Impact of Maintainability Defects on Code Inspections

Özlem Albayrak
Computer Technology and Information Systems
Bilkent University
Ankara, Turkey
+90 (312) 290 5039
ozlemal@bilkent.edu.tr

David Davenport
Computer Engineering
Bilkent University
Ankara, Turkey
+90 (312) 290 1248
david@cs.bilkent.edu.tr

ABSTRACT
Software inspections are effective ways to detect defects early in the development process. In this paper, we analyze the impact of certain defect types on the effectiveness of code inspection. We conducted an experiment in an academic environment with 88 subjects to empirically investigate the effect of two maintainability defects, i.e., indentation and naming conventions, on the number of functional defects found, the effectiveness of functional defect detections, and the number of false positives reported during individual code inspections.

Results show that in cases where both naming conventions and indentation defects exist, the participants found minimum number of defects and reported the highest number of false positives , as compared to the cases where either indentation or naming defects exist. Among maintainability defects, indentation seems to significantly impact the number of functional defects found by the inspector, while the presence of naming conventions defects seems to have no significant impact on the number of functional defects detected. The presence of maintainability defects significantly impacts the number of false positives reported. On the effectiveness of individual code inspectors we observed no significant impact originated from the presence of indentation or naming convention defects.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging – *Code inspections and walk-throughs*;

General Terms
Code inspections and walk-throughs

Keywords
Source code inspection, code inspection effectiveness.

1. INTRODUCTION
Formal software inspections have been established as an effective way to detect defects and thus decrease software development costs [2, 7]. Code inspection is an important part of software inspections, as a structured quality verification process to identify defects in the source code [2, 3, 10, 18].

ESEM'10, September 16–17, 2010, Bolzano-Bolzen, Italy.

Manual inspections are considered too laborious for widespread adoption. Studies have been conducted on how to optimize the inspection process to increase effectiveness of code inspections [11, 17]. Considerable research has been carried out on computer supported code inspection tools [4, 8, 15, 16].

In [5] researchers proposed development of agent-based tools to automate parts of the code inspection process. It has been shown that advances in static program analysis can reduce the inspection time required [2]. For more effective code inspections, techniques for object-oriented code inspections were suggested in [6].

An inspection of the source code usually follows a checklist [1, 2, 9]. While many checklists and coding standards exist, the final decision is generally based on the experience of the inspector [13]. Several recent studies have concluded that most of the defects are found during individual inspections [8]. In this study, we focus on individual code inspectors, using checklists.

This paper does not propose a new method to improve effectiveness of code inspections. It presents results of an examination of whether the presence of some maintainability defects impact individual code inspection effectiveness. Previous studies show that most of the defects found in code inspections are not problems that could have been uncovered by latter phases in testing or field usage because these defects have little or nothing to do with the visible execution behavior of the software. Code inspections do not only detect defects, but they also improve the readability of the code, and hence maintainability. Rather, they improve the maintainability of the code by making the code conform to coding standards, minimizing redundancies, improving language proficiency, improving safety and portability [12, 2]. To our knowledge, no previous studies have been conducted to observe if the presence of maintainability defects has an impact on individual code inspector's effectiveness.

The purpose of this experiment was to find out *whether or not maintainability defects, due to indentation and naming defects, have an impact on the number of functional defects detected,* and on the *effectiveness of code inspection* to detect functional defects, and the number of *false positives* reported.

We undertook a study to investigate the following research questions: *For code inspections*,

> *RQ1) Is the number of functional defects detected by individuals affected by the presence of either (a) Indentation defects or (b) Naming defects?*

RQ2) Is the individual's effectiveness of functional defect detection affected by the presence of indentation and naming defects?

RQ3) Is the number of false positives reported affected by the presence of indentation and naming defects?

2. EXPERIMENTAL SETUP

For initial work to observe the impact of different types of maintainability defects on the number of functional defects detected correctly and false positives reported, and on the effectiveness of individual code inspections, we used four different versions of a Java source code.

In this study, we examined the following hypotheses. (Due to space limitations, we do not include the alternative hypotheses here and only present the null hypotheses):

$H1a_0$: The number of functional defects detected by an inspector is not affected by the presence of indentation defects in the source code.

$H1b_0$: The number of functional defects detected by an inspector is not affected by the presence of naming defects in the source code.

$H2_0$: The effectiveness of functional defect inspection by an inspector is not affected by the presence of indentation or naming defects in the source code.

$H3_0$: The number of false positives is not affected by the type of maintenance defects in the source code.

2.1 Experimental Variables

Maintainability defects are defects that do not affect the visible functionality of software, but impact maintainability by making the code easier to understand and modify. Naming defects include violation of naming conventions and using meaningless names in the code. Indentation defects are related to misuse of spaces in the source code lines, within the same line and between lines. The dependent variables included:

Functional defects: The total number of functional defects detected correctly in the source code. (We do not use the defect detection ratio as defined in [15], because in our experiment there were equal number of functional defects injected into the code.)

Inspector effectiveness: The number of functional defects correctly found divided by the time spent in the inspection.

False positives: The total number of false positives reported by the individual inspector.

2.2 Subjects

We first conducted a pilot study with 16 senior students enrolled into CTIS494-Software Quality Assurance course. The main purpose of this pilot study was to validate our checklist, set an appropriate time-period for the experiment, and collect feedback on the whole experiment moving forward.

The actual experiment included 88 subjects who were volunteer freshmen students of three different sections of a second Java programming course (CS102-Algorithms and Programming) of the Computer Engineering department at Bilkent University during the Spring semester of 2010. The subjects had two semester programming experience in an academic environment and no experience in code inspections.

2.3 Materials

The experiment used four versions of the same artifact, about 100 LOC of Java source code. All four versions included the same functional defects. We then injected different defect types composed of indentation and naming defects to each version. Table 1 presents the distribution of defect types injected to the different versions of the artifact.

Table 1. Distribution of defect types injected

Defect Type	V0	V1	V2	V3
Indentation	0	0	10	4
Naming	0	10	0	6
Functional	6	6	6	6

2.4 Study Design

During the lecture immediately preceding the experiment, the subjects were informed about software inspections. They were trained on code inspection using checklists. The study was conducted on the same day after the training. Each student inspected one artifact using the checklist provided to them to detect naming, indentation and functional defects. Each version of the artifact contained the same total number of defects, except for the first group which only had the 6 functional defects injected. All of the subjects started the study at the same time and they used a common timer application to record the time they detected each defect. We placed the subjects and distributed artifacts to them.

3. DATA ANALYSIS AND RESULTS

We collected 88 valid inspection documents from 88 subjects. The subjects' assignment to treatment groups was not totally random. To balance the number of students per course section, they were seated with respect to their CS102 sections, some of the students did not appear in the experiment, thus each group does not have equal number of participants. We counted the number of defects reported and the number of defects detected by each inspector. The descriptive statistics are presented in Table 2, while the box-plots in Fig. 1 shows graphically the number of functional defects detected and false positives reported for different versions of the artifact.

Table 2. Descriptive statistics for functional defects

Artifact version	False Positives Reported		Functional Defects Detected		N
	Mean	Std. Dev.	Mean	Std. Dev.	
V0	5.61	2..808	2.22	1.313	23
V1	4.73	2.676	2.32	1.249	22
V2	4.74	2.663	1.79	1.512	19
V3	6.96	2.941	1.63	1.173	24

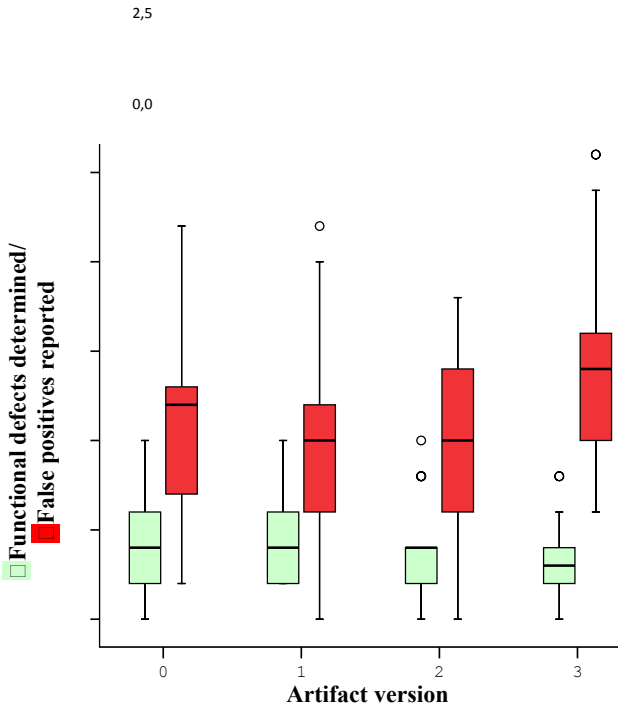

Figure 1. Box-plot for functional defects and false positives.

For all statistical tests reported in this paper, we have used one-way ANOVA and an alpha value of 0.05.

3.1 H1: Functional Defects

Table 3 presents results of the ANOVA to test $H1a_0$.

Table 3. Test of the ANOVA ($H1a_0$-Indentation Defects)

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	7.119	1	7.119	4.255	0.042
Within Groups	143.870	86	1.673		
Total	150.989	87			

R Squared = 0.047 (Adjusted R Squared=0.036)

The ANOVA was significant F(1, 88) = 4.255, p = 0.042, η2 = 0.047. This result allows $H1a_0$ to be rejected. The presence of indentation defects impacts the number of functional defects detected in the code. In the presence of indentation defects, the subject found less functional defects. On the average, the presence of indentation defects resulted in 0.56 (25%) fewer functional defects being defected.

Table 4 presents results of the ANOVA to test $H1b_0$. For impact of the presence of naming defects, the ANOVA was not significant F(1, 88) = 0.057, p = 0.812, η2 = 0.001.

Table 4. Test of the ANOVA ($H1b_0$-Naming Defects)

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	0.099	1	0.099	0.057	.812
Within Groups	150.889	86	1.755		
Total	150.989	87			

R Squared = 0.001 (Adjusted R Squared=0.011)

3.2 H2: Inspector Effectiveness

Table 5 presents results of the ANOVA to test $H2_0$. The ANOVA was not significant F(3, 83) = 2.191, p = 0.096, η2 = 0.077. Those inspected V1 were the most, and V3 were the least effective.

Table 5. Test of the ANOVA (Inspector effectiveness)

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	1.35E-005	3	4.50E-006	2.191	0.096
Within Groups	.000	79	2.06E-006		
Total	0.000	82			

R Squared = 0.077 (Adjusted R Squared=0.042)

3.3 H3: False Positives

Table 6 presents results of the ANOVA to test $H3_0$.

Table 6. Test of the ANOVA (False positives)

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	75.106	3	25.035	3.233	0.026
Within Groups	650.484	84	7.744		
Total	725.591	87			

R Squared = 0.104 (Adjusted R Squared=0.071)

The ANOVA was significant F(1, 88) = 3.233, p = 0.026, η2 = 0.104. This result allows $H3_0$ to be rejected. The type and existence of maintainability defects impacts the number of false positives reported. In the presence of only one type of maintainability defects, in versions V1 and V2, the mean and the standard deviations of the false positives reported by the inspectors are almost the same (Table 3). While in the presence of both indentation and naming defects, as in V3, the subject determined more false positives than the other versions. The inspectors whose artifact did not contain any maintainability defects, version V0, reported more false positives than the inspectors using V1 and V2 versions of the artifact. On the average, the presence of both naming and indentation defects resulted in 2.23 (47%) more false positives being reported.

4. THREATS to VALIDITY

This experiment exhibits a number of threats to internal and external validity. External validity deals with generalizations and internal validity investigates if the treatment causes the outcome. Common to any empirical study, researchers cannot draw general conclusions based solely on the results of one study.

The subjects were students who may not represent real developers, and thus may have been the major source of the observed results. The source code artifact used may not be reflective of a typical Java source code, but we selected commonly made defects in the code. The likelihood of experimenter's bias is limited by having two researchers. The experiment was conducted all at once and by the same collector, thus we do not have location and instrumentation threats.

5. CONCLUSIONS and FURTHER WORK

The primary goal of this study was to investigate the impact of the maintainability defects, indentation and naming conventions defects, on the number of functional defects identified correctly, on the number of false positives reported, and on the effectiveness of individual code inspectors.

Based on empirical data collected during an experiment in an academic setting with 88 students, our analysis showed that the inspectors detected less functional defects when their code included indentation defects. Thus, the presence of indentation defects has a significant negative impact on the number of functional defects found, while the presence of naming defects does not appear to have the same impact.

The significant impact of the maintainability defects on the number of false positives reported is an important observation. If the code contains only a single type of maintainability defects, the inspectors' effectiveness are equal. The presence of indentation and naming defects does not have significant impact on the effectiveness of individual code inspections. The results reveal initial information on the importance of indentation defects in the code. The presence of indentation defects decreases the readability and significantly reduces the defects found.We aim to run similar experiments using different programming languages and with professional developers in the future.

6. ACKNOWLEDGMENTS

We thank Dr. Jorge L. Díaz-Herrera for commenting on earlier versions of the paper, and the anonymous reviewers.

7. REFERENCES

[1] Almeida Jr., J. R., Camargo Jr., J. B., Basseto, B. A., and Paz, S. M. 2003. Best practices in code inspection for safety-critical software. *IEEE Softw.* 20, 3 (May. 2003), 56-63. DOI= http://dx.doi.org/10.1109/MS.2003.1196322.

[2] Anderson, P., Reps, T., Teitelbaum, T., Zarins, M . 2003. Tool support for fine-grained software inspection . *IEEE Softw*. 20, 4. 42 – 50. DOI= 10.1109/MS.2003.1207453.

[3] Barnard, J. and Price, A. 1994. Managing code inspection information. *IEEE Softw.* 11, 2 (Mar. 1994), 59-69. DOI= http://dx.doi.org/10.1109/52.268958

[4] Brothers, L., Sembugamoorthy, V., and Muller, M. 1990. ICICLE: groupware for code inspection. In *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work* (Los Angeles, California, United States, October 07 - 10, 1990). CSCW '90. ACM, New York, NY, 169-181. DOI= http://doi.acm.org/10.1145/99332.99353

[5] Chan, L., Jiang, K., and Karunasekera, S. 2005. A tool to support perspective based approach to software code inspection. In *Proceedings of the 2005 Australian Conference on Software Engineering* (March 29 - April 01, 2005). ASWEC. IEEE Computer Society, Washington, DC, 110-117. DOI= http://dx.doi.org/10.1109/ASWEC.2005.10

[6] Dunsmore, A., Roper, M. and Wood, M. 2003.The development and evaluation of three diverse techniques for object-oriented code inspection. *IEEE Transactions on Software Engineering*, 29, 8. 677 – 686.

[7] Fagan, M. E., 1999. Design and code inspections to reduce errors in program development. *IBM Systems Journal*. 38, 2. 258 – 287. DOI: 10.1147/sj.382.0258

[8] Harjumaa, L., Hedberg, H., and Tervonen, I. 2001. A path to virtual software inspection. In *Proceedings of the Second Asia-Pacific Conference on Quality Software* (December 10 - 11, 2001). APAQS. IEEE Computer Society, Washington, DC, 283.

[9] Hatton, L., 2008.Testing the value of checklists in code inspections. IEEE Software, 25, 4. 82 – 88. DOI= 10.1109/MS.2008.100.

[10] Laitenberger, O., 1998. Studying the effects of code inspection and structural testing on software quality. In *Proceedings of the The Ninth International Symposium on Software Reliability Engineering* (November 04 - 07, 1998). ISSRE. IEEE Computer Society, Washington, DC, 237.

[11] Land, L.P.W., C. Sauer, and Jeffrey, R. 2000.The use of procedural roles in code inspections. *Empirical Software Engineering*, 5, 11 – 34.

[12] Mantyla, M.V., Lassenius, C., 2009.What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*. 35, 3. 430 – 448. DOI= 10.1109/TSE.2008.71.

[13] Nelson, S., and Schumann, J. 2004. What makes a code review trustworthy. In *Proceedings of the 37th Hawaii International Conference on System Sciences*. HICSS 2004, 1 – 10.

[14] Porter, A.A., Siy, H.P., Toman, C.A., and Votta, L.G., 1997. An experiment to assess the cost-benefits of code inspections in large scale software development . *IEEE Transactions on Software Engineering*. 23, 6. 329 – 346.DOI= 10.1109/32.601071.

[15] Remillard, J. 2005. Source Code Review Systems. *IEEE Softw.* 22, 1 (Jan. 2005), 74-77. DOI= http://dx.doi.org/10.1109/MS.2005.20

[16] Rodgers, T. L., Vogel, D. R., Purdin, T., and Saints, B. 1998. In search of theory and tools to support code inspections. In *Proceedings of the Thirty-First Annual Hawaii international Conference on System Sciences - Volume 3* (January 06 - 09, 1998). HICSS. IEEE Computer Society, Washington, DC, 370 - 378. DOI= http://dx.doi.org/10.1109/HICSS.1998.656306.

[17] Seaman, C. B. and Basili, V. R. 1997. An empirical study of communication in code inspections. In *Proceedings of the 19th international Conference on Software Engineering* (Boston, Massachusetts, United States, May 17 - 23, 1997). ICSE '97. ACM, New York, NY, 96-106. DOI= http://doi.acm.org/10.1145/253228.253248.

[18] Siy, H. and Votta, L. 2001. Does the modern code inspection have value? In *Proceedings of the IEEE international Conference on Software Maintenance (ICSM'01)* (November 07 - 09, 2001). ICSM. IEEE Computer Society, Washington, DC, 281.