risk areas? (See the sample measures and baselines in Figure 2).

Set goals: Define the goals to be achieved. For example, what are the expectations, relative to the baselines, what do we hope to learn or gain, how will Cleanroom perform with respect to changing requirements? (See the sample expectations in Figure 2).

Choose process: Select the best mix of methods and techniques to achieve the goals relative to the environment. That is, how should the Cleanroom process be modified and tailored relative to the environment? For example, formal methods are hard to apply and require skill; we may have insufficient data to measure reliability; therefore, we might allow back-out options for unit testing certain modules.

Execute: Collect and analyze data based upon the goals, making changes to the process in real time.

Analyze: Try to characterize and understand what happened relative to the goals; write lessons learned.


	Sample Measures	Sample Baseline	Sample Expectation
PROCESS	Effort distribution Change profile		Increased design % due to emphasis on peer review process
	Productivity Level of rework Impact of spec changes	Historically, 26 DLOC per day	No degradation from current level
COST	Error rate Error distribution Error source	Historically, 7 errors per KDLOC	Decreased error rate

Figure 2. Sample measures, baselines, and expectations.

Package: Modify the process for future use.

There were many lessons learned during this first application of the Cleanroom approach in the SEL. However, the most relevant to reading were that the failure rate during test was reduced by 25% and productivity increased by about 30%, mostly due to fact that there was a reduction in the rework effort, i.e., 95% as opposed to 58% of the faults took less than 1 hour to fix. About 50% of code time was spent reading, as opposed to the normal 10%. All code was read by 2 developers. However, even though the developers were taught reading by step-wise abstraction for coding reading, only 26% of the faults were found by both readers. This implied to us that the reading technique was not applied as effectively as it should have been, as we expected a more consistent reading result.

During this case study, problems, as specified by the users, were recorded and the process was modified in real time. As well, notes were made as to how to improve the process for its next application. For example, better training and skill development was needed for the methods and techniques, better mechanisms were needed to upload the code to the testers and testers needed to be able to add requirements to help them analyze output.

Based upon the success of the first Cleanroom case study, we began to define new studies with the goal of applying the reading technique more effectively. A second and third Cleanroom project were initiated. Changes to the process involved better training, a solution to the uploading problem, and allowing testers to add requirements. The project leaders for the first project became process modelers for the next two and we began to generate the evolved version of the SEL Cleanroom Process Model. Thus, experimentally, we moved from a case study to a multi-project analysis study.

Figure 3 gives an overview of the projects studied to date. Figure 4 gives the effects of Cleanroom on error rate and productivity. Like the first Cleanroom project, the second was done in-house at NASA, and was successful with regard to reducing error rate but was not as productive as the first. The third project was done totally by the contractor. It appeared to be less successful on both counts, partly because it was our first experience with a project of that size (160 KLOC) and partly because it was done off site with less access to support. Based upon these projects, other modifications were made to the method, e.g., allowing a clean compile before reading.

A fourth Cleanroom project was recently completed. Again, like the third, it was large and totally developed by the contractor. As can be seen in Figure 4, the results here were very positive.

Cleanroom has been successful in the SEL. Although there is still room for improvement in reading and abstracting code formally, a more major concern is the lack of techniques for reading documents other than code, e.g., requirements, design, test plans.

This has generated a motivation for the continual evolution of reading techniques in the SEL, both inside and outside the Cleanroom life cycle model. Specific emphasis is on improving reading technology for requirements and design documents.

2.4 Scenario-Based Reading

The experiments described above convinced us that reading is a key, if not the key technical activity for verifying and validating software work products. However, there has been little research focus on the development of reading techniques, with the possible exception of reading by step-wise abstraction, as developed by Harlan Mills.

	1st Cleanroom	2nd Cleanroom	3rd Cleanroom	4th Cleanroom
Project	ACME	SAMPEX	WINDPOLAR	SOHO AGSS
Size (developed lines)	40 KSLOC	23 KSLOC	160 KSLOC	141 ICSLOC
Size (total lines)	66 KLOC	39 KLOC	201 KLOC	485 KLOC
Dates	1988-90	1990-91	1990-92	1993-1995
Function	Attitude determination (Math)	Telemetry processing (data processing)	Full attitude (two missions)	Full attitude

Figure 3. Multi-project analysis study of cleanroom in the SEL.

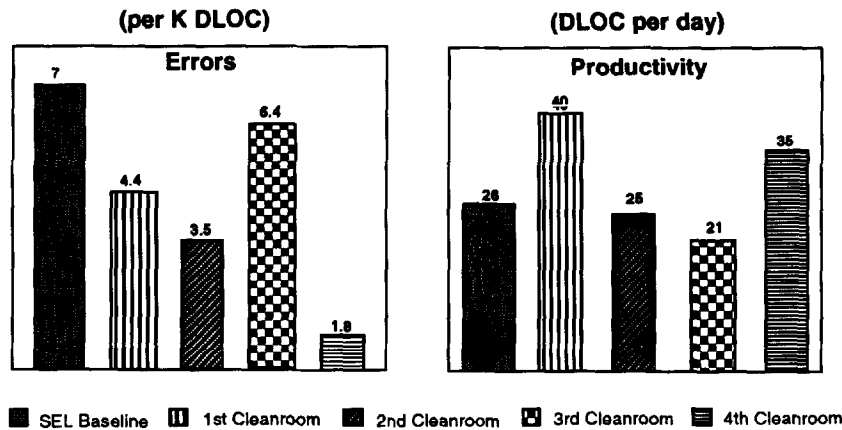


Figure 4. Effects of Cleanroom on error rate and productivity.

The ultimate goal here is to understand the best way to read for a particular set of conditions. That is, we are not only interested in how to develop techniques for reading such documents as requirements documents, but under what conditions are each of the techniques most effective and how might they be combined in a method, such as inspections, to provide a more effective reading technology for the particular problem and environment.

The idea is to provide a flexible framework for defining the reading technology so that the definer of the technology for a particular project has the appropriate information for selecting the right techniques and method characteristics. Thus, the process definition may change depending on the project characteristics. For example, if a higher number of omission faults are expected, we might emphasize a traceability reading approach embedded in design inspections; when embedding traceability reading in design inspections, we might make sure a traceability matrix exists.

As stated in the introduction, we believe there are many factors that affect the way a person reads, e.g., the reviewer's role, the reading goals, the work product. Based upon these studies, we also believe that techniques can be developed that will allow us to better define how we should read, and that using these techniques, effectively embedded in the appropriate methods, can improve the effects of reading. For example, reading techniques for end-users reading a software requirements document should be different than the reading techniques for software testers reading a requirements document; reading techniques for developers reading for interface faults should be different than reading techniques for developers reading for missing initialization. Also, if we know that reading by step-wise refinement is

more effective for interface faults, and, based upon past history, we anticipate a large number of interface faults for a particular project, then we can assign more than one reader to use step-wise abstraction reading in our inspection team.

Thus we need to improve the reading of all kinds of documents from various points of view. To do this, we need to more deeply understand the relationship between techniques and methods and the dimensions of both. That is, what are the things we can vary when dealing with a technique? For example, consider the following dimensions of a reading technique:

Input object: any document, e.g., requirements, design, code, test plan, etc.

Output object: a set of defects or anomalies

Technique: some specific procedure, e.g., sequential reading, path analysis, step-wise abstraction, etc.

Formality: the degree of rigor, e.g., proof, correctness demonstration, etc.

Goals: the purpose for reading, e.g., fault detection, traceability, performance, understanding reuse, etc.

Method: the method the technique is embedded in, e.g., walk-through, inspections, reviews, etc.

Perspective: the role of the reader, e.g., user, designer, tester, maintainer, etc.

Context: anticipated problems, application domain, organization, etc.

Product qualities: correctness, reliability, efficiency, portability, etc.

Process qualities: process conformance, integration with other processes, etc.

When defining a technique, what are the values of the various dimensions? We have been developing

and studying reading techniques that take into account the various dimensions, as well as the historical data of the environment where the technique will be applied. The goal is to define a set of reading techniques that can be tailored to the document being read and the goals of the organization for that document, and that are usable in existing methods, such as inspections or reviews.

To this end, we have been working on an approach to generating families of reading techniques, based upon the values of different dimensional attributes. At the top level, each family of techniques is based upon combining two primary dimensions, e.g., the goal and the perspective, to generate a procedure, or operational scenario (Figure 5). The operational scenario requires the reader to (1) create an abstraction (based on a model building or abstraction dimension) of the product, and (2) answer questions (based on an analysis dimension) while building that abstraction. Each reading technique in the family can be based upon a different abstraction and question set.

Each family (and thus each technique) is tailored based upon other dimensions as well, e.g., the input dimension, the context dimension. So, based upon the input dimension, a family of techniques can be instantiated for a particular document (e.g., requirements, design) and notation (e.g., English text, a formal notation) in which the document is written. Based upon the context dimension, a family of techniques can be tailored to react appropriately to the project and environment characteristics. The choice of primary, and secondary dimensions, as well as abstractions and the types of questions asked depend on the organization's needs and concerns.

Thus each technique within the family is (1) tailorable, based upon the values of various dimensions, (2) detailed, in that it provides the reader a well-defined set of steps to follow, (3) specific, in

that the reader has a particular purpose or goal for reading the document and the procedures support that goal, (4) focused, in that it provides a particular coverage of the document, and a combination of techniques in the family provides coverage of the entire document, (5) studied empirically to determine if and when it is most effective.

So far, two different families of reading techniques have been defined for requirements documents: defect-based reading and perspective-based reading.

Perspective-based reading focuses on different product customer perspectives, e.g., reading from the perspective of the software designer, the tester, the end-user, the maintainer, the hardware engineer, representing the perspective dimension. The analysis questions were generated by focusing predominantly on various requirements type errors, e.g., incorrect fact, omission, ambiguity, and inconsistency (Basili and Weiss 1981), representing the goal dimension.

Defect-based reading focuses on a model of the data and functions of the requirements in a form of state machine notation. The different model views were based upon focusing on a variation of the defect classes given above: data type inconsistency, incorrect functions, an ambiguity or missing information, representing the goal dimension. The analysis questions were generated by combining/abstracting a set of questions that were used in checklists for evaluating the correctness and reliability of requirements documents, representing an existing technique dimension.

To provide a little more detail into the approach for generating reading techniques, consider the following example of the generation of test-based reading, one member of the family of perspective-based reading. The object is the requirements document, the model-base is a testing technique, (e.g., equiva-

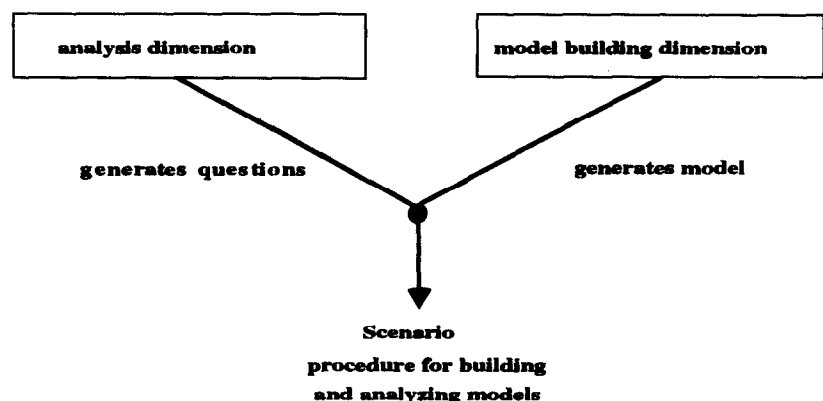


Figure 5. Building focused tailored reading techniques.

lence partitioning, boundary-value testing), and the analysis dimensions are the correctness, completeness, consistency, and unambiguity of the requirements.

The operational scenario of reading procedure is defined as follows: for each requirement, make up a test or set of tests that will allow you to ensure that the implementation satisfies the requirement. Use equivalence partitioning, boundary-value testing criteria to make up the test suite.

The second dimension is based upon defect classes, specifically incorrect fact, omission, ambiguity, and inconsistency. These generated the following questions, which the reader should ask while building the test plan model:

- (a) Do I have all the information necessary to divide the requirement into a valid equivalence class and invalid equivalence classes? Can I make up reasonable test cases for each based upon the criteria?
- (b) Can I be sure that the test I generated will yield the correct value in the correct units?
- (c) Does the requirement make sense from what I know about the application and from what is specified in the overview?
- (d) Are there other interpretations of this requirement that the implementor might make based upon the way the requirement is defined?
- (e) Is there another requirement for which the equivalence class is defined differently, i.e., in which the test case you generate should give a contradictory response for the other equivalence class?

The model for developer-based reading might be to perform a high level design using structured analysis or object oriented design. The model for the use-based reading might be to develop a user's manual. Although in each case the questions are derived from trying to identify omission, incorrect facts, etc., the opportunities for such discoveries, and thus the questions, will vary, depending on the model used.

Specific members of each of the families have been studied experimentally. In the defect-based reading study, the goal was to analyze *defect-based reading*, *ad hoc reading* and *checklist-based reading* in order to *evaluate and compare* them with respect to their *effect on fault detection effectiveness in the context of an inspection team* from the viewpoint of the *researcher*. The three defect-based reading techniques stated above were applied. The study was applied using graduate students at the University of Maryland. The requirements documents were written in the SCR notation (Henninger 1980). They

were a Water Level Monitoring System and a Cruise Control System. The experimental design is a blocked subject-project: Partial factorial design, replicated twice with a total of 48 subjects (Porter, et al., 1995).

Major results were that (1) the defect-based readers performed better than ad hoc and checklist readers with an improvement in defect detection rate of about 35%, (2) the defect-based reading procedures helped reviewers focus on specific fault classes but were no less effective at detecting other faults, and (3) checklist reading was no more effective than ad hoc reading.

In the perspective-based reading study, the goal was to analyze *perspective-based reading* and *NASA's current reading technique* in order to *evaluate and compare* them with respect to their *effect on fault detection effectiveness in the context of an inspection team* from the viewpoint of the *researcher and the SEL*. Three perspective-based reading techniques (test-based, developer-based, and use-based reading) were defined and studied. Studies have been performed in the SEL environment using generic requirements documents written in English (ATM machine, Parking Garage) and NASA type functional specifications (two ground support AGSS sub-systems). The experimental design is again a blocked subject-project using a partial factorial design. It has been applied twice, with a total of 25 subjects (Basili, et al., 1996).

Major results are that perspective-based reading (1) is effective for generic documents both at the individual and team level, i.e., taking each technique in the family individually as compared with the standard approach, and combining the three perspectives for full coverage against a team of standard readers, (2) catches different types of defects depending on the perspective, (3) is effective for the NASA documents at the team level. It was felt that the techniques could be better tailored for the NASA style document to improve individual scores.

We will continue to evolve and study various families and various techniques within the families. The first series of experiments described above is aimed at discovering if scenario-based reading is more effective than current practices. Early results are promising. A second series will be used to discover under which circumstances each of the various scenario-based reading techniques, or families of techniques, is most effective.

We hope to replicate these experiments in different environments, replacing the NASA documents with documents from other organizations. We also hope to run a case study at NASA to better understand how to tailor the techniques to the documents.

Scopes of Evaluation

		# Projects	
		One	More than one
# of Teams	One	3. Cleanroom (SEL Project 1)	4. Cleanroom (SEL Projects, 2,3,4,...)
	More than one	2. Cleanroom at Maryland	1. Reading vs. Testing 5. Scenario reading vs. ...

Figure 6. Series of studies.

We will continue to develop operational scenarios for other document types, e.g., design document, and test their effectiveness in experiments. We will eventually consider tool support for the techniques developed.

3. CONCLUSION

In our attempt to better understand the effects of software reading techniques, we have run the experimental gamut from blocked subject-project experiments (reading vs. testing) to replicated projects (University of Maryland Cleanroom study) to a case study (the first SEL Cleanroom study) to multi-project variation (the set of SEL Cleanroom projects) and now back to blocked subject-project experiments (for scenario-based reading). (See Figure 6).

As we learn, as we move through each cycle of the Quality Improvement Paradigm, the level of sophistication of our reading goals is maturing. Our ability to understand things about reading is evolving. A pattern of knowledge is being built from a series of experiments.

Various groups at different sites are already replicating some of the experiments. Most of these are members of ISERN, the International Software Engineering Research Network, whose goal is specifically to perform and share the results of empirical studies.

ACKNOWLEDGMENTS

I would like to thank Forrest Shull and Carolyn Seaman for reviewing the drafts of this paper. This work was supported in part by NASA grant NSG-5123, UMIACS, NSF grant 01-5-24845.

REFERENCES

Basili, V. R., Green, S., Laitenberger, O. U., Lanubile, F., Shull, F. Sorumgaard, S. and Zelkowitz, The Empirical

Investigation of Perspective-Based Reading. *Journal of Empirical Software Engineering*, Volume 1, Issue 2 (1996).
 Basili, V. R. and Green, S., Software Process Evolution at the SEL. *IEEE Software*, pp. 58-66 (1994).
 Basili, V. R., Zelkowitz, M. V., McGarry, F., Pajerski, R., Page, J., Waligora, S., SEL'S Software Process-Improvement Program. *IEEE Software*, pp. 83-87 (1994).
 Basili, V. R., Zelkowitz, M. V., McGarry, F., Pajerski, R., Page, J., Waligora, S., SEL'S Software Process-Improvement Program. *IEEE Software*, pp. 83-87 (1994).
 Basili, V. R., Caldiera, G., McGarry, F., Pajerski, R., Page, G., Waligora, S., The Software Engineering Laboratory —An Operational Software Experience Factory, 14th International Conference on Software Engineering (1992).
 Basili, V. R., Software Development: A Paradigm for the Future, COMPSAC '89, Orlando, Florida, pp. 471-485 (1989).
 Basili, V. R., and Rombach, H. D., The TAME Project: Towards Improvement-Oriented Software Environment. *IEEE Transactions on Software Engineering*, vol. 14, no. 6 (1988).
 Basili, V. R., and Selby, R., Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering*, pp. 1278-1296 (1987).
 Basili, V. R., Quantitative Evaluation of Software Methodology, Keynote Address, First Pan Pacific Computer Conference, Melbourne, Australia (1985).
 Basili, V. R. and Weiss, D. M., A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, pp. 728-738 (1984).
 Basili, V. R., Weiss, D. M., Evaluation of a Software Requirements Document by Analysis of Change Data, Proceedings of the Fifth International Conference on Software Engineering, pp. 314-323 (1981).
 Currit, P. A., Dyer, M. and Mills, H. D., Certifying the Reliability of Software. *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 3-11 (1986).
 Henninger, K. L., Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE TSE*, vol. SE-6, no. 1, pp. 2-13 (1980).
 Linger, R. C., Mills, H. D. and Witt, B. I., Structured

- Programming: Theory and Practice. *IEEE TSE*, Reading, MA: Addison-Wesley (1979).
- Myers, G. J., A Controlled Experiment in Program Testing and Code Walkthrough Inspections. *Communications ACM*, pp. 760-768 (1978).
- Hetzel, W. C. An Experimental Analysis of Program Verification Problem Solving Capabilities as They Relate to Programmer Efficiency. *Computer Personnel*, vol. 3, pp. 10-15 (1972).
- Porter, A. A., Votta, L. G. and Basili, V. R., Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. *IEEE Transactions on Software Engineering*, vol. 21, no. 6, pp. 563-575 (1995).
- Selby, R., Basili, V. R. and Baker, T., Cleanroom Software Development: An Empirical Evaluation. *IEEE Transactions on Software Engineering*, pp. 1027-1037 (1987).