A Model and Tool Features for Collaborative Artifact Inspection and Review

ABDUSALAM NWESRI, KHAIRUDDIN HASHIM
Software Engineering Department, Tenaga Nasional University
MALAYSIA
khairuddin@uniten.edu.my

Abstract: - Inspection offers an opportunity to detect and remove defects at various points during software development. Early detection will reduce the effect of propagation and amplification of defects into the later phases of software development. Collaborative inspection on the web aims to eliminate the time factor needed to assemble the inspection or review team at a physical location. Through the collaborative mode, software teams can perform software inspection and review from geographically separated places asynchronously. These newly introduced practices have proven that collaborative inspection and review of artifacts on the web is feasible. This paper provides a model for collaborative inspection and review including possible features of model and tool that will support collaborative inspection and review on the web.

Key-Words: artifact, collaborative, software inspection, review

1. Introduction

Artifacts produced in software development activities are required to be inspected and reviewed. Otherwise, defects get amplified and propagated to the following phases. To minimize this, review and inspection are required.

With size of projects increase in magnitude and cost, effective methods must be introduced to ensure defects are detected early in the development phase. Quality of final products usually depends on the quality of procedures employed. Comprehensive documented procedures ensure that step by step requirements are adhered to. Considerations on process requirements ensure that a process is executed accurately with required focus and considerations.

Nowadays, it is quite common for team members of a software development project to be separated geographically. As such, methods and tools have to be developed to support this new requirement [1]. A collaborative approach should facilitate asynchronous activities due to time zone difference. Software inspections have been proven to improve software quality and reduce software development costs.

The process as introduced by Fagan in 1967 [2], involves a group of competent people working together to check required changes on any milestone deliverable in software development. Inspections are among the most mature and perhaps best-studied practices in software engineering [3]. Inspection is applied in various domains including the software security domain as reported in [4]. Although in general software inspections have been accepted in the software industry as a cost-effective approach, many remain reluctant to implement inspection [5].

Inspection methods can be more effective than informal reviews and require less effort than formal proof, but success depends on having a sound and systematic procedure for conducting inspection [6]. If this is effective, maintenance cost may also be reduced. There were many requirements suggested by previous tools' developers. Some of these requirements vary from one tool to another. In this paper we first present the traditional

inspection process, discuss several research approaches to inspection and then define some requirements towards effective collaborative inspection on the web.

2. Traditional Inspection Process

The original process as proposed by Fagan [2] goes through five phases: *overview*, *preparation*, *inspection*, *rework* and *follow-up*. In the first phase, *overview,* the author presents his product to the whole team. Then, the document and any related work such as the source document and checklists are distributed to the team members. Individually, each team member investigates the document in order to understand it but not to detect defects.

In the following phase, Inspection Meeting, the document is paraphrased by the reader. During this process, the inspectors can raise issues regarding the document. If a consensus that an issue is a defect is reached, the issue is then classified as missing, wrong, or extra and its severity is also classified as either major or minor. The defect is then recorded by the recorder. The meeting moves on until the inspection team finishes inspecting the document or within a time limit not more than two hours.

The moderator will hand over the defect list to the author who makes the necessary corrections. This phase is usually called *rework*. In the following phase, follow-up, the moderator ensures that all required changes have been made. The moderator thereafter decides upon the state of the inspection and whether or not a re-inspection is required. It is at this point that the inspection process is considered complete.

In the updated version [7] of the inspection process, the planning phase was added. In the planning phase, details of the product to be inspected, the inspection team, time schedule and defect detection approaches are identified. The process is depicted in Fig. 1.

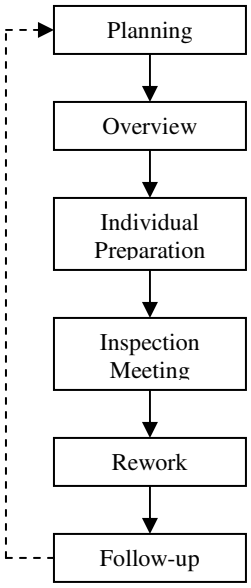

Fig. 1: Fagan's Inspection Process

3. The Research Landscape in Software Inspection

A study [8] proposed an inspection quality enhancement method. It suggested rules for better implementation of inspection procedures. The rule set has 8 categories and 30 indexes grouped by similarity of rule characteristic. Table 1 shows the review items for code inspection.

Category	**No**	**Index Item**
Readability & Maintenance	1	Macro Naming
	2	Function Naming
	3	Enum Constants Naming
	4	Global Variable Naming
	5	Local Variable Naming

	6	File Naming
	7	#define or #undef within a block
	8	File Comments
Dead Code	9	Failure Definition Global Variables
	10.	Non-null Statements
Potential Error	11	Default in Switch
	12	Floating Point Comparison
	13	Uninitialized Pointer
	14	Variable Initialization
	15	Null Pointer Assignment
	16	Assignment in Boolean Expression
	17	Braces of Loop Body
	18	Three Expressions of a For Statement
Control Error	19	Unreachable Code
	20	Goto Statement
	21	Empty Block Body
	22	Loop Counter Type
Performance	23	Debug Statement
Storage Management	24	Dynamic Heap Memory
Interface	25	Number of Arguments and Parameters
	26	External Definition Object
	27	External Definition Function
	28	Internal Linkage of Object
	29	Internal Linkage of Function
Security	30	Observe Prohibition Function

Table 1: Review Items for Code Inspection

The question of whether or not inspection meeting offsets its cost has been controversial. Many articles, questioning the meeting value, were published [9], [10]. Some researchers have anticipated that asynchrony will replace the inspection meeting in the future [11]. Some others reached the conclusion that the inspection meeting is of doubtful value. Many experiments were conducted to check that doubtfulness.

Votta at AT&T Bell Labs [9] observed a series of inspection meetings involving software professionals working on industrial projects. His analysis from the data gathered suggested that the number of defects found in meetings is only 4% greater than the number discovered during individual preparation. He also conducted subsequent cost-benefit study to compare meeting based inspections with a process based around individual defect depositions.

He reported that the potential benefit of finding more defects in a meeting was not adequately offset by the higher cost incurred in organizing the meeting. Votta also noticed that only two of the inspectors can interact in the meeting at any one time. Straightforwardly, he concluded that around 30%-80% of other inspectors' time was spent listening to the conversation.

McCarthy et al. [12] conducted a series of experiments to investigate the notion that the

inspection meeting is responsible for uncovering many defects and the effectiveness of the meeting in finding defects compared with other defect detection techniques. They utilized three detection techniques for testing the hypothesis namely Preparation – Inspection (PI), Detection – Collection (DC) and Detection – Detection (DD).

PI is a technique whereby the inspectors at first try to understand or only browse the document and only look for defects later in the meeting. DC is a technique where inspectors go through the document to find defects during the meeting. The individual inspector simply reports his/her own findings.

DD is a technique where another round of checking is added after the initial individual checking. Amongst the three methods used, DD was found to be the best. The DD technique recorded a detection rate of 46% followed by DC (23%) and finally PI (19%). Upon this finding, they concluded that meetings are not necessarily vital to successful inspections. However, they reported that further study is needed to confirm this finding. Porter et al.

Porter et al. [13] came to a conclusion that inspection meeting gains is approximately zero. In a series of experiments, they compared the number of defects found for the first time at the meeting (meeting gains) and the number of the defects found before the inspection meeting by any of the inspectors that have not been found in the meeting. They found that there is no significant difference between the two findings. Similar results have been concluded by the replication of the same experiment both in Italy [14] and in UK [15].

In another study, Porter and Johnson [16] compared two experimental studies of software review meetings. The experiments compared the performance of two different groups performing inspections: "real" inspection groups, which were involved in the normal inspection meeting, and "nominal" inspection groups, in which the result of the inspection is the correlation of individual inspectors' results.

They performed the comparison to test five hypotheses under the context that the real groups will outperform the nominal group. However, the studies failed to discover any significant difference in the number of defects found by the two groups. Instead, they found that the number of issues produced by the nominal groups were significantly more than the ones produced by the real groups.

This has raised a doubt about the effectiveness of the inspection meeting. On the other hand, these studies revealed that the group meeting is more effective than a meeting-less inspection in identifying the false positive defects. Also, inspection meeting was identified to be more effective in finding some certain types of defects. They concluded that the inspection meeting does not in itself increase nor decrease the detection capability of the inspection process.

Land et al. [17] confirmed the findings obtained by Porter. They found that the number of new defects reported by interacting groups (IG): groups that interact during the inspection meeting is low. According to this result, they discounted the synergy to be the important factor behind the inspection meeting. They also reported that there are defects discovered by individuals that have not been found by groups. The most important point they concluded is that interacting groups are the best at the discrimination between true and false positive defects. Therefore, they still have the performance advantage over the nominal groups in terms of the net defects.

Land et al. [18] reconfirmed these results and have demonstrated that interacting groups are the preferred choice over the average individuals and the nominal groups.

Finally, based on the above arguments, Glass concluded that inspection meeting is of doubtful value [19]. The above results resulted in development of new models. These models have reduced the load on the inspection meeting by distributing the process and discharging it from the inspection process.

Some asynchronous inspection models have been developed. The main concern of these models was to practice the software inspection without the need for all the inspectors to be present at the same time or to convene at the same place.

A model developed by Philip Johnson [20] has three identified roles: *moderator*, *producer* and *reviewer*. The moderator is the person who is in charge of the overall process, the producer is the author of the document and the reviewer is the person who performs the checking. The process goes through seven phases: *setup*, *orientation*, *private review*, *public review*, *consolidation*, *group review meeting* and *conclusion*.

In the *setup* phase, the inspection team is identified and the work product is made available using a computer tool called CSRS [21]. In the following phase, *orientation*, the inspection team is briefed about the inspection materials and objectives. In the *private review*, reviewers check the document and create annotations. In this phase, the annotations are kept private. However, they publicly become available in the *public review* phase.

Reviewers can view all comments and also add comments. New annotations can be added at this phase as well. When the team resolves all issues, or the moderator decides to terminate the discussion, this phase is considered complete. The *consolidation* phase then follows in which the moderator analyses the results of the private and public review phases, and summarizes unresolved issues. Based on the results, the moderator will decide whether or not a *group reviews meeting* will take place. The final phase is the *conclusion* where the moderator produces the final inspection report and the inspection metrics reports.

Another model was introduced by Mashayekhi *et al*. [22]. This model makes use of Humphrey's inspection process [23]. The process goes through the same phases except the inspection meeting, which has been substituted by a sequence of defect discussions. In some cases synchronous inspection meeting is held to resolve issues that have not been resolved asynchronously. The process starts with the *initialisation* phase where the moderator makes the inspection materials available to the whole inspection team.

Afterwards, the reviewers start to check the work product in what is called *fault collection*. A fault list is usually produced by each reviewer. In the following phase, *correlation*, the producer correlates the fault lists in one list. The correlated list is then posted to the inspection team for further asynchronous discussion. This phase is usually called *asynchronous meeting*. If the team manages to resolve all the issues in the correlated fault list, an action-item list with the resolved issues and suggested resolutions is forwarded to the producer for *rework*.

On the other hand, if the team failed to resolve some issues, then the moderator decides whether or not a synchronous meeting phase should take place. The action item list is passed to the author to make the necessary corrections. This phase is called *rework*. The moderator, as in the traditional *follow-up* phase, assures that all the changes have been properly made.

A model introduced by Paul Murphy and James Miller [24, 25] replaces the inspection meeting with another round of individual inspection. The process starts with the *planning* phase in which the moderator prepares for the inspection. The first round of *individual preparation* then starts. Here,

inspectors individually go through the document looking for defects. When reaching a pre-stated deadline, each inspector circulates his/her own defect list to the rest of the team and the moderator for review.

Using a communication mechanism such as email, inspectors then discuss those defects. A second round of *individual review* is followed. Learning from others inspectors' defects, an inspector can generate new defects, reclassify or delete his/her old ones. The outcome of the second round is then submitted to the moderator who collates the defect lists into one list.

This list is sent to the author for rework. In contrast to Humphrey's inspection process, the model opposes the idea that the author can participate in the defect detection or collation. The final phase is the traditional follow-up activities.

4. Evaluation Criteria for Software Inspection Tools

Hedberg and Lappalainen [26] introduced the DESMET method of evaluation on the functional requirements of a software inspection tool covering the following criteria: artifact management, defect management, process management, process improvement support and quality aspects. In their paper, a brief description of each criterion was given.

In order to develop a good software inspection tool that can survive major changes in the software development process, we have to consider the nature of software inspection that would be performed. One good article linking inspection with formal technical review process was written by Philip Johnson [27]. His article gave some important insights of tools that cannot be overlooked and have to be taken into account in developing any software inspection tool. He provided seven recommendations for formal technical reviews as follows:

Providing tighter integration between inspection and the development method: integrating the inspection method with development in use improves the software development process as well as the quality of the inspection method. For example, building inspection checklists when using the spiral model might differ when using the waterfall model.

Minimizing meeting and maximizing asynchronous meeting: moving towards asynchronous meetings may improve the inspection process in some aspects. Asynchronous meetings relax the time and the place factors, support the review of larger artifacts, and ameliorate the interval time problem identified in the synchronous meetings.

Shifting the focus from defect removal to improved developer quality: inspection methods should not focus on the author as much as on the products of review. For example, discussing issues, instead of raising issues but not resolving in the inspection meeting, helps improve the quality of the inspection process.

Building organizational knowledge based on review: software inspections should generate and maintain an organizational knowledge based on guidelines, checklists and others. This knowledge base should be available to other reviewers and developers.

Outsourcing review and insourcing of review knowledge: the service of an external consultant with specialized knowledge might replace the insource review in organizations. In addition, organizations could insource their review knowledge by buying or building review guideline database.

Investigating computer-mediated review technology: using automation to accomplish the reengineering of software inspections. It facilitates the points pointed above.

Breaking the boundaries in review group size: group size should not be restricted to 6-9 participants as in current approaches. The use of computer-mediation allows for increase in the number of participants, allowing the review to be carried out effectively and efficiently.

Based on the points presented above, we can arrive at the following conclusion:

• Software inspection is moving towards asynchronous mode. Therefore, software inspection tools have to implement a good asynchronous inspection model to make the process more effective.

• Software inspection tools should maintain an inspection knowledge base that can be used in the improvement of the software inspection process.

5. Attributes of Collaborative Artifact Inspection Model and Tool

Considering various requirements, we propose the following model features for collaborative artifact inspection and review.

Artifact Type. The model and tool should support any type of documents allowing the different milestones' deliverables to be inspected. In addition, it should be capable of displaying any number of documents required to accomplish the inspection task. The model and tool should be able to support inspection and review of artifacts such as software requirements document and systems documentation.

Inspection Management. The Inspection process should be managed to ensure quality process design and execution. Process steps should be well documented.

Linked Annotations. The model and tool have to implement an annotation mechanism that connects annotations to the relevant locations in the respective documents.

Implementing a good inspection meeting model. The model and tool are to implement a good asynchronous inspection meeting model supported by a workflow feature. This will ensure steps are adhered to through effective completion of tasks.

Query & Reporting. The model and tool should be able to report inspection results and statistics that can be used in the assessment of the inspection process. Reports should include the following:

- Identification of annotations made by
    - all inspectors/reviewers
    - any single inspector/reviewer
- Identification of annotations, based on certain keywords
- Identification of annotations made by all or some inspectors/reviewers on
    - selected sections of document
    - selected pages

Checklist supported inspection/review. The model and tool should make use of checklists to guide the steps through the inspection or review process. This will help structure a review process that is consistent through standardization.

Support Functions. The inspection process should be supported by tool functions such as search and glossary. This will allow reference to words be clarified during the process of inspection.

Reaching consensus. The model and tool should allow the inspection team to initiate discussions, comment and reach consensus regarding issues. The model and tool should include mechanisms to record preferences and decisions.

Administration. The model and tool need to support authorization of different roles with responsibilities and restrictions along the process steps. This includes incorporation of selected annotations to create draft and final versions of artifacts.

Attachments. The model and tool should allow for attachment of files linked to annotations. For instance, a video or audio clip that gives an overview of a task to be inspected should help put things in perspective without having to read a document. It is also necessary for author and inspector(s) who might want to attach files to support or demonstrate comments made.

Process Improvement. The model and tool should collect inspection data for use in future projects, for example, common causes of defect relating to requirements domain or project team. The tool should maintain a good inspection database that can be used in software process improvement.

Availability, Accessibility & Security. The system should be available 24x7 to allow for inspectors and reviewers located in different time zones to have secure access. A web-based implementation is suitable.

6. Conclusion

Inspection offers the opportunity to detect and remove defects in the software development process. Early detection will reduce the time delay and cost implications resulting from late detection. Effective facilitation and execution of the process will ensure timely change, when and where necessary.

Multi-site software development projects separated geographically in different time zones require collaborative tools. This paper presents the basic requirements for a collaborative artifact inspection and review model. We finally proposed twelve features of model and tool for collaborative artifact inspection and review.

References:

[1] A. Nwesri, K. Hashim, A Model for Collaborative Artifact Inspection and Review, *Proc. of the 12th. WSEAS Int. Conf. on Computers*, Greece, July, 2008.

[2] M. Fagan, Design and Code Inspections to Reduce Error in Program Development, *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182-211.

[3] F. Shull, C. Seaman, Inspecting the History of Inspections: An Example of Evidence-Based Technology Diffusion, *IEEE Software*, Volume 25, 2008, pp. 88 – 90.

[4] A. AlAzzazi, A. E. Sheikh, Security Software Engineering: Do it the right way, *Proceedings of the 6th. WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems*, Greece, 2007.

[5] C. Denger, F. Shull, A Practical Approach for Quality-Driven Inspections, *IEEE Software*, Volume 24, Issue 2, 2007, pp. 79-86.

[6] D. L. Parnas, The Role of Inspection in Software Quality Assurance, *IEEE Transactions on Software Engineering*, Vol. 29, No. 8, 2003, pp. 674-676.

[7] M. E. Fagan, Advances in Software Inspection, *IEEE Transactions of Software Engineering*, 12(7): 744-751, July, 1986.

[8] T. Kyung, S. Kim, A Study on the Development of Rules for Effective Code Inspection: Case Study of Company "A" Information System, *Proc. of WSEAS Int.*

Conf. on Applied Computer and Applied Computational Science, China, 2008.

[9] L. G. Votta, Does Every Inspection Need a Meeting? In *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 107–114, December 1993.

[10] P. M. Johnson and D. Tjahjono, Does Every Inspection Really Need a meeting? *Journal of Empirical Software Engineering*, Volume 4, Number 1, January, 1998.

[11] P. M. Johnson, *Reengineering Inspection: The Future of Formal Technical Review*, *Communications of the ACM*, 41(2):40-52, February, 1998.

[12] P. McCarthy, A. A. Porter, H. Siy, and L. Votta, An Experiment to Assess Cost Benefits of Inspection Meetings and Their Alternatives. *Technical Report*, Computer Science Dept., University of Maryland, 1995.

[13] A. 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, 1995.

[14] F. Lanubile and G. Visaggio, Assessing Defect Detection Methods for Software Requirements Inspections Through External Replication. *Technical Report*, Dept. of Informatica, University of Bari, 1996.

[15] J. Miller, M. Wood, M. Roper, and A. Brooks, Further Experiences with Scenarios and Checklists. *Technical Report*, Dept. of Computer Science, University of Strathclyde, 1996.

[16] A. A. Porter and P. M. Johnson, Assessing Software Review Meeting: Results of Comparative Analysis of Two Experimental Studies. *IEEE Transactions on Software Engineering*, 23(3):129–145, March 1997.

[17] L. Land, C. Sauer, and R. Jeffery, Validating the Defect Detection Performance Advantage of Group Designs for Software Reviews: Report of a Laboratory Experiment Using Program Code. In Jazayeri M. and Schauer H., editors, *Sixth European Software Engineering Conference Held Jointly with the Fifth ACM SIGSOFT Symposium on Foundations of Software Engineering*, Number LNCS 1301 in Lecture Notes in Computer Science, pp. 295-309. Springer, September, 1997.

[18] L. Land, R. Jeffery and C. Sauer, Validating the Defect Detection Performance Advantage of Group Designs for Software Reviews: Report of Replicated Experiment. In Bailes P. A., editor, *Proceedings of Australian Software Engineering Conference, IEEE* Computer society, pp. 17-26, October 1997.

[19] R. L. Glass, Inspections-Some Surprising Findings, *Communications of the ACM*, 42(4):17–19, April 1999.

[20] P. M. Johnson, An Instrumented Approach to Improving Software Quality Through Formal Technical Review. In *Proceedings of the 16th International Conference on Software Engineering*, May 1994.

[21] P. M. Johnson and D. Tjahjono (1993). CSRS Users Guide. Technical Report ICS-TR-93-16, Collaborative Software Development Laboratory, Department of Information and Computer Sciences, University of Hawaii, 1993.

[22] V. Mashayekhi, C. Feulner, and J. Reidl, CAIS: Collaborative Asynchronous Inspection of Software. In *Proceedings of the Second ACM SIGSOFT Symposium on the Foundations of Software Engineering*, December 1994.

[23] W. S. Humphrey, *Managing the Software Process*, Chapter 10, pp. 171–190. *Addison-Wesley*, 1989.

[24] P. Murphy and J. Miller, Asynchronous Software Inspection, Technical Report EfoCS-27-97, Department of Computer Science, University of Strathclyde, 1997.

[25] P. Murphy and J. Miller, A Process for Asynchronous Software Inspection. In *Proceedings of the 8th International Workshop on Software Technology and Engineering Practice*, pp. 96–104, July 1997.

[26] H. Hedberg and J. Lapalainen, A Preliminary Evaluation of Software Inspection Tools with the DESMET Method, *Proc. Fifth International Conference on Quality Software*, 2005.

[27] P. M. Johnson, Reengineering Inspection: The Future of Formal Technical Review, *Communications of the ACM*, Volume 41, Issue 2, 1998. pp. 49-52.