Evaluating Efficiency and Effectiveness of Code Reading Technique with an Emphasis on Enhancing Software Quality

Syed Usman Ahmed
Department of Information Technology
Jodhpur Institute of Engg. & Technology, Jodhpur

Rajendra Purohit
Department of Information Technology
JIET School of Engg. & Technogy for Girls, Jodhpur

ABSTRACT
Code review is done to identify bugs or errors in a pre-released source code of any software work product. However it is also clear that some code review techniques that we follow are not totally effective and efficient in nature. This paper proposes a way to evaluate code review technique using Analysis of Variance (ANOVA) technique. This evaluation finds out the effect of experience of subject, Lines of code review, order of code review and day of code review on efficiency and effectiveness of code review technique. This evaluation will in turn be used to analyze the null hypothesis that will be created using ANOVA technique. Based on the significance value (p-value) obtained in the ANOVA test we will accept or reject the null hypothesis that we created to test the efficiency and effectiveness of code reading or review technique**.**

Keywords
Code review, ANOVA technique, effectiveness, efficiency, Static testing.

1. INTRODUCTION
Testing is the most important and widely used aspect of software engineering discipline today. Although finding defects is the ultimate goal of any type of testing, however it also serves as a helping hand for validation and verification activities. Improving the efficiency of defect removal will surely increase the reliability of software product and in turn are the goals of software testing. Software developers use various methods and techniques for finding defects in software work products. For finding errors or defects they may execute a program on a computer to reveal and observe failure or use their experienced vision to identify the defects and errors in the program source code [1]. Performing sound testing has now become a matter of reputation this is very well exemplified by a automobile giant, Toyota Company when it called of approximately 133,000 Prius and 14,500 Lexus vehicles to update the antilock breaking system [2]. Despite of various other ways of testing and controlling the software quality, software testing still upholds the torch of maintaining software quality assurance and control in the industry.

2. LITERATURE SURVEY
Software development is aimed to provide either software or a service for its clients. In future of software engineering (FOSE) a road map for testing was presented [4]. This road map laid stress on some fundamental research work and one of the parameter of fundamental research was demonstrating effectiveness of testing techniques using empirical studies. In FOSE 2007 [3] it was mentioned that additional research was needed to provide three types of evidences: analytical, statistical or empirical of the effectiveness of test selection criteria in revealing faults in order to understand the classes of faults for which the criteria are useful. In FOSE 2007 empirical body of evidence was identified as one of the important challenges. It is mentioned in [3] that in every topic of software engineering research, empirical studies are essential to evaluate proposed techniques and practices, to know how and when they work and to improve on them. This research work got its motivation for developing an empirical body of knowledge which is at the basis for building and evolving the theory for testing.

Moreover in a official report "State of code review 2013" [5] released by SmartBear software revealed that over 70% of respondent said that they do collaborative review in some capacity and those who do review are twice as likely as highly satisfied with their overall software quality. Over 90% of respondent said that conducting code review is important.

As per the current industry standard the software testing techniques can be classified into two basic categories: *static testing* and *dynamic testing*. If in a testing technique we require to execute the actual code and find out the bug or defects or errors then it falls under dynamic testing technique, whereas those testing technique in which execution of final code is not required for locating defects or bugs or errors are called as static testing techniques [7]. ***Code review*** is a systematic examination of source code and it is intended to detect and isolate mistakes overlooked in the development phases. Code review improves both the quality of software and the developers' skills. There are various forms of reviews: peer review (*informal*), walkthrough (*informal*), inspection (*formal*).

Dynamic Testing / Execution based techniques focus on the range of ways that are used to ascertain software quality and validate the software through actual executions of the software under test [9].

3. RELATED WORK
The research on the comparison of testing technique traces back to as early as 35 years ago with Hetzel making a start in 1976 by conducting a controlled experiment in order to analyze three defect detection methods [8]. The most commonly studied factors in the experiments evaluating testing techniques are their effectiveness (i.e., number of detected defects) and efficiency (i.e., effort required to apply the technique) in programs [9]. By tracing the major research results that have contributed to the growth of software testing techniques we can analyze the maturation of software testing techniques research. We can also assess the change of research paradigms over time by tracing the types of research questions and strategies used at various stages [10]. Three directions of research have been found related to evaluation of testing techniques [9]:

1) Actual evaluations and comparisons of testing techniques based either on analytical or empirical methods.
2) Evaluation frameworks or methodologies for comparing and/or selecting testing techniques.
3) Surveys of empirical studies on testing techniques which have summarized available work and have highlighted future trends.

However, the most significant study was conducted by [11]. This experiment studied the effectiveness and efficiency of different code evaluation techniques. The work of Basili and Selby was first replicated by [1]. This replication assumed the same working hypotheses as in initial experiment, but the experiment changed the programming used of the source code. A fault isolation phase was also added in the experiment [9]. Their work was replicated again by [12]. Their experiment followed exactly the same guidelines as the experiment run by Kamsties and Lott (who had built a laboratory package to ease external replication of the experiment), although new analyses were added [9]. Further the experiment was replicated by [13]. Their experiment stressed on the fault types and did not considered efficiency of testing techniques.

4. GOAL AND HYPOTHESIS

We replicated the experiment which was actually carried out by [1] and further replicated by [12] which include fault isolation phase in addition to fault detection phase with the sole intention of studying the impact of code reading technique on programs of varying length from the package created by Kamsties and Lott. Detection refers to the observation that the programs observed behavior differs from the expected behavior. Isolation means to reveal the root cause of the difference in behavior. In our case, it is faults in the code. The experiment package built by Kamsties and Lott [1] was used, although some experimental conditions like hypothesis, choice of testing technique, programs selection are changed.

GQM (Goal-Question-Metrics) approach was used to state the goals of this experiment. Accordingly we define our main goal of the experiment as:

Analyze **code review technique** for detecting software defects in varying lines of code for the purpose of comparison with respect to effectiveness and efficiency from the point of view of a researcher in the context of a controlled experiment.

In addition we want to analyze the effect of various parameters on varying lines of code. Accordingly the two main hypotheses are:

MH_{01}: Code review with varying lines of code does not differ in their effectiveness.

MH_{11}: Code review with varying lines of code differs in their effectiveness.

MH_{02}: Code review with varying lines of code does not differ in their efficiency.

MH_{12}: Code review with varying lines of code differs in their efficiency.

The goal of the experiment can be stated as follows:

- **Find out the effectiveness in revealing failures**
- **Find out the efficiency in revealing failures**
- **Find out the effectiveness in isolating faults**
- **Find out the effectiveness in isolating faults**

The questions that were used to test these hypotheses were:

1. What influence does each independent variable have on effectiveness of failure observation and fault isolation?
2. What influence does each independent variable have on the time to observe failure, time to isolate failure and the total time?
3. What influence does each independent variable have on the efficiency of failure observation and fault isolation?

Table 1. Average percentage of defect detected

	Effectiveness		
	Code Reading	Functional	Structural
Hetzel	37.3	47.7	46.7
Myers	38	30	36
Basili and Selby	54	54.6	41.2
Kamsties and Lott Replication 1	**43.5**	**47.5**	**47.4**
Kamsties and Lott Replication 2	**52.8**	**60.7**	**52.8**
Roper et al	32.1	55.2	57.5
Juristo and Vegas Replication 1	19.98	37.7	35.5
	-	75.8	71.4

The experiment is explained in detail in section V below. However our main aim was to evaluate the code review techniques with respect to independent parameters like subject, day, group (order) and program using ANOVA technique. The average percentage of defects detected in existing experiments is shown in table 1 above.

Also average defect detection rate in existing experiment is shown in table 2 below:

As we are using the same guidelines laid down by [1], our results is expected to be in between the range of replica 1 and replica 2 of Kamsties and Lott experiment as shown in bold in table 1 and table 2.

Table 2.
Average defect detection rate

	Effectiveness		
	Code Reading	Functional	Structural
Hetzel	-	-	-
Myers	0.8	1.62	2.07
Basili and Selby	Depends on program	Depends on program	Depends on program
Kamsties and Lott Replication 1	**2.11**	**4.69**	**2.92**
Kamsties and Lott Replication 2	**1.52**	**3.07**	**1.92**
Roper et al	1.06	2.47	2.20
Juristo and Vegas Replication 1	-	-	-
	-	-	-

5. EXPERIMENTAL DESIGN

A procedure that is used to execute a experiment serves as a baseline to guarantee the accuracy of the experiment in the given environment. The procedure may involve training activities, execution of experiment, collecting data, providing feedback etc. A total of twenty one subjects joined this experiment, all these subjects were aware of the fact that their association with the experiment is solely for gaining domain knowledge of software testing field. The subjects for the experiment are selected based on the experience in software testing and knowledge of software engineering and C language. The **table 3** summarizes the overall statistic for the subjects of the experiment.

Table 3.
Subject selection criteria

Selection criteria	**Post Graduate Level (M. C. A)**	**Graduate Level (B. Tech)**
No. of Students	14	7
Experience in software testing	6 Months of industrial training	1 Month of industrial training
Knowledge of SE and C	Yes, Good	Yes, Good

We have taken the same time limit as described by [6], i.e. 240 minutes to each group, so that the results can be compared with each other. Moreover this experiment is a bit different for the one used by [6] as we have used different programs and that too of varied length so examine the effect of increasing lines of code with respect to subject, group and program. Each subject have to see different program on each day, however all subject reviewed all three programs by end of the experiments.

No programs from the Kamsties and Lott package were used in the training session. Instead some trivial simple source code was used in the learning phase. These codes were seeded with almost all types of faults. The programs we used in our experiment are have one different program from the one used by Kamsties and Lott and Roper et al in their live experiment, as in this experiment we were concerned with varying lines of code. The following programs were used in actual experiments and they were part of Kamsties and Lott package [1]:

1. **Cmdline:** evaluates a number of options that are supplied on the command line. The functions in that program fill a data structure with the results of the evaluation, which the driver function prints out upon completion.
2. **N-tree***:* implements an abstract data type, namely a tree with unbounded branching. The functions support creating a tree, inserting a node as a child of a named parent, searching the tree for a node, querying whether two children are siblings, and printing out the contents of the tree. The driver function reads commands from a file to exercise the functions .
3. **Count:** it counts the number of lines, words, and characters in the named files. Words are sequences of characters that are separated by one or more spaces, tabs, or line breaks (carriage return). If a file supplied as argument does not exist, a corresponding error message is printed and processing of any other files continues. If no file is supplied as an argument, count reads from the standard input.

All the programs were written in a C language with which the subjects were familiar. Table 4 gives size data for the programs.

The defects used in our experiment were supplied with Kamsties and Lott package. Most of the defects present in the program as a part of Kamsties and Lott package. We also classify the faults using the two-faceted fault-classification scheme from the [11] experiment. Facet one (type) captures the absence of needed code or the presence of incorrect code (omission, commission). Facet two (class) partitions software faults into the six classes:

1. Initialization
2. Control
3. Computation
4. Interface
5. Data
6. Cosmetic

Table 4.
Size and other relevant information for programs

	cmdlines.c	nree.c	count
Total Lines	272	212	44
Blank lines	26	38	2
Lines with comments	0	4	0
Non-blank non commented lines	246	170	42
Preprocessor directives	4	5	1

Figure 1, Figure 2 and Figure 3 shows the fault distribution for program cmdline, ntree and count respectively. Figure 4 shows the collective fault distribution for all the programs.

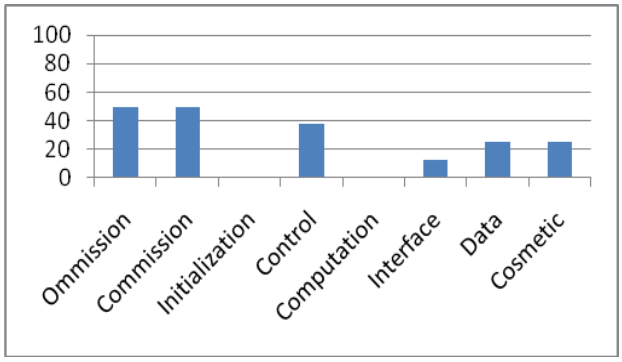

Figure 1. Fault distribution percentage for n-tree program

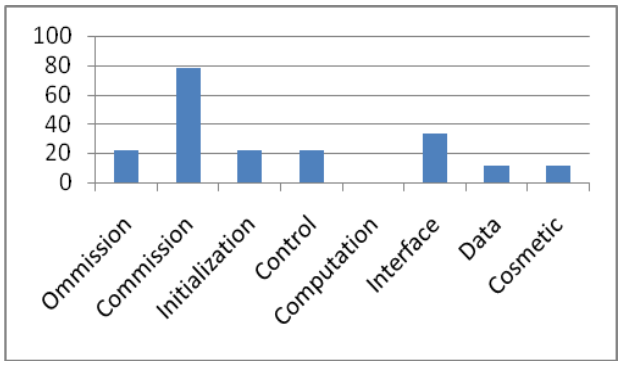

Figure 2. Fault distribution percentage for cmdline program

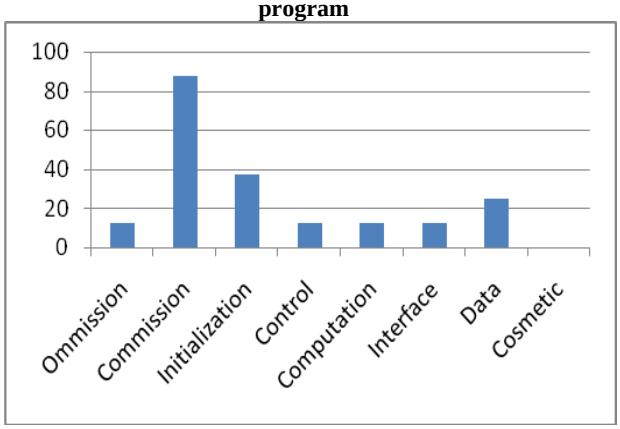

Figure 3. Fault distribution percentage for count program

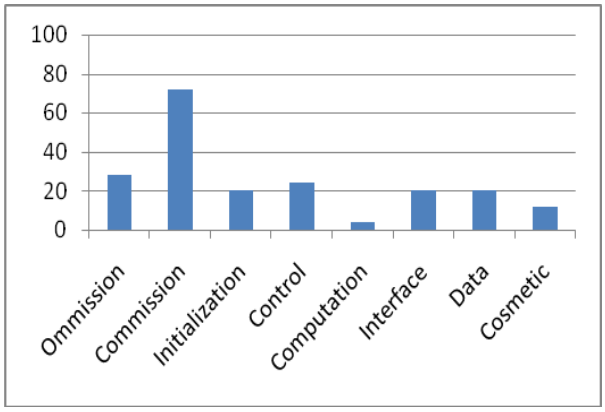

Figure 4. Oevrall Fault distribution percentage

We have applied the same process as mentioned in [1, 6] for code reading i.e. using stepwise abstraction in a 3 step process.

Step 1: subjects were given line numbered printed source code. They read the source code and write their own specification using stepwise abstraction by identifying prime subroutines (consecutive LOC), after writing their own specifications, the subjects receive the official specification.

Step 2: Now the subjects matches the official specification with their own specification to observe inconsistencies (failure observation) between specified and expected program behavior (analog to failures in the other techniques).

Step 3: the subjects begin to isolate the faults that led to the inconsistencies which were observed in step 2. No special technique is specified for the fault-isolation activity.

Finally, subjects hand in a list of identified inconsistencies and isolated faults.

The subjects apply code review technique to three different **programs** (first independent variable) in different **orders/groups** (second independent variable). The subject required three days to complete the experiment and all subject work on same defect detection technique on same day. So the variable **technique** is confounded with **day** and not considered as separate variable. So we have a separate variable as **Day (Technique)** (Third independent variable). Finally the **subject** is the forth independent variable, which is however an uncontrolled independent variable.

6. RESULTS

The ultimate goal of this evaluation approach is to generate metrics for assessing code review technique using ANOVA technique. There are seven metrics that can be generated from the raw data collected in the experiment. The metrics are:

1. Percentage of faults detected
2. Percentage of faults isolated
3. Time to detect faults
4. Time to isolate faults
5. Total time to detect and isolate faults
6. No. faults found / time
7. No. of faults isolated / time

The results showed in table 5 below summaries the effect of independent variables on code review metrics.

Table 5.
Effect of independent variable(s) of test metrics

Metrics	Independent Variable(s)
Percentage of faults detected	Program, Subject
Percentage of faults isolated	Program
Time to detect faults	Program, Group
Time to isolate faults	Program
Total time to detect and isolate faults	Program, Group
No. faults found / time	Program
No. of faults isolated / time	Program

The results of the experiment shown in table 5 points on the fact that complexity and length of the source code affects the most in case of code review efficiency and effectiveness. As we have mentioned in section IV that the results of our experiment should be around the two replica of Kamsties and Lott experiment as we have taken the same guidelines of the authors. The results of our experiments in terms of average percentage of defects detected are shown in table 6 and average rate of defect detection is shown in table 7.

Table 6.
Comparison of Average number of defects detected

	Effectiveness		
	Code Reading	Functional	Structural
Kamsties and Lott Replication 1	**43.5**	47.5	47.4
Kamsties and Lott Replication 2	**52.8**	60.7	52.8
Our experiment	**50.48**	-	-

Table 7.
Comparison of Average number of defects detection rate

	Efficiency		
Kamsties and Lott Replication 1	**2.11**	4.69	2.92
Kamsties and Lott Replication 2	**1.52**	3.07	1.92
Our experiment	**2.02**	-	-

7. CONCLUSION

This experiment was carried out to test the effectiveness and efficiency of code review technique with respect to varying lines of code. We found the not only lines of code has a huge impact on code review technique but complexity of program also affect the effectiveness and efficiency of code review. The effect of program was significant in all the cases. The group (order) and subject also affect the efficiency and effectiveness of code review however their affect was not uniform, rather subject affect the effectiveness of failure observation while group affect the mean failure observation time and total time to detect failure and isolate faults.

We do agree with the previous research done in this field that effectiveness and efficiency depends on program and faults, however in our experiment two more factors also got highlighted that group and subject may also affect the code review technique with respect to efficiency and effectiveness. Roper et al [12] has very rightly quoted that "*as the programs and faults vary, so do the results*".

We suggest other researchers to use **standard code review checklist** to monitor the effect on the efficiency and effectiveness. However these experiments should be carried out in accordance with the given schema [1] so that the outcome of the experiment can be compared to the standard work done by other researchers.

8. REFERENCES

[1] Erik Kamsties and Christopher M. Lott, "An Empirical Evaluation of Three Defect-Detection Techniques", Experimental study, 1995.

[2] Malik, Qaisar Ahmad, "Combining Model-Based Testing and Stepwise Formal Development", Turku Centre for Computer Science. 2010.

[3] Bertolino, "Software testing research: Achievements, challenges, dreams", In future of software engineering 2007, FOSE'07, page 85-103 IEEE 2007.

[4] Mary Jean Harrold, "Testing: A Roadmap", In Future of Software Engineering, 22nd International Conference on Software Engineering, June 2000

[5] "State of code review 2013", SmartBear Software Inc., SB-C-041713-WEB.

[6] Sheikh Umar Farooq, S.M.K. Quadri, "Evaluating Effectiveness of Software Testing Techniques With Emphasis on Enhancing Software Reliability", Journal of Emerging Trends in Computing and Information Sciences, ISSN 209-8407, Vol 2, No.12, 2011.

[7] M. Roper, Software testing, McGraw-Hill, Inc. , 1995.

[8] Hetzel, W. An experimental analysis of program verification methods. 1976.

[9] Juristo, N., Moreno, A., and Vegas, S. Reviewing 25 years of testing technique experiments. Empirical Software Engineering, 9(1):7-44, 2004.

[10] Luo, L. Software testing techniques. Institute for software research international Carnegie mellon university, Pittsburgh, PA, 2001.

[11] Basili, V. and Selby, R. Comparing the effectiveness of software testing strategies. Software Engineering, IEEE Transactions on, (12):1278-1296, 1987.

[12] Roper, M., Wood, M., and Miller, J. An empirical evaluation of defect detection techniques. Information and Software Technology, 39(11):763 - 775, 1997.

[13] Juristo, N., Moreno, A., and Vegas, S. Limitations of empirical testing technique knowledge. SERIES ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 12:1-38, 2003.

[14] R. Panneerselvam, Research Methodology, ISBN 81-203-2458-8, pp:71-81.

[15] Ahmed, Syed Usman and Azmi, Muhammad Asim, "A Novel Model Based Testing (MBT) approach for Automatic Test Case Generation", International Journal of Advanced Research in Computer Science, 4(11), pp 81-83, 2013.

[16] Ahmed, Syed Usman, Sahare, Sneha Anil and Ahmed, Alfia, "Automatic test case generation using collaboration UML diagrams", World Journal of Science and Technology, Vol-2, pp4-6, 2012.