

Improving the Software Inspection Process

Tor Stålhane, PhD and Tanveer Husain Awan, MSc

Norwegian University of Science and Technology
stalhane@idi.ntnu.no
BEKK Consulting, Oslo, Norway
Tanveer.Awan@bekk.no

Abstract. In this paper we look at the results from three experiments. We discuss the results and combine them into advices for code inspection. The main observations are that 1) it is beneficial to use large inspection groups in order to have access to a large amount of diverse experience and knowledge, 2) hands-on experience is more important than general knowledge and experience and 3) if left on their own, large groups tend to use a voting-like mechanism when deciding which defects to report after the group meeting.

1 Introduction

Code reading and code inspections is a hot topic in software development. We already know that the number of defects found during inspection varies greatly – the problem is why they do so and what we should do about it. Following the TQM philosophy, we see this variation as an opportunity – some persons find many defects. Thus, we know it is possible. How can we repeat their successes? We also know that many persons find quite few defects. Thus, we know there is a danger. How can we avoid it? The same questions apply to inspection meetings – some work well while some do not contribute in the defect detection process at all. What can we learn here?

The discussions and results presented in this paper are based on a set of code inspection experiments done by one of the authors during his sabbatical stay at the CSE, UNSW [1] plus two experiments run at NTNU by the other author as part of his master degree in software engineering [2], [3].

The rest of this paper is organized as follows: First we present a summary of earlier code inspection experiments. We then give a short description of the UNSW experiment and a detailed description of the experiments done at NTNU. We go on to present the data analyses and the main results before presenting our conclusions and some ideas for further work.

2 Related Work

Code reading and code inspection has been a popular field for experiments. There are several reasons for this – for instance that the technique is important and that the results from the experiments are easy to analyze using simple statistical methods.

Already in 1978, Meyers run an experiment at the IBM Systems Research Institute where he compared testing and inspections [4]. The experiment showed no significant differences between the methods when it comes to error detection efficiency but testing showed a higher variability than inspection.

In 1987, Basili presented an experiment where he compared code inspection to two testing methods [5]. While code reading came out on top among professional developers, the results varied quite a lot for the students participating in the experiment. Porter and Votta ran an experiment in 1994 where they compared scenario-based inspection to checklist-based inspection [6]. The experiment used both individual inspections and a following inspection group meeting. For the defects touched by the scenarios, the scenario-based inspection was the most effective. For the other defects, no significant difference was observed. A related method – perspective-based code reading, was used in an experiment by Laitenberg in 1999 [7] where he found that this method outperformed checklist-based inspection.

Basili, Porter and Votta have later run several related experiments, Basili in 1996 [8] and in 2001 [9] and Porter and Votta in 1997 [10]. Votta found that the inspection effectiveness was not affected by team size and there was a great variation between the groups in the number of defects found. The same was observed by Kelly [11].

The reported works have made a lot of observations and some interesting conclusions. What is missing is a thorough analysis of the causes for the variations and how they can be used in an SPI process to improve inspection. Our contribution is to add this perspective.

3 Code Reading, Variation and SPI Opportunities

The statistical control view of improvement insists that we must first reduce the variation, then understand the cause – effect relationship and then improve the process. The main argument behind this approach is that a great variation will mask any possible effect of an improvement action.

There is, however, another way to look at variations – they tell you how good or bad things might turn out. One of the first to raise this issue was E. Auråen [12]. Following his idea, we can use the results from the inspection experiments as follows: Some inspections or inspection groups find more defects than the rest. We should learn how to repeat their success. Some inspections or inspection groups find fewer defects than the rest. We should learn how to avoid their failures.

In the same way, we have observed that several organizations, including large organizations like Sun Microsystems, have skipped the inspection meeting since they, on the average, do not contribute to the number of defects identified. Again we have a case where an activity sometimes helps and sometime has no beneficial effect whatsoever. Again the SPI opportunity has been missed – why does the inspection meeting sometimes work, while at other occasions it has turned out to be counterproductive?

If we do not grab the SPI opportunities offered by the experiments and experience but instead build our decisions on discussions around average values, we will miss an important opportunity to achieve substantial improvements.

4 The Experiments in Detail

4.1 The UNSW Experiments

One of the foci for these experiments was to see if the group meeting had a beneficial effect on the number of defects found. The three experiments were all run in the same way - the participants (120 students) were given a piece of code with a set of seeded defects. They inspected the code and reported the defects found before meeting as a group to decide on the final inspection report. In each experiment, all participants inspected the same code containing the same seeded defects.

In order to study the effects of the inspection meetings we introduce the two terms nominal group (NG) and real group (RG). The nominal group contains the number of defects identified by the group members *before* the inspection meeting – $NG = |\text{Union of defects found by each group member}|$ - while the real group contains the number of defects reported by the group after the inspection meeting.

The difference between the number of defects found by the nominal group (NG) and the defects found by the real group (RG) is a measure of the effect of the group meeting. The NG - RG distribution for each experiment is shown in figure 1 below.

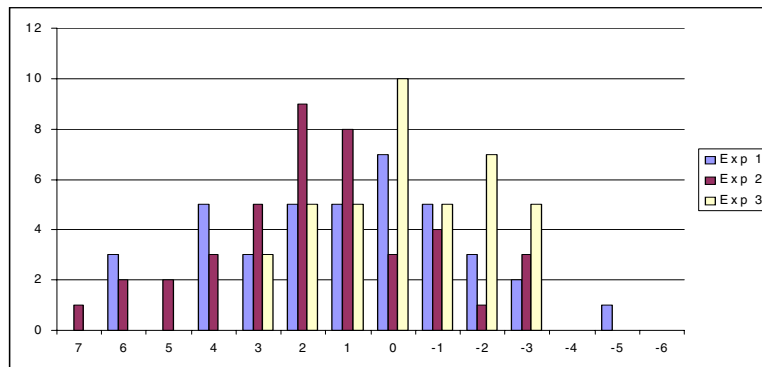


Fig. 1. The distribution of the difference between nominal and real groups, NG - RG

The differences between the numbers of defects found in the nominal group and in the real group from all three experiments yielded the same standard deviation. The mean value of the differences changed, however, significantly from experiment 2 with a mean of 1.6 to experiment 3 with a mean of 0.3. This difference is significant at the 1.4% level and it seems that the gain / loss value moves towards a symmetrical distribution.

The most surprising info from the data shown in figure 1 is the size of the variation - $SD = 2.5$. In some real groups they find much more defects – up till five defects more and thus have a process gain – while in some real groups they find much fewer defects – up till seven defects less and thus have a process loss. If we pool all the data shown in figure 1 together, we find a probability of 53% for process loss (64/120) and a probability of 30% for process gain (36/120).

The picture becomes much clearer if we make a table showing the number of defects reported from the group meeting (RG) and the number of group members that identified each defect during their individual inspections.

Table 1. How many persons in a group identified each defect?

	Number of persons who found a defect during individual inspection				
	Experiment	0	1	2	More than 2
Defects reported by real group	1	138	134	82	54
	2	109	145	48	8
	3	90	87	32	17
Defects <i>not</i> reported by real group	1	896	149	26	3
	2	1070	160	16	2
	3	454	69	10	1

Based on the data shown in table 1 we can estimate $P(\text{in RG} \mid \text{found by } n \text{ in NG})$. The result is shown in the graph below.

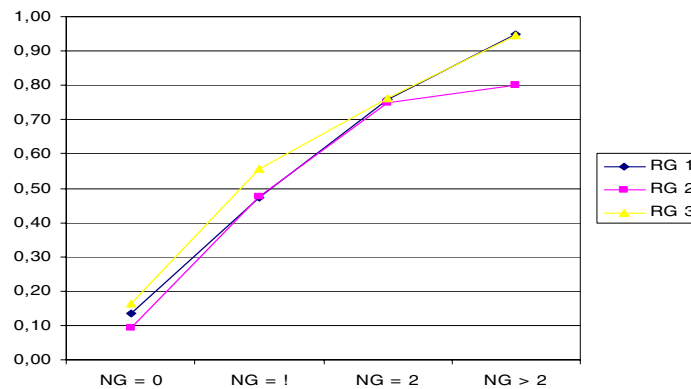


Fig. 2. The probability of reporting a defect as a function of persons that found it

Even though all persons participating in the inspection were instructed to include all defects in the final report, it seems that they ignored this instruction.

If nobody in the nominal group had found the defect during individual inspection, there was a 10% probability that the group would find it during the meeting. If only one person in the group found the defect there was a 50% probability that the group would report it. If more than two persons found the defect – mostly meaning all in the group – the probability of reporting the defect rose to between 80 and 95%. The group does not necessarily use a voting process – group pressure would give the same effect.

We can use these data in two ways – the standard solution is to skip the final inspection meeting since there is a 53% probability of process loss as opposed to only a 30% probability of process gain. The other, more productive way of using the results, is to ask the question – how can we assure a process gain in the inspection meeting? We already know that it is possible.

If we take a closer look at the data, an interesting pattern is revealed. The diagram below show three percentages from the group process: defects not found during individual inspections but identified during the inspection meeting – denoted New – the defects found during individual inspections and reported from the meeting – denoted Retained – and the defects found during individual inspections but not included in the inspection report – denoted Removed.

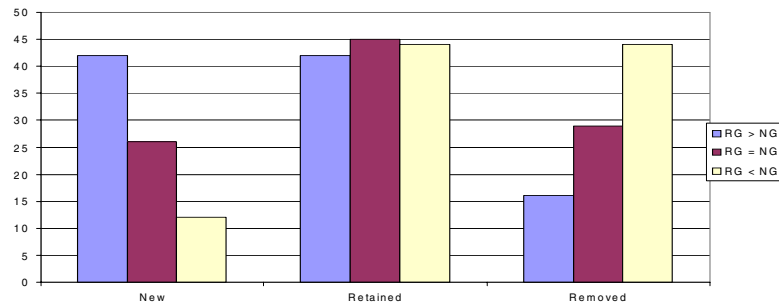


Fig. 3. New, Retained and Removed defects in experiment 3

The three columns show the data for three categories of groups – the real group performed significantly better than the nominal group – $RG > NG$ – the real group and the nominal group were about equal – $RG = NG$ – and the real group performed significantly worse than the nominal group – $RG < NG$. We see that the three categories keep about the same percentage of the defects from the nominal group – 44% on the average. What make the difference, however, is that on the average, in groups were we experience process

- Loss, we find few new defects (12%) but remove many old ones (44%).
- Gain, we find a lot of new defects (42%) and reject few of the old (16%).
- Stability, we find some new defects (26%) but remove approximately the same amount of the old ones (28%).

We thus have two important mechanisms that must be understood in order to improve code inspections – why are already identified defects removed and why do they find so few new ones? In our opinion, both effects can be satisfactory explained by the mechanism shown in figure 2 – the voting-like mechanism. Strong tendencies to use this mechanism in the group process will both make it easy to throw out already identified defects and make it difficult to include new ones.

4.2 NTNU Experiment 1

The first NTNU experiment [2] was concerned with two issues – group size and the use of checklists. Checklists have been heavily used in inspection experiments and, as should be expected, the results vary. The experiments were run with 20 NTNU students and had two phases – one where the student inspected the code alone and one where the students got together in groups to construct a final list of defects, just as in

the UNSW experiment. Ten students used a tailor-made checklist and ten students used an ad-hoc approach. The code they inspected was 130 lines of Java code with 13 seeded defects.

We used a t-test to compare the results from the students using checklist and the students that used an ad-hoc approach. As expected, the groups that used the checklists did significantly better – the checklist group found on the average 8.4 defects while the ad-hoc group found on the average 6.7 defects. The difference is significant at the 1.4% level.

In order to check that we have enough participants we use the standard relationship between the sample size, effect size and the probabilities for a type I or type II error – commonly denoted α and β respectively. If we use $\alpha = 0.05$ and $\beta = 0.20$, we have

$$N = \frac{32}{ES^2} \quad (1)$$

N is the sample size and ES is the effect size. Since we already have used the t-test, we will compute the effect size based on t is the t-statistics and the degrees of freedom for the test - df. See [18]:

$$ES = \frac{2t}{\sqrt{df}} \quad (2)$$

In our case, we have $t = 2.42$ and $df = 16$. Thus $ES = 1.21$ and we have that $N > 21$. We have just 20 participants and we should thus be a little careful when concluding on this experiment.

What is more interesting, however, is the effect of the size of the groups that produced the final list of defects. We used groups of size two, three and five. We will focus on the groups with two and five persons in order to get as great a contrast as possible. The data are organized as shown in table 2 below. We have used the standard experimental design notation – see for instance [13].

Table 2. Two-factor experiment table

Group size A	Use of checklists B	A X B	Number of defects reported
-	-	+	7
-	+	-	9
+	-	-	13
+	+	+	11

We can now estimate the effects of each factor and interaction in the usual way – see for instance [13]. We find

- Group size effect = $[(11 + 13) - (9 + 7)] / 2 = 4$
- Checklist effect = $[(9 + 11) - (13 + 7)] / 2 = 0$
- Interaction effect = $[(7 + 11) - (13 + 9)] / 2 = -2$

In order to conclude, we need the standard deviation of the effects. This is usually estimated based on the assumption that the higher order interactions – order of three or more – equal zero. Since we have two factors and thus only one interaction, this approach can not be used here. Instead we have used the standard deviation value for

the pooled data of all individual inspections, which is 1.7. Two standard deviations – 3.4 – give us an approximate confidence level of 5% which means that we will reject both the checklist effect and the effect of the checklist – size interaction, and only accept the size effect as significant.

Since the checklist has a significant effect on single person inspections, it seems that the checklist is used as an ersatz for diverse experiences – albeit an inferior one.

4.3 NTNU Experiment 2

The second NTNU experiment [3] was performed to study the influence of experience and the effect on three common types of defects – wrong code, missing code and extra code. In this experiment, we used only individual inspections. In order to increase the available experience, we supplied a checklist. As observed in NTNU experiment 1, using a checklist will have a beneficial effect for small groups.

We had 21 persons with high experience and 21 persons with low experience. A simple graph shows the effect of experience on the probability of detecting the tree defect types, depending on the level of experience:

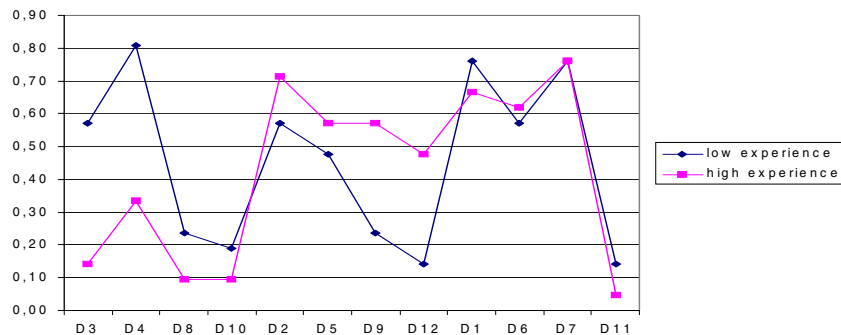


Fig.4. Defect types, experience level and the probability of defect detection

The four first defects are missing code; the next four defects are extra code while the four last defects are wrong code. See appendix A for a description of each defect. The first important question is – did those with more experience find more defects? The rather surprising observation is that the persons with low experience found more defects than did the personnel with high experience – 5.5 versus 5.1. The difference is, however, not significant. Thus, it seems like we cannot improve the code inspections by including more experienced personnel; at least not personnel with more general software engineering and development experience.

If we, instead of looking at the overall data, look at the data from each defect category, we find that personnel with

- Low experience is better at finding missing code. The difference is statistically significant – $p < 0.01$.
- High experience is better at finding extra code statement. The difference is statistically significant with $p = 0.01$

- Low – or high – experience are no better than the others in finding wrong code statements since $p = 0.38$.

The results are consistent with the results reported by V. Basili [9]. Now that we know this, how can we use it to improve code inspections? First we have to look beyond the rather broad word “experience”. The main difference in experience between the high level and the low level participants was general experience. The participants in the high experience groups were PhD students, while the participants in the low experience groups were third and fourth year students. A closer look at the entry questionnaire for the two groups shows that the low experience group had more recent hands-on experience in writing and debugging Java code than the high experience group.

The data indicates that the high experience subjects focused more on the overall functionality of the software and did not really read all the code. Thus, they missed simple but low level defects like the missing keyword “static” – D3 – which anyone that has ever coded Java *must* know is a defect. Trivial missing code defects were not even considered, while they found defects related to extra code.

What should worry us most are the two missing statement defects D8 and D10 and the wrong statement defect D11. These defects were found by few persons in both experience categories and are the defects most likely to be missed. The difference between the number participants finding these three defects and finding the rest of the defects is statistically significant with $p < 0.01$.

An explanation to this effect can be deduced from the way the checklist is organized. The checklist has 47 questions all in all, spread over 12 sections, covering two full pages. No question in the checklist is related to D11 and the questions related to defects D8 and D10 are in the last section of the checklist. The observed effect is most properly named the fatigue effect and seems to occur if we have long checklists or large amount of code.

5 Threats to Validity

Since all the three experiments reported here were student experiments run at a university, we will discuss the threats to validity for all three experiments together. Most of the following discussion is taken from [3].

5.1 Conclusion Validity

The conclusion validity is concerned with to what extent the conclusions are statistically valid. From equations (1, 2), we see that we need 22 participants or more given our chosen probabilities for type I and type II errors. Since we have only 20 participants, we feel that the conclusion validity in this case is medium. Threats related to our selection of subjects are taken care of since both groups – high and low experience – were homogenous with respect to education and relevant experiences.

5.2 Internal Validity

The internal validity is concerned with whether we are able to show a cause – effect relationship in our experiment. It is, in general impossible to prove a cause - effect relationship by means of statistical analysis alone. It is likewise impossible to show this by a discussion. What we can show is that we have taken reasonable precautions concerning some of the dangers that an experiment meets when trying to observe an effect of a treatment and have chosen to focus on selection, and instrumentations.

The groups in the NTNU 1 experiment were made homogeneous by using the participants' profiles. For the NTNU 2 experiment we had a set of subjects that were quite homogeneous from the start, except for the amount of experience. For the UNSW experiment we had no control over the experience and knowledge profile of the group other than that they all belonged to the same course. On the other hand – the fact that they had the same amount of software engineering education makes them a quite homogeneous group from the start. Thus, in all three cases we feel confident that the groups were reasonable homogeneous.

The instrumentation threat to validity arises out of differences in the way we perform pre- and post-measurements. Since this experiment is only concerned with counting defects found, this threat can only arise if we define or count defects in different ways. We had defined the defects before the experiment started and registered the seeded defects from the forms turned in by the subjects. The code base has been used for quite some time and no defects other than the seeded defects have been observed.

5.3 Construction Validity

Construction validity is concerned with the extent to which we measure the data relevant to our hypotheses. Our main concerns are whether the metrics we used captured the attributes of interest – the ability to identify defects in a piece of code. The metric used – number of faults found – will measure the participants' ability to find defects and is thus considered to be reliable. Threats like hypothesis guessing, researcher expectancy and evaluation apprehension are not relevant for our experiments.

5.4 External Validity

External validity is concerned with one of the most important validities – can the results be generalized? The largest threat to generalization is that all three experiments used students as subjects. Students with low experience may not be representative for professional software developers. The students with high experience, on the other hand, have quite a lot of industrial experience through the jobs they have had during their summer vacations or through part-time jobs.

All in all, the choice of persons to participate in the experiment is the largest threat to validity. This problem is discussed in several papers - see for instance [15], [16] and the interested reader should consult these papers. The main conclusion is worth repeating here: the main difference is not related to professionals versus students but to the amount of knowledge and experience. People with knowledge and experience

in a certain area will perform better than those without this knowledge and experience. Thus, a student with recent hands-on experience with Java programming will perform better at code inspection than a highly experienced project leader who has not coded for years.

The code used in the experiment may not be representative for real-world software code when it comes to size and complexity. The code used is 130 lines of Java. On the other hand, the time used was only one hour and there is thus a reasonable relation between time used and code size. Fagan suggested 125 LOC per hour [17] and this is the same as we have used. Even large pieces of code are broken down into smaller pieces for inspection and thus, our choice of code size and time used is well within the limits that one should expect to find in an industrial setting.

6 SPI Opportunities

This chapter sums up our most important advices for how to do inspections.

Hands-on experience is important. As we have seen, the class of defects detected depends on the reviewers' experience. There seems to be little need for general experience - what we need is theme-specific experience. Thus, when constructing an inspection team, we need to consider the relationship between the participants' experience and the type of defects we are trying to detect.

Large groups do better than small groups. This is not in agreement with what Votta found in [10]. His reason for stating that there is no difference between small and large inspection teams is the possibility of process loss – see chapter 4.1 above – and a low return from extra time spent in inspection meetings. We have shown that an inspection meeting can give process gain as well as process loss but the main reason why we get a different result from the one published by Votta et al. [10] is in our opinion that on the average, a large group contains a larger amount of experience and knowledge. The first NTNU experiment shows that large groups in general perform better than small groups, even if the small groups use a checklist. If we can put together a group that is diverse enough to cover all important topics, we will find more defects.

A review must be understood as a social process. Even though we can lay down rules for how to perform an inspection, the group will, in general, behave as they like. Thus, we need to build a spirit of trust and cooperation in the inspection teams. It is for instance important that all defects identified are reported. As the UNSW experiment shows, this might not always be the case.

The ordering of items in a checklist is important. There are some indications that many of the subjects used the checklist in a sequential way. Defects pertaining to topics mentioned towards the end of the checklist were found by significantly fewer subjects – for instance D8 and D10.

Long checklists should be split up into smaller checklists covering each area of concern – the fatigue effect. The new, shorter checklists should be distributed among the participants according to their experience and goal for the review. In this way, each reviewer can use both his experience and the checklist to focus on the topics where he should be able to identify the most defects.

7 Further Work

The most important questions raised by the experiments are why people ignore or are unable to detect specific classes of defects and why an inspection group seems to report defects after a voting process instead of reporting any defect found by at least one member. In order to understand these effects it is not enough to conduct the type of experiments described in the experiments reported above – we need to observe the groups in action and conduct post-experiment interviews.

Since specific experience and knowledge is an important factor for the result of an inspection, it might be possible to tune an inspection team and the checklists used to a defect profile – what kinds of defects are usually found in our code?

Lastly – we might improve our inspection process by using defects that are found later, for instance during testing and operation, and do a focused post mortem asking “Why didn’t we find this defect during inspection and how can we improve our inspection process so that we are more efficient the next time?”

We plan to perform experiments pertaining to these three problem areas during the autumn 2005 and spring 2006.

References

1. Stålhane, T. et al., Teaching the Process of Code Review, ASWEC 2004, Melbourne, Australia, April 13 – 16, 2004
2. Awan, T.H., Sources of variation in software inspections; An empirical and explorative study. TDT 4735, Software Engineering, NTNU, Norway, 2003.
3. Awan, T.H., Sources of variation in software inspections: An empirical research project. Master thesis, NTNU, Norway, 2004.
4. Meyers, G.J., Experiment in Program Testing and Code Walkthrough/Inspection, IBM Systems Research Institute, 1978
5. Basili, V.R and Selby, R.W., Comparing the Effectiveness of Software testing Strategies, IEEE Transactions on Software Engineering, SE-12 (7), Dec. 1987.
6. Porter, A.A. and Votta, L.G., An experiment to assess different defect detection methods for software requirements inspection. NASA research paper, 1994.
7. Laitenberg, O., et al., An Experimental Comparison of Reading Techniques for Defect detection in UML Design Documents. National research Council Canada, 1999.
8. Basili, V.R. et al., The Empirical Investigation of Perspective-Based Reading, 1996.
9. Basili, V.R et al., Investigating the effect of Process Experience on Inspection Effectiveness. University of Maryland, Institute for Advanced Computer Studies, 2001
10. Votta, L.G. et al., An Experiment to Assess the Cost – Benefits of Code Inspection in Large Scale Software Development. IEEE Transactions on Software Engineering, SE 23 (6), 1997.
11. Kelly, D. and Shepard, T., Task-Directed Software Inspection Technique: An Experiment and case Study, Royal Military College of Canada, 1997.
12. Auråen, E. Manufacturing World Commodities at the '94 Internet Conference – Dynamic Leadership through project management. Oslo, Norway, June 9 – 11.
13. Box, G.E.P. et al., Statistics for Experimenters. John Wiley and Sons, Inc., 1978.
14. Wohlin, C. et al., Experimentation in Software Engineering. An Introduction, Kluwer Academic Publishers, 2000.

15. Sølvsberg A. et al.: Evaluating the quality of information models. Empirical testing of a conceptual model quality framework, Proceedings of ICS, 2003, Portland, Oregon
16. Arisholm, E. and Sjøberg, D.I.K.: Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-oriented Software, IEEE Transaction on Software Engineering, vol. 30, no. 8, August 2004.
17. Fagan, M.E., Design and code inspection to reduce errors in program development. IBM Systems Journal, no. 15, 1976.
18. Rosenthal, R. & Rosnow, R. L. (1991). Essentials of behavioral research: Methods and data analysis (2nd ed.). New York: McGraw Hill.

19. Appendix A – Seeded Defects

- D1: wrong statement – wrong port number
- D2: extra statement – unused variable
- D3: missing statement – main method is missing the keyword “static”
- D4: missing statement – no closing of out-stream
- D5: extra statement – unused variable
- D6: wrong statement – sends wrong parameter value
- D7: wrong statement – wrong parameter value
- D8: missing statement – no exception handling
- D9: extra statement – unnecessary parameter
- D10: missing statement - no closing of output file
- D11: wrong statement – wrong parameter value
- D12: extra statement – code not needed