An Experiment To Assess Different Defect Detection Methods For Software Requirements Inspections

A.A. Porter

Computer Science Department
University of Maryland
Institute for Advanced Computer Science
College Park, Maryland 20472
aporter@cs.umd.edu

L.G. Votta

Software Production Research Department
AT&T Bell Laboratories

Naperville, IL 60566
votta@research.att.com

January 21, 1994

ABSTRACT

Software requirements specifications (SRS) are usually validated by inspections, in which several reviewers read all or part of the specification and search for defects. We hypothesize that different methods for conducting these searches may have significantly different rates of success. Therefore, we designed and performed an experiment to assess the effect of different detection methods on the probability of detecting defects in a software specification document.

Preliminary results of our first run with 24 reviewers, assembled in 3 member teams, indicate that a scenario-based detection method, in which each reviewer executes a specific procedure to discover a particular class of defects is more effective than either ad hoc or checklist methods.

We describe the design, execution, and analysis of the experiment so others may reproduce it and test our results for different kinds of software developments and different populations of software engineers.

This work is supported in part by the National Aeronautics and Space Administration under grant NSG–5123

1 Introduction

One of the most common ways of validating a software requirements specification (SRS) is to submit it to an inspection by a team of reviewers. Many organizations use a three-step inspection procedure: defect[1] detection, collection and repair[2]. [7, 15] A team of reviewers reads the SRS, identifying as many defects as possible. Newly identified defects are collected, usually at a team meeting, and then sent to the document's authors for repair.

We are focusing on the methods used to perform the first step in the inspection process, defect detection. For the purposes of this article, we define a defect detection method as a set of defect detection procedures coupled with an assignment of responsibilities to individual reviewers.

Defect detection procedures are the strategies reviewers use to discover defects. These may range in sophistication from intuitive, nonsystematic techniques, such as ad hoc or checklist procedures, to explicit and highly systematic procedures, such as formal proofs of correctness.

A reviewer's responsibility may range from general to specific. For example, a reviewer whose goal is to identify as many defects as possible has a general responsibility. In contrast, a reviewer who is only concerned with a limited set of issues, say insuring appropriate use of hardware interfaces, identifying untestable requirements or checking conformance to coding stan-

[1] We have consciously chosen the word *defect* instead of the word *fault* even though this does not adhere to the IEEE Standards on Software Engineering Terminology [8]. The reason is that *fault* has a context implying software–only one of the several places where inspections are used.

[2] Depending on the exact form of the inspection, they are sometimes called reviews or walkthroughs. For a more thorough description of the taxonomy see [7] pp. 171*ff* and [9].

dards has a specific responsibility.

In practice, reviewers often use ad hoc or checklist detection procedures to discharge general reviewer responsibilities. Some authors, notably Parnas and Weiss[12] have argued that using more systematic detection procedures with selective reviewer responsibilities would be a more effective method.

Until now, however, there have been no reproducible, quantitative studies comparing alternative detection methods for software inspections. We have conducted such an experiment and our initial results strongly indicate that the choice of defect detection method significantly affects inspection performance. Furthermore our experimental design may be easily replicated by interested researchers.

Below we describe the relevant literature, several alternative defect detection methods which motivated our study, our research hypothesis, and our experimental observations, analysis and conclusions.

1.1 Inspection Literature

A summary of the origins and current practice of inspections may be found in [7]. Consequently, we will discuss only work directly related to our current efforts.

Fagan[5] defined the basic software inspection process. While most writers have endorsed this approach[2, 7], Parnas and Weiss are more critical [12]. In part they argue that effectiveness suffers because individual reviewers are not assigned specific responsibilities and because they lack specialized techniques for meeting those responsibilities.

To address these concerns – at least for software designs – they introduced the idea of active design reviews. The principal characteristic of active design reviews is that individual reviewers read for a specific purpose using specialized questionnaires. Their proposal forms the motivation for the scenario-based detection method described in Section 2.2.

1.2 Detection Methods

The two most frequently used defect detection methods employ ad hoc or checklist procedures with no explicit (i.e., general) assignment of reviewer responsibilities. Ad hoc detection methods are characterized by the absence of systematic detection techniques and a very general reviewer focus. Checklist methods reuse important "lessons learned" within an environment or application. Their purpose is to define the reviewer responsibilities and suggest ways for reviewers to identify defects. A checklist might list characteristic defects, prioritize different defects, or pose questions that help reviewers discover defects, such as "Are all interfaces clearly defined?" or "If input is received at a faster rate than can be processed how is this handled?"

An alternative approach might combine systematic detection methods with selective reviewer responsibility. This approach would give individual reviewers specific detection responsibilities and specialized techniques for meeting them.

To explore this alternative we developed a set of defect-specific procedures called scenarios. A scenario is a collection of procedures that operationalize strategies for detecting particular classes of defects. Each reviewer executes a single scenario, so multiple reviewers are needed to achieve broad coverage of the document.

1.3 Hypothesis

The remainder of this article describes a controlled experiment comparing several detection methods. The goals of this experiment are twofold: to characterize the behavior of current approaches and to assess the potential benefits of scenario-based methods. The hypothesis underlying this experiment is depicted in Figure 1: that is, that nonsystematic procedures with general reviewer responsibility may lack reviewer coordination, leading to overlap and gaps, and thereby lowering the overall inspection effectiveness. On the other hand, systematic approaches with specific responsibilities might improve coordination and reduce gaps – increasing the overall detection effectiveness of the inspection.

2 The Experiment

We designed and conducted a multitrial experiment to compare the effectiveness of several detection methods. We ran the experiment in the Spring of 1993 with 24 subjects – students from A. Porter's graduate course on formal specification methods, who acted as reviewers. The complete run consisted of (1) a training phase in which the subjects were taught inspection methods and the experimental procedures, and in which they inspected a sample SRS, and (2) an experimental phase in which the subjects conducted two monitored inspections.

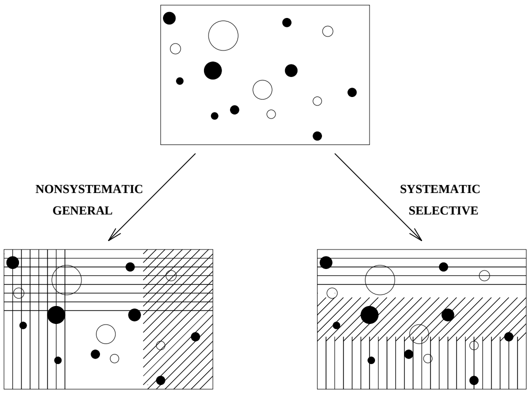


Figure 1: **Systematic Inspection Research Hypothesis.** This figure represents a software requriements specification before and after a *nonsystematic, general responsibility* and a *systematic, selective responsibility* inspection. The points and holes portray various defects. The line-filled regions indicate the coverage achieved by different members of the inspection team. Our hypothesis is that systematic, selective responsibility inspections acheive broader coverage and minimize reviewer overlap, resulting in higher defect detection rates and greater cost benefits than nonsystematic methods.

2.1 Experimental Design

The design of the experiment is somewhat unusual. To avoid misinterpreting the data it is important to understand the experiment and the reasons for certain elements of it's design [3].

2.1.1 Variables

The experiment manipulates four independent variables:

1. the detection method used by a reviewer (*ad hoc, checklist, or scenario*);
2. the team composition (who is assigned to which team);
3. the specification to be inspected (two are used during the experiment);
4. the order in which specifications are inspected by each team of reviewers;

Table 1 shows the settings of the independent variables.

The dependent variable is the team defect detection rate. That is the number of defects detected by the team divided by the total number of defects known [4] to be in the specification. The closer that value is to 1, the more effective we judge the detection method to be.

2.1.2 Design

The experiment uses a randomized, partial factorial design The design is randomized because the treatments of the teams are chosen at random. It is partial factorial because all combinations of independent variables are present.

2.1.3 Threats to Internal Validity

A potential problem in any experiment is that some factor may affect the dependent variable without the researcher's knowledge. These effects must be minimized whenever possible. We considered three main threats to internal validity: (1) selection effects, (2) maturation effects, and (3) instrumentation effects.

Selection effects are due to natural variation in human perfomance. For example, random assignment

[3] The interested reader should consult chapter 4 of reference [10] for an excellent discussion of randomized social experimental designs.

[4] No defects were intentionally seeded into the specifications. All defects are naturally occurring.

Trial/Specification

Detection Method		Trial 1		Trial 2	
		WLMS	CRUISE	WLMS	CRUISE
	ad hoc	B,D,G,H	A,C,E,F	A	D
	checklist			E	B,H
	scenarios			F,C	G

Table 1: This table shows the settings of the independent variables. Each team inspects both documents, one per round, using one of three detection methods. For the first trial all teams used the *ad hoc* method. In the second trial, methods were randomly assigned.

may accidentally creates elite teams. Therefore, differences in the team's natural abilities will mask differences in the detection methods. To limit this effect, we rated each reviewer's background knowledge and experience as either low, medium, or high. Teams of three were then formed by selecting one individual at random from each category.

This approach has its own drawbacks. In future replications we will opt for purely random assignment. We then avoid any possibility, no matter how remote, that the method of categorizing inspection experience is responsible for the observed outcome of the experiment.

Maturation effects are due to individuals learning from previous experience as the experiment proceeds. We have manipulated the detection method used and the order in which the documents are inspected so that the presence of this effect can be detected.

As will be seen in Section 3, neither selection, nor maturation significantly explains variation in the defect detection rate.

Finally, the instrumentation effects may result from differences in the specification documents. Such variation is impossible to avoid, but we control for it by having each team inspect both documents.

2.1.4 Threats to External Validity

Threats to external validity limit our ability to generalize the results of our experiment to industrial practice. We identified three threats to external validity:

1. the reviewers in the first run of our experiment may not be representative of software programming professionals,
2. the specification documents may not be representative of real programming problems,
3. the inspection process in our experimental design may not be representative of software development practice.

The first two threats are real. To surmount them we will replicate our experiment with software programming professionals inspecting industrial work products. Nevertheless, laboratory experimentation is a necessary first step because it greatly reduces the risk of transferring immature technology.

We avoided the third threat by modeling the experiment's inspection process after the design inspection process described in Eick, et. al., [4] which is used by several development organizations at AT&T; therefore, we know that at least one professional software development organization practices inspections in this manner.

2.1.5 Analysis Strategy

Our analysis strategy has two steps. The first step was to find those independent variables that individually explain a significant amount of the variation in the dependent variable. This was accomplished by using an analysis of variance technique as discussed in Box, et. al. ([3], pp. 165*ff*).

The second step was to evaluate the combined effect of the variables shown to be significant in the initial analysis. Again, we followed Box, et. al. closely ([3], pp. 210*ff*).

Once these relationships were discovered and their magnitude estimated, we examined other data, such as correlations between categories of defects detected and detection methods that would confirm or reject (if possible) a causal relationship between each independent variable and the dependent variable.

2.2 Experiment Instrumentation

We developed several instruments for this experiment: Three small software requirements specifications (SRS), instructions and aids for each detection method, and a data collection form.

2.2.1 Software Requirements Specifications

The SRS we used describe three event-driven process control systems: an elevator control system, a water level monitoring system, and an automobile cruise control system. Each specification has four sections: Overview, Specific functional requirements, External interfaces, and a Glossary. The overview is written in natural language, while the last three sections are specified using the SCR tabular requirements notation [6].

For this experiment, all three documents were adapted to adhere to the IEEE suggested format [9]. All defects present in these SRS appear in the original documents or were generated during the adaptation process; no defects were intentionally seeded into the document. The authors discovered forty-two defects in the WLMS SRS; 26 in the CRUISE SRS. The authors did not inspect the ELEVATOR SRS since it was only used during training exercises.

Elevator Control System (ELEVATOR) describes the functional and performance requirements of a system for monitoring the operation of a bank of elevators [16]. 16 pages.

Water Level Monitoring System (WLMS) describes the functional and performance requirements of a system for monitoring the operation of a steam generating system [14]. 24 pages.

Automobile Cruise Control System (CRUISE) describes the functional and performance requirements for an automobile cruise control system [11]. 31 pages.

2.2.2 Defect Detection Methods

To make a fair assessment of the three detection methods (ad hoc, checklist, and scenario) each method needed to search for the same population of defects. This population was defined using a general defect taxonomy.

The taxonomy is a composite of two schemes developed by Schneider, et.al. [13] and Basili and Weiss [1]. Defects are divided into two broad types: omission in which important information is left unstated and commission in which incorrect, redundant or ambiguous information is placed in the SRS. Omission defects were subdivided into four categories: Missing Functionality (MF) , Missing Performance (MP), Missing Environment (ME), and Missing Interface (MI). Commission defects were divided into four categories: Ambiguous Information (AI), Inconsistent Information (II), Incorrect or Extra Functionality (IF), and Wrong Section (WS). (See Appendix A for complete taxonomy.)

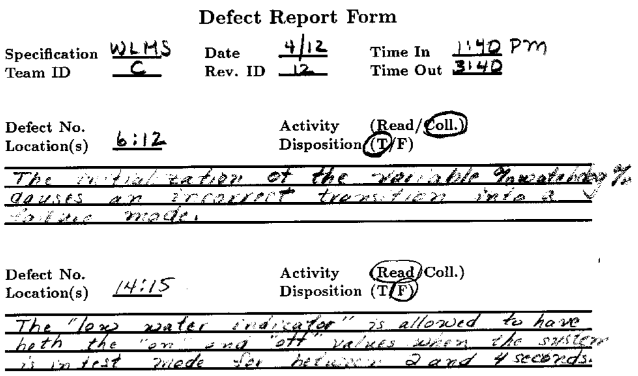
We provided a copy of the taxonomy to each reviewer. Reviewers using the ad hoc method received no further assistance.

We constructed a single checklist, based on the defect taxonomy. The checklist items are detailed questions culled from several industrial checklists. Thus, they are very similar to checklists used in practice. All checklist reviewers used the same checklist. (See Appendix B for the complete checklist.)

Finally, we developed three groups of scenarios to cover the defect taxonomy. These scenarios are intended to detect data type inconsistencies, incorrect functionality, and ambiguous or missing functionality. The defects targeted by the procedures are a subset of those covered by the taxonomy. We provide a complete description of all scenarios in Appendix C.

2.2.3 Defect Report Forms

We also developed a Defect Report Form. Whenever a potential defect was discovered – during the defect detection or collection activities – an entry was made on the form. The entry captured information such as: Inspection activity (Detection, Collection), Defect Location (Page and Line Numbers), Defect Disposition, (Defect can be a True Defect or a False Positive), and a prose Defect Description.



Defect Report Form

Specification WLMS Date 4/12 Time In 1:40 PM
Team ID C Rev. ID 12 Time Out 3:40

Defect No.
Location(s) 6:12
Activity (Read/Coll.)
Disposition (T/F)

The initialization of the variable [illegible]
causes an incorrect transition into a
[illegible] mode.

Defect No.
Location(s) 14:15
Activity (Read/Coll.)
Disposition (T/F)

The "low water indicator" is allowed to have
both the "on" and "off" values when the system
is in test mode for between 2 and 4 seconds.

Figure 2: **Reviewer Defect Report Form.** This is a small portion of the defect report that each reviewer completed during the reviewers detection phase of the experiment. Shown are defects number 10 and 11 found by reviewer 12 of team C for the WLMS specification document.

A small portion of a Defect Report appears in Figure 2.

2.3 Experiment Preparation

Twenty four subjects participated in the study. All were graduate students at the University of Maryland, taking a course on Formal Verification Methods from A. Porter.

The students were given a series of lectures on software requirements specifications, the SCR tabular requirements notation, inspection procedures, the defect classification scheme and the filling out of data collection forms. The references for these lectures were [12, 5, 9].

Each participant filled out a personal experience survey of their academic status, relevant computer science course work, and professional experience. These surveys were used to create teams with roughly equal skills. (See Section 2.1.3 for details.) Within each team, members were randomly assigned to act as the moderator, the recorder, or the reader during the collection meeting.

2.4 Conducting the Experiment

2.4.1 Training

For the training exercise, each team inspected the Elevator SRS. Individual team members read the specification and recorded all defects they found on a Defect Report Form. Their effort was restricted to two hours. Later we met with the students and answered any questions about the procedure. Afterwards, each team conducted a supervised inspection meeting and filled out a master Defect Report Form for the entire team. The ELEVATOR SRS was not used in the remainder of the experiment.

2.4.2 First Trial: Characterizing Ad Hoc Detection

The instruments used in the first trial are the WLMS and CRUISE specifications discussed in Section 2.2.1 and the defect report forms described in Section 2.2.3. Four of the eight teams were asked to inspect the CRUISE specification; the remaining four teams inspected the WLMS specification. The defect detection step was limited to two hours and all potential defects were to be reported on the defect report form. After defect detection, all materials were collected. [5] Once all team members had finished their detection phase, the team's moderator arranged for the collection meeting. At the meeting, the documents were reread and defects discussed. The team's recorder maintained the team's master defect report form. The collection meeting was also limited to 2 hours. The entire trial was completed in a one week period.

[5] We set aside 14 two-hour time slots during which inspection tasks could be done. Participants performed each task within a single two-hour session and were not allowed to work at other times.

2.4.3 Second Trial: Detection Methods

The instruments used in the second trial were the specifications used in trial 1, a checklist, three defect-based scenarios, and the defect report form. The development of the checklist and scenarios is described in Section 2.2.2. The same checklist and scenarios were used for both documents.

The eight teams were assigned to experimental groups as follows: teams that inspected the CRUISE specification in trial 1, inspected the WLMS specification in trial 2, and vice versa. Of the four teams inspecting the CRUISE specification, one used the ad hoc method, two used checklists, and the fourth team used the scenarios. For the WLMS specification, one team used the ad hoc method, one team used the checklist, and two teams used the scenarios. The experimental procedures and data collection were exactly the same as for trial 1. After the trial was completed we gathered and analyzed all the defect report forms.

3 Data and Analysis

3.1 Data

Two sets of data are important to our study. The first set shows whether or not a team discovered a particular defect. This data is created at the team meetings and is used to assess the effectiveness of each defect detection method. Figure 3.1 graphically summarizes this data, showing the defect detection rate and detection method used for each team. As the Figure shows, the *scenarios* detection methods resulted in the highest defect detection rates, followed by *ad hoc* detection methods, and finally by *checklist* detection methods.

The second data set is the summaries of the individual Defect Report Forms filled out when the reviewer read the specification document. This data provides a log of the defects each reviewer discovered and is used to assess whether procedures improve a reviewer's ability to identify specific classes of defects.

3.2 Analysis

Table 2 contains the results of the analysis as outlined in Section 2.1.5. The independent variables are listed in order from the least to the most significant. The maturation, replication, and selection effects are negligible. The anticipated instrument bias is present, as is the significant effect of the detection method.

We use an analysis of variance as described in Section 2.1.5 to assess the simultaneous effect of **Instrumentation** and **Detection Methods** on the defect detection rate. Table 3 shows the input to this analysis. The table contains 6 cells (3 detection methods, 2 specification documents) where each cell contains the average for all teams with the detection method and specification document. As the Table shows, the *scenarios* detection methods resulted in the highest defect detection rates, followed by *ad hoc* detection methods, and finally by *checklist* detection methods.

Table 4 shows the analysis of variance for the specification and detection method independent variables. The analysis assumes that there is no interaction between the independent variables [6]. The analysis of variance confirms our first analysis. The detection methods explain roughly 3 to 4 times more variance in the defect detection rate than do the instruments (the difference between the specification documents).

From both these analyses we conclude that the *scenario* detection method is the most effective and the *checklist* method the least effective.

An analysis of the second data set gives one explanation for why scenarios increase defect detection rates. Table 5 categorizes the defects in both documents and compares the average detection rates of scenario and all other reviewers. We see that in most cases reviewers using scenarios are more effective at finding the defects their scenario was designed to uncover. At the same time, all reviewers are equally effective at detecting defect that are not targeted by the scenarios. ("Not Observable" category).

4 Summary and Conclusions

Our experimental design for comparing defect detection methods is flexible and economical, and allows the researcher to measure the effect of several potential threats to the experiment's internal validity. In particular, we determined that neither the experience gained by the reviewers as the experiment progressed nor the composition of the teams had a significant effect on inspection effectiveness. However, differences in the SRS did have an effect.

The results from our initial experiment are drawn from only sixteen observations. We are currently replicating

[6] This assumption may appear naive; but, until we can replicate the experiment, we do not have enough data points to analyze nonlinear interactions.

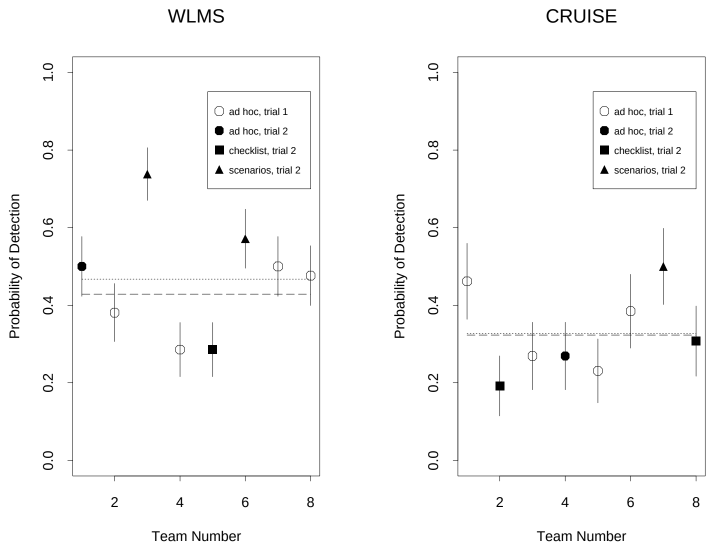


Figure 3: **Defect Detection by Detection Method.** The observed defect detection rates are displayed. The open symbols are trial 1, the filled symbols are trial 2. The vertical line segment through each point indicates one standard deviation in the estimate of the rate (assuming each defect was a Bernoulli trial). The dashed (dotted) lines display the average of the defect detection rates for trial 1 (trial 2).

Independent Variable	SS_T	ν_T	SS_R	ν_R	$(SS_T/\nu_T)(\nu_R/SS_R)$	Significance Level
Inspection round–maturation	.009	1	.315	14	.40	.537
Team composition–selection	.160	7	.164	8	1.120	.434
Specification– instrumentation	.081	1	.243	14	4.678	.048
Detection Method	.186	2	.139	13	8.71	.004

Table 2: **Analysis of Variance for Each Independent Variable.** The analysis of variance of team defect detection rate on the four independent variables shows the significance of the detection method and specification for explaining variation in the dependent variable.

the experiment to increase the size of our data sets. Keeping in mind the limited number of observations, we tentatively draw several conclusions.

The results from a single run of this experiment indicate that the defect detection probability when using scenarios is superior to that obtained with ad hoc or checklist methods – an improvement of roughly 35%. The data also demonstrate that the superiority of scenarios is due in part to the ability of reviewers to focus on specific classes of defects, in accordance with the suggestion of Parnas and Weiss. Readers should note however, the scenarios appeared to be better suited to the defect profile of the WLMS SRS than the CRUISE. This indicates that poorly designed scenarios risk poor inspection performance. Another important result is that the checklist method – the industry standard, was the poorest detection method. Since the scenarios were constructed from the checklists, the obvious explanation that the checklist is poorly designed seems incorrect.

The results of this work have several implications for software practitioners. They indicate that overall inspection performance can be improved when individual reviewers use systematic procedures to address a small set of specific issues. This contrasts with the usual practice where reviewers have neither systematic procedures nor clearly defined responsibilities.

Instrument	Detection Method		
	ad hoc	*checklist*	*scenario*
WLMS	.38, .29, .50, .48, .50*	.29*	.50*, .74*
(average)	.43	.29	.62
Cruise	.39, .23, .27, .46, .27*	.19*, .31*	.50*
(average)	.32	.25	.50

Table 3: **Team Defect Detection Rate Data.** The nominal and mean defect detection rates for all 16 teams are arranged in the appropriate cells. The rates without (with) asterisks are for trial 1 (trial 2).

Source	Sum of Squares	Degrees of Freedom	Mean Square
average	.9680	1	.9680
specification	.0122	1	.0122
detection method	.0862	2	.0431
residuals	.0019	2	.0010
TOTALS	1.0683	6	

Table 4: **Analysis of Variance of Detection Method and Document.** The table displays the results of an analysis of variance of the average team defect detection rates given in Table 3

From the point of view of software researchers the work demonstrates the feasibility of constructing and executing inexpensive experiments to validate fundamental research recommendations. Since economical designs are necessary to allow replication in other environments with different populations of software engineers, this is an important consideration.

5 Future Work

The experimental data raise many interesting questions for future study.

- In many instances a single reviewer identified a defect at preparation, but the defect was not subsequently recorded on the team log. Are single reviewers who observe a defect sometimes forgetting to mention a defect they observed or is the reviewer being "talked out" of the defect at the team meeting?

- In the same analysis we notice that very few defects are initially discovered during defect collection. As the collection activity normally occurs as a meeting, we can ask, "Does every inspection need a meeting?" Interestingly, this data is consistent with an industrial study performed by L. Votta [15].

- Consider the effectiveness of scenarios as shown in Table 5. More than half of the defects in both specifications are not addressed by the scenarios used in this study. What other scenarios are necessary to achieve a broader defect coverage?

- Previous work by Schneider, et al., has shown that a N-fold inspection (i.e., performing an inspection with several independent teams working in parallel) can improve performance. A close examination of our data supports this conclusion, but it appears that the improvement may be much less pronounced than previously reported.

We also believe that the results of any individual experiment can be challenged. Experiments gain credibility only through replication and reproduction. Each new run reduces the probability that results can be explained by natural human variation or experimental error. Consequently, we are replicating the experiment at the University of Maryland. We are also creating a laboratory (i.e., a package containing all the experimental materials) so that other groups may replicate the experiment easily. We expect to have all the materials publicly available by June 1994. This "lab" will allow us to reproduce the experiments with other university researchers in Japan, Germany, and Austrailia and with industrial developers at AT&T Bell Laboratories and Motorola Inc.

	WLMS			CRUISE		
Scenario	Total	Scen.	Other	Total	Scen.	Other
Data type checking	14	7.5	4.1	10	6	2.6
Incorrect functionality	5	1.5	0.7	3	0	0.3
Missing functionality	5	3.5	0.8	1	0	0
Not Observable	18	5.0	3.9	12	2.6	2.0

Table 5: **Average Number of Defects Found by Defect Type**. The table shows the average defect detection rate per individual reviewer for both scenario and all other detection methods. For the defects that the scenario was designed to detect the reviewers using the method were more effective than reviewers using either the ad hoc or checklist. All reviewers were equally effective at finding the defects in the "Not Observable" category–defects where no scenario aided in their detection.

Acknowledgements

The work reported in this article is part of a long-term effort to create software development processes that can be engineered to fit the goals and needs of their users. The research is the product of a collaborative effort between Victor Basili, Adam Porter, and Lawrence Votta. Due to a mistake on our part, Victor Basili does not appear as an author on this work. Therefore, we would like to take this opportunity to acknowledge his many contributions to this research.

We would also like to recognize the hard work of all the experimental participants – an excellent job was done by all. We are grateful to Mark Ardis, John Kelly, and David Weiss, who helped us to identify sample requirements specifications and inspection checklists. We also wish to recognize the valuable technical comments provided by John Gannon, Richard Gerber, Clive Loader, Eric Slud and Scott VanderWeil Finally, Art Caso's editing is greatly appreciated.

References

[1] V. R. Basili and D. M. Weiss. Evaluation of a software requirements document by analysis of change data. In *Proceedings of the Fifth International Conference on Software Engineering*, pages 314–323, San Diego, CA, March 1981.

[2] Barry W Boehm. *Software Engineering Economics.* Prentice Hall, Englewood Cliffs, NJ, 1981.

[3] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters.* John Wiley & Sons, New York, 1978.

[4] Stephen G Eick, Clive R Loader, M. David Long, Scott A Vander Wiel, and Lawrence G Votta. Estimating software fault content before coding. In *Proceedings of the 14th International Conference on Software Engineering*, pages 59–65. Association for Computing Machinery, May 1992.

[5] M E Fagan. Design and code inspections to reduce errors in program development. *IBM Syst. J.*, 15(3):182–211, 1976.

[6] Kathryn L. Heninger. Specifying Software Requirements for Complex Systems: New Techniques and their Application. *IEEE Transactions on Software Engineering*, SE-6(1):2–13, January 1980.

[7] Watts S. Humphery. *Managing the Software Process.* Addison-Wesley Publishing Co., 1989. Reading, Massachusetts.

[8] *IEEE Guide for the Ude of IEEE Standard Dictionary of Measures to Produce Reliable Software.* Soft. Eng. Tech. Comm. of the IEEE Computer Society, 1989. IEEE Std 982.2-1988.

[9] *IEEE Standard for software reviews and audits.* Soft. Eng. Tech. Comm. of the IEEE Computer Society, 1989. IEEE Std 1028-1988.

[10] Charles M. Judd, Eliot R. Smith, and Louise H. Kidder. *Research Methods in Social Relations.* Holt, Rinehart and Winston, Inc., Fort Worth, TX, sixth edition, 1991.

[11] J. Kirby. Example nrl/scr software requirements for an automobile cruise control and monitoring system. Technical Report TR-87-07, Wang Institute of Graduate Studies, July 1984.

[12] Dave L Parnas and Dave M Weiss. Active design reviews: principles and practices. In *Proceedings*

of the 8th International Conference on Software Engineering, pages 215–222, Aug. 1985.

[13] G. Michael Schnieder, Johnny Martin, and W. T. Tsai. An experimental study of fault detection in user requirements. *ACM Transactions on Software Engineering and Methodology*, 1(2):188–204, April 1992.

[14] J. van Schouwen. The a-7 requirements model: Re-examination for real-time systems and an application to monitoring systems. Technical Report TR-90-276, Queen's University, Kingston, Ontario, Canada, May 1990.

[15] Lawrence G Votta. Does every inspection need a meeting? In *Proceedings of ACM SIGSOFT '93 Symposium on Foundations of Software Engineering*. Association for Computing Machinery, December 1993.

[16] William G. Wood. Temporal logic case study. Technical Report CMU/SEI-89-TR-24, Software Engineering Institute, Pittsburgh, Pa, August 1989.

A Ad Hoc Detection

The defect taxonomy primarily from the work of Schneider, et al., A small portion is due to Basili and Weiss. We have borrowed this directly from previously published sources.

- Omission
  - Missing Functionality (MF): Information describing the desired internal operational behavior of the system has been omitted from the SRS.
  - Missing Performance (MP): Information describing the desired performance specifications has either been omitted or described in a way that is unacceptable for acceptance testing.
  - Missing Interface (MI): Information describing how the proposed system will interface and communicate with objects outside the the scope of the system has been omitted from the SRS.
  - Missing Environment (ME): Information describing the required hardware, software, database, or personnel environment in which the system will run has been omitted from the SRS
- Commission
  - Ambiguous Information (AI): An important term, phrase or sentence essential to the understanding of system behavior has either been left undefined or defined in a way that can cause confusion and misunderstanding.
  - Inconsistent Information (II): Two sentences contained in the SRS directly contradict each other or express actions that cannot both be correct or cannot both be carried out.
  - Incorrect Fact (IF): Some sentence contained in the SRS asserts a facts that cannot be true under the conditions specified in the SRS.
  - Wrong Section (WS): Essential information is misplaced within the SRS

B Checklist Method

- General
  - Are the goals of the system defined?
  - Are the requirements clear and unambiguous?
  - Is a functional overview of the system provided?
  - Is an overview of the operational modes provided?
  - Have the software and hardware environments been specified?
  - If assumptions that affect implementation have been made, are they stated?
  - Have the requirements been stated in terms of inputs, outputs, and processing for each function?
  - Are all functions, devices, constraints traced to requirements and vice versa?
  - Are the required attributes, assumptions and constraints of the system completely listed?
- Omission
  - Missing Functionality
    * Are the described functions sufficient to meet the system objectives?
    * Are all inputs to a function sufficient to perform the required function?
    * Are undesired events considered and their required responses specified?
    * Are the initial and special states considered (e.g., system initiation, abnormal termination)?
  - Missing Performance
    * Can the system be tested, demonstrated, analyzed, or inspected to show that it satisfies the requirements?
    * Have the data type, rate, units, accuracy, resolution, limits, range and critical values
    * for all internal data items been specified?
    * Have the accuracy, precision, range, type, rate, units, frequency, and volume of inputs and outputs been specified for each function?
  - Missing Interface
    * Are the inputs and outputs for all interfaces sufficient?
    * Are the interface requirements between hardware, software, personnel, and procedures included?
  - Missing Environment
    * Have the functionality of hardware or software interacting with the system been properly specified?
- Commission
  - Ambiguous Information
    * Are the individual requirements stated so that they are discrete, unambiguous, and testable?
    * Are all mode transitions specified deterministicly?
  - Inconsistent Information
    * Are the requirements mutually consistent?
    * Are the functional requirements consistent with the overview?
    * Are the functional requirements consistent with the actual operating environment?
  - Incorrect or Extra Functionality
    * Are all the described functions necessary to meet the system objectives?
    * Are all inputs to a function necessary to perform the required function?
    * Are the inputs and outputs for all interfaces necessary?
    * Are all the outputs produced by a function used by another function or transferred across an external interface?
  - Wrong Section
    * Are all the requirements, interfaces, constraints, etc. listed in the appropriate sections.

C Scenarios

C.1 Data Type Consistency Scenario

1. Identify all data objects mentioned in the overview (e.g., hardware component, application variable, abbreviated term or function)
   (a) Are all data objects mentioned in the overview listed in the external interface section?
2. For each data object appearing in the external interface section determine the following information:
   - Object name:
   - Class: (e.g., input port, output port, application variable, abbreviated term, function)
   - Data type: (e.g., integer, time, boolean, enumeration)
   - Acceptable values: Are there any constraints, ranges, limits for the values of this object
   - Failure value: Does the object have a special failure value?
   - Units or rates:
   - Initial value:

   (a) Is the object's specification consistent with its description in the overview?
   (b) If object represents a physical quantity, are its units properly specified?
   (c) If the object's value is computed, can that computation generate a non-acceptable value?
3. For each functional requirement identify all data object references:
   (a) Do all data object references obey formatting conventions?
   (b) Are all data objects referenced in this requirement listed in the input or output sections?
   (c) Can any data object use be inconsistent with the data object's type, acceptable values, failure value, etc.?
   (d) Can any data object definition be inconsistent with the data object's type, acceptable values, failure value, etc.?

C.2 Incorrect Functionality Scenario

1. For each functional requirement identify all input/output data objects:
   (a) Are all values written to each output data object consistent with its intended function?
   (b) Identify at least one function that uses each output data object.
2. For each functional requirement identify all specified system events:
   (a) Is the specification of these events consistent with their intended interpretation?
3. Develop an invariant for each system mode (i.e. Under what conditions must the system exit or remain in a given mode)?
   (a) Can the system's initial conditions fail to satisfy the initial mode's invariant?
   (b) Identify a sequence of events that allows the system to enter a mode without satisfying the mode's invariant.
   (c) Identify a sequence of events that allows the system to enter a mode, but never leave (deadlock).

C.3 Ambiguities/Missing Functionality Scenario

1. Identify the required precision, response time, etc. for each functional requirement.
    (a) Are all required precisions indicated?
2. For each requirement, identify all monitored events.
    (a) Does a sequence of events exist for which multiple output values can be computed?
    (b) Does a sequence of events exist for which no output value will be computed?
3. For each system mode, identify all monitored events.
    (a) Does a sequence of events exist for which transitions into two or more system modes is allowed?