CODE INSPECTION: A REVIEW

Robson Ytallo Silva de Oliveira, Paula Gonçalves Ferreira
CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife - PE, Brazil

Alexandre Alvaro, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira
C.E.S.A.R – Recife Center for Advanced Studies and Systems, Recife – PE, Brazil

Keywords: Code Inspection, Quality Assurance, Software Reuse, Maintainability.

Abstract: The software inspection process is generally considered a software engineering best practice. For a long time, it had the goals of finding and fixing defects as soon as possible. For this reason, techniques are suggested for use in a software reuse process in order to improve the quality of the assets developed and reused. Thus, the code become itself easier to understand and changeable, and improving its maintainability, minimizing redundancies and improving language proficiency, safety and portability. In this way, looking for analyzing this area, this paper presents a survey of code inspection research.

1 INTRODUCTION

Code inspections have emerged as one of most effective quality assurance techniques in software engineering. The primary goal of an inspection is to find defects before the testing phase; and hence strongly contribute to improve the quality of software with budget and time benefits (DeMarco, 1982). A real example of code inspection benefits could be seen at Jet Propulsion Laboratory that gives support to NASA (Wohlin et al., 2002). They saved US$ 7,5 million performing around 300 inspections.

The main goal of this survey is study the state-of-the-art in code inspection research, in an attempt to analyze this possible trend and to provide insights in proposal of a well-defined software components code inspection process. Thus, the main techniques, process, methods and results were identified and analyzed.

Besides this introduction, the reminder of this paper is organized as follows. The Section 2 presents the most relevant techniques for the code inspection activity that will be considered in the works found in the literature. Section 3 splits the code inspection area in two ages and surveys the state-of-the-art related to code inspection research. Finally, Section 4 presents the concluding remarks and directions for future work.

2 CODE INSPECTION TECHNIQUES

There are a lot of techniques that very important to guide the code inspection process. They have been studied and used during last years. The most relevant code inspection technique was presented by Michael Fagan in 1976 (Fagan, 1976). The technique was called FTR (Formal Technical Review) and consists of five steps, as follows:

- **Overview:** The author presents the scope and the proposal of his software product;
- **Preparation:** In this step, the reviewers only understand the code;
- **Inspection Meeting:** The reviewers work together in order to identify errors and reporting them;
- **Rework:** The author repairs the issues reported in the last step;
- **Follow-up:** A moderator analyses the rework step and judge if it is necessary to repeat the process.

Usually a practical model, based on FTR, is used when the cost of the code inspection process is bigger than its benefits, which is composed of three steps, as follows:

Ytallo Silva de Oliveira R., Gonçalves Ferreira P., Alvaro A., Santana de Almeida E. and Romero de Lemos Meira S. (2007).
CODE INSPECTION: A REVIEW.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 537-540
DOI: 10.5220/0002366505370540

- **Preparation:** The reviewers understand the code. In addition, they need to find defects in this step in contrast to FTR;
- **Collection:** The errors discovered are analyzed and issues are reported when a defect is considered a real one;
- **Repair:** In this phase, the reported issues are addressed to the author repair them.

Some practical model variants try contributing through their characteristics in the Cost-Benefit relation. The most known of them are:

- **Active Design Review (ADR):** It is a technique based on multiple sessions, where each session is divided in short phases which are independent;
- **Phased Inspection (PI):** As well as ADR, PI is based on multiple sessions, but its phases are sequential. The reparation is done after each phase;
- **N-Fold Inspections:** This uses the concept of N-teams which are designated to perform the same task of inspection. In the end of process, a moderator is responsible to remove the redundancies. In general, N-Fold Inspections have a high price due to people are addressed to develop the same task.

3 CODE INSPECTION: A SURVEY

Analyzing the techniques presented and how they have been used during the life of software development, it is possible to divide the code inspection process in two ages: ***Before 1976*** and ***From 1976 until today***. However, this survey will consider just the second age due to this area is scarce of works and results before Fagan's model.

First, in 1995, Haton (Hatton, 1995) suggests the static inspection idea, which is purely the concept of the code inspection. If suppose a dynamic inspection, the concepts of test and inspection can be mixed. Sometimes, the defect is evident into the software, but the reviewers does not know where they are, being necessary to run it. To understand the idea, it is used an analogy: an engineer walks along a stationary train, punching its wheels. If he has experience, he is able to identify if there is any crack in the train, if it is prejudicial and where. In the code inspection, a static process identifies language inconsistencies, furthermore becoming the code easier to be understood and documented, contributing, consequently, with software reuse. The great disadvantage, in accordance with this work, it is the necessity to have an experienced professional in the language and the system.

Porter et al., in 1997 (Porter et al., 1997), presented a report related to cost-benefits of the code inspection techniques. The work had the follow results:

- No difference in time and effectiveness between groups with two or four people;
- Two groups with two people are not more efficient than one group with two people;
- "Multiple sessions" do not have more effectiveness than "single sessions".

Still under 1997, (Porter and Johnson, 1997), was motivated for a work from 1985 (Eick et al., 1992). The oldest work discussed about the relation between the steps of Preparation and Inspection Meeting from FTR model. According to (Eick et al., 1992), the most of defects were found in the second step due to using the first one to find errors too, opposed to the original model. The new work analysed the relation between Real and Nominal Group into a code inspection process.

Before the conclusions, it will be defined the both kinds of groups (Porter and Johnson, 1997):

- Real: "participants meet face-to-face and interact with each other to accomplish the group task".
- Nominal: "participants work individually without interacting with each other, and their individual results are pooled together to accomplish the group task".

Two experiments was performed and the more significant result did not find any difference in effectiveness between real and nominal groups, nevertheless preparing and joining people – real group – have a high price.

In 1987, (Basili and Selby, 1987) compared and combined code inspection techniques. After ten years, (Wood et al., 1997) performed a similar work, obtaining the same results: the individual techniques are very important, but the best results are generated through techniques combination.

Several works about code inspection are related or use the FTR model. Few significant modifications are determined. But in 1998, (Johnson, 1998) wrote some recommendations for Fagan's model. Although the recommendations weren't being explicitly referenced into the future works, it is easy to identify some of them adopted, such as: provide tighter integration between FTR and development model; Minimize meetings and maximize asynchronicity; Shift the focus from defect removal to improved developer quality; Build organizational knowledge bases on review; Outsource review and insource review knowledge; Investigate computer-mediated review technology; and Break the boundaries on review group size.

In (Brykczynski, 1999), Brykczynski listed a range of checklists, suggesting what to be used or avoided. By suggestions, it is possible to identify the Johnson's recommendation (Johnson, 1998) to build organizational knowledge bases on review. Although (Johnson, 1998) is not referenced, Brykczynski suggests updating the checklist to maintain an updated knowledge base.

In 1999, the main concern is about Object-Oriented (OO) Designs. Usually, when an OO code is inspecting, only a piece of it is analyzed. It can be very difficult to understand only that piece if we consider some concepts like polymorphism and inheritance. Travassos et al. (Travassos et al., 1999) cited the importance of reading techniques in OO context. The great problem for OO code inspection is the preparation phase. Sometimes it is not well-defined or sometimes it has a high price to be performed.

In 2000, Kimble and White (Kimble and White, 2000) describes an alternative source code analysis in order to identify automatically the existence of recursion, missing functionality and logical errors. The method used a parser to massage the code to a set of conventions. A modified notation was created and from this a Control Flow Diagram (CFD) was built and the thread analysis was done easier due to well-defined flows.

Still in 2000, Adams (Adams, 2000) presented an interesting work, that could be considered very radical, but his words make sense. According to Adams, six years before Michael Fagan publish his work describing the FTR model, Apollo 11 was launched to space. Considering the Flight Control Software that helped guide Eagle, Adams concludes that many complex software systems was developed and some kind of inspection process was adopted before 1976. Thus, he creates some questions about if inspections and reviews are really necessary.

Another work developed in 2000, Nandivada and Dutta (Nandivada and Dutta, 2000) describes a new model for code reviews. It is called "The 9 Quadrant model" and it is based on statistical techniques such as control charts. The model provides a single framework to generate a scatter chart between yield and cost of code review. 9Q model serves as an excellent tool for process decision-making and indirectly addresses planning of code reviews, determining their efficiency and improving the process.

In 2001, the Bifel et al. (Bifel et al., 2001) work was concerned in evaluating re-inspection. A re-inspection repeats the inspection process and is often believed to be less efficient. Thus, a cost-benefit model was proposed in this work to help whether a re-inspection justifies its cost. The work concluded that more than 80% of re-inspections weren't necessary and the remaining were necessary due to the reviewers weren't familiar with the system or with the process.

In 2002, according to Wohlin et al. (Wohlin et al., 2002), the advantages of inspections were not well perceived by management. Knowing that "benchmarking is a continuous improvement process rather than a competitive comparison" (Wohlin et al., 2002), and "It is a widely used business practice and has been accepted as a key component for organizations to search for improvement in quality, competitive position or market share" (Wohlin et al., 2002), benchmarking could compare tools, people, environments, and so on to build a well-defined code inspection. Several tools for software benchmarking have also been developed. Although makes no reference to (Johnson, 1998), it follows the recommendation to build organizational knowledge bases on review.

Again, the relation between Real and Nominal Groups was discussed by Tyran and George (Tyran and George, 2002) in 2002. This time was suggested to use a Group Support System (GSS), attending the recommendation of Johnson (1998) to investigate computer-mediated review technology. The GSS's are responsible to address group process issues in collaborative groups, helping to minimize problems associated with the meeting process. In addition, according to Tyran and George (2002), to use a GSS-supported, teams perform best the code inspection, detecting more defects.

The Usage-Based Reading (UBR) techniques was discussed by Thelin et al. (Thelin et al., 2003), in 2003. According to work, the UBR which uses the traditional inspection concepts, use cases and operational profile testing, is more effective and efficient than the checklist-based method.

In 2005, Remillard (Remillard, 2005) presented his experiences using Source Code Review Systems. He describes the tools Bugzilla and CodeStriker, showing their main features. According to work, the CodeStriker looks like having more advantages considering the other compared systems. This assumption is giving emphasis by license type, revision control system integration, data storage and kind of inspections supported.

4 CONCLUSION AND FUTURE REMARKS

This paper has presented a survey related to the state-of-the-art in the code inspection research. As we can observe, the FTR Model was predominant in the works analyzed. Still on, a set of works found into literature presented some variants but all of them follow the Fagan's principles.

For future work, we planned to establish a well-defined process for software component code inspection in conjunction with RiSE[1] projects in order to evaluate this one process. One of the main motivations for us is due to the fact that we did not find into the literature a code inspection process specific for software components. However, this is one of the three bases adopted in our group for quality assurance in software components, composed of a component certification process (Alvaro et al., 200) (Alvaro et al., 2006) and test's component. The code inspection requirements and the process will be described in future papers.

REFERENCES

Adams, T. (2000). The God of Inspection. *ACM SIGSOFT - Software Engineering Notes*, 25, 02, 30.

Alvaro, A., Almeida, E. S., Meira, S. L. (2005). Software Component Certification: A Survey. In *31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track*, Porto, Portugal. *IEEE Press*.

Alvaro, A., Almeida, E. S., Meira, S. L. (2006). Towards a Software Component Certification framework. In *5th International Conference on COTS-Based Software Systems (ICCBSS)*, Poster Session, Florida, EUA. *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag.

Basili, V., Selby, R. (1987). Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering*, 1278-1296.

Bifel, S., Freimut, B., Laitenberger, O. (2001). Investigating the Cost-Effectiveness of Reinspections in Software Development. In *IEEE International Conference on Software Engineering (ICSE)*, 155-164.

Brykczynski, B. (1999). A Survey of Software Inspection Checklists. *ACM SIGSOFT - Software Engineering Notes*, 24, 01, 82-89.

DeMarco, T. (1982). Controlling Software Projects. *Yourdon Press*. New York.

Eick, S.G., Loader, C.R., Long, M.D., Vander Wiel, S.A., Votta, L.G. (1992). Estimating Software Fault Content Before Coding, In *14th Internacional Conference on Software Engineering (ICSE)*, 59-65.

Fagan, M. (1976). Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15, 3, 182-211.

Hatton, L. (1995). Static Inspection: Tapping the wells of software. *Programming Research Ltd*. 85-87.

Johnson, P. (1998). Reengineering Inspection. *Communications of the ACM*, 41, 02, 49-52.

Kimble, J., White, L. (2000). An Alternative Source Code Analysis. In *The International Conference on Software Maintenance (ICSM)*, 64-75.

Nandivada, R., Dutta, S. (2000). The 9 Quadrant Model for Code Reviews. In *IEEE Asia Pacific Conference on Quality Software*, 188-193.

Porter, A., Johnson, P. (1997). Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies. *IEEE Transactions on Software Engineering*, 23, 03, 129-145.

Porter, A., Siy, H., Toman, C., Votta, L. (1997). An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development. *IEEE Transactions on Software Engineering*, 23, 06, 329-346.

Remillard, J. (2005). Source Code Review Systems. *IEEE Software*, 22, 01, 74-77.

Thelin, T., Runeson,, P., Wohlin, C. (2003). Prioritized Use Cases as a Vehicle for Software Inspections. *IEEE Software*, 20, 04, 30-33.

Travassos, G., Shull, F., Fredericks, M., Basili, V. (1999). Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. In *14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 34, 10.

Tyran, C., George, J. (2002). Improving Software Inspections with Group Process Support. *Communications of the ACM*, 45, 09, 87-92.

Wohlin, C., Aurum, A., Petersson, H., Shull, F., Ciolkowski, M. (2002). Software Inspection Benchmarking - A Qualitative and Quantitative Comparative Opportunity. In *8th IEEE Symposium on Software Metrics,* 118-127.

Wood, M., Roper, M., Brooks, A., Miller, J. (1997). Comparing and Combining Software Defect Detection Techniques: A Replicated Empirical Study. In *6th European Software Engineering Conference / 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1301, 162-277.

[1] Reuse in Software Engineering (RiSE) group – http://www.rise.com.br