On Empirical Comparison of Checklist-based Reading and Adhoc Reading for Code Inspection

R. O. Oladele
Department of Computer Science
University of Ilorin, P. M. B. 1515, Ilorin

H. O. Adedayo
Department of Computer Science
University of Ibadan, Ibadan

ABSTRACT

Inspection is a proven approach that is commonly used to manage software quality. To this end many inspection techniques such as Checklist-Based Reading (CBR), Perspective-Based Reading (PBR), Usage-Based Reading (UBR), and Defect-Based Reading (DBR) have been proposed in the literature. Unfortunately, plethora of empirical studies carried out to evaluate these reading techniques have produced inconsistent and conflicting results. Consequently, ad hoc reading and CBR still remain the standard reading techniques in software organizations. This paper investigates the performance of ad hoc and CBR techniques in a traditional paper-based environment. Seventeen undergraduate students of computer science majority of whom are in their final year were used as subjects in a 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.

General Terms

Empirical Software Engineering, Software Inspection.

Keywords

Checklist-Based Reading, Ad hoc Reading, Inspection Techniques, Code Inspection

1. INTRODUCTION

Software engineers have a common objective of improving software quality. Realizing this objective is not however an easy task and research is still on-going on how best to decrease defects and increase quality in software. There are primarily three strategies for decreasing defects in software namely defect prevention, defect detection and correction, and defects' impacts reduction. Dynamic testing, Inspection and Automated analysis are the key methods currently being used to detect and correct defects. Each of these strategies has its own strengths and limitations and should be used together in the process of verification and validation.

Software Inspection was first proposed by Fagan at IBM in the 1970s. It is now a fairly widely used method of program verification, especially in critical systems engineering [1]. It is a non-execution-based test process in which a software artifact is examined to find defects, omissions and anomalies.

It is generally believed that empirical studies of software inspection have to be conducted in different contexts – different environments, using different people, languages, cultures, documents, etc., to ensure credibility and validity. The motivation for this work is based on this belief.

Inspections are an old idea whose major objective is defect detection. In recent time, there are empirical evidences that defect detection is more of individual activity than a group activity. 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 [2]. 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 [3]. Reading techniques may range from intuitive, non-systematic procedures such as ad hoc or checklist-based techniques, to explicit and highly systematic procedures such as formal proofs of correctness [4]. The role of individual reviewer can be general – to discover as many defects as possible, or specific – to concentrate on a couple of issues such as ensuring appropriate use of hardware interfaces, identifying ambiguous requirements, or identifying two or more conflicting requirements.

Ad hoc reading and CBR remain the dominant reading techniques in the industry. In fact, CBR is considered to be the standard reading technique in software organizations [5]. In ad hoc reading, typically to technical support is provided. Reviewers rely on their own skills, knowledge and experience to identify defects. At most inspectors are taken through some session of training in program comprehension before the review process starts [6]. In CBR, the artifact is examined with the aid of pre-defined list of questions – a checklist. This technique is more structured and is believed to offer more support to the reviewer compared to ad hoc reading [6]. There have been many empirical studies involving either ad hoc reading and another reading technique, CBR and another reading technique, or ad hoc reading, CBR and another reading technique. However, empirical studies directly investigating the comparative effectiveness of CBR and ad hoc reading are rare.

Perspective-Based Reading (PBR) was investigated in [7, 8, 9] by different authors and they all observed that PBR is significantly more effective than ad hoc reading. In [10] and [11], PBR was investigated in relation to CBR by different authors and they all discovered that there is no significant difference between the effectiveness of PBR and CBR. Comparing and extrapolating all these findings one would expect that there will be a significant difference between the effectiveness of CBR and ad hoc reading. Surprisingly, Akinola and Osofisan [3] did an empirical, comparative study on CBR and ad hoc reading and observed that none of the two reading techniques outperforms each other in the tool-based environment studied. There is therefore the need for more empirical studies on CBR and ad hoc reading with a view to assessing their relative performance.

The objective of this paper is to compare and hence evaluate how well CBR performs in comparison to ad hoc reading. The paper presents a controlled experiment where CBR is compared to ad hoc reading. 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.

2. EXPERIMENTAL SETTING

2.1 Subjects

Seventeen (17) undergraduate students of computer science of the University of Ilorin (in Nigeria) were used as reviewers in the study. Nine (9) of the reviewers used ad hoc reading, without providing any aid for them in the inspection. The remaining eight (8) reviewers used CBR. The students were believed to have some working knowledge of java programming and software development.

2.2 Experimental Artifacts

The artifacts used for this experiment was a 99 lines of java code which accepts data into 3-dimensional arrays. This small-scale code was used because the reviewers had no prior experience in code inspection, though they were taken through some training sessions on code inspection before the commencement of the experiment. The program implemented key matrix operations such as sum, difference, product, determinant and transpose. The code was firstly written, verified and validated by the investigators before it was finally mutated with fifteen (15) errors of which eight (8) 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, then reports the result of the computation if there were no errors, should there be any errors the program reports appropriate error log for the operation in question.

2.3 Experimental Purpose

The goal of the experiment is to estimate the performance differences of CBR and ad hoc reading vis-à-vis effectiveness, efficiency, and number of false positive. In particular, we wanted to investigate if there is any significant difference in the performance of ad hoc reading and CBR.

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.

3. EXPERIMENTAL RESULTS

This section reports the results obtained from the experiment as well as analysis of 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 were collated. 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 inspection experiment.

Table 3.1: Inspection Results

	AD HOC READING					CBR			
REVIEWERS	NO OF FALSE POSITIVE	EFFECTIVENESS (%)	EFFICIENCY (%)	EFFORT (Min)		NO OF FALSE POSITIVE	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					

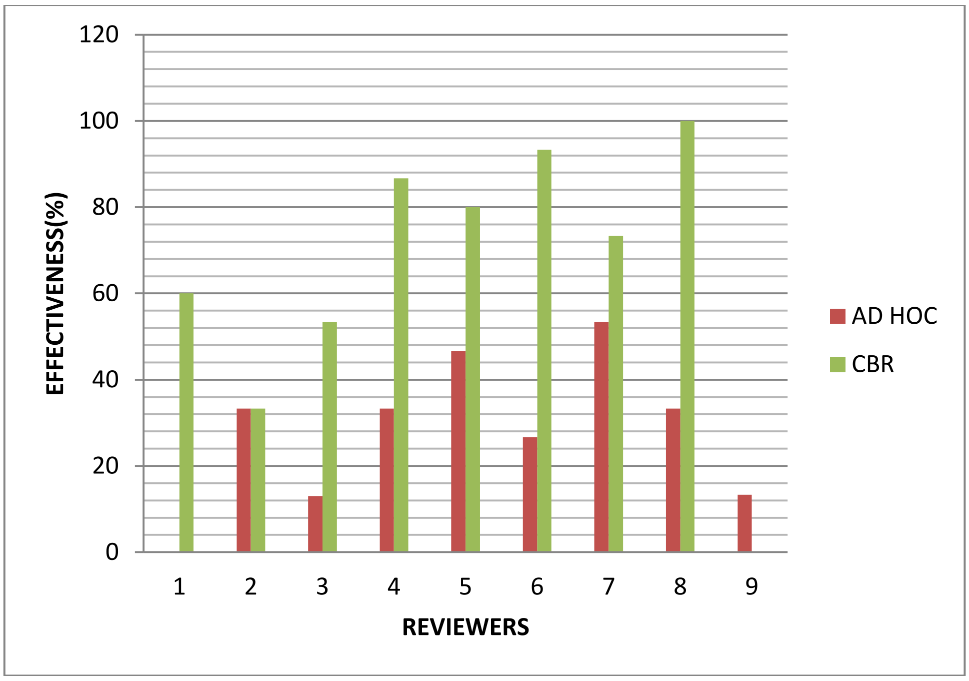

Figure 3.1: **A bar chart showing the difference in effectiveness of CBR and Ad hoc Reading**

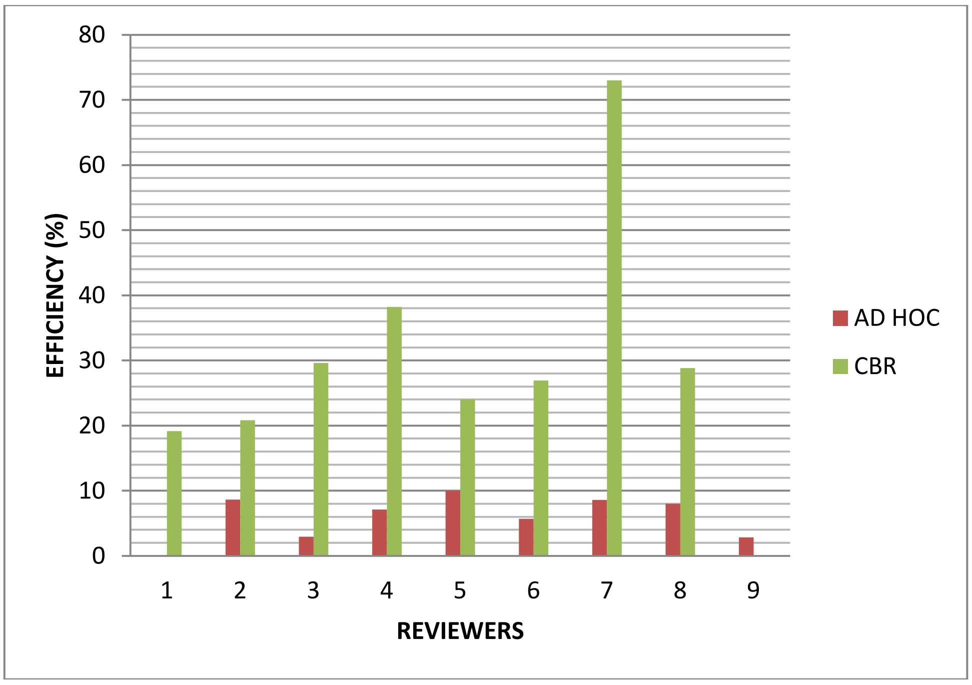

Figure 3.2: A bar chart showing the difference in efficiency of CBR and Ad hoc Reading

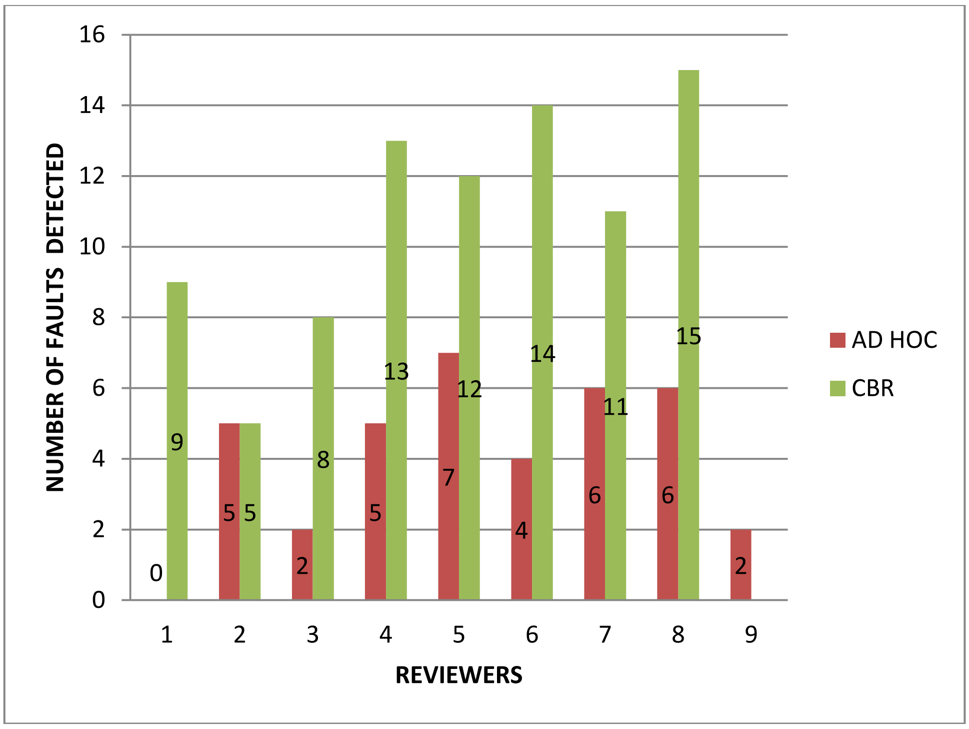

Figure 3.3: A bar chart showing the difference in fault detection capacity of CBR and Ad hoc Reading

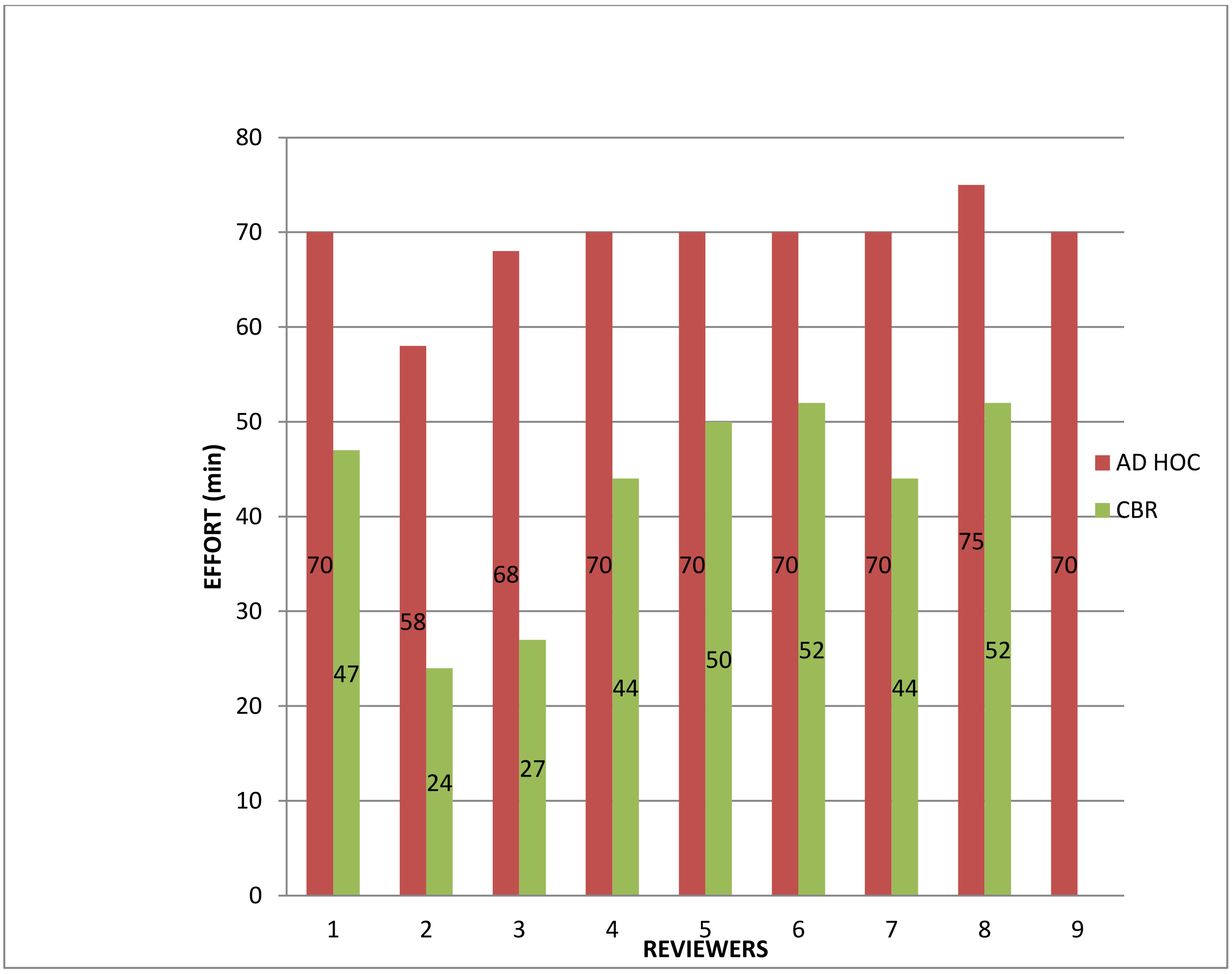

Figure 3.4: A bar chart showing the difference in effort required by CBR and Ad hoc Reading

From Figure 3.1, it is clear that each reviewer that used CBR was significantly more effective than his counterpart that used ad hoc reading with the exception of only one case where CBR and ad hoc reading tied in effectiveness. Similarly, Figure 3.2 shows each reviewer that used CBR was significantly more efficient than his counterpart in the ad hoc group. Figure 3.3 reveals that all reviewers in the CBR group detected much more defects than their counterparts in the ad hoc group though there is one exception where readers from both group detected 5 defects each. Figure 3.4 depicts that code inspection using ad hoc reading took much longer than code inspection using CBR. Also, it can be observed from Table 3.1 that the average number of false positive found using ad hoc is twice the average found using CBR. It is pertinent to point out that our findings are not in consonance with the findings in [3] where the authors observed that there was no significant difference between the performance of ad hoc reading and that of CBR though the experiment was performed in a tool-based environment

4. CONCLUSION

This paper compares the performance of ad hoc and checklist-based reading techniques vis-à-vis their 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 as subjects, large real-life code documents and much more subjects. With CBR being said to rank the same as PBR, UBR, DBR, and other scenario-based reading techniques, in performance [16, 17], it will therefore be interesting to investigate, in the future, the performance of ad hoc reading in relation to these scenario-based reading techniques.

5. REFERENCES

[1] Sommerville, Ian 2007. Software Engineering. Eighth Edition. Addison-Wesley.

[2] Laitenberger, Oliver 2002. A Survey of Software Inspection technologies, handbook on Software Engineering and knowledge Engineering, vol. II, 2002.

[3] Akinola, O. S., Osofisan, A. O. 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

[4] Porter, A. A, Votta, L.G. and Basili, V. R. 1995. Comparing detection methods for software requirements inspections: A replicated experiment. IEEE Trans. on Software Engineering, 21(Harvey, 1996):563-575.

[5] Laitenberge, O., and DeBaud, J. M. 2002. An Encompassing life cycle centric survey of Software Inspection. Journal of systems and software, 50, 5-31.

[6] Oladele, R. O. 2010. Reading Techniques for Software Inspection: Review and Analysis. Journal of Institute of Mathematics and Computer Sciences (Computer Science Series), India, 21(2): 199 – 209

[7] Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. and Zelkowitz, M. V. 1996. The Empirical Investigation of Perspective-Based Reading. Empirical Software Engineering: An International Journal, 1(2):133-164.

[8] Ciolkowski, M., Differding, C., Laitenberger, O., and Münch, J. 1997. "Empirical Investigation of Perspective-Based Reading: A Replicated Experiment", ISERN Report no. 97-13.

[9] Shull, F. 1998. Developing Techniques for Using Software Documents: A Series of Empirical Studies, PhD Thesis, Computer Science Department, University of Maryland, USA.

[10] Biffl, S. 2001. Software Inspection Techniques to Support Project and Quality Management, Habilitationsschrift, Shaker Verlag, Austria.

[11] Halling, M., Biffl, S., Grechenig, T. and Köhle, M. 2001. Using Reading Techniques to Focus Inspection Performance. Proc. of the 27th Euromicro Workshop on Software Process and Product Improvement, 248-257.

[12] Parnas, D. L. and Weiss, D. M. 1985. Active design reviews: Principles and practices. In Proceedings of the 8th International Conference on Software Engineering, Aug. 1985, 215 – 222

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

[14] Fagan, M. E. 1976. Design and Code Inspections to reduce errors in Program Development. IBM Systems Journal, 15(3):182-21

[15] Fagan, M. E. 1986. Advances in Software Inspection, IEEE Trans. On Software Engineering, SE-12(7):744-751.

[16] Denger, C., Ciolkowski, M., Lanubile, F. 2004. Does Active Guidance Improve Software Inspections? A Preliminary Empirical Study; 2004; Proceedings of the IASTED International Conference SOFTWARE ENGINEERING February 17-19, 2004, Innsbruck, Austria; 408-413

[17] Berling, T., Thelin, T. 2004. A Case Study of Reading Techniques in a Software Company; Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)