An Empirical Study on Software Error Detection: Voting, Instrumentation, and Fagan Inspection *

Sunsup So, Yongseop Lim, Sung Deok Cha, and Yong Rae Kwon
Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1, Kusong-dong, Yusong-gu, Teajon 305-701, Korea
E-mail: {triples, yslim, cha, yrkwon}@salmosa.kaist.ac.kr

Abstract

This paper presents the results of an experiment that compared error detection capability of voting, instrumentation, and Fagan inspection methods. Several experiments have measured effectiveness of various error detection methods. However, most experiments have used different programs; consequently, the results are generally incompatible and do not allow one to make objective comparison on the cost-effectiveness of various approaches. No software can be developed using unlimited amount of resources, practitioners need empirical and objective data on the cost-effectiveness of various error detection methods to decide which methods to use during software development.

Results of this experiment is significant because these methods have been applied to the same program. Futhermore, the participant's educational and industrial experience are comparable to that of the previous experiments.. We confirmed the previous finding that detecting errors in reliable programs is difficult; none of the three methods detected more than half of all the known errors in the programs. Of the three methods employed, participants detected more errors by using Fagan inspection method than they did by voting or instrumentation. When the average number of hours needed to detect an error was compared, Fagan inspection method was shown to be more cost-effective than instrumentation method.

1. Introduction

Critical digital systems can fail because of errors in either software or hardware. Although hardware unreliability was known to be a major contributor to system failures in the past, it is fast becoming an insignificant factor with advances in hardware technology and commercial availability of fault-tolerant computers[1]. Instead software unreliability has become a major bottleneck in further improving the system reliability. In order to enhance software reliability in cost-effective manner, it is important to detect any residual software errors at the earliest possible opportunity.

Many proposals have been made in literature on how errors in software code or documentation can be detected. Some of frequently used error detection methods include voting (also known as back-to-back testing), Fagan inspection, and instrumentation (also known as self-checking code)[2,3,4,5]. Since no software can be developed using unlimited amount of resources, practitioners need empirical and objective data on the cost-effectiveness of various error detection methods to decide which methods to use during software development. Unfortunately, relatively little empirical data exist to assist practitioners in making such decisions wisely. Although several controlled experiments have measured cost-effectiveness of various error detection methods, these results remain largely incompatible (i.e., have been applied on different problems). There have been only a couple of relevent controlled experiments [6,7,8] that compared several error detection methods using the same program.

Knight and Leveson [5] developed 27 Launch Interceptor Program (referred to as LIP hereafter) versions to determine if independently developed versions fail in statistically independent manner. Leveson et al. used eight of the 27 LIP versions in a subsequent experiment [6] to compare error detection capabilities of voting and instrumentation methods. In order to conduct an instrumentation experiment, they hired 24 graduate students majoring in computer science at UCI and UVA for a week. Each version was assigned 3 students who reviewed specification, performed informal and individual code reading, and developed self-checks to detect software errors. They placed no restrictions on the participants as to how many self-checks to develop and

* This work is partially supported by Korea Institute of Nuclear Safety under contract number KS95-018 and by Korea Atomic Energy Research Institute under contract number KAERI/CM-103/94

0-8186-7171-8/95 $04.00 © 1995 IEEE

where to place the self-checks. The conclusion from the instrumentation experiment was that voting and instrumentation methods are complementary error detection methods to each other in that each method detected the same number of but different types of errors in the programs.

Another controlled experiment on the subject was conducted by Shimeall [7, 8]. His experiment compared error detection capability of functional testing, back-to-back testing, instrumentation, code reading, and static data-flow analysis. He found that voting and functional testing techniques detected more errors than the other techniques did and that each of the surveyed techniques were complementary to others in that each technique tended to detect largely distinct types of errors.

Although there have been several controlled experiment on Fagan Inspection methods, there have been no experiments that compared Fagan inspection method against other error detection methods such as voting or instrumentation *using the same program*. Since Fagan inspection method is gaining acceptance among practitioners and researchers alike as a promising a cost-effective method, we felt important to extend the previous experiment [6] to include Fagan inspection method.

The rest of this paper is organized as follows. Section 2 briefly summarizes Fagan inspection method and its industrial application experiences. Section 3 describes our experimental design, and Section 4 discusses the results obtained from the experiment. Section 5 concludes the paper and discusses future research directions.

2. Fagan Inspection

Fagan inspection [9,10,11,12] process involves systematic group review of code (or related artifacts such as requirements or design documents). Each team member participating in Fagan inspection has well-defined roles, and the team conducts reviews following specific guidelines. An inspection team is most productive when its team members work in harmony and fulfill the assigned roles.

Typical Fagan inspection team consists of moderator, reader, inspector, and author. Moderator manages the inspection meetings and is responsible for scheduling and result archiving. Moderator plays especially important leadership roles and must ensure that the team stays focused on detecting errors without being sidetracked (e.g., suggesting necessary corrections or desirable enhancements). Properly trained and technically experienced personnel who have worked on similar projects to the one being inspected would be a good candidate to serve as a moderator. Reader's role is to paraphrase the product being reviewed at a reasonable pace. Otherwise, an inspection team might be tempted to inspect software too quickly and superficially; consequently, the team may not detect as many errors as they could otherwise. Author's presence in the inspection meetings is generally considered beneficial because; (1) author can assist the inspection team to understand the product and (2) author can understand exact nature of the errors the inspection team found. An inspector's role is to examine software from a tester's viewpoint. All team members, including moderator and reader, also play inspector's role and should actively participate in error detection process.

Fagan inspection process consists of the following steps, each with a specific objectives: planning, overview, preparation, inspection, rework, and follow-up.

- **PLANNING:** When materials to be inspected pass entry criteria (i.e., source code successfully compiles without syntax errors), inspection team members are selected, and inspection schedules (e.g., time and place) are established.
- **OVERVIEW:** Team members are briefed on the material to be inspected, and roles are assigned.
- **PREPARATION:** Team members review the material individually to prepare themselves to fulfill the assigned roles.
- **INSPECTION:** The team conducts an inspection meeting to find errors and records errors detected. Purpose of the inspection meeting is solely on the error detection, and any attempts to find alternative solutions should be strongly discouraged by the moderator.
- **REWORK:** The author revises the product to fix all the detected errors.
- **FOLLOW-UP:** Moderator or the entire inspection team reviews the product again to assure that all fixes are effective and that no additional defects have been introduced during rework.

Experiences obtained from industrial application of Fagan inspections strongly suggests that omitting or combining any of the six inspection steps is undesirable in that degraded inspection efficiency outweighs apparent short-term cost saving. Since Fagan inspection process is quite labor-intensive, each session is recommended not to last more than two hours, and a team is recommended not to schedule more than two inspection sessions per day.

Since not all inspectors start off being good detectives, a checklist that describes the types of frequently detected errors is provided to assist inspection teams and to classify the detected errors. Typical checklist include the following types of errors: data reference (e.g., out-of-bound array subscript), data declaration, computation, comparison, control-flow,

interface, input/output, logic, bad styles, and others. Detected errors are normally recorded as follows:

"In module:XXX, Line: YYY, NAME-CHECK is performed one less time than required --- LO/W/MAJ."

Error description is self-explanatory. Error classification "LO/W/MAJ" means that this is a error in Logic and that an incorrect (i.e., Wrong) logic is detected - as opposed to Missing or Extra - and that it is a major defect - as opposed to Minor - that would result in a malfunction or unexpected result if left without being corrected. Minor errors are usually critiques on programming styles.

3. Experimental Design

Our experiment was carried out as a term project in a graduate-level software engineering class at the Korea Advanced Institute of Science and Technology (KAIST). Participants in the experiment were volunteers, and unfortunately, we were unable to recruit sufficient number of volunteers to perform Fagan inspections on all 8 versions the previous experiment used. We were also unable to find volunteers to formulate "full" Fagan inspection teams. Therefore, we selected 4 out of the 8 LIP programs used in the previous experiment [7] and formed three different inspection teams (referred to as the team A, B, and C hereafter) so that each team inspected all of the four versions. They were known as version 3, 6, 12, and 25 and had 757, 643, 573, and 906 lines of Pascal code, respectively, including the comments. Each team had only two members who actively played the roles of a moderator and a reader as well as inspectors. Because the LIP versions had been developed elsewhere, the authors could not have been a part of the inspection teams. Each inspection team tried to fully adhere to the principles and recommendations of Fagan inspection technique as practically as possible.

Team A consisted of a first-year graduate student in software engineering at KAIST and a technical staff member from Agency for Defense Development (ADD) with seven years of industrial experience and a M.S. degree in software engineering. Team B consisted of two technical staff members from ADD each with four years of industrial experience. Team C consisted of two graduate students at KAIST. One had 3 years of industrial and 5 years of graduate study experiences in computer science, while the other was enrolled in a second year in the M.S. program in computer science with no industrial experiences. None of the participants had prior knowledge on Fagan inspection method, LIP programs, or errors that have been found on the LIP programs.

Because rework and follow-up phases were unnecessary for our experiment, only the first four phases of Fagan inspection were applied. During the overview and preparation phases, experiment administrators briefed the team members on the purpose of the experiment, Fagan inspection method, the checklist, and the LIP specification. These sessions took three hours for team A and 1 hour for teams B and C, respectably. Checklist was used merely as a guideline, and the inspection teams were not restricted to report only the errors mentioned in the checklist. Care was taken to accurately record the amount of time each team spent on various inspection activities. At the end of the inspection meetings, list of detected errors was submitted to the experiment administrators along with the time sheet.

4. RESULTS

Results obtained from the experiment are presented in two parts. We present data on the amount of time spent on Fagan inspection by each team as well as number and types of detected errors. In the second part, using results reported in [6], we compare error detection capabilities of voting, instrumentation, and Fagan inspection methods.

4.1 LIP Errors Detected by Fagan Inspection

Figure 1 shows the number of hours spent by each Fagan inspection team as well as the average number of source code lines inspected per hour.

Figure 2 presents the number of major and minor errors detected by three teams. Major errors refer to the ones that would cause production of incorrect outputs while minor errors refer to those that would have no impact on the correctness of the outputs but could influence other aspects of software quality such as maintainability, performance, etc. Therefore, we consider only the major errors when the effectiveness of Fagan inspection method is compared against that of voting and instrumentation.

We performed a statistical analysis, known as *t* test, to determine if there were significant relationship between the number of hours spent on inspection and the number of errors detected. We found no statistically significant relationship between the number of errors detected and the number of hours spent. That is, more time spent on the inspections didn't necessarily result in detecting more errors. However, Figure 2 shows significant variation on the number of errors detected by each team, and the team's (or team member's) ability seems to be the most significant factor in detecting errors.

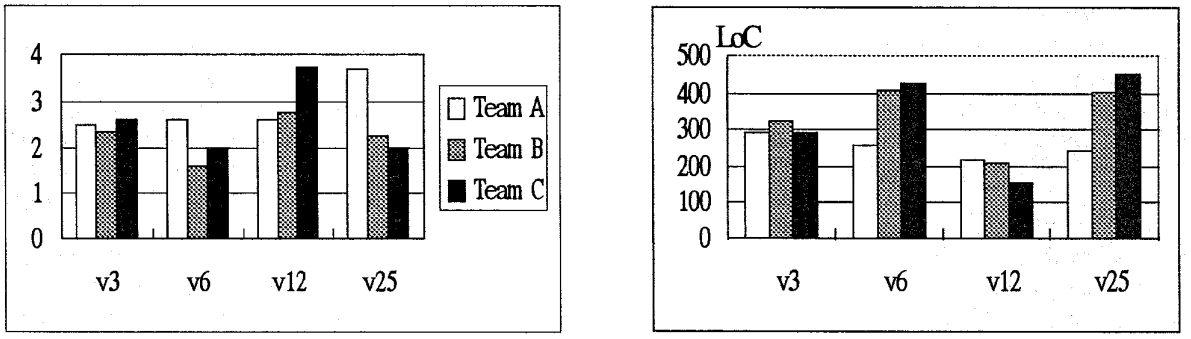

Figure 1. Number of hours spent by each team on Fagan Inspection (left) and Lines of Code inspected per hour by each team (right)

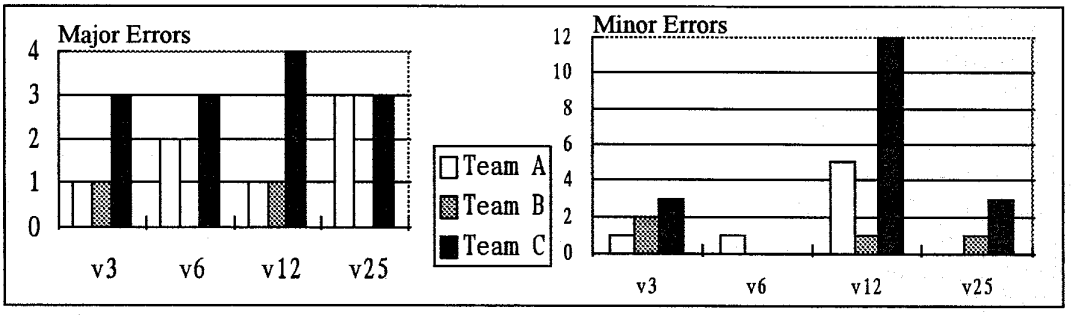

Figure 2. Major (left) and Minor (right) errors detected in each program

Table 1 shows how many of the previously known errors in the LIP programs were detected by each Fagan inspection. List of known faults are derived from the results reported in [6, 13]. It shows that nearly half of all the known faults were detected by Fagan inspection method. Surprisingly, Fagan inspection method detected one error in version 6 (referred to as error 6.5 hereafter) that had ***not*** been detected despite extensive voting and instrumentation effort that had been applied on this version. This finding is impressive because (1) LIP versions were known for their high (i.e., over 99.99% on the average) reliability when they were initially developed; (2) each version had been tested on more than 1,100,000 randomly generated test cases based on realistic operational profiles; and (3) each version had been subject to informal code reading by three participants during instrumentation experiment. Erroneous code fragment, designed to compute the slope between the two points *(xa,ya)* and *(xb,yb)*, is shown below:

```
. . . .
else if (realcompare(xa-xb,0.0) <> eq ) and
        (realcompare(xb-xc,0.0) = eq) then
            begin
                . . . .
                mab := (ya-yb)/(xa-xc);
                mab := -1.0/mab;
. . . .
```

This code fragment is unstable under a very few condition even though **xb** is *considered to be equal* to **xc** (i.e., realcompare (xb-xc)=eq). More specifically, this formula is incorrect because actual computation *mab* do not depend on finite precision arithmetic that **realcompare** function uses. Therefore, the value of *xc* is approximately (but not exactly) equal to that of *xb*, and the value of *mab* is assigned an incorrect value. This error is quite subtle and left undetected. Had the program been executed with test cases that would have caused failures, voting method would have detected the error. As for instrumentation, one of the three participants failed to detect the error during informal code reading but added a valid self-check that would have detected the error. However, the valid self-check was never activated when tested by test cases generated based on operational profile. This result shows that randomly generated test cases alone may be insufficient criteria to effectively expose residual software errors.

Unfortunately, Fagan inspection teams made several inaccurate or incorrect error identification. Such possibility is perhaps inevitable given that Fagan inspection involves group and manual review of source code by humans whose fallibility is well-understood. In

our experiment, team A indicated a correct code block as a faulty one in the version 25, and team C identified two correct code fragments as faulty ones in the version 3(see table 1). We believe that the possibility of such shortfalls can be reduced if Fagan inspection process is augmented with a set of software tools such as dynamic testing tools to assist inspection activities.

4.2 Comparisons with Instrumentation and Voting

Table 2 compares the errors in the LIP programs detected by voting, instrumentation, and by Fagan inspection method. Fault numbers in the table are assigned the same ones assigned in the previous experiments [6]. Fault 6.5 refers to the error that had not been detected in the previous experiments. Voting is considered to have detected an error if any of the voting triples or pairs detected the error (i.e., failing to produce correct and majority result). Likewise, instrumentation and Fagan inspection methods were considered to have detected an error if at least one of the three instrumented versions or Fagan inspection teams correctly reported the presence of the error, respectively.

Fagan inspection technique detected a few more errors than voting and inspection methods did. This result demonstrates that Fagan inspection method is as effective (if not more) as voting or instrumentation in detecting errors. Fagan inspection method detected all the errors (except error 3.4) that voting pairs or triples did. Similarly, all but two errors detected by instrumentation method were also detected by Fagan inspection method. If one were to compare the number of errors detected during individual code reading performed during instrumentation experiment (shown as $ in the table) against the number of errors detected by Fagan inspection method, Fagan inspection is shown to be clearly superior method to the code reading. Success of Fagan inspection method is even more impressive if one takes into consideration that Fagan inspection teams were shorthanded. inspection method is even more impressive if one takes into consideration that Fagan inspection teams were shorthanded.

Another important consideration is the cost-effectiveness of each method. We used the average number of staff hours needed to detect an error as the criterion. This criterion excludes the possibility of direct and quantitative comparison of cost-effectiveness between voting and the other methods. Voting, in principle, may seem to cost effective because no further human intervention is required once all the independent versions and voter have been implemented. However, we have several reasons to believe that voting is not as cost-effective as instrumentation or Fagan inspection methods: (1) multiple (2- or 3-) versions must be developed, and software development processes well-known to be a costly activity. While one might argue that specification and testing costs need not be duplicated, we note that extra overhead is needed to manage multiple development teams and that such cost would not be insignificant cost. (2) Executing two- or three- versions together and comparing the outputs do not necessarily result in "complete" error detection. For example, if two versions failed to produce a majority (unanimous in this case) result, we have no option but to manually examine which of the two versions (or both) was incorrect.

Table 1. Number of faults detected and wrong identification

Number	Already Known Faults		Other Faults	Wrong
	Present	Detected Faults	Detected Faults	Identification
3A	4	1	0	0
3B		1	0	0
3C		3	0	2
6A	4	2	1	0
6B		0	0	0
6C		3	0	0
12A	4	1	0	0
12B		1	0	0
12C		4	0	0
25A	4	3	0	1
25B		0	0	0
25C		3	0	0
Total	48	22	1	3

Table 2. Number of errors detected by voting, instrumentation, and Fagan inspection. (Note: $ in the instrumentation column means that the error was detected during informal and individual code reading phase of the instrumentation.)

Error	Voting	Instrumentation	Inspection
3.1	✓		✓
3.2	✓		✓
3.3	✓	✓ $	✓
3.4	✓		
6.1	✓	✓ $	✓
6.2	✓	✓ $	✓
6.3	✓		✓
6.4		✓	
6.5 *			✓
12.1	✓	✓ $	✓
12.2	✓		✓
12.3		✓	✓
12.4		✓	✓
25.1		✓ $	✓
25.2	✓	✓ $	✓
25.3	✓	✓ $	✓
25.4		✓	
Total	11	11 (7)	14

That is, voting method can reveal that one or more errors exist in the program but provides little help in pinpointing exact error location. Based on our experience, we do not believe voting method to be a cost-effective error detection method.

Cost-effectiveness of instrumentation and Fagan inspection method can be objectively compared, and Table 3 shows data obtained from our experiment. The number of hours spent on Fagan inspection include time for planning, overview, preparation, and inspection phases. Similarly, the number of hours spent on instrumentation includes, as reported in [6], times spent reading specification, developing self-checks, implementing self-checks, and debugging instrumented versions.[1]

This result clearly shows that Fagan inspection method is more cost-effective than instrumentation in detecting errors. We note that average number of hours needed to detect an error using Fagan inspection in our experiment is comparable to what others have reported from the industrial applications of Fagan inspection method [11,12,14]. However, we urge readers to interpret our data with caution.

[1] On the other hand, one of the three participants who instrumented version 25 did not submit a time sheet and was excluded in the comparison.

Our experiment is just one sample, and more data on cost-effectiveness of instrumentation method are needed before we can make more convincing and objective conclusions on relative cost-effectiveness.

5. CONCLUSIONS

Based on our experience and empirical data we have collected to date, we believe the Fagan inspection method (or its variations) to be more cost-effective than voting or instrumentation methods. Therefore, we would, as others who applied Fagan inspection method on industrial projects almost unanimously did, highly recommend systematic application of Fagan inspection method throughout software development life cycle phases.

Despite positive results on Fagan inspection we report in this paper, we see several enhancements that can be introduced to the Fagan inspection process to further enhance its effectiveness. For example, Table 2 shows that there are two errors (errors 6.4 and 25.4) that were detected during instrumentation experiment but by none of the Fagan inspection teams. Incomplete analysis seems to be the primary factor in failing to detect error 6.4. However, interestingly enough, error 25.4 is *identical in nature* to error 12.4 which was successfully detected by the Fagan inspection method and the Fagan inspection teams reviewed the version 12 *before* they reviewed the version 25. Yet, error 25.4 was not detected while error 12.4 was successfully detected. There appears to be two contributory factors at work: (1) although these errors had common semantics, their appearances didn't share such explicit similarities; and (2) Fagan inspection teams may have forgotten about the error that had been detected earlier. Developing a tool that maintains a database of known errors and allows dynamic testing of the selected code fragment would be useful to Fagan inspection teams. Such tools could be a part of an environment whose goal is to maximize the productivity of the inspection method.

Further empirical studies that can provide *objective* comparison of various error detection methods are needed. Our experiment extended a previous experiment [6] to compare voting, instrumentation, and Fagan inspection method. We are in the process of conducting another empirical experiment, extending Shimeall's experiment[7], to compare effectiveness of Fagan inspection against functional testing, voting, instrumentation, and static analysis.