# Comparing Code Reading and Testing Criteria: A Replication of Experimental Studies

José Carlos Maldonado
ICMC-USP, Caixa Postal 668,
13.560 São Carlos, SP, Brazil
jcmaldon@icmc.usp.br

Sandra Fabbri
DC-UFSCar, Caixa Postal 676
13565-905 São Carlos, SP, Brazil
sfabbri@dc.ufscar.br

Manoel Mendonça
UNIFACS
41950 Salvador, BA, Brazil
mgmn@unifacs.br

Emerson Dória
UNOESTE
19050-680 Pres. Prudente, SP, Brazil
emerson@icmc.sc.usp.br

Luciana Martimiano
ICMC-USP, Caixa Postal 668,
13.560 São Carlos, SP, Brazil
luciana@icmc.usp.br

Jeffrey Carver
Mississippi State University
Box 9637, MS 39762, USA
carver@cse.msstate.edu

Forrest Shull
Fraunhofer Center for Experimental
Software Engineering
College Park, MD 20740, USA
fshull@fc-md.umd.edu

Victor Basili
Department of Computer Science,
University of Maryland
Maryland, USA
basili@cs.umd.edu

## ABSTRACT

In this paper, we describe a variation of the Basili & Selby [1] and Kamsties & Lott [2] experiments. Named *ITonCode*, this experiment was conducted in the context of the Readers Project [3], a collaborative research initiative on software defect detection techniques supported by Brazil (CNPq) and US (NSF) research agencies. This experiment aims to compare inspection and testing techniques at code level. It extends previous experiments by introducing incremental testing, composed of control flow, data flow and mutation (error-based) criteria. Although data-flow and mutation based criteria have been introduced, code reading presented the best percentage of detected faults. On the other hand, functional testing performed worst than incremental testing. Moreover, the combination of any two techniques performed better or equal to the techniques alone. Similar results were obtained for defect isolation.

## Categories and Subject Descriptors

D.2.4 [**Testing and Debugging**]: Code inspections, walk-throughs and testing.

## General Terms

Experimentation, Validation.

## Keywords

Experimental Studies, VV&T techniques characterization.

## 1. INTRODUCTION

The task of choosing the best software engineering techniques, methods and tools to achieve a set of quality goals under a given scenario is not a simple endeavor. Experimental studies are fundamental for executing cost-benefit analyses of software engineering approaches. Based on empirical evidence, one can construct experience bases that provide qualitative and quantitative data about the advantages and disadvantages of using these software engineering techniques, methods and tools, in different sets and domains. According to Basili et al [4] experimentation in Software Engineering is necessary because hypotheses without proof are neither safe nor reliable as a knowledge source. Replication is an important activity in this scenario.

According to Fusaro et al [6], replicate means to reproduce as faithfully as possible a previous experiment, usually run by other researchers, in different environment and conditions. When the results generated by a replication are coincident with the ones of the original experiment, they contribute to strength the hypotheses being studied. Otherwise, other parameters and variables should be investigated.

This paper reports the results obtained in an experiment aimed at comparing software inspection and testing techniques. This experiment, here on named the *ITonCode* experiment, was based on two previous experiments conducted by Basili & Selby [1] and Kamsties & Lott [2]. The *ITonCode* experiment is not a replication but rather an extension of the other studies. It adds increasingly strict structural and mutation-based testing criteria to the original Basili-Selby experiment, in an approach named Incremental Testing.

The problem of conducting effective coordinated replications has been addressed by the Readers Project, a collaborative research effort to develop, validate and package reading techniques for software defect detection through experimentation. This project, supported by the Brazilian (CNPq) and American (NSF) national research foundations, investigates techniques for software document review in diverse technical and cultural settings [3]. It is important to highlight that this experiment contributed to the establishment of the Experimental Knowledge Sharing Model

(EKSM) [10]. The EKSM defines the *types* of knowledge sharing that must occur during experimental replications.

This paper is organized as follows. Section 2 comments on the two previous studies. Section 3 characterizes *ITonCode* and describe the main results; and Section 4 presents the conclusions of this work.

## 2. THE ORIGINAL EXPERIMENT

### • Basili & Selby Experiment

Studied Techniques: code reading, applying stepwise abstraction; functional testing, applying equivalence partition and boundary value analysis; and structural testing, applying All-Nodes criteria.

Software Artifacts: three Fortran programs with 169, 147 and 365 lines of code with 9, 6 and 12 faults, respectively.

Objectives: comparing three different fault detection techniques concerning effectiveness (the average percentage of faults found by subjects) and efficiency (the average number of faults found by each subject per hour) to observe faults.

Participants: The experiment was carried out three times: the first two with 42 advanced students from "Software Design and Development" course at the University of Maryland and the last one with 32 developers from NASA and Computer Science Corporation.

Experimental Project: in the execution phase each technique was applied in a different session where the participants could identify the faults. The artifacts were applied in such a way that it avoided the exchange of information between the subjects.

Main results: code reading, followed by functional testing, had the best percentage of detected faults.

### • Kamsties & Lott Experiment

Techniques: code reading, applying stepwise abstraction; functional testing, applying equivalence partition; and structural testing, applying all-branches, all-multiple condition, all-loops and all-relational operators criteria

Software Artifacts: three C programs with 44, 89 and 127 lines of code in the training sessions and other three ones with 260, 279 and 282 lines of code and 11, 14 and 11 defects, respectively, in the experiment execution.

Objectives: comparing three different fault detection techniques concerning effectiveness and efficiency to observe faults and isolate defects.

Participants: The experiment was carried out two times, both with undergraduate students, with 27 and 23 students, respectively.

Experimental Project: it was based on Basili & Selby [1] experimental project, with a step added to isolate defects. The training was elaborated to be applied into two parts (theoretical and practical). The artifacts were applied in such a way that it avoided the exchange of information between the subjects.

Main results: code reading and functional testing had the same percentage of detected faults while structural testing had a smaller one. Besides, functional testing had a better performance in relation to the efficiency to isolate defects.

## 3. THE *ITonCode* EXPERIMENT

The aim of this study is to address the results of previous experiment on comparing reading and testing techniques adding the perspective of data-flow and mutation based testing criteria. To achieve this objective the previous lab package had to be downloaded from the original experimenters and updated: the planning of the experiment, the training material and data collection forms. To run this experiment two tools that support the unit Testing of C programs were used: *Proteum* [8] that supports mutation analysis; and *Poketool* [9] that supports data-flow criteria.

Because of the level of effort involved in running even a replicated or extended experiment, the authors were not willing to undertake a full experiment without testing the material and concepts in their own environment. Although the pilot study also required an investment of effort and resources, it was considered appropriate and necessary to ensure the quality and conformance of the experimental process. The pilot study was very useful for identifying tacit knowledge issues. The pilot study involved three experienced people. Their performance determined the initial expected lower bound for the experiment running time and upper bound for the expected defect detection effectiveness.

Techniques: code reading, applying stepwise abstraction; functional testing, applying equivalence partition and boundary value analysis; and incremental testing. Incremental Testing means incrementally evolving the test case set, obtaining adequate test set for the all-nodes, all-branches, all-uses, all-potential-uses and mutation analysis criteria.

Software Artifacts: the same artifacts of Kamsties and Lott [2] experiment, both for training and execution phases.

Objectives: comparing three different fault detection techniques concerning effectiveness and efficiency to observe faults and isolate defects. In this paper we only present data on the effectiveness.

Participants: 12 graduate students from UFSCar and ICMC-USP.

Experimental Project: Essentially the same project defined by Basili and Selby with the isolation step introduced by Kamsties and Lott. The training was carried out in two consecutives days and for each of the five techniques (code reading, functional testing, control-flow testing, data-flow testing and mutation testing) the participants had 30 minutes of theory training and 60 minutes of practical training. The execution was carried out along three consecutives weeks. The artifacts were applied in such a way that it avoided the exchange of information between the subjects.

Main results: Although data-flow and mutation based criteria have been introduced, code reading presented the best percentage of isolating defects. On the other hand, functional testing performed worst than incremental testing. Moreover, the combination of any two techniques performed better or equal to the techniques alone, for all the programs (see Figure 1). Similar results were obtained for detected faults, in accordance to Basili and Selby's results. We

observe that the differences among the techniques have yet to be tested for statistical significance.

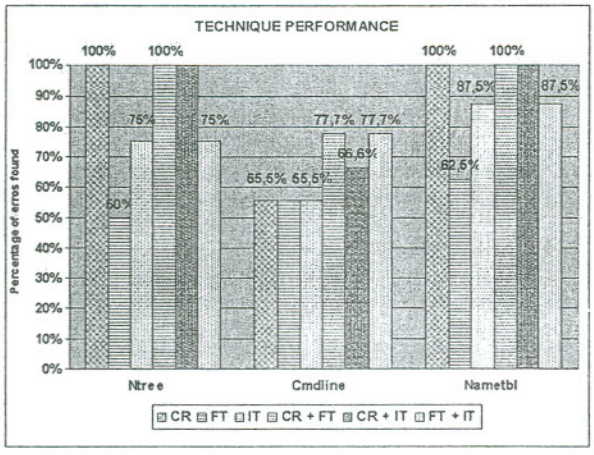


**Figure 6. Better/Worst Performance by technique and program or a combination of techniques.**

## 4. CONCLUSION

Threats and limitations should be taken in account when using the results discussed in this paper. For instance, subject selection represents an internal threat for our experiment because subjects may have different degrees of ability and expertise. This effect is mitigated by the fact that all subjects apply all techniques. Also, the subject survey form allows taking the subject experience into consideration. Outliers were analyzed against the subject background information. Training time is another point that should be further addressed. Different learning curves for the techniques may interfere in their performance. The results may reflect a transient state of learning and consequently of effectiveness of the techniques. We mitigate some of the learning effect on the experiment internal validity by applying the programs in increasing order of complexity. There is also a possible interaction between learning and the order in which the subjects apply the techniques. This is mitigated by the permutation of the ordering of the techniques among the subject. However, there is no way to avoid the fact that a subject accumulates defect detection experience from one experimental trial to the other.

In spite of code reading presenting the best percentage of detected faults and defect isolation, the fact that any combination of two techniques performed better or equal to the techniques alone points out that the complementary aspects of the techniques should be explored in the establishment of VV&T strategies.

In the near future we intend to explore these set of experiments in an industrial set, considering also the OO paradigm as well.

## 5. ACKNOWLEDGMENTS

Our thanks to the members of the Readers Project and to the funding agencies CNPq. FAPESP and NSF.

## 6. REFERENCES

[1] Basili, V.; Selby, Richard W. "Comparing the Effectiveness of Software Testing Strategies". IEEE Transactions on Software Engineering. n. 12, vol. SE-13 (1987), 1278-1296.

[2] Kamsties, Erik. E Lott, Christopher M. "An Empirical Evaluation of Three Defect-Detection Techniques." Technical Report ISERN 95-02. Department of Computer Science, University of Kaiserslautern, 67653, Kaiserslautern, Germany, May 1995.

[3] Maldonado, J.C.; Martiniano, L. A. F.; Dória, E.S.; Fabbri, S.C.C.P.F.; Mendonça, M. "Readers Project: Replication of Experiments –A Case Study Using Requeriments Documents". In: ProTeM-CC-Project Evaluation Workshop – International Cooperation, CNPq, Rio de Janeiro, RJ, October, 2001, pp. 85-117.

[4] Basili, V. R.; Green S.; Laitenberger, O.; Lanubile, F.; Shull, F.; Sorumgard, S.; Zelkowitz, M. "Packging Researcher Experience to assist Replication of Experiments". In: ISERN Meeting, Sydney, Australia, 1996

[5] Law, D., and Naem, T., "DESMET: Determining and Evaluation Methodology for Software Methods and Tools", Proceedings of BSc Conference on CASE - Current Practice, Future Prospects, Cambridge, England, March, 1992.

[6] Fusaro, Pierfrancesco; Filippo Lanubile; Viusaggio Giuseppe. "A Replicated Experiment to Assess Requirements Inspection Techniques", Empirical Software Engineering Journal, vol. 2, nº. 1, p.39-57, 1997

[7] Dória, E. S. "Experimental Studies Replication in Software Engineering". MsC Thesis, ICMC-USP, São Carlos, 2001

[8] Delamaro, M. E. "Proteum – A Testing Environment Base don Mutation Analysis". MsC Thesis. ICMC / USP, São Carlos, SP, October, 1993.

[9] Chaim, M.L. "POKETOOL – A Tool to Support Programs Structural Testing Base don Data-Flow Analysis". MsC Thesis, DCA/FEE/UNICAMP, Campinas, SP, April, 1991.

[10] Shull, F., M. Mendonca, V. Basili, J. Carver, J. Maldonado, S. Fabbri, G. Travassos, and M. Ferreira, "Knowledge-sharing Issues in Experimental Software Engineering." Empirical Software Engineering - An International Journal, 2004. 9(1): p. 111-137.