Evaluation of Code Inspection on an Outsourced Software Project in Mauritius

Geshwaree Huzooree
Department of IT
Charles Telfair Institute
Moka, Mauritius

Vimla Devi Ramdoo
Dept. of Computer Science and Engineering
University of Mauritius
Réduit, Mauritius

ABSTRACT
Software inspection is a proven methodology that enables the detection and removal of defects in software artifacts and thus contributes towards software quality assurance. It eventually leads to significant budget and time benefits. To be most effective, inspections must be an integral part of the software development life cycle and form part of the development schedule. This study considers code inspection as it is the most frequently used inspection in the software development process. For the implementation part of this research, code inspection is carried out in a selected outsourced project in a company situated in Mauritius, and the results are evaluated based on the inspection process and feedback of people involved in the inspection process.

General Terms
Software Code Inspection.

Keywords
Software Inspection, Defects, Software Verification and Validation, Software Quality Assurance

1. INTRODUCTION
Software plays a major role in modern organizations and many of the systems on which our lives and livelihoods depend are run by software. Unfortunately, the development of software is a major headache for organizations around the world, even Mauritius is not an exception. Often software is delivered late and does not meet user requirements as it should. Many software problems are due to the development of low quality software that is characterized by numerous defects. Software experts have suggested that the way to address these types of software problems is to improve software quality through quality assurance methods and one of the most well-known software quality techniques is indeed software inspection [1] [2] [3].

Though well-established for finding defects, software inspections are not universally used by software industries. This is due to several reasons such as the lack of training on how to carry out inspections, the need for project managers to move resources away from testing into inspections, and the large amount of paperwork that formal inspections require [4].

Along with audits, reviews, walkthroughs and configuration management, software inspection is equally important to the software verification and validation (SVV) process for finding defects as early as possible. It usually involves activities in which a team of qualified personnel determines whether the created artifact is of sufficient quality. Detected quality deficiencies are subsequently corrected, and in this way an inspection can not only contribute towards software quality improvement, but also lead to significant budget and time benefits.

In this paper, as a first step, an evaluation of the current use of software inspection around the world is introduced. Then the implementation of the code inspection process on an outsourced software project in an organization found in Mauritius is carried out, and an evaluation of the results is performed.

2. LITERATURE REVIEW
2.1 Software Inspection
Software inspection, which is a software quality technique, was developed by Michael Fagan in 1972 at IBM (Fagan, 1976). It is a group meeting which is conducted to uncover defects in a software work product (for example the software requirements specification, software design specification, code, and test plan). The approach is a formally defined process involving a series of well-defined inspection steps and roles, a checklist to aid error detection, and the formal collection of process and product data.

Industries conducting software inspection have found it to be effective way to uncover defects. The detection rate for an inspection varies depending on the type of work product being inspected and the specific inspection process used. Studies have found that 30 to 90 percent of the defects in a work product may be uncovered through the inspection process [5].

Early detection of defects can lead to cost savings. For example, one study has estimated that inspection-based techniques at Hewlett-Packard have yielded a cost saving of $21 million [6]. It should also be noted that inspections may take up a significant portion of a project's time and budget if performed on a consistent basis throughout the life of a project. According to an industry estimate from AT&T, project teams may devote four to fifteen percent of project time to the inspection process [7]. While allocating this amount of time on inspections may seem high, the benefit of reducing software defects has been found to outweigh the cost of conducting inspections.

The favorable attitude that many in the software development field have towards the inspection process is underscored by a statement made by Barry Boehm (a well-known expert in the field of systems development), who has written that "the [software inspection] has been the most cost effective technique to date for eliminating software errors." This statement was backed up (see Figure 1) of Fagan, 1986 [8] showing the difference it makes with and without inspections.

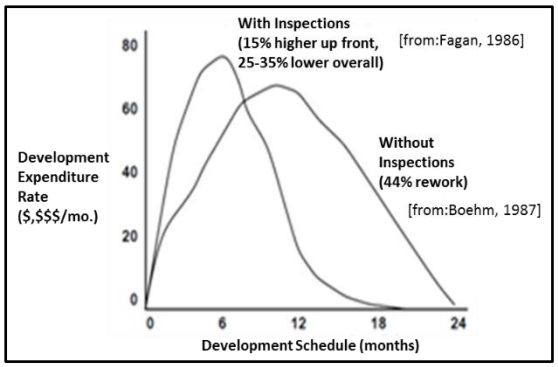

Fig 1: Development With and Without Inspections

2.2 Phases of Software Inspections

Software inspections are carried out in several defined steps. The inspection process that consists of five steps namely (see Figure 2):

1. Planning;
2. Overview meeting (optional);
3. Preparation;
4. Inspection / Examination meeting;
5. Rework / Follow-up.

For each step, the following information is included:

1. Objectives: the purpose of the step

2. Entry criteria: the conditions that must be met to begin the step

3. Activities: the activities that occur as part of the step

4. Exit criteria: the conditions that must be met to complete the step

5. Metrics: the product and process data that should be collected.

2.3 Benefits of Software Inspections

The inspection process was designed to help the developing organization to produce better products in terms of quality as defects were found early and fixed when they were less expensive. The effectiveness of the test activity is increased and less time may have to be devoted to testing the product. Another important benefit of inspections is the immediate evaluation and feedback to the author from his peers which will bring about improvements in the quality of future products.

2.4 Software Inspections and Rate of Defects

Research has shown that with inspections, defects can be managed and reduced. The curve A (A-Injected) represents the defects injected in the software. The Curve C (C-Detected with Inspections) represents defects remaining after removal by inspections for the volume of defects injected whereas the curve B (B-Detected without Inspections) represents the defects remaining without inspections [9] (see Figure 3).

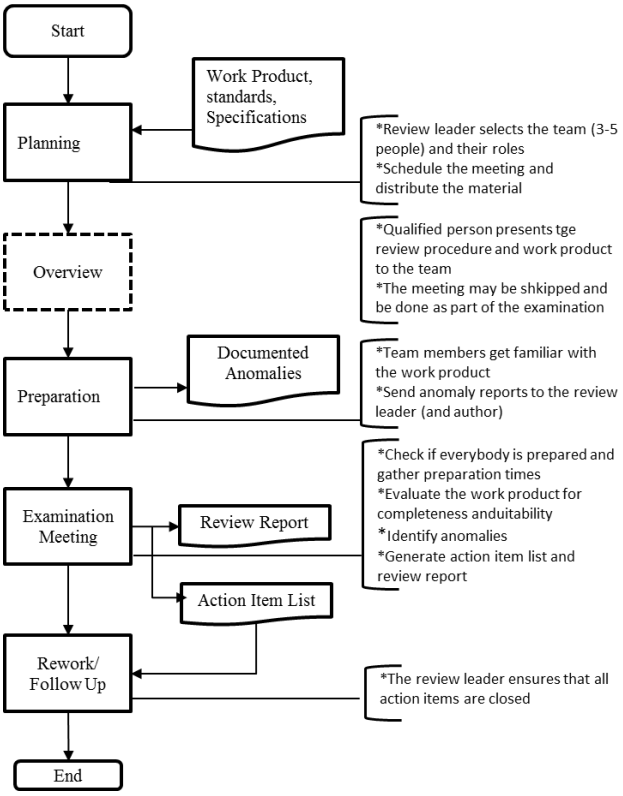

Fig 2: Inspection Process

2.5 Code Inspection

Code inspections are highly efficient test methods which cannot be substituted by any other test methods. It is time consuming but according to statistics it will find up to 90% of the contained errors, if done properly. However it all depends on the methods and checks applied and on the diligence of the inspectors. A proper code inspection may take several days and needs the help of tools. Proper inspections can be applied for almost all work products in the software life cycle. At the first glance they may look very time consuming. Nevertheless, statistical evaluations have shown that over the whole life cycle of the software development they even save resources and thus money and improve the quality of the product.

2.6 Code Inspection Metrics

It is difficult to monitor and analyze code inspection without measurements. To be able to plan, monitor and improve inspection (see Table 1), nine key metrics have been proposed by AT&T laboratories, namely Bell using the Goal-Question-Metric paradigm [10].

Table 1. Goals, questions and metrics

Goals, Questions and Metrics		
Goals	**Questions**	**Metrics**
Plan	How much does the Inspection process cost?	Average effort per KLOC
		Percentage of re-inspection
	How much calendar time does the inspection process take?	Average effort per KLOC
		Total KLOC inspected

Monitor and Control	What is the quality of the inspected software?	Average fault detected per KLOC
		Average inspection rate
		Average preparation rate
	To what degree did the staff conform to the procedures?	Average inspection rate
		Average preparation rate
		Average lines of code inspected
		Percentage of re-inspection
	What is the status of the inspection process?	Total KLOC inspected
Improve	How effective is the inspection process?	Defect removal efficiency
		Total KLOC inspected
		Average inspection rate
		Average preparation rate
		Average lines of code inspected
	What is the productivity of the inspection process?	Average effort per fault detected
		Average inspection rate
		Average preparation rate
		Average lines of code inspected

The nine key metrics are as follows:

1. Total non-comment lines of source code inspected, in thousands (KLOC) for simplicity.

$$\text{Total KLOC inspected} = \frac{\sum_{i=1}^{N} LOC\ Inspected_i}{1000}$$

2. Average lines of code (LOC) inspected.

$$\text{Average LOC inspected} = \frac{Total\ KLOC\ inspected * 1000}{N}$$

3. Average preparation rate: the average rate, expressed in lines of code per hour, at which an inspector studies the inspection material.

$$\text{Average preparation time} = \frac{Total\ KLOC\ inspected * 1000}{\frac{\sum_{i=1}^{N} Preparation\ \ time_i}{Number\ \ of\ Inspectors_i}}$$

4. Average inspection rate: the average lines of code inspected per meeting hour.

$$\text{Average inspection rate} = \frac{Total\ KLOC\ inspected * 1000}{\sum_{i=1}^{N} Inspection\ \ duration_i}$$

5. Average effort per KLOC: the average hours spent in an inspection activities by an inspection team for one thousands lines of codes of non-commented source code.

$$\text{Average effort per KLOC} = \frac{\sum_{i=1}^{N} Inspection\ \ effort_i}{Total\ KLOC\ inspected}$$

6. Average effort per fault detected.

$$\text{Average effort fault detected} = \frac{\sum_{i=1}^{N} Inspection\ \ effort_i}{Total\ faults\ \ detected_i}$$

7. Average faults detected per KLOC.

$$\text{Avg faults detected per KLOC} = \frac{\sum_{i=1}^{N} Total\ faults\ \ detected_i}{Total\ KLOC\ inspected}$$

8. Percentage of re-inspections.

$$\text{\% of re- inspections} = \frac{No\ of\ reinspections\ \ dispositions * 1000}{Number\ \ of\ inspections}$$

Note: Number of re-inspections dispositions = Number of re-inspections disposition of type Re-Inspect + Number of re-inspections disposition of type Rework

9. Defect-removal efficiency.

$$\text{Defect removal efficiency} = \frac{\sum_{i=1}^{N} Total\ faults\ \ detected_i}{Total\ coding\ \ faults\ \ detected}$$

Note: Total Faults Detected = Total number of faults detected at the inspection

Total Coding Faults Detected = Total no of faults detected at the inspection + Faults identified in inspected Code (during testing) + Faults detected by customers.

The importance of inspection in the software development life cycle has been seen, and the main metrics used in code inspection have been enquired. The research is continued by finding out the potential causes of software defects in outsourced software projects of a company situated in Mauritius.

3. COMPANY PROFILE

The company chosen for the purpose of this research is an international Consulting and Systems Integration (CSI) company having a branch in Mauritius where CRM projects are outsourced. Currently, in the software development phase of the CRM projects, there is no code inspection as such that are carried out in the organization in Mauritius, but there are time-to-time code reviews (informal) that are carried out by the team leads. When non-compliant codes are found, the team leads communicate informally to the developers concerned so that they can modify the code, thus making it compliant to the standards set for the particular project. PHP is the main development language used in the organization.

4. METHODOLOGY

4.1 Causes of software defects

The potential causes of software defects are analyzed and shown on the Ishikawa or Fish bone diagram (see Figure 5). For this study, the "Coding Errors" cause are explored via a code inspection process. A code inspection checklist was used to perform the code inspection.

4.2 Implementation of Code Inspection

The following sample criteria were used to choose the module of code to inspect (both important to the project and the organization.

1. Criticality: functions that were critical to the operation of the software developed.
2. Complexity: modules that were more complex that other modules.

3. Past history: modules that have shown a high number of bugs in the past.
4. Experience level of software engineer: code written by inexperienced software engineer.

The measures were collected as follows:

Inspection Rate (LOC/Hour) = Reviewed LOC/Review

Time

Preparation Rate per Reviewer (LOC/Hour per Reviewer) = Reviewed LOC/ (Total Preparation Time/Number of Reviewers)

Number of Reviewers (Reviewers) = All Participants at the Review, excluding the Presenter.

The code inspection was carried out on selected samples of code of 150 and 300 LOC respectively, and the documents used are the Inspection Problem Report Form and the Inspection Summary Process Report. After the inspection process, a questionnaire was set up to gather feedback from the people involved in the process and the results were then evaluated.

4.3 Research Limitations and Assumptions

During the implementation of inspection, the sources of limitations identified (hence assumptions taken) were as follows:

1. Incorrect and/or bias information.
2. Time constraint.
3. Sampling errors.

5. RESULTS AND DISCUSSIONS

In order to evaluate the effectiveness of the inspection program, the data collected from inspections are analyzed in order to reveal trends. The data are the amount of product inspected at a meeting, the time taken in preparation and in the inspection meeting, total defects found per inspection and the types of defects found in the development phase of the software life cycle. The data for trending is normally collected by the moderator, using forms provided for this purpose.

5.1 Result: Perceived Effectiveness of Code Inspections

The result of the perceived effectiveness of code inspection from 5 team members sampled to answer the questionnaire. The mean value of the answers was computed to generate the graph (see Figure 6)

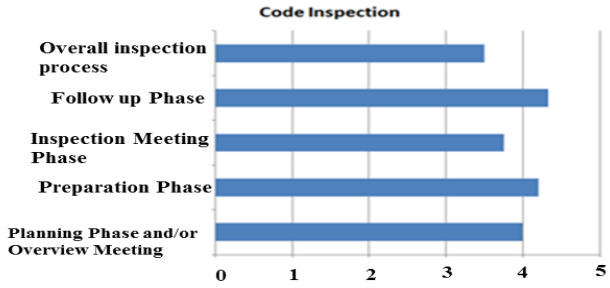

Fig 6: Perceived effectiveness of code inspection

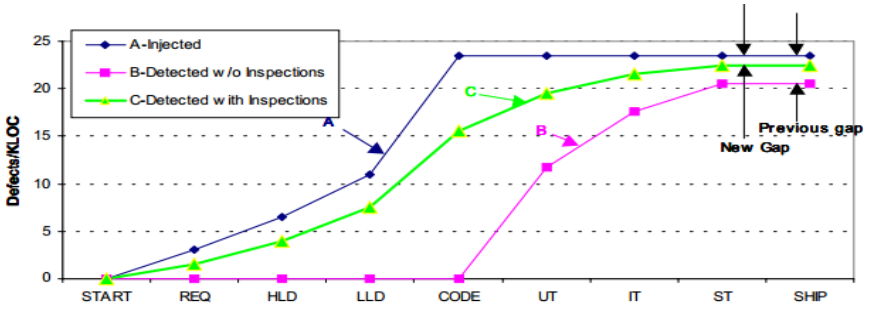

Fig 3: Rate of defects

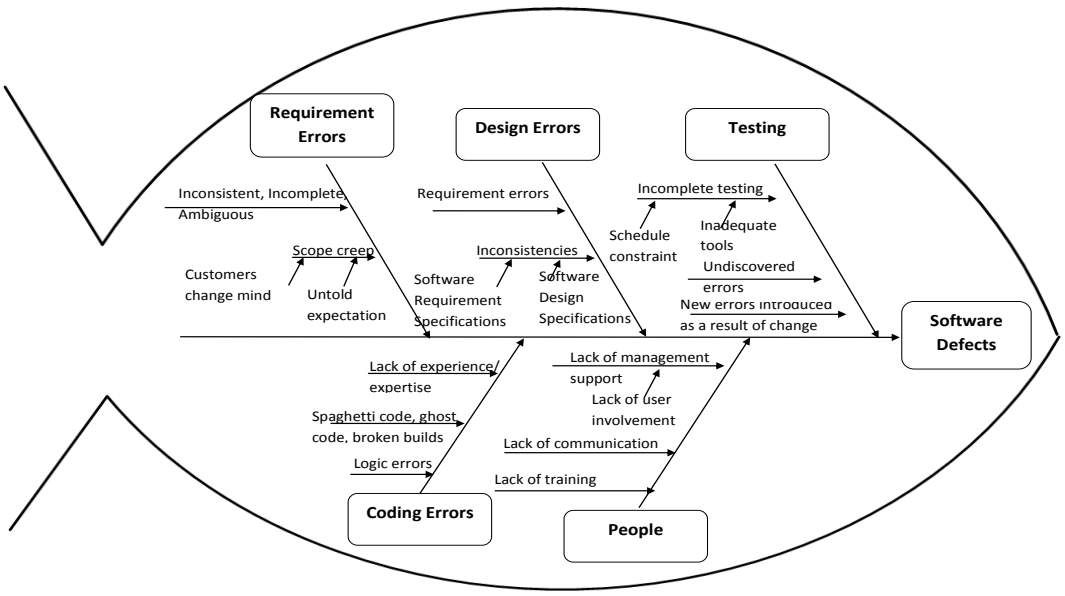

Fig 5: Ishikawa Diagram showing the Potential causes of software defects

5.1.1 Evaluation of Category 1: Planning Phase and/or Overview Meeting (optional)

It can be seen that the planning phase is on an average of 4 on the rating scale of 1-5 used, where 1 stands for "strongly disagree" and 5 stands for "strongly agree"(see Figure 6). It means that the team members were nearly satisfied with the planning phase, for example they agreed that the moderator and the inspection team were selected based on their expertise. They also agreed on the commitment from the team and the responsibilities fulfilled by the moderator as adequate.

Nevertheless, the members did not quite agree from the fact the inspectors from other range of disciplines were considered, as according to them, was not the case.

5.1.2 Evaluation of Category 2: Preparation Phase

The team agreed with the familiarity that they were supposed to have for the inspection, but they did not however keep track of the preparation time. The main reason was that they were preoccupied with other work, so they could not dedicate a specific amount of time for the preparation, but rather find some time whenever possible for the preparation.

5.1.3 Evaluation of Category 3: Inspection Meeting Phase

Based on the team feedback, the overall inspection meeting was adequately done as expected. Some team members did not agree from the fact that the follow-up after the inspection meeting was evaluated as it should. On the other hand, all the team members agreed that the moderator did not distribute any meeting minutes, but rather said it informally.

5.1.4 Evaluation of Category 4: Follow-up Phase

Almost all the team members were satisfied of the follow-up phase that they found small but adequate. They appreciated the fact that the moderator communicated the completion of the inspection to the team via a formal motivational email.

5.1.5 Evaluation of Category 5: Overall inspection Process

On average, the overall inspection process was judged as 3.5 on a scale of 1 (strongly disagree) to 5 (strongly agree). The objectives set were met and the team agreed that the inspection process helped them promote quality deliverables as they found defects in their code and thus learned from their previous mistakes. Some team members agreed that there were still ways for improvement in the inspection process for example on the way conflicts are resolved. Some team members on the other hand did not appreciate their deliverables being inspected by peers.

5.2 Result: Implementation of Code Inspection

According to the metrics identified in Code Inspection Metrics of the literature review (see Table 2), the code inspection was implemented on two set of an outsourced project modules and its effectiveness evaluated.

Table 2. Metrics used

Metrics	Results	
	Module 1	**Module 2**
Total KLOC inspected	0.3	.015
Average LOC inspected	300	150
Average Preparation Rate	150	150
Average Inspection Rate	150	150
Average Effort per KLOC	73	73
Average Effort per Fault Detected	4.4	2.8
Average faults detected per KLOC	17	27
Percentage of re-inspections	0	0
Defect-removal efficiency	71%	100%

Inspection Scenario:

Two modules of PHP code were inspected, and the assumptions are as follows:

- Preparation time ≈ 50 lines of source code/hour
- Inspection rate of about 100-200 lines of source code/hour
- Rework time = 0
- Assumption: No disposition of type re-inspect/rework

The data gathered for the two modules of code are as follows (see Table 3):

Table 3. Data gathered from PHP modules

Data Gathered	Module 1	Module 2
No of Inspection, N	1	1
No of Inspectors	3	3
No of participants	8	8
LOC	300	150
Preparation Time	6hrs	3hrs
Inspection Duration	2hrs	1hr
Total Fault Detected	5	4
Total Coding Fault Detected	7	4

The results from the code inspection implementation highlight the fact after introducing a code inspection process within the organization, it is equally important to assess its viability and effectiveness with the organization. Despite the inspectors have spent twice the time to prepare and inspect 300LOC compared to the sample code of 150LOC (see Figure 7); the defect removal efficiency is still 71% (see Figure 8). The results shows that code inspections with fewer LOC are more effective in defect removal and less time consuming in terms

of preparation time and inspection duration compared to code inspections with greater LOC.

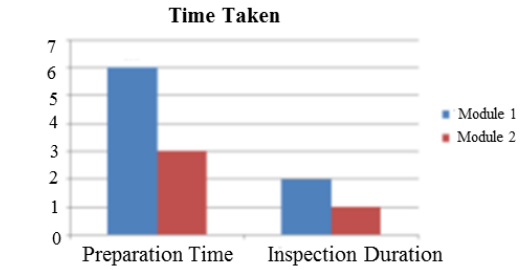

Fig 7: Time taken to prepare and inspect modules

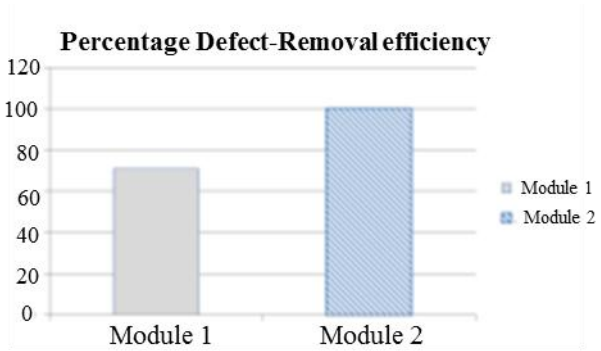

Fig 8: Percentage Defect-removal efficiency

6. CONCLUSION

Formal inspections have been demonstrated by many organizations to be an effective method for finding and removing defects in software products. However, just putting a formal inspections program in place does not guarantee that the program will operate at maximum efficiency. It is important to evaluate the implementation of the formal inspections process and to improve it by fine tuning the procedures that are followed.

The study showed that developers who participate in the inspection of their own product actually create fewer defects in subsequent work. Because inspections formalize the development process, productivity-enhancing and quality-enhancing tools such as automated tools can be adopted more easily and rapidly. So to further improve quality in the software product, automated tool can be used wherever possible for inspections such as automated code inspection. Moreover, some custom-built inspection processes can be used as well based on lessons learned after carrying out inspections on projects of the particular organization.

Results have shown that after the inspection, everyone has a better understanding of the work done. For the outsourced project's modules inspected, we found that inspection also encourages collaboration between the project team members thus increasing communication.

Thus, organizations should be aware that introducing the inspection process to improve effectiveness generally lowers productivity, but the cost of this decrease is negligible when compared with the cost of removing defects later in development or testing phases. However, some improvements in effectiveness also increase productivity. Code inspections can further be used to improve quality of the software and also reinforce coding standards, besides reviewing of codes.

This study can be extended whereby code inspections can be further implemented and evaluated by varying the preparation time, the inspection duration and number of inspectors with respect to the LOC to estimate the efficiency of code inspection.

7. REFERENCES

[1] Kamei, Yasutaka, et al. 2013. A large-scale empirical study of just-in-time quality assurance. *Software Engineering, IEEE Transactions on* 39.6: 757-773.

[2] Fagan, Michael E. 2001. Advances in software inspections. *Pioneers and Their Contributions to Software Engineering*. Springer Berlin Heidelberg: 335-360.

[3] McIntosh, Shane, et al. 2014. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM.

[4] Remillard, Jason. 2005. Source code review systems. Software, IEEE 22.1: 74-77.

[5] Brykczynski, Bill, Reginald Meeson, and David A. 1994. Wheeler. Software Inspection: Eliminating Software Defects. IDA, Inst. for Defense Analyse.

[6] Johnson, Philip M. 1998. Reengineering inspection. Communications of theACM 41.2: 49-52.

[7] Wiegers, Karl E. 2002. Seven truths about peer reviews. Cutter IT Journal15.7: 31-37.

[8] Fagan, Michael E. 1976. Design and code inspections to reduce errors in program development. IBM Systems Journal 15.3: 182-211.

[9] Software Inspections, Ron Radice, article published in Methods & ToolsSummer. 2002. http://www.methodsandtools.com/PDF/dmt0202.pdf [Accessed: 21 October 2012].

[10] Barnard, J. and Price, A. 1994. Managing Code Inspection Information. IEEE Software, 59-69.