# ON EMPIRICAL COMPARISON OF CHECKLIST-BASED READING AND ADHOC READING FOR CODE INSPECTION

**R. O. Oladele[1] and H. O. Adedayo[2]**

**[1]Department of Computer Science, University of Ilorin, P. M. B. 1515, Ilorin**
**roladele@yahoo.com**
**[2]Department of Computer Science, University of Ibadan, Ibadan**
**wunmaf@yahoo.com**

***ABSTRACT***
***Software inspection is a necessary tool for software quality assurance. To this end a number of inspection techniques have been proposed in the literature with the ad hoc and Checklist-Based Reading (CBR) being the most widely used. This paper investigates the performance of ad hoc and CBR techniques in a traditional paper-based environment. Seventeen undergraduate students of computer science most of whom are in their final year were used as subjects in the controlled experiment. Results of the experiment indicate that CBR is significantly superior to ad hoc reading in terms of effectiveness, efficiency, effort, and number of false positives. On the average, 4 faults were detected in 69 minutes using ad hoc reading while 11 faults were detected in 42.5 minutes using Checklist-based reading. Also the average number of false positive is about 3.13 in checklist-based approach as against about 6.44 in ad hoc approach.***

KEYWORDS: Software Inspection, Quality Assurance, Defects, Reading Techniques, Software Artifact.

**DOI:** https://doi.org/10.54043/laujet.2013.07.02.05

## INTRODUCTION

Increasing software quality is a common objective for software engineers, however, the goal is not easy to achieve and there have still been many research efforts addressing how best to decrease defects and increase quality in software. Basically, there can be three main strategies for decreasing defects in software: defect prevention, defect detection and correction, and reducing impacts of defects. Automated analysis, Inspection, and execution testing are the main methods to detect errors, these methods have their own characteristics, and based on the project situation, they can be used selectively or together.

Software Inspection has become widely used since it was first introduced by Fagan at IBM. This is due to its potential benefits for software development, increased demand for quality certification in software, (for instance, ISO 9000 compliance requirements), and the adoption of the Capability Maturity Model (CMM) as a development methodology.

Software inspection is a proven method for software quality assurance. It involves strict and close examinations carried out on development products to detect defects, violations of development standards and other problems. The development products could be specifications, source code, contracts, test plans and test cases.

It has been hypothesized that in order to gain credibility and validity, software inspection experiments have to be conducted in different environments, using different people, languages, cultures, documents, and so on. In other words, the experiments must be repeated in some other environments. The motivation for this work therefore arises from this hypothesis.

Software inspection is as old as Program Development itself. It was proposed in the 70's by IBM, which pioneered its early adoption and later evolution [4, 5]. It is a means is of detecting faults in software artifacts such as requirements, design, code, test cases, etc. In recent time, empirical studies have shown that defect detection is more an individual activity than a group activity as assumed by many inspection methods and refinements. Suffice it to say that inspection results are completely determined by the inspectors themselves, their strategies for understanding the documents being inspected, and the tools or support available to them during inspection exercise.

A defect detection or reading (as it is popularly called) technique is defined as the series of steps or procedures whose purpose is to guide an inspector in acquiring a deep understanding of the inspected software product. The comprehension of inspected software product is a prerequisite for detecting subtle and / or complex defects, those often causing the most problems if detected in later life cycle phases.

According to Porter et al [21], defect detection techniques range from intuitive, non-systematic procedures such as ad hoc or checklist-based techniques, to explicit and highly systematic procedures such as scenario or correctness proofs. A reviewer's individual task may be general, to identify as many defects as possible, or specific, to focus on a limited set of issues such as ensuring appropriate use of hardware interfaces, identifying un-testable requirements, or checking conformity to coding standards.

The most frequently used detection methods are ad hoc and checklist. Ad hoc reading offers very little reading support at all since a software product is simply given to inspectors without any direction or guidelines on how to proceed through it and what to look for. However, Ad hoc does not mean that inspection participants do not scrutinize the inspected product systematically. The word 'Ad hoc' only refers to the fact that no technical support is given to them for the problem of how to detect defects in a software artifact. In this case, detection fully depends on the skill, the knowledge, and the experience of an inspector. In this case, defect detection depends fully on the skill, the knowledge, and the experience of an inspector. Training session in program comprehension may help inspectors develop some of these capabilities to alleviate the lack of reading support.

Checklists offer stronger, boilerplate support in the form of questions for inspectors while reading the documents. These questions concern quality aspects of the document. Checklists are advocated in many inspection works. For example, Fagan [4, 5], Dunsmore [6], Sabaliauskaite [7], Humphrey [8], and Gilb and Grahams' manuscript [9] to mention a few.

O. S. Akinola and A. O. Osofisan [13] did an empirical, comparative study on checklist-based and ad hoc code reading techniques in a distributed groupware environment. Their findings show that none of the two reading techniques outperforms each other in the tool-based environment studied. The remainder of this paper is organized as follows. In section 2 we present the experimental setting. In section 3, experiment results are reported while the paper is concluded in section 4.

## EXPERIMENTAL SETTING

Subjects

Seventeen (17) students of computer science department were employed in the study. Nine (9) of the student-reviewers used ad hoc reading technique, without providing any aid for them in the inspection. The remaining eight (8) student-reviewers used checklist-based reading technique.

Experimental Artifacts

The artifacts used for this experiment was a 99 lines of java code which accepts data into 3-dimensional arrays. This small size code was used because the student involved in the experiment had their first experience in code inspection with this experiment, even though they were given some formal training on the code inspection prior to the exercise. The experiment was conducted by some students in the department of computer science, university of Ilorin, Nigeria. The students were believed to have some working knowledge of java programming and software development. The arrays were used as matrices. Major operations on matrices were implemented in the program such sum, difference, product, determinant and transpose. All conditions for these operations were tested in the program. The code was developed and tested okay before it was finally seeded with fifteen (15) errors of which eight (8) of those errors were logical errors, four (4) were syntax errors, and three (3) were numerical errors. The program accepts data into three arrays (say A, B, C), then perform some operations like addition, subtraction, multiplication on A and B only, while other operations like determinant and transpose are performed on C. these operations report the output result of the computation if there were no errors, if there were errors in the form of operational condition not being fulfilled for any of the operations, the program reports appropriate error log for that operation.

Experimental Purpose

The goal of the experiment is to evaluate the effectiveness and the efficiency of checklist-based reading (CBR) when it comes to finding faults in a program code. The evaluation is performed by means of comparing CBR inspections with Ad hoc reading. We wanted to determine if there is any significant difference in the performance of ad hoc reading and CBR vis-à-vis their effectiveness and efficiency

In particular the experiment investigated if CBR inspections are cost effective by measuring the time taken to conduct the inspection, and the number of faults that the reviewers detected within that time.

Effectiveness of the inspection technique is defined as fault finding rate and is calculated by dividing the number of found faults with the total number of existing faults in the inspected code document. The efficiency is defined as the number of found faults per hour. It is worth mentioning that effectiveness and efficiency of the inspection technique is measured in a similar way in number of other studies on software inspections.

## EXPERIMENTAL RESULTS

This section shows the results obtained from the experiment as well as analysis following the results. The inspection records provided several data items, i.e. data on the time when the inspection session started and finished as well as the time when a certain fault was found, description of each identified fault, and fault location in the requirements specification.

In order to find values for inspection effectiveness and efficiency for each subject the number of

identified faults and total time of the inspection was collected. Each reported fault was evaluated so as to make sure that it was not a false positive. A false positive is a reported fault that does not qualify as a fault in relation to the inspected code document.

Table 3.1 shows the results obtained from the experiment for both Ad hoc and Checklist-based inspections.

Table 3.1: Experiment Results for Ad hoc Reading and CBR

| | AD HOC READING | | | | | CBR | | | |
|---|---|---|---|---|---|---|---|---|---|
| S/N | NO OF FP | EFFECTIVENESS (%) | EFFICIENCY (%) | EFFORT (Min) | | NO OF FP | EFFECTIVENESS (%) | EFFICIENCY (%) | EFFORT(Min) |
| 1 | 5 | 0.00 | 0.00 | 70 | | 4 | 60.00 | 19.15 | 47 |
| 2 | 4 | 33.33 | 8.62 | 58 | | 3 | 33.33 | 20.83 | 24 |
| 3 | 7 | 13.00 | 2.94 | 68 | | 3 | 53.33 | 29.63 | 27 |
| 4 | 10 | 33.33 | 7.14 | 70 | | 1 | 86.67 | 38.24 | 44 |
| 5 | 7 | 46.67 | 10.00 | 70 | | 5 | 80.00 | 24.00 | 50 |
| 6 | 8 | 26.67 | 5.71 | 70 | | 4 | 93.33 | 26.92 | 52 |
| 7 | 4 | 53.33 | 8.57 | 70 | | 3 | 73.33 | 73.00 | 44 |
| 8 | 8 | 33.33 | 8.00 | 75 | | 2 | 100.00 | 28.85 | 52 |
| 9 | 5 | 13.33 | 2.86 | 70 | | | | | |

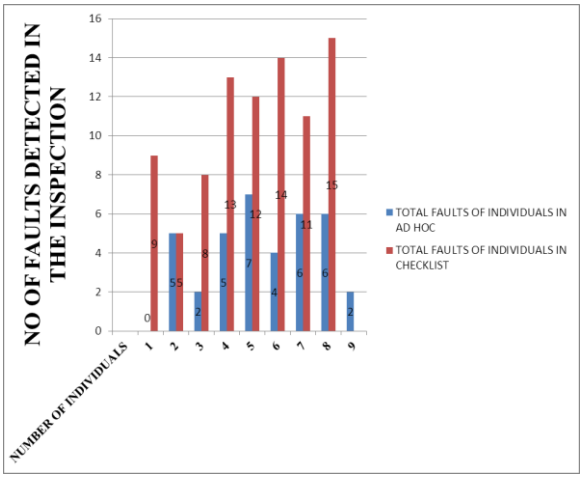


Figure 3.1: A Bar Chart showing the differences in number of faults detected

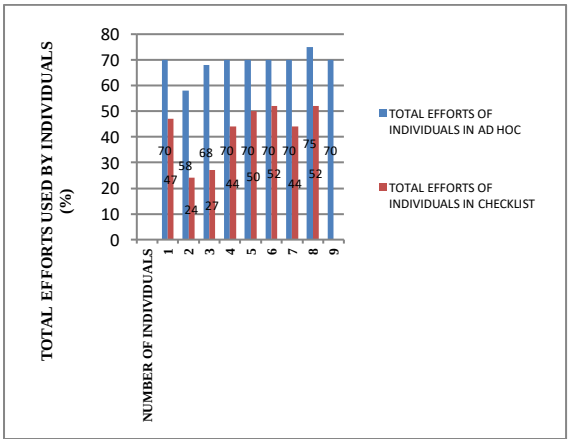


**Figure 3.2: A bar chart showing the differences between the total efforts per individual.**

## CONCLUSION

This work demonstrates the quality of ad hoc and checklist-based reading techniques vis-à-vis their defect detection effectiveness, efficiency, effort taken and number false positives. The results obtained show that checklist-based reading significantly outperforms ad hoc reading. However, results of this study need further experimental validations especially in industrial settings with professionals and large real-life code documents.

## REFERENCES

Laitenberger, O., and DeBaud, J.M., (2000): An Encompassing Life-cycle Centric Survey of Software Inspection. Journal of Systems and Software, 50, 5-31

Lanubile and Giuseppe Visaggio (2000): Evaluating defect Detection Techniques for Software Requirements Inspections.

Thomas Thelin and Per Runeson's Experimental Comparison of Usage- Based and Checklist-Based Reading book in the *department of Communication Systems,Lund University.*

Michael E. Fagan (1976): Design and Code Inspections to reduce errors in Program Development. IBM Systems Journal, 15(3):182-211

Michael E. Fagan (1986): Advances in Software Inspection, *IEEE Trans*. On Software Engineering, SE-12(7):744-751.

Alastair Dunsmore, Marc Roper and Murray Wood (2003): Practical Code Inspection for Object Oriented Systems, *IEEE Software* 20(4), 21 – 29.

Giedre Sabaliauskaite, Fumikazu Matsukawa, Shinji Kusumoto, Katsuro Inoue (2002): "An

Experimental Comparison of Checklist-Based Reading and

Perspective-Based Reading for UML DesignDocument Inspection," *ISESE*, p. 148, 2002 International Symposium on Empirical Software Engineering (ISESE'02), 2002

Watts S. Humphrey (1989): *Managing the Software Process*, chapter 10. Addison-Wesley Publishing Company.

Tom Gilb and Dorothy Graham (1993): *Software Inspection*. Addison-Wesley Publishing Co.

Laitenberger Oliver (2002): A Survey of Software Inspection technologies, handbook on Software Engineering and knowledge Engineering, vol. II, 2002.

Laitenberge, O., and DeBaud, J.m., (2002): An Encompassing life cycle centric survey of Software Inspection. Journal of systems and software, 50, 5-31.

David L. Parnas and David M. Weiss (1985): Active design reviews: Principles and practices. In *Proceedings of the 8th International Conference on Software Engineering*, pages 215-222, Aug. 1985.

O. S. Akinola and A. O. Osofisan (2009): An Empirical Comparative of Checklist-based and Ad hoc Code Reading Techniques in a Distributed Groupware Environment. *International Journal of Computer Science and Information Security, 5(1): 25-35*

Victor Basili, the role of Experimentation in Software Engineering: past, present, and future, keynote address, 18th international conference on software engineering berlin 1996.

Victor Basili, Richard Selby, comparing the effectiveness of software testing techniques, IEEE Transactions of software engineering, vol. 13(12) pp.1278-1296, December 1987.

Richard C. linger, Harlan D. mills, Bernard I. Witt, structured programming: theory and practice, Addison Wesley publishing company, 1979.

Adam A. porter, Lawrence G. Votta, and Victor R. Basili (1995): comparing detection methods for software requirements inspections: A replicated experiment. IEEE Trans. On software engineering, 21 (Harvey 1996): 563-575.

Paulk, M., Curtis, B., Chrissis, M.B. and Weber, C. V. (1993): "Capacity Maturation Model for Software", Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

R. O. Oladele, Reading Techniques for Software Inspection : Review and Analysis. *Journal of Institute of Mathematics and Computer Sciences (Computer Science Series), India*, 2010; 21(2):

Dewayne, E. Perry, Adam A. Porter and Lawrence G. Votta(2000): Empirical studies of software engineering: A Roadmap, *Proc. of the 22nd Conference on Software Engineering*, Limerick Ireland, June 2000.

Adam A. Porter, Lawrence G. Votta, and Victor R. Basili (1995): Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Trans. on Software Engineering*, 21(Harvey, 1996):563-575.

Harlan D. Mills (1972): Evolving and Packaging Reading Techniques through Experimentation: *IEEE Trans. on Software Engineering.*