Comparing Detection Methods For Software Requirements Inspections: A Replication Using Professional Subjects

Adam Porter*
Computer Science Department
University of Maryland
College Park, Maryland 20742
aporter@cs.umd.edu

Lawrence Votta
Software Production Research Department
Lucent Technologies
Naperville, Illinois 60566
votta@research.bell-labs.com

Abstract

Software requirements specifications (SRS) are often validated manually. One such process is inspection, in which several reviewers independently analyze all or part of the specification and search for faults. These faults are then collected at a meeting of the reviewers and author(s).

Usually, reviewers use Ad Hoc or Checklist methods to uncover faults. These methods force all reviewers to rely on nonsystematic techniques to search for a wide variety of faults. We hypothesize that a Scenario-based method, in which each reviewer uses different, systematic techniques to search for different, specific classes of faults, will have a significantly higher success rate.

In previous work we evaluated this hypothesis using 48 graduate students in computer science as subjects.

We now have replicated this experiment using 18 professional developers from Lucent Technologies as subjects. Our goals were to (1) extend the external credibility of our results by studying professional developers, and to (2) compare the performances of professionals with that of the graduate students to better understand how generalizable the results of the less expensive student experiments were.

For each inspection we performed four measurements: (1) individual fault detection rate, (2) team fault detection rate, (3) percentage of faults first identified at the collection meeting (meeting gain rate), and (4) percentage of faults first identified by an individual, but never reported at the collection meeting (meeting loss rate).

For both the professionals and the students the experimental results are that (1) the Scenario method had a higher fault detection rate than either Ad Hoc or Checklist methods, (2) Checklist reviewers were no more effective than Ad Hoc reviewers, (3) Collection meetings produced no net improvement in the fault, and detection rate – meeting gains were offset by meeting losses,

Finally, although specific measures differed between the professional and student populations, the outcomes of almost all statistical tests were identical. This suggests that the graduate students provided an adequate model of the professional population and that the much greater expense of conducting studies with professionals may not always be required.

Keywords: Inspection, Controlled Experiment, Replication.

*This work is supported in part by a National Science Foundation Faculty Early Career Development Award CCR–9501354.

1 Introduction

One way 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 for eliminating faults : detection, collection, and repair[1]. [11, 19] A team of reviewers reads the SRS, identifying as many faults as possible. Newly identified faults 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 this process, fault detection. For this article, we define a fault detection method to be a set of fault detection techniques coupled with an assignment of responsibilities to individual reviewers.

Fault detection techniques may range in prescriptiveness from intuitive, nonsystematic procedures, such as Ad Hoc or Checklist techniques, to explicit and highly systematic procedures, such as formal proofs of correctness.

A reviewer's individual responsibility may be general – to identify as many faults as possible – or specific – to focus on a limited set of issues such as ensuring appropriate use of hardware interfaces, identifying untestable requirements, or checking conformity to coding standards.

These individual responsibilities may be coordinated among the members of a review team. When they are not coordinated, all reviewers have identical responsibilities. In contrast, the reviewers in coordinated teams may have separate and distinct responsibilities.

In practice, reviewers often use Ad Hoc or Checklist detection techniques to discharge identical, general responsibilities. Some authors, notably Parnas and Weiss[15], have argued that inspections would be more effective if each reviewer used a different set of systematic detection techniques to discharge different, specific responsibilities.

1.1 Preliminary Research

In earlier work [16] we conducted an experiment to compare alternative detection methods for software inspections. Our results suggest that the choice of fault detection method significantly affects inspection performance.

Our subjects for that study were 48 graduate students in computer science. Initially we use students rather than professional because cost considerations severely limit our opportunities to conduct studies with professional

[1]Depending on the exact form of the inspection, they are sometimes called reviews or walkthroughs. For a more thorough description of the taxonomy see [11] pp. 171*ff* and [12].

developers. Therefore we prefer to refine our experimental designs and measurement strategies in the university before using them in industry. This approach also allows us to do a kind of bulk screening of our research hypotheses. That is, we can conduct several studies in university, but only rerun the most promising ones in industry. Intuitively, we feel that hypotheses that don't hold up in the university setting are unlikely to do so in the industrial setting.

Of course, this reasoning is asymmetrical. It may or may not be true that results derived in the university apply in industry. Therefore, we still need to conduct studies with professional subjects. Consequently, to improve the external validity of our initial results we have replicated the experiment using professional software developers as subjects. We have also compared the performances of the student and professional populations to better understand how generalizable the original results were. This is important because experiments using professional subjects are far more costly than those using student subjects.

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

1.2 Inspection Literature

A summary of the origins and the current practice of inspections may be found in Humphrey [11]. Consequently, we will discuss only work directly related to our current efforts.

Fagan[7] defined the basic software inspection process. While most writers have endorsed his approach[4, 11], Parnas and Weiss are more critical [15]. In part, they argue that effectiveness suffers because individual reviewers are not assigned specific responsibilities and because they lack systematic techniques for meeting those responsibilities.

Some might argue that Checklists are systematic because they help define each reviewer's responsibilities and suggest ways to identify faults. Certainly, Checklists often pose questions that help reviewers discover faults. However, we argue that the generality of these questions and the lack of concrete strategies for answering them makes the approach nonsystematic.

To address these concerns – at least for software designs – Parnas and Weiss introduced the idea of active design reviews. The principal characteristic of an active design review is that each individual reviewer reads for a specific purpose, using specialized questionnaires. This proposal forms the motivation for the detection method proposed in Section 2.2.2.

1.3 Detection Methods

Ad Hoc and Checklist methods are two frequently used fault detection methods. With Ad Hoc detection methods, all reviewers use nonsystematic techniques and are assigned the same general responsibilities.

Checklist methods are similar to Ad Hoc, but each reviewer receives a checklist. Checklist items capture important lessons learned from previous inspections within an environment or application. Individual checklist items may enumerate characteristic faults, prioritize different faults, or pose questions that help reviewers discover faults, such as "Are all interfaces clearly defined?" or "If input is received at a faster rate than can be processed, how is this handled?" The purpose of these items is to focus reviewer responsibilities and suggest ways for reviewers to identify faults.

1.4 Hypothesis

We believe that an alternative approach which gives individual reviewers specific, orthogonal detection responsibilities and specialized techniques for meeting them will result in more effective inspections.

To explore this alternative we developed a set of fault-specific techniques called Scenarios – collections of procedures for detecting particular classes of faults. Each reviewer executes a single scenario and multiple reviewers are coordinated to achieve broad coverage of the document.

Our underlying hypothesis is depicted in Figure 1: that nonsystematic techniques with general reviewer responsibility and no reviewer coordination, lead to overlap and gaps, thereby lowering the overall inspection effectiveness; while systematic approaches with specific, coordinated responsibilities reduce gaps, thereby increasing the overall effectiveness of the inspection.

2 The Experiment

To evaluate our systematic inspection hypothesis we designed and conducted a multi-trial experiment. The goals of this experiment were twofold: to characterize the behavior of existing approaches and to assess the potential benefits of Scenario-based methods. Originally we ran the experiment twice. Both runs used 24 subjects each – students taking a graduate course in formal methods who acted as reviewers. We ran the experiment a third time

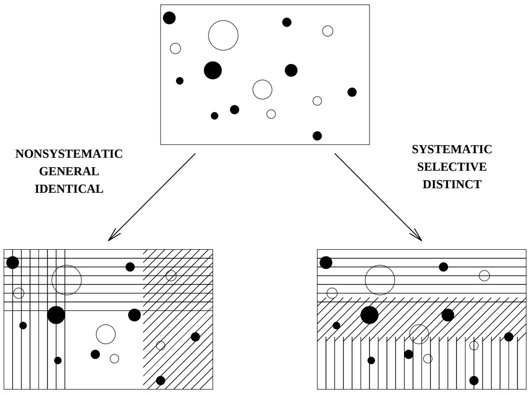

Figure 1: **Systematic Inspection Research Hypothesis.** This figure represents a software requirements specification before and after a *nonsystematic* technique, *general* and *identical* responsibility inspection and a *systematic* technique, *specific* and *distinct* responsibility inspection. The points and holes represent various faults. The line-filled regions indicate the coverage achieved by different members of the inspection team. Our hypothesis is that systematic technique, specific and coordinated responsibility inspections achieve broader coverage and minimize reviewer overlap, resulting in higher fault detection rates and greater cost benefits than nonsystematic methods.

as part of a professional training course at Lucent Technologies, using 18 professional developers as subjects.

Each complete experimental 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.

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 its design [2].

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 specification to be inspected (two are used during the experiment);
3. the inspection round (each reviewer participates in two inspections during the experiment);
4. the order in which the specifications are inspected (either specification can be inspected first).

[2] See Judd, et al. [13], chapter 4 for an excellent discussion of randomized social experimental designs.

The detection method is our treatment variable. The other variables allow us to assess several potential threats to the experiment's internal validity. For each inspection we measure four dependent variables:

1. the individual fault detection rate,
2. the team fault detection rate [3],
3. the percentage of faults first identified at the collection meeting (meeting gain rate), and
4. the percentage of faults first identified by an individual, but never reported at the collection meeting (meeting loss rate).

2.1.2 Design

The purpose of this experiment is to compare the Ad Hoc, Checklist, and Scenario detection methods for inspecting software requirements specifications.

When comparing multiple treatments, experimenters frequently use fractional factorial designs. These designs systematically explore all combinations of the independent variables, allowing extraneous factors such as team ability, specification quality, and learning to be measured and eliminated from the experimental analysis.

Had we used such a design each team would have participated in three inspection rounds, reviewing each of three specifications and using each of three methods exactly once. The order in which the methods are applied and the specifications are inspected would have been dictated by the experimental design.

Such designs are unacceptable for this study because they require some teams to use the Ad Hoc or Checklist method after they have used the Scenario method. Since the Ad Hoc and Checklist reviewers create their own fault detection techniques during the inspection (based on their experience or their understanding of the checklist), our concern was that using the Scenario method in an early round might imperceptibly distort the use of the other methods in later rounds. Such influences would be undetectable because, unlike the Scenario methods, the Ad Hoc and Checklist methods do not require reviewers to perform specific, auditable tasks.

We chose a partial factorial design in which each team participates in two inspections, using some combination of the three detection methods, but teams using the Scenario method in the first round must continue to use it

[3] The team and individual fault detection rates are the number of faults detected by a team or individual divided by the total number of faults known to be in the specification. The closer that value is to 1, the more effective the detection method. No faults were intentionally seeded into the specifications. All faults are naturally occurring.

Round/Specification

Detection Method		Round 1		Round 2	
		WLMS	CRUISE	WLMS	CRUISE
	ad hoc	1B, 1D, 1G 1H, 2A, 3C	1A, 1C, 1E 1F, 2D, 3E, 3F	1A, 3E	1D, 2B
	checklist	2B, 3A	2E, 2G, 3D	1E, 2D, 2G, 3D	1B, 1H, 3C
	scenarios	2C, 2F, 3B	2H	1F, 1C, 2E 2H, 3F	1G, 2A, 2C 2F, 3A, 3B

Table 1: This table shows the settings of the independent variables. Each team inspects two documents, the WLMS and CRUISE, one per round, using one of the three detection methods. Teams from the first replication are denoted 1A–1H, teams from the second replication are denoted 2A–2H. Teams from the third replication (the professional subjects) are denoted 3A–3F.

in the second round. Table 1 shows the settings of the independent variables.

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. This possibility must be minimized. We considered five such threats: (1) selection effects, (2) maturation effects, (3) replication effects, (4) instrumentation effects, and (5) presentation effects.

Selection effects are due to natural variation in human performance. For example, random assignment of subjects may accidentally create an elite team. Therefore, the difference in this team's natural ability will mask differences in the detection method performance. Our strategy is to assign teams to detection methods on a random basis. However, teams that used Scenarios in the first round were constrained to use them again in the second round. This compromise provides more observations of the Scenario method and prevents the use of the Scenario method from affecting the use of the Ad Hoc or Checklist methods. However we can't determine whether or not the teams that used only the Scenarios have greater natural ability than the other teams.

Maturation effects are due to subjects learning 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 discovered and taken into account.

Replication effects are caused by differences in the materials, participants, or execution of multiple replications. In the student studies we limited this effect by using only first and second year graduate students as subjects - rather than both undergraduate and graduate students. Across the student and professional populations we attempted to maintain consistency in the experimental procedures used by packaging the experimental procedures as a classroom laboratory exercise. This helped to ensure that similar steps were followed for all replications.

As we will show in Section 3, variation in the fault detection rate is not explained by selection, maturation, or replication effects.

Finally, instrumentation effects may result from differences in the specification documents. Such variation is impossible to avoid, but we controlled 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 such threats:

1. The subjects in our initial runs may not be representative of software programming professionals. Although more than half of the subjects have 2 or more years of industrial experience, they are graduate students, not software professionals. Furthermore, as students they may have different motivations for participating in the experiment. This shouldn't be a problem in the replication using professional subjects.

2. The specification documents may not be representative of real programming problems. Our experimental specifications are atypical of industrial SRS in two ways. First, most of the experimental specification is written in a formal requirements notation. (See Section 2.2.) Although several groups at AT&T and elsewhere are experimenting with formal notations [2, 8], it is not the industry's standard practice. Secondly, the specifications are considerably smaller than industrial ones.

3. The inspection process in our experimental design may not be representative of software development practice. We have modeled our experiment's inspection process after the one used in several development organizations within AT&T [6]. Although this process is similar to a Fagan-style inspection, there are some differences. One difference is that reviewers use the fault detection activity to to find faults, not just to prepare for the inspection meeting. Another difference is that during the collection meeting reviewers are given specific technical roles such as test expert or end-user only if the author feels there is a special need for them.

   Our process also differs slightly from the AT&T process. For example, the SRS authors are not present at our collection meetings, although, in practice, they normally would be. Also, industrial reviewers may bring more domain knowledge to an inspection than our student subjects did.

2.1.5 Analysis Strategy

Our analysis strategy had two steps. The first step was to find those independent variables that individually explain a significant amount of the variation in the team detection rate. The second step was to evaluate the combined effect of the variables shown to be significant in the initial analysis. Both analyses use standard analysis of variance methods (see [5], pp. 165*ff and 210ff* or [9]). Once these relationships were discovered and their magnitude estimated, we examined other data, such as correlations between the categories of faults detected and the detection methods used that would confirm or reject (if possible) a causal relationship between detection methods and inspection performance.

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 other three sections are specified using the SCR tabular requirements notation [10].

For this experiment, all three documents were adapted to adhere to the IEEE suggested format [12]. All faults present in these SRS appear in the original documents or were generated during the adaptation process; no faults were intentionally seeded into the document. The authors discovered 42 faults in the WLMS SRS; and 26 in the CRUISE SRS. The authors did not inspect the ELEVATOR SRS since it was used only for training exercises.

Elevator Control System (ELEVATOR) [20] describes the functional and performance requirements of a system for monitoring the operation of a bank of elevators (16 pages).

Water Level Monitoring System (WLMS) [18] describes the functional and performance requirements of a system for monitoring the operation of a steam generating system (24 pages).

Automobile Cruise Control System (CRUISE) [14] describes the functional and performance requirements for an automobile cruise control system (31 pages).

2.2.2 Fault Detection Methods

To make a fair assessment of the three detection methods (Ad Hoc, Checklist, and Scenario) each method should search for a well-defined population of faults. To accomplish this, we used a general fault taxonomy to define the responsibilities of Ad Hoc reviewers.

The checklist used in this study is a refinement of the taxonomy. Consequently, Checklist responsibilities are a subset of the Ad Hoc responsibilities.

The Scenarios are derived from the checklist by replacing individual Checklist items with procedures designed to implement them. As a result, Scenario responsibilities are distinct subsets of Checklist and Ad Hoc responsibilities. The relationship between the three methods is depicted in Figure 2.

The taxonomy is a composite of two schemes developed by Schneider, et al. [17] and Basili and Weiss [3]. Faults are divided into two broad types: omission – in which important information is left unstated and commission – in which incorrect, redundant, or ambiguous information is put into the SRS by the author. Omission faults were further subdivided into four categories: Missing Functionality, Missing Performance, Missing Environment, and Missing Interface. Commission faults were also divided into four categories: Ambiguous Information, Inconsistent Information, Incorrect or Extra Functionality, and Wrong Section. (See Appendix A for complete taxonomy.) We provided a copy of the taxonomy to each reviewer. Ad Hoc reviewers received no further assistance.

Checklist reviewers received a single checklist derived from the fault taxonomy. To generate the checklist we populated the fault taxonomy with detailed questions culled from several industrial checklists. Thus, the checklist items are similar in style to those found in several large organizations. All Checklist reviewers used the same checklist. (See Appendix B for the complete checklist.)

Finally, we developed three groups of Scenarios. Each group of Scenarios was designed for a specific subset of the Checklist items:

1. Data Type Inconsistencies (DF),
2. Incorrect Functionalities (IF),
3. Missing or Ambiguous Functionalities (MF).

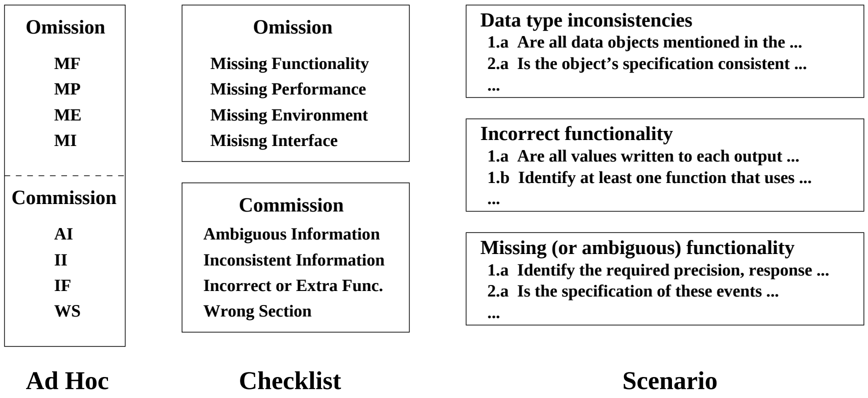

Figure 2: **Relationship Between Fault Detection Methods.** The figure depicts the relationship between the fault detection methods used in this study. The vertical extent represents the coverage. The horizontal axis labels the method and represents the degree of detail (the greater the horizontal extent the greater the detail). Moving from Ad Hoc to Checklist to Scenario there is more detail and less coverage. The gaps in the Scenario and Checklist columns indicate that the Checklist is a subset of the Ad Hoc and the Scenarios are a subset of the Checklist.

After the experiment was finished we applied the Scenarios ourselves to estimate how broadly they covered the WLMS and CRUISE faults (i.e., what percentage of defects could be found if the Scenarios are properly applied.) We estimated that the Scenarios address about half of the faults that are covered by the Checklist. Appendix C contains the complete list of Scenarios.

2.2.3 Fault Report Forms

We also developed a Fault Report Form. Whenever a potential fault was discovered – during either the fault detection or the collection activities – an entry was made on the form. The entry included four kinds of information: Inspection Activity (Detection, Collection); Fault Location (Page and Line Numbers); Fault Disposition, (Faults can be True Faults or False Positives); and a prose Fault Description. A small sample of a Fault Report appears in Figure 3.

2.3 Experiment Preparation

We attempted to ensure that the operation of the experiment was the same for all replications of the experiment. However, as we describe in the following Sections, we made several allowances for the schedules of our professional subjects.

The participants were given two, 75 minute lectures on software requirements specifications, the SCR tabular

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. **Activity** (Read/Coll.)
Location(s) 6:12 **Disposition** (T/F)

The initialization of the variable %watchdog% causes an incorrect transition into a failure mode.

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

The "Low Water indicator" is allowed to have the "on" and "off" values when the system is in test mode between 2 and 4s.

Figure 3: **Reviewer Fault Report Form.** This is a small sample of the fault report form completed during each reviewer's fault detection. Faults number 10 and 11, found by reviewer 12 of team C for the WLMS specification are shown.

requirements notation, inspection procedures, the fault classification scheme, and the filling out of data collection forms. The references for these lectures were Fagan [7], Parnas [15], and the IEEE Guide to Software Requirements Specifications [1]. The participants were then assembled into three-person teams – 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 faults they found on a Fault Report Form. Their efforts were restricted to two hours. Later we met with the participants and answered questions about the experimental procedures. Afterwards, each team conducted a supervised collection meeting and filled out a master Fault Report Form for the entire team. The ELEVATOR SRS was not used in the remainder of the experiment.

2.4.2 Experimental Phase

This phase involved two inspection rounds. The instruments used were the WLMS and CRUISE specifications discussed in Section 2.2.1, a checklist, three groups of fault-based scenarios, and the Fault 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.

During the first Round, one half of the teams were asked to inspect the CRUISE specification; the remaining teams inspected the WLMS specification. The detection methods used by each team are shown in Table 1. Fault detection was limited to two hours, and all potential faults were reported on the Fault Report Form. After fault detection, all materials were collected. For the student subjects we set aside 28 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. For the professional subjects we allowed each team to schedule their own working times and to control access to their experimental materials. We asked them to follow the time guidelines and to complete each task in one sitting. In post-experiment interviews none of the professional subjects told us that they were unable to comply with our instructions.

Once all team members had finished fault detection, the team's moderator arranged for the collection meeting. At the collection meeting, the reader paraphrases each requirement. During this paraphrasing activity, reviewers may bring up any issues found during preparation or discuss new issues. The team's recorder maintained the team's master Fault Report Form. Collection was also limited to 2 hours and the entire Round was completed in one week. The collection meeting process is the same regardless of which fault detection method was used during fault detection.

The second Round was similar to the first except that teams who had inspected the WLMS during Round 1 inspected the CRUISE in Round 2 and vice versa.

3 Data and Analysis

3.1 Data

Three sets of data are important to our study: the fault key, the team fault summaries, and the individual fault summaries.

Rev	Method	Sum	1	2		21		32		41	42
42	Data inconsistency	9	0	0		0		0		0	0
43	Incorrect functionality	6	0	1		0		0		0	0
44	Missing functionality	18	0	0	•••	1	•••	0	•••	0	0
Team	Scenario	23	0	1		0		1		0	0
Key			AH	DT		MA		AH		DT	AH

Figure 4: **Data Collection for each WLMS inspections.** This figure shows the data collected from one team's WLMS inspection. The first three rows identify the review team members, the detection methods they used, the number of faults they found, and shows their individual fault summaries. The fourth row contains the team fault summary. The fault summaries show a 1 (0) where the team or individual found (did not find) a fault. The fifth row contains the fault key which identifies those reviewers who were responsible for the fault (AH for Ad Hoc only; CH for Checklist or Ad Hoc; DT for data type inconsistencies, Checklist, and Ad Hoc; IF for incorrect functionality, Checklist and Ad Hoc; and MF for missing or ambiguous functionality, Checklist and Ad Hoc). Meeting gain and loss rates can be calculated by comparing the individual and team fault summaries. For instance, fault 21 is an example of *meeting loss*. It was found by reviewer 44 during the fault detection activity, but the team did not report it at the collection meeting. Fault 32 is an example of *meeting gain*; it is first discovered at the collection meeting.

The fault key encodes which reviewers are responsible for each fault. In this study, reviewer responsibilities are defined by the detection techniques a reviewer uses. Ad Hoc reviewers are responsible (asked to search for) for all faults. Checklist reviewers are responsible for a large subset of the Ad Hoc faults[4]. Since each Scenario is a refinement of several Checklist items, each Scenario reviewer[5] is responsible for a distinct subset of the Checklist faults.

The team fault summary shows whether or not a team discovered a particular fault. This data is gathered from the fault report forms filled out at the collection meetings and is used to assess the effectiveness of each fault detection method.

The individual fault summary shows whether or not a reviewer discovered a particular fault. This data is gathered from the fault report forms each reviewer completed during their fault detection activity. Together with the fault key it is used to assess whether or not each detection technique improves the reviewer's ability to identify specific classes of faults.

We measure the value of collection meetings by comparing the team and individual fault summaries to determine the meeting gain and loss rates. One team's individual and team fault summaries, and the fault key are represented in Figures 4 and Figure 5.

[4] i.e., faults for which an Ad Hoc reviewer is responsible.
[5] i.e., reviewers using Scenarios.

Rev	Method	Sum	1	2	•••	14	•••	17	•••	25	26
42	Ad Hoc	7	0	1		0		0		1	0
43	Ad Hoc	6	0	1		0		0		1	0
44	Ad Hoc	4	0	0		0		0		0	0
Team	Ad Hoc	10	0	1		1		0		1	0
Key			AH	MF		AH		AH		AH	DT

Figure 5: **Individual and Team Fault Summaries (CRUISE)**. This figure shows the data collected from one team's CRUISE inspection. The data is identical to that of the WLMS inspections except that the CRUISE has fewer faults – 26 versus 42 for the WLMS – and the fault key is different.

Our analysis is done in two steps: (1) We compared the team fault detection rates to determine whether the detection methods have the same effectiveness and (2) we analyzed the effectiveness of collection meetings to further understand differences in each method's performance.

3.2 Analysis of Team Performance

Tables 4 shows the raw team data. Six of the cells contain the average detection rate for teams using each detection method and specification (3 detection methods applied to 2 specifications). Figure 6 summarizes this data. As depicted, the Scenario detection method resulted in the highest fault detection rates, followed by the Ad Hoc detection method, and finally by the Checklist detection method. For the student population the performances of the Ad Hoc and Checklist methods were statistically indistinguishable. For the professional population the performances of the Ad Hoc method was statically superior to that of the Checklist method.

Tables 2 and 3 present a statistical analysis of the team performance data as outlined in Section 2.1.5. The independent variables are listed from the most to the least significant. For both the professionals and the students the Detection method used is significant. For the students, but not the professionals, Specification is also significant. For both groups the Round, Replication, and Order are not significant.

We also analyzed the combined Instrumentation and Treatment effects for the student performances. Since Method was the only significant independent variable for the professional subjects we did not perform this analysis on their data. The results indicates that the interaction between Specification and Method is not significant. This means that although the average detection rates varied for the two specifications (for the student subjects only), the effect of the detection methods is not linked to these differences.

Based on the preceding analyses we reject the null hypothesis that the detection methods have no effect on

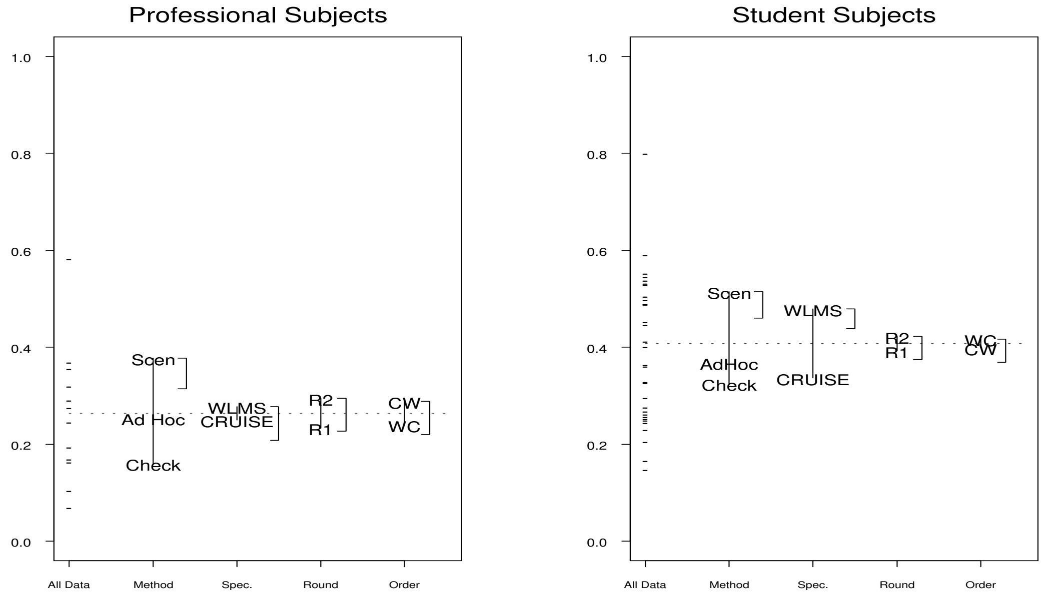

Figure 6: **Fault Detection Rates by Independent Variable.** The dashes in the each panel's far left column show each team's fault detection rate for the WLMS and CRUISE. The horizontal line is the average fault detection rate. The plot demonstrates the ability of each variable to explain variation in the fault detection rates. For the Specification variable, the vertical location of WLMS (CRUISE) is determined by averaging the fault detection rates for all teams inspecting WLMS (CRUISE). The vertical bracket,], to the right of each variable shows one standard error of the difference between two settings of the variable. The plot indicates that for both the professional and student subjects Method is significant, for the students, but not the professionals, Specification is significant; and for neither group is Round, Replication, or Order significant.

inspection performance.

3.3 Analysis of Collection Meetings

In this Section, we measure the benefits of collection meetings by comparing the team and individual fault summaries to determine the meeting gains, meeting losses and net meeting gain/loss. (See Figure 4 and Figure 5).

A "meeting gain" occurs when a fault is found for the first time at the collection meeting. A "meeting loss" occurs when a fault is first found during an individual's fault detection activity, but it is subsequently not recorded during the collection meeting. Meeting gains may thus be offset by meeting losses and the difference between meeting gains and meeting losses is the net improvement due to collection meetings. Our results indicate that collection meetings produce no net improvement for either the professional or student populations.

Independent Variable	SS_T	ν_T	SS_R	ν_R	$(SS_T/\nu_T)(\nu_R/SS_R)$	Significance Level
Detection Method – treatment	.200	2	.359	29	8.064	< .01
Specification– instrumentation	.163	1	.396	30	12.338	< .01
Inspection round – maturation	.007	1	.551	30	.391	.54
Experimental run – replication	.007	1	.551	30	.391	.54
Order – presentation	.003	1	.003	30	.141	.71
Team composition – selection	.289	15	.268	16	1.151	.39

Table 2: **Analysis of Variance for Each Independent Variable (Student subjects).** The analysis of variance shows that only the choice of detection method and specification significantly explain variation in the fault detection rate. Team composition is also not significant.

Independent Variable	SS_T	ν_T	SS_R	ν_R	$(SS_T/\nu_T)(\nu_R/SS_R)$	Significance Level
Detection Method – treatment	.095	2	.053	9	7.942	< .01
Specification– instrumentation	.002	1	.147	10	.158	.70
Inspection round – maturation	.011	1	.137	10	.837	.38
Order – presentation	.007	1	.141	10	.510	.49
Team composition – selection	.099	5	.051	6	2.37	.16

Table 3: **Analysis of Variance for Each Independent Variable (Professional subjects).** The analysis of variance shows that only the choice of detection method is significant.

Specification	**Detection Method**		
	Ad Hoc	**Checklist**	**Scenario**
WLMS	(.19) (.29) .29 .38 .45 .48 .5 .5	(.17) (.17) .29 .33 .5 .52	(.31) .4 .55 .55 (.55) .57 .62 .74
(average)	(.24) .43	(.17) .41	(.43) .57
Cruise	.23 .23 .27 .27 (.27) (.27) .35 .38 .46	(.12) .19 (.19) .23 .23 .31	(.27) .35 (.38) .42 .42 .5 .54
(average)	(.27) .31	(.15) .24	(.33) .45

Table 4: **Team Fault Detection Rate Data.** The nominal and average fault detection rates for all 24 teams. The performances of the professional subjects are enclosed in parentheses.

3.3.1 Meeting Gains

Figure 7 displays the meeting gain rates for all inspections. Overall the meeting gain rate is 5.0% $\pm$ 6.3% (3.9% $\pm$ 3.4%) for professionals (students). The meeting gain rate is 6.7% $\pm$ 10.2% (4.7% $\pm$ 5.3%) for WLMS inspections and 3.2% $\pm$ 7.2% (3.1% $\pm$ 4.3%) for CRUISE inspections. The rates are not significantly different between different populations or different specifications. It is interesting to note that these results are consistent with an earlier industrial case study by Votta [19].

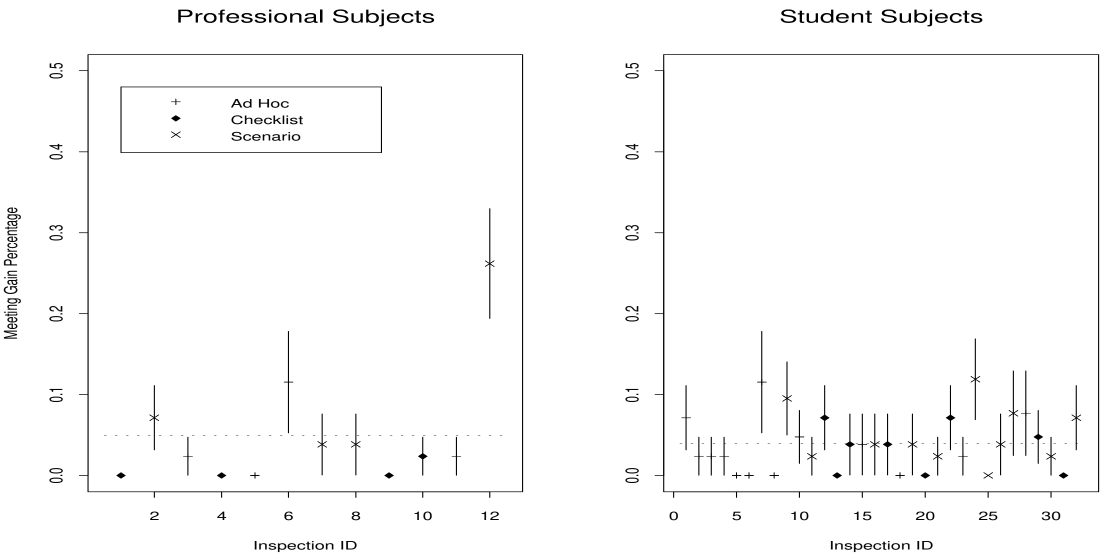

Figure 7: **Meeting Gains for all Inspections.** Each point represents the meeting gain rate for a single inspection, i.e., the number of faults first identified at a collection meeting divided by the total number of faults in the specification. Each rate is marked with symbol indicating the inspection method used. The vertical line segment through each symbol indicates one standard deviation in the estimate (assuming each fault was a Bernoulli trial). This information helps in assessing the significance of any one rate. The average meeting gain rate is $5.0\% \pm 6.3\%$ for the professionals. ($3.9\% \pm 3.4\%$ for the students.)

3.3.2 Meeting Losses

The overall average meeting loss rates were $6.7\% \pm 7.2\%$ and $7.2\% \pm 4.6\%$ for the professionals and students respectively. (See Figure 8.) The meeting loss rates for the WLMS were $8.3\% \pm 11.2\%$ ($6.8\% \pm 6.3\%$) for the professionals (students), while the loss rates for the CRUISE were $5.1\% \pm 9.0\%$ ($7.7\% \pm 6.6\%$). Again there was no statistically significant difference between the loss rates of the different populations or the different specifications. One cause of meeting loss might be that reviewers are talked out of the belief that something is a fault. Another cause may be that during the meeting reviewers forget or can not reconstruct a fault found earlier.

This effect has not been previously reported in the literature. However, since the interval between the detection and collection activities is usually longer in practice than it was in our experiment (one to two days in our study versus one or two weeks in practice), this effect may be quite significant.

3.3.3 Net Meeting Improvement

The average net meeting improvement is $-1.7\% \pm 3.9\%$ for professional inspections and $-3.3\% \pm 1.4\%$ for student inspections. For the WLMS the net improvement was $-1.5\% \pm 8.5\%$ ($-2.0\% \pm 2.9\%$) for professionals and students

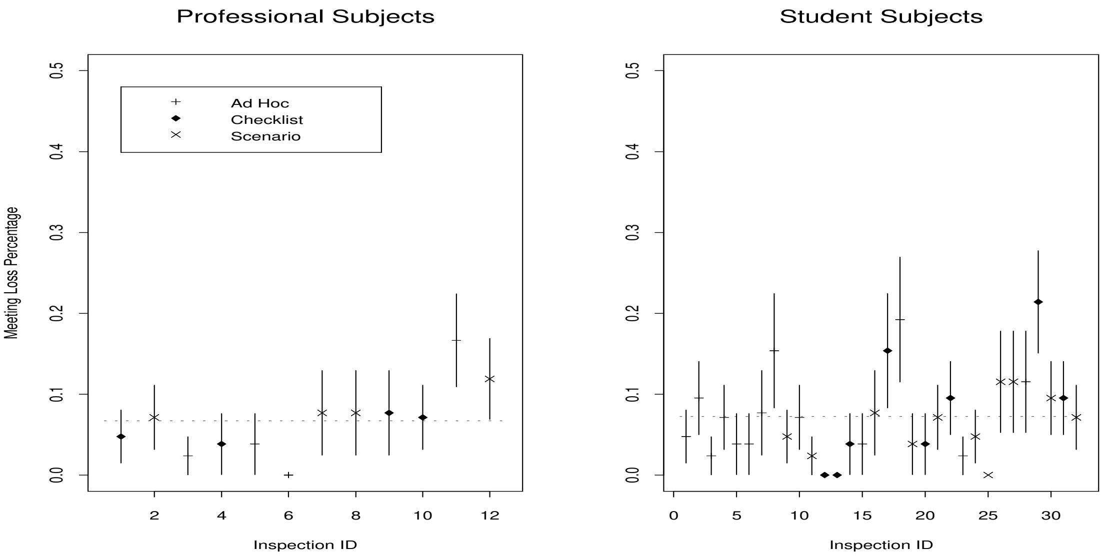

Figure 8: **Meeting Loss Rate for all Inspections.** Each point represents the meeting loss rate for a single inspection. The meeting loss rate is the number of faults first detected by an individual reviewer divided by the total number of faults in the specification. Each rate is marked with a symbol indicating the inspection method used. The vertical line segment through each symbol indicates one standard deviation in the estimate of the rate (assuming each fault was a Bernoulli trial). This information helps in determining the significance of any one rate. The average team loss rate is $6.7\% \pm 7.2\%$ for the professionals. ($7.2\% \pm 4.6\%$ for the students).

respectively. For the CRUISE the net improvement was $-1.9\% \pm 6.6\%$ ($4.5\% \pm 2.8\%$). (Figure 9 displays the net meeting improvement for all inspections.) We found no correlations between the loss, gain, or net improvement rates and any of our experiment's independent variables.

4 Summary and Conclusions

This article presents the results from a replication of an experiment to compare different defect detection methods for inspecting software requirements specifications. One possible limitation of the original experiment was that it used graduate students in computer science as subjects. If, during inspections, students behave very differently than software professionals, then the original experiment's results will be invalid. To address this concern we reran the experiment using software development professionals as subjects. One of our major findings is that, although the performances of the student and professional populations were different, all of the hypothesis tests gave the same results. This doesn't imply that studies with professional are no longer needed, but it suggests that student studies shouldn't automatically be discounted. This is very important because studies with professionals

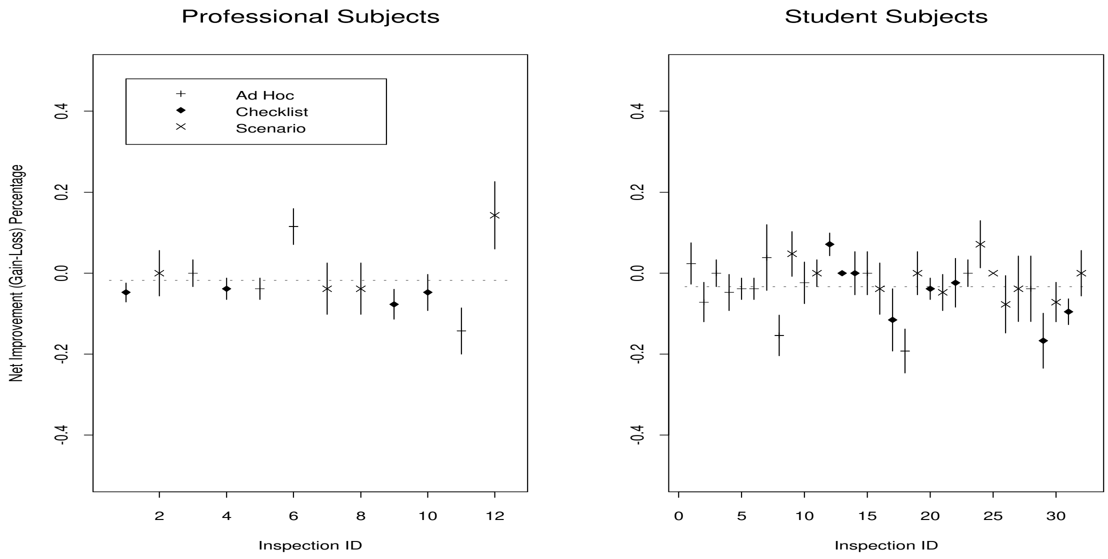

Figure 9: **Net Meeting Improvement for all Inspections.** Each symbol indicates the net meeting improvement for a single inspection. The average net meeting improvement rate is $-1.7\% \pm 3.9\%$ for the professionals. ($-3.3\% \pm 1.4\%$ for the students). These rates are not significantly different from 0.

are much more expensive than are studies with student subjects.

In both the student and professional populations we found the following results:

1. **The fault detection rate when using Scenarios was superior to that obtained with Ad Hoc or Checklist methods – an improvement from 21% to 38% in the professional population and from 35% to 51% in the student population.**

2. **The Checklist method – the industry standard, was no more effective than the Ad Hoc detection method when used by either subject population.**

3. **On the average, collection meetings contributed nothing to fault detection effectiveness.**

The results of this work have important implications for software practitioners. The indications are 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, in which reviewers have neither systematic procedures nor clearly defined responsibilities.

Economical experimental designs are necessary to allow replication in other environments with different populations. For software researchers, this work demonstrates the feasibility of constructing and executing inexpensive experiments to validate fundamental research recommendations.

These results also call into question the common practice of disregarding studied done with student subjects. The far more important question is clearly when do student subjects provide an adequate model of the professional population.

5 Future Work

The experimental data raise many interesting questions for future study.

- Very few faults are initially discovered during collection meetings. Therefore, in view of their impact on development interval (calendar time to complete development), are these meetings worth holding?
- More than half of the faults are not addressed by the Scenarios used in this study. What other Scenarios are necessary to achieve a broader fault coverage?
- We strongly suspect that Scenarios will have to be developed and customized to individual environments. These experiments only evaluate the concept of Scenarios, but do not give sufficient detail to allow others to develop their own. We are currently working to formalize the Scenario approach and to create a methodology for developing them.

Acknowledgments

We would like to thank Victor Basili for his contributions to this work. We would also like to recognize the efforts of the experimental participants – an excellent job was done by all.

References

[1] *IEEE Guide to Software Requirements Specifications.* Soft. Eng. Tech. Comm. of the IEEE Computer Society, 1984. IEEE Std 830-1984.

[2] Mark A. Ardis. Lessons from using basic lotos. In *Proceedings of the Sixteenth International Conference on Software Engineering*, pages 5–14, Sorrento, Italy, May 1994.

[3] 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.

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

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

[6] 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, May 1992.

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

[8] S. Gerhart, D. Craigen, and T. Ralston. Experience with formal methods in critical systems. *IEEE Software*, 11(1):21–28, January 1994.

[9] R. M. Heiberger. *Computation for the Analysis of Designed Experiments.* Wiley & Sons, New York, New York, 1989.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] Dave L. Parnas and David M. Weiss. Active design reviews: principles and practices. In *Proceedings of the 8th International Conference on Software Engineering*, pages 215–222, Aug. 1985.

[16] Adam Porter, Lawrence G. Votta, and Victor Basili. Comparing detection methods for software requirement inspections: A replicated experim ent. *IEEE Transactions on Software Engineering*, 21(6):563–575, June 1995.

[17] G. Michael Schneider, 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.

[18] J. vanSchouwen. 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.

[19] 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.

[20] 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 fault taxonomy is due to the work of Schneider, et al., and Basili and Weiss.

- Omission
  - Missing Functionality: Information describing the desired internal operational behavior of the system has been omitted from the SRS.
  - Missing Performance: Information describing the desired performance specifications has either been omitted or described in a way that is unacceptable for acceptance testing.
  - Missing Interface: 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: 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: 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: 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: Some sentence contained in the SRS asserts a facts that cannot be true under the conditions specified in the SRS.
  - Wrong Section: 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 Or 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?