The Significance of Participant Experience when Evaluating Software Inspection Techniques

David A. McMeekin, Brian R. von Konsky, Michael Robey, David J.A. Cooper
Digital Ecosystems and Business Intelligence Institute
Curtin University of Technology
Suite 3 - Enterprise Unit 4, De' Laeter Way
Technology Park, Bentley WA 6102, Australia
{D.McMeekin, B.vonKonsky, M.Robey, David.Cooper}@curtin.edu.au

Abstract

Software inspections have been used to improve software quality for 30 years. The Checklist Based Reading strategy has traditionally been the most prevalent reading strategy. Increased Object Oriented usage has raised questions regarding this technique's efficacy, given issues such as delocalisation. This study compared two OO inspection techniques: Use-Case Reading and Usage-Based Reading, with Checklist Based Reading. Students and industry professionals were recruited to participate in the study. The effectiveness of each reading strategy was analysed, and the effect experience had on inspection efficacy. The results showed no significant difference between inspection techniques, whether used by student or professional developers but a significant difference was identified between student and professional developers in applying the different techniques. Qualitative results highlighted the differences in ability between industry and students with respect to what each group considered important when inspecting and writing code. These results highlight the differences between student and industry professionals when applying inspections. Therefore, when selecting participants for empirical software engineering studies, participant experience level must be accounted for within the reporting of results.

1. Introduction

For the past 30 years, software inspections have been shown to reduce defects and improve software quality. The development of what became known as Checklist Based Reading (CBR) or 'Fagan Inspections' occurred during the time when the procedural programming paradigm was dominant. However, this has changed over the past 15 years as the Object Oriented (OO) paradigm's usage has increased. Dunsmore *et al.* [7, 8, 9, 10] highlighted current issues in inspection efficacy, believed to be caused by the changing paradigm.

Past research that developed inspection techniques specifically for OO code reported inconclusive results regarding which technique had the greatest efficacy in detecting defects and defects due to delocalisation in OO code [9, 10]. The work reported here applied the OO inspection technique, Usage-Based Reading (UBR), developed by Thelin *et al.* [31] as a refinement to the Use-Case Reading (UCR) technique. This study's two research goals were:

1. to compare the UBR, UCR and CBR techniques in a code inspection using both:
   (a) students, and
   (b) industry professionals, as participants
2. to measure differences in defect detection between industry professionals and students when using the described inspection techniques

These goals were met using three metrics for each inspection technique in each participant grouping:

1. the total number of defects detected
2. the number of defects due to delocalisation detected, and
3. the number of false positives generated

The contribution of this paper is an analysis of the effectiveness of the reading strategies, and the effect that experience has on strategy efficacy.

2 Background

A software inspection is the process of reading software artefacts to detect defects. A defect is defined as 'a deviation from the specification that would go on to cause failure

1530-0803/09 $25.00 © 2009 IEEE
DOI 10.1109/ASWEC.2009.13

IEEE computer society

(some undesired behaviour of either a trivial or catastrophic nature) if left uncorrected' [6]. Inspections are considered one of the most effective software engineering practices to ensure quality [18]. Since their formalisation, inspections have been shown to reduce defect numbers in both design and code, reduce testing time, and improve production cost effectiveness [11, 12, 32, 18, 17].

The OO programming paradigm introduced many features not commonly available in procedural programming, such as function and operator overloading, inheritance, dynamic (late) binding and polymorphism. It was expected that these features would improve software quality with better data abstraction and information hiding, more readily allowing for concurrency and adaptability to changes in the real world. These added features increased program complexity, causing other problems within these systems [2, 19, 35, 36].

2.1 Delocalisation

A major challenge faced by OO code inspectors is delocalisation. First described by Soloway *et al.* [28], delocalisation is where related code is spread throughout many different locations.

OO programs cannot simply be read from top to bottom. An OO program written in a delocalised manner becomes difficult to understand as inspecting a single class may rely on several other classes for it to be fully understood. Code relied upon by the class under inspection may be contained in another class. If the class containing this code is not within the scope of the inspection, inspectors make assumptions about it. These assumptions may be correct or incorrect. Accurate assumptions are difficult to make when inspecting code in delocalised components. This problem is not unique to OO programs. Programs written in a procedural technique also face similar issues with function calls. However, it appears that the OO paradigm greatly increased the delocalisation problem [10].

2.2 Reading Techniques

A reading technique is a step by step procedure implemented by an inspector or inspection team to detect defects within a software system. Implementing a reading technique allows methodical examination of the artefact under inspection and provides avenues for correction and feedback, hence improving final product quality [25].

2.2.1 Checklist Based Reading (CBR)

CBR is considered the standard inspection technique used by software developers[18] and was first described by Fagan [11]. The inspector reads a pre-determined amount of code. During this reading phase, checklist questions guide the inspector in identifying common coding errors. A 'yes' answer indicates no defect while a 'no' answer indicates there may be a defect. Checklist questions should be developed from the accumulation of historical defect data [13, 16]. As CBR is considered standard, it should be used as a baseline within empirical studies comparing inspection techniques [31].

2.2.2 Use Case Reading

The UCR technique was designed for inspecting OO systems, taking into account dynamic interaction with collaborating objects. The goal is to ensure that every object responds in a correct manner given their role in executing the scenario. That is, the correct methods are called and state changes are consistent and correct [10].

Inspectors create multiple scenarios using each system use case. Each scenario is then traced through a sequence diagram, examining the method calls and changes in the object state to see how the class under inspection implements them. When encountering the class under inspection within the sequence diagram, the inspector changes artefacts and inspects the code to verify the expected behaviour. At the completion of the inspection, the object's state should match the expected state listed in the scenario. If the states do not match, it suggests there is a defect within the code. In this manner, the object usage context is also inspected. The assumption is that incorrect, missing or error-prone method calls will be discovered using this technique. This implementation means that certain methods within a class will not be inspected from the scenarios created. The uninspected methods must be inspected by using other techniques [10].

2.2.3 Usage-Based Reading

UBR is an inspection technique from within the Scenario-Based Reading family SBR [1]. UBR is similar to UCR (described earlier) but was derived from the Statistical Usage Inspection (SUI) technique. The SUI technique focuses on product reliability certification by testing it according to the system's expected usage [20]. UBR attempts to guide the inspectors in finding defects considered to have the most negative impact upon the program's usability [31].

In UBR, use cases are prioritised according to the way in which the system will be used. Using the requirements document as the reference point, the use case with the highest priority is selected and traced by manual execution through the design document. Tracing the execution ensures that the use case goals are achieved and if not this is noted. This inspection is repeated until each use case has been inspected.

2.2.4 Prior Studies

Dunsmore *et al.* [9, 10] conducted an OO code inspection, comparing CBR with the two techniques, UCR and Abstract Driven Reading, developed for OO code. Dunsmore *et al.* reported a significant difference at the 10% level in the number of detected defects between the different techniques.

Thelin *et al.* [31] conducted an inspection on OO requirements and design documents comparing the CBR technique with the UBR inspection technique. They reported a significant difference between the techniques in the number of defects detected at the 5% level in detected defects.

2.3 Participants: Students or Industry

Software engineering empirical research aims to improve software quality through developing and improving design and development processes and methodologies, inspection and maintenance techniques as well as tool support needed for these processes [27]. This means research results should be transferable from academia into industry and generalizable to the wider software engineering community.

Table 1 displays a summary of software engineering research that discussed differences in results when using student and industry participants within the studies. The summary indicates results generated from within empirical studies should be interpreted in the context of participants involved within the study.

Table 2 lists empirical research that highlights results and recommendations from the literature relating to the use of student and/or novice programmers.

Tables 1 and 2 demonstrate using both student and professionals as participants within software engineering empirical research has advantages and disadvantages. A major advantage is the homogenous nature of the participants in that an experience baseline can be set for students, each having successfully completed a certain section of their degree program. Using industry professionals within a study is advantageous as results can be generalised to the wider developer community. The anomaly introduced by professionals is the *extent* and *variation* of their experience within the area being studied.

Examining the studies and results listed in Tables 1 and 2 this study uses both students and industry professionals. Results from this study can therefore be used to: examine students' application of inspection techniques, examine industry professionals' application of inspection techniques and permit results to be compared student to student, professional to professional and student to professional.

The studies of Dunsmore *et al.* [10] and Thelin *et al.* [31] have been well cited within the literature and because of this, our study has focussed specifically on those two studies. Table 3 shows a comparison between the different aspects of Dunsmore *et al.* [9], Thelin *et al.* [31] and this study.

Author	**Summary of comments**
Weinberg [34]	Using trainees for studies gives results about trainees
Curtis [4]	Generalizations about professionals should be careful when novices are used
Curtis [5]	Results about novices should produce better teaching methods and tools
Potts [23]	Research occurs, results derived and then industry should be involved in the application of the results
Glass [14]	Software engineering research has been done independent of industry
Votta [33]	Highlights that recommendations from [5] haven't been implemented
Sjøberg [26]	Studies should reflect industrial setting
Sjøberg [27]	The lack of reality in studies can cause technology transfer from academia to industry to be very slow

Table 1. Listing of articles discussing participants within empirical research.

3 Methodology

This empirical study compared the UBR, UCR and CBR inspection techniques and the impact of inspector experience level on their efficacy. It investigated the differences reported in Dunsmore *et al.*and Thelin *et al.*'s prior research [9, 10, 31]. The prior research was linked by controlling two independent variables found in the previous studies:

1. the reading technique implemented by each participant
2. the participants' experience level.

3.1 Artefacts

For this study, the artefacts from Dunsmore *et al.* [9, 10] were chosen. This enabled a comparison of inspection results returned from the participants' code inspections using different inspection techniques. The code contained 14 seeded defects. A composite checklist combining the salient features used by Dunsmore *et al.* [9, 10] and Thelin *et al.* [31] was also developed. The use cases were prioritised for use with the UBR inspection technique.

Author	**Objectives**	**Participants**	**Conclusions**
Porter *et al.* [22]	Tested different inspection methods	Stud.	Scenario method returned best results
Porter & Votta [21]	Replicated previous experiment	Industry	Students and industry results similar
Höst *et al.* [15]	Factors affecting lead time	Stud./Ind	Same under certain conditions
Carver *et al.* [3]	Why use students in research	Stud.	Useful to test hypotheses with
Runeson [24]	Using PSP to measure improvement	Stud./Ind	Same but more studies needed
This study	**OO code inspection**	**Stud./Ind**	**Significantly different results**

Table 2. Listing of articles discussing results of using students within empirical research.

	Thelin *et al.* [31]	**Dunsmore *et al.* [10]**	**This study**
Inspected artefacts	Design documents	Source code	Source code from [10]
Participants	4^{th}year Masters with industry experience	3^{rd}year Honours	3^{rd} and 4^{th}year students, industry professionals
Inspection techniques	Prioritised Use Cases (UBR), Checklist (CBR)	Non–prioritised Use Cases (UCR), Checklist (CBR), Abstract Driven (ADR)	Prioritised Use Cases (UBR), Checklist (CBR), Non–prioritised Use Cases (UCR)
Defects	Categorised in importance	Uncategorised	Uncategorised
Checklists	Created for their study	Created for their study	Combined the two studies checklists

Table 3. Breakdown of the Studies.

The artefacts presented to participants included a natural language specification, a class specification, a class diagram of the entire system, the Java code to be inspected, a defect reporting form and a feedback form. Access to four other classes within the system was provided as well as access to the Java API documentation. The code to be inspected contained approximately 200 lines of code which falls within the recommended number for a 120-minute inspection period [11, 13, 29]. Participants were given an instruction sheet that described the inspection method they were to use. Those performing the checklist inspection were presented with a checklist while those performing the use case and usage based inspections were presented with use case scenarios, a sequence diagram and a scenario sheet. The use cases were prioritised for the usage based inspection.

3.2 Participants

Participants for this study were from two differing groups: 1) students and, 2) industry professionals. The student participants volunteered for the study and took part in their own time. It was not part of formal course work. Student participants were enrolled within one of three degrees: Bachelor of Science (Computer Science), Bachelor of Science (Information Technology) and Bachelor of Engineering (Software Engineering). Student participants were required to be in the third or fourth year of their degree and have successfully passed two introductory Java courses and two Software Engineering courses. By using this criteria as a baseline for student participants, an assumption was made that students had a similar base knowledge.

Practising IT professionals from local industry also took part in this study. Table 4 shows the number of industry participants with their corresponding years of experience using each of the technologies listed. Participants experience varied based on companies for which they had worked. As an example, participants had worked with companies currently building mission critical Java systems, enterprise Java systems, and C++ mobile phone applications.

Participants within both groupings had no prior knowledge of the system artefacts to be inspected.

3.3 Procedure

The code inspection was an individual task and participants were asked not to interact with others during the inspection. Student participants were also asked not to discuss the inspection with other students after the study as this could effect results if those spoken to chose to participate at a later stage. Participants noted their start and finish times as well as the time they discovered a defect and a brief description of the defect. Providing a fix for the defect was not

	<1yr	1-2yrs	2-4yrs	>4yrs
In industry	4	5	1	17
Working with Java	0	5	4	18
Working with UML	4	5	10	8
Working with OO	0	3	5	19

Table 4. Number of industry participants' experience levels in years with the different technologies (all industry participants' had experience with the 4 different technologies).

required as the inspectors' task is simply to identify them. Participants were told that the code compiled and executed. Because inspections are a static task, participants were not permitted to compile and run the code themselves. To commence, participants read the natural language specification and class descriptions of the system and its functionality.

Participants completing the CBR inspection answered each question on the checklist and participants completing the UCR and UBR inspections used the provided use cases to guide them as they read through the sequence diagrams. A reference in the sequence diagram to the code being inspected required the participant to examine that code section for correctness. The UBR inspection participants were informed of the use case prioritisation and asked to take that into consideration as they inspected the code.

3.4 Data Collection and Analysis

Defect data was collected by participants entering a detected defect into a defect recording sheet. They recorded a defect number, the class and line number where it was located and a brief description of the defect. Each participant completed a demographic questionnaire prior to starting and a second questionnaire upon finishing the inspection.

The statistical data was analysed using the R software package.

3.5 Threats to Validity

Several internal threats were identified in this study, similar to those listed in [9]. The first one was a potential selection effect resulting from experience levels and participant diversification not being evenly distributed. With the student participants, information that would have clarified this was not directly available to the researchers. Instead, student participants were asked if they met the minimum requirements. Industry professionals were also asked if they met the minimum requirements.

The demographic breakdown of the students, and the inspection method used may have also affected the reported results. To minimise this threat, the inspection technique allocation was randomised by assigning the technique to the participant as they entered the room. The first person was given CBR, the second UCR, the third UBR and then they cycle started again.

Participant motivation to discover the defects may have been increased with the knowledge that defects existed within the system.

A learning curve involved with using the techniques may have been involved. In [9, 10] participants were familiar with the system under inspection and they had participated in training sessions. In both [9, 10] and [31] the study was run as part of an academic unit contributing to their overall university qualification. The student participants were also trained using the inspection technique and were enrolled in units related to inspections, verification and validation. This was not the case with this study, as students may have received a single lecture that discussed inspection, so it was expected that the overall mean from this study would be lower than those reported in [9, 10] and [31].

The first external threat to validity was that the code used may not have reflected industry software. The code inspected was part of a larger system thus increasing system complexity and somewhat reducing this effect. The second threat was that seeded defects differed from those that would be seen in industry software. In mitigating this threat, defects were seeded based on literature surveys, an industrial survey and prior studies [9, 10].

4 Results

4.1 Students

There were 36 student participants: 12 completed the UBR inspection, 14 completed the CBR inspection, and 10 completed the UCR inspection. Figure 1 depicts the average number of times each seeded defect was discovered by inspectors using the different techniques. Table 5 presents an overall summary of the results.

Table 5 shows students using CBR and UBR techniques returned similar results with very little difference between the total number of defects detected by students using those two techniques. Comparing UBR and UCR, UBR appears to have performed better than UCR in detecting defects and defects due to delocalisation. Both techniques returned the same average result for the number of false positives generated. Students using CBR, generated more false positives compared to both UBR and UCR.

		Inspection Technique		
		UBR	**CBR**	**UCR**
	N	**12**	**14**	**10**
Total defects (14)	Mean	3.4	3.2	1.7
	Std. Deviation	2.7	2.5	1.7
	Std. Error Mean	0.8	0.7	0.5
	Minimum	0	0	0
	Maximum	9	10	5
Delocalised Defects (8)	Mean	2.2	1.5	1.0
	Std. Deviation	1.9	1.4	1.1
	Std. Error Mean	0.5	0.4	0.3
	Minimum	0	0	0
	Maximum	3	5	5
False Positives	Mean	0.8	2.0	0.8
	Std. Deviation	1.7	1.9	1.9
	Std. Error Mean	0.5	0.5	0.6
	Minimum	0	0	0
	Maximum	6	6	6

Table 5. Student participants results summary.

Figure 2 shows the median detection values were close together, and an outlier within the CBR data. A close examination of participant 21's (the outlier) defect reporting form indicated that this participant took 78 minutes for the inspection, 24 minutes longer than any other student using this technique. The participant reported no false positives and no other data was found to indicate any anomaly with the participant's results, hence their results remained within the data set.

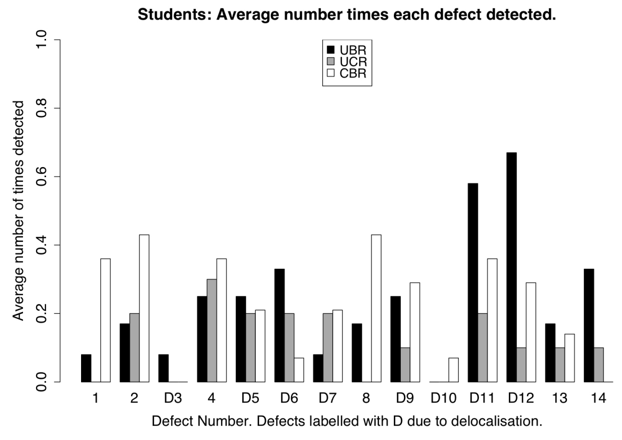

Figure 1. Students: Average number of times each defect was discovered.

The collated data was then analysed using a Kruskal-Wallis test to determine if there was a significant difference between the three inspection techniques: for all defects, for delocalised defects, and for false positive generation.

The Kruskal-Wallis test for each metric showed the Asymptotic Significance to be 0.2. This indicates no statistically significant difference between the UBR, CBR and UCR inspection techniques for the three metrics. The three inspection techniques therefore did not outperform nor under perform in enabling or hindering total defect discovery using student participants.

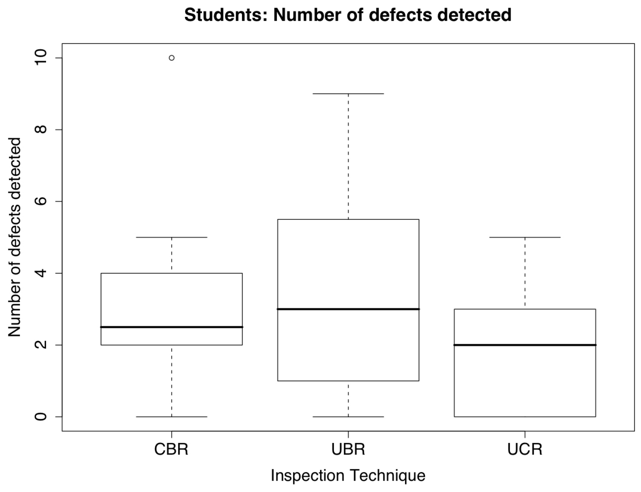

Figure 2. Students: Boxplot of number of defects discovered by each technique.

4.2 Industry

There were 26 industry professionals: 8 completed the UBR inspection, 8 completed the CBR inspection, and 10 completed the UCR inspection. Figure 3 depicts the average number of times each defect was detected using the different inspection techniques and Figure 4 shows the different distributions of results for the various techniques. Table 6 contains the results' descriptive statistics for the overall defects, the defects due to delocalisation and the number of false positives generated by participants using the different techniques.

The three techniques returned similar means to each other for all defects, and defects due to delocaisation. The UCR technique however returned zero false positives, but both UBR and CBR returned very low false positive means. Figure 4 shows the three techniques' median detection rates lay close together. It also shows a possible outlier. Examining the outlier's reporting form indicated no reason to justify removing their data from the sample and so the data was included.

A Kruskal-Wallis test was used to determine if there was a statistically significant difference between the three inspection techniques, first for all defects, then for delocal defects and finally for false positive generation. The

		Inspection Technique		
		UBR	CBR	UCR
Total defects (14)	***N***	**8**	**8**	**10**
	Mean	5.2	5.6	4.4
	Std. Deviation	3.6	1.9	1.9
	Std. Error Mean	0.7	0.5	0.8
	Minimum	0	2	1
	Maximum	10	8	7
Delocalised Defects (8)	Mean	2.9	2.6	2.4
	Std. Deviation	2.3	1.1	1.4
	Std. Error Mean	0.8	0.4	0.4
	Minimum	0	1	1
	Maximum	6	4	5
False Positives	Mean	1.1	0.5	0.0
	Std. Deviation	1.9	1.9	1.7
	Std. Error Mean	2.8	0.5	0.0
	Minimum	0	0	0
	Maximum	8	2	0

Table 6. Industry participants results summary.

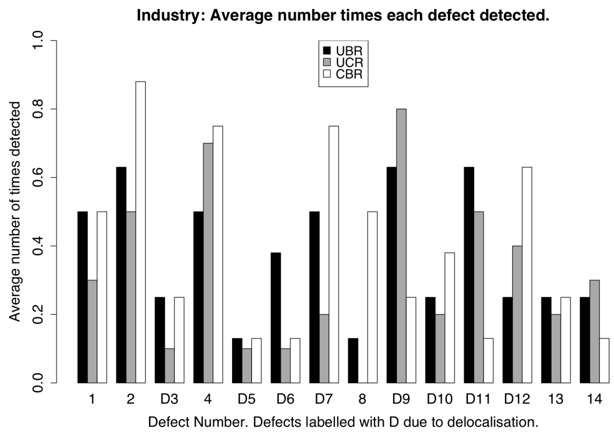

Figure 3. Industry: Average number of times each defect was discovered.

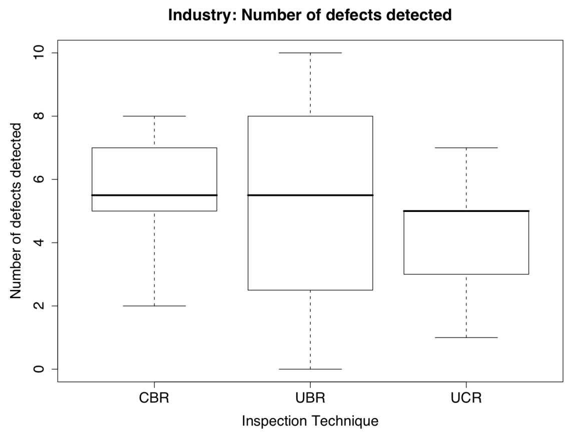

Figure 4. Industry: Boxplot of number of defects discovered by each technique.

test results (Asymptotic Significance = 0.4, 0.7 and 0.1 respectively) indicated no statistically significant difference between the inspection techniques. The application of the three inspection techniques, by industry participants for defect detection did not outperform nor under-perform each other in either hindering or aiding defect discovery.

4.3 Students vs Industry

This section compares the overall results returned by the student grouping with the results returned by the industry professionals.

Table 7 breaks down the data from the two groupings. It shows that industry professionals returned better results than students for all three inspection techniques with respect to: detecting defects, detecting defects due to delocalisation and false positive generation.

	Technique		
Students	**UBR**	**CBR**	**UCR**
All defects	3.4	3.2	1.7
Delocal	2.2	1.5	1.0
False positives	0.8	2.0	0.8
Industry	**UBR**	**CBR**	**UCR**
All defects	5.2	5.6	4.4
Delocal	2.9	2.6	2.4
False positives	1.1	0.5	0.0

Table 7. Mean of defects results summary: Student vs Industry.

A statistical analysis was conducted using the Mann-Whitney test comparing student and industry professional results for defect detection, detection of defects due to delocaisation and the number of false positives generated. The test results (Asymptotic Significance = 0.001, 0.02 and 0.01 respectively) indicated that professional developers performed significantly better than student participants all three tested areas.

4.4 Qualitative Data

After completing the inspection task, both participant groupings gave feedback about the task. Over 60% of professionals noted that both the code and diagrams contained no documentation. Professionals noted this as an important defect. From the student participants, only 17% made any reference to the lack of documentation and none noted it as a defect.

Eight industry participants noted that IDE usage would have simplified the inspection, yet no student made reference to IDE usage. Industry participants referred to the need for access to testing data to aid the inspection. It was not that the testing data would replace the inspection, but with the data it would assist with focusing their effort. No student mentioned the need for access to testing data.

Industry participants found CBR restrictive and focussed them on program structure rather than business logic. This, despite the fact that the quantitative data within this study showed industry inspectors using CBR, on average, detected more defects than other inspectors. Industry participants also identified that UCR and UBR allowed for defects to be missed as certain code was not included within the use case scenarios.

Students found CBR easy to follow and gave a great structure for how to go about inspecting code.

5 Discussion

Within both the student and industry groups the results showed no statistically significant difference between the three inspection techniques in defect detection, detection of defects due to delocalisation and false positive generation. The UBR's high mean number of defects compared to UCR, although not statistically significant, supports Dunsmore *et al.* [9] conclusions that further refinement of UCR is needed. UBR is a refined version of the UCR technique and performed somewhat better than the original technique.

The overall results were consistent with those found by Dunsmore *et al.* [9, 10], with no statistically significant difference between the techniques compared at the 5% level. These results differ from those reported by Thelin *et al.* [31] and in [30] and [37] which replicated the original study by Thelin *et al.* [31]. They reported that the UBR method performed significantly higher than the CBR technique. It must be noted that those studies were performed on design documents and not code and this may account for the significantly different results.

Another point to note: students using CBR generated a slightly higher false positive count than students using the other techniques, on average. Although it wasn't significantly higher, in an industry setting this could become expensive if developers are deployed to fix these defects only to discover they aren't actually defects.

Examining comments participants demonstrated other differences between students and industry participants not reflected by the quantitative analysis. This data demonstrates that differences between students and industry professionals cannot be quantitatively analysed in isolation. Quantitative results should be examined in conjunction with the qualitative data to be completely understood.

Industry participants appeared to have an intuitive understanding that business logic was more important than simply finding structural defects. Also, industry participants appeared to intuitively prioritise what they believed to be important within the code whereas students did not.

In and of itself the results from both the quantitative and qualitative data, are not surprising. These results strongly indicate though the continued and increased need for industry participation within software engineering empirical research.

6 Conclusion

Regarding the first research goal stated in this study: there was no statistical difference between the tested inspection techniques, UBR, UCR and CBR, in the areas of defect detection, defects due to delocalisation detection and the number of false positives generated within each participant group. Looking at this result alone could lead to an incorrect conclusion that inspection technique has no impact upon the number of defects discovered. The qualitative data demonstrates that the quantitative data should be interpreted in light of the qualitative data.

Regarding the second research goal state in this study: an inspectors experience level has a significant impact upon their ability to detect: defects, defects due to delocalisation and the number of false positives they generate. This should be accounted for when examining quantitative data as the qualitative data may reflect differences that would go unnoticed without this data collected and analysed.

Sjøberg *et al.* [27] state there is need for the software engineering community to set benchmarks for empirical research. This study provides further evidence to support this statement. Benchmarks will improve ways to compare results between studies and provide a way for meta-analysis to occur upon the empirical study data that is reported. These benchmarks need to include how, when, and where student and industry professionals should be used within studies and how studies can generalise results according to artefacts used and participants involved. Establishing these benchmark types may enable a greater transfer of technology from academia into industry.

References

[1] V. Basili. Evolving and packaging reading technologies. *Journal of Systems Software*, 38(1):3–12, 1997.

[2] G. Booch. Object-oriented development. *IEEE Transactions on Software Engineering*, 12(2):211–221, 1986.

[3] J. Carver, L. Jaccheri, and F. S. Morasca, S.; Shull. Issues in using students in empirical studies in software engineering education. In *Proceedings: Ninth International Software Metrics Symposium 2003.*, pages 239–249. IEEE, 2003.

[4] B. Curtis. Measurement and experimentation in software engineering. In *Proceedings of the IEEE*, pages 1144–1157. IEEE, 1980.

[5] B. Curtis. By the way, did anyone study any real programmers? In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 256–262, Norwood, NJ, USA, 1986. Ablex Publishing Corp.

[6] A. Dunsmore. *Investigating effective inspection of object-oriented code*. PhD thesis, University of Strathclyde, Glasgow, 2002.

[7] A. Dunsmore, M. Roper, and M. Wood. Object-oriented inspection in the face of delocalisation. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 467–476, Limerick, Ireland, 2000.

[8] A. Dunsmore, M. Roper, and M. Wood. Systematic object-oriented inspection - an empirical study. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 135–144, Toronto, Ontario, Canada, 2001.

[9] A. Dunsmore, M. Roper, and M. Wood. Further investigations into the development and evaluation of reading techniques for object-oriented code inspection. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 135–144, Orlando, Florida, U.S.A, 2002.

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

[11] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, Mar. 1976.

[12] M. E. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751, July 1986.

[13] T. Gilb and D. Graham. *Software Inspection*. Addison–Wesley, Wokingham, 1993.

[14] R. Glass. The software-research crisis. *Software*, 11(6):42–47, Nov. 1994.

[15] M. Höst, B. Regnell, and C. Wohlin. Using students as subjects–a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201–214, Nov. 2000.

[16] W. Humphrey. *A Discipline for Software Engineering*. Addison–Wesley, Boston, 1995.

[17] W. S. Humphrey. *Introduction to the team software process*. Addison–Wesley, Massachusetts, 2000.

[18] O. Laitenberger and J. DeBaud. An encompassing life cycle centric survey of software inspection. *Journal of Systems and Software*, 50(1):5–31, 2000.

[19] M. Lejter, S. Meyers, and S. P. Reiss. Support for maintaining object-oriented programs. *IEEE Transactions on Software Engineering*, 18(12):1045–1052, 1992.

[20] M. Olofsson and M. Wennberg. Statistical usage inspection. Master's thesis, Department. of Communication Systems, Lund Institute of Technology and Ericsson Telecom AB, 1996.

[21] A. Porter and L. Votta. Comparing detection methods for software requirements inspections: A replication using professional subjects. *Empirical Softw. Engg.*, 3(4):355–379, 1998.

[22] A. Porter, L. Votta, and V. Basili. Comparing detection methods for software requirements inspections: a replicated experiment. *IEEE Transactions on Software Engineering*, 21(6):563–575, June 1995.

[23] C. Potts. Software-engineering research revisited. *Software*, 10(5):19–28, Sept. 1993.

[24] P. Runeson. Using students as experiment subjects an analysis on graduate and freshmen student data. In *EASE'03 - Proceedings 7th International Conference on Empirical Assessment and Evaluation in Software Engineering*, pages 95–102. BCS Publishing, 2003.

[25] F. Shull, I. Rus, and V. Basili. Improving software inspections by using reading techniques. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 726–727, Toronto, Ontario, Canada, 2001.

[26] D. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E. Koren, and M. Vokac. Conducting realistic experiments in software engineering. In *ISESE'02: Proceedings of the 2002 International Symposium on Empirical Software Engineering*, pages 17–26. IEEE, 2002.

[27] D. Sjøberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasavanovic, N. Liborg, and A. Rekdal. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9):733–753, Sept. 2005.

[28] E. Soloway, J. Pinto, S. Letovsky, D. Littman, and R. Lampert. Designing documentation to compensate for delocalized plans. *Communications of the ACM*, 31(11):1259–1267, Nov. 1988.

[29] I. Sommerville. *Software Engineering*. Addison–Wesley, Harlow, sixth edition, 2001.

[30] T. Thelin, C. Andersson, P. Runeson, and N. Dzamashvili-Fogelstrom. A replicated experiment of usage-based and checklist-based reading. In *METRICS '04: Proceedings of the Software Metrics, 10th International Symposium on (METRICS'04)*, pages 246–256, Washington, DC, USA, 2004. IEEE Computer Society.

[31] T. Thelin, P. Runeson, and C. Wohlin. An experimental comparison of usage-based and checklist-based reading. *IEEE Transactions on Software Engineering*, 29(8):687–704, Aug. 2003.

[32] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili. Detecting defects in object-oriented designs: using reading techniques to increase software quality. In *OOPSLA '99: Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 47–56, Denver, Colorado, United States, 1999.

[33] L. Votta. By the way, has anyone studied any real programmers, yet? In *Proceedings of the Ninth International Software Process Workshop*, pages 93–95. IEEE, 1994.

[34] G. M. Weinberg. *The psychology of computer programming silver anniversary edition*. Dorset House Publishing, New York, NY, USA, 1998.

[35] N. Wilde and R. Huitt. Maintenance support for object-oriented programs. *IEEE Transactions on Software Engineering*, 18(12):1038–1044, 1992.

[36] N. Wilde, P. Matthews, and R. Huitt. Maintaining object-oriented software. *IEEE Software*, 10(1):75–80, Jan. 1993.

[37] D. Winkler, M. Halling, and S. Biffl. Investigating the effect of expert ranking of use cases for design inspection. In *EUROMICRO '04: Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, pages 362–371, Washington, DC, USA, 2004. IEEE Computer Society.