

Preliminary Evaluation of Tool-based Support for Distributed Inspection

Filippo Lanubile

Teresa D. Mallardo

University of Bari
Dipartimento di Informatica
Bari, Italy
lanubile@di.uniba.it

ABSTRACT

Software inspections are a software engineering “best practice” for defect detection and rework reduction. In this paper, we describe our initial empirical evaluation with using a tool aiming to provide Internet groupware support for distributed software inspections. The tool is based on a restructured inspection process to reduce synchronization and coordination problems.

1. INTRODUCTION

Global software development [6] is a phenomenon started in the early 1990s, whose main drivers are the access to government contracts in foreign countries, the integration of groups from alliances and acquisitions, and the difficulty of staffing new projects with adequately skilled people. These forces are contrasted by negative factors such as higher costs and time delays, because distance causes overheads for management and affects teamwork cooperation, thus resulting in reduced productivity and higher development time [5].

Distributing software projects across multiple, geographically separated teams questions established best practices of software engineering and asks for new solutions. For example, focusing on verification activities in a global telecommunication company, the authors of a repeated case study reported that collocated peer reviews outperformed distributed peer reviews with respect to efficacy and efficiency [3]. We claim that traditional software processes, including peer reviews, should be significantly reengineered for being distributed across multiple sites, using the Internet as a collaborative infrastructure.

The focus of our work is on software inspection, an industry-proven type of peer review for detecting defect and reducing rework. Because of its roots on manual activities and face-to-face meetings, the software inspection is an excellent example of a traditional software process that might benefit from geographically distribution, on condition that it is reorganized to reduce synchronization and coordination, and then supported by some Internet-mediated environment.

In [2], we described an ongoing project to develop a tool, called the Internet-Based Inspection System (IBIS), aiming to provide groupware support for distributed software inspections. After one year, IBIS has become ready for use and this paper reports our

first experiences with using IBIS for distributed software inspections.

2. THE UNDERLYING MODEL OF SOFTWARE INSPECTION

Software inspections were developed first by Michael Fagan at IBM [4] and then adopted with many variants [7] for the purpose of defect detection and process improvement. Software inspections are distinguished from other types of peer reviews in that they rigorously define:

- a phased process to follow (e.g., planning, overview, preparation, meeting, rework, follow-up);
- roles performed by peers during review (e.g., moderator, author, recorder, reader, and inspector);
- a reading toolset to guide the review activity (e.g., defect taxonomies, product checklists, or scenario-based reading techniques);
- forms and report templates to collect product and process data.

Recently, a reorganization of the inspection process has been proposed [9] to shorten the overall cost and total time of the inspection process, on the basis of behavioral theory of group performance and the results of many independent empirical studies that argue the need for traditional meetings [1, 8].

The alternative design for software inspections mainly consists of replacing the preparation and meeting phases of the classical inspection process with three new sequential phases: defect discovery, defect collection and defect discrimination (see Figure 1).

The first, defect discovery, reflects the historical shift of goal for the preparation phase, that has changed from pure understanding to defect detection, and so inspectors have to individually take notes of defects. The other two inspection phases are the result of separating the activities of defect collection (i.e., putting together defects found by individual reviewers) from defect discrimination (i.e., removing false positives), having removed the goal for team activities of finding further defects.

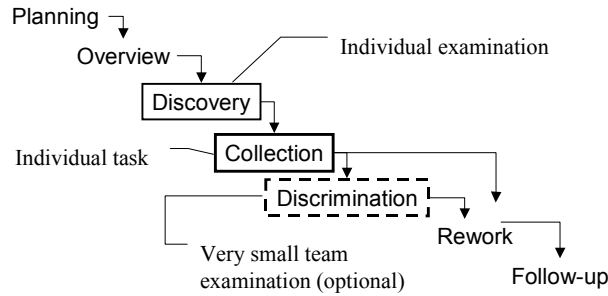


Figure 1. The reengineered inspection process

Defect collection independent of defect discrimination only requires either the moderator or the author himself. Defect discrimination may be skipped either entirely, passing all the collected defects directly to the author for rework, or partially skipped, excluding from the discussion any duplicate defects found during collection. Another change for saving time and diminishing coordination is introduced by reducing the number of discussants to those that can be recognized as experts, on the basis of the analysis of individual findings. The behavioral theory and empirical findings suggest that a single expert reviewer paired with the author may be as effective in the discrimination task as larger groups.

By reducing the need for synchronous communication between reviewers and the overload of coordination on the moderator's shoulders, the reengineered inspection process provides the model upon which to build an automated support for running distributed inspections.

3. IBIS

The IBIS architecture adopts a lightweight approach to achieve the maximum of simplicity of use and deployment. Since it is mainly a web application, IBIS uses technologies based on web standards, as specified by the World Wide Web Consortium (W3C).

All persistent data related to inspections are stored as structured XML documents, programmatically accessed via the DOM API, and automatically manipulated by XSL transformations. Thus, IBIS does not require any DBMS for accessing the data repository, except for archival purposes. All groupware functionalities are developed from dynamic web pages on the basis of scripts and server-side components. Event notification is achieved by generating email messages. Clients only need to use a browser and, optionally, an email reader. IBIS does not require any additional installation, neither manual nor automatic, and can be used throughout a company firewall.

In the following, we describe the use of the tool during the seven phases of the reengineered inspection process.

3.1 Planning

In the Planning stage, the moderator determines whether the product to be inspected meets the entry criteria and tailors the template inspection process: inspection goal, names for phases

and roles, categories for defect types and severity, and reading support items (at the current date, checklist questions are supported). In addition, a determination is made on whether to hold an overview and a list of contacts is invited by email to be part of the inspection team.

3.2 Overview

In the Overview stage, the inspection team can access to background information on the inspection process or the product being inspected. This stage may not be necessary if the team is already familiar with both the process and product.

3.3 Discovery

In the Discovery stage, members of the team perform individually the review task and record potential defects on a discovery log. Checklists can be used for guidance on typical types of defects to be found in the product being inspected. An inspector can add, modify or delete defects while as they are discovered. If it is the first access of the inspector to the discovery stage, the discovery form is shown with an empty defect list, otherwise it is shown with the defect list from the last session. When an inspector has declared the end of discovery, a notification message is sent to the moderator. The moderator can browse the discovery logs of all the inspectors to determine whether team members have adequately performed the reviews, and eventually invite inspectors for additional discovery.

3.4 Collection

In the Collection stage, all the discovery logs are collapsed into a unique defect inspection list. Either moderator or the author can set as duplicates identical defects from multiple inspectors. Looking for duplicates is helped by the ability to sort the merged defect list with respect to location fields and checklist questions. Duplicates are excluded from the discrimination stage and go directly to the rework stage. Looking at the inspection defect list without duplicates, the moderator may select which defects are worth to be discussed in the discrimination stage and which inspectors have to participate to the discussion. This decision is supported by the display of individual findings statistics, such as total number of reported defects and number of unique defects.

3.5 Discrimination

In the Discrimination stage, discussion takes place asynchronously as in a discussion forum. Each defect in the discrimination list is mapped to a threaded discussion. Invited inspectors may add their comments inside the threads and, when a consensus has been reached, either the moderator or the author can remove false positives.

3.6 Rework

In the Rework stage, the author is invited to correct defects found. He fills in defect resolution entries including information on how the defect has been fixed or, if not, the explanation. At the end, the author can upload the revised document and a notification message is sent to the moderator.

3.7 Follow-up

In the Follow-up stage, the moderator determines if defects found have been corrected by the author and that no additional defects have been introduced. In order to decide whether exit criteria have

been met or another rework cycle is needed, the moderator may invite additional reviewers among inspection team members. Once the inspection is closed with a final recommendation, the system sends a notification mail to the team members including a summary and a detailed report as a feedback for participants.

4. FIRST EXPERIENCE WITH IBIS

We have done two types of inspections enacted by IBIS with the goals of testing the tool itself and gather a first feedback from users. In the first inspection, ten reviewers had to find defects in the IBIS requirements document (19 pages long). In the second inspection, eight reviewers had to detect programming style violations in a dynamic web document (694 lines of markup elements mixed to scripting code), which was part of the IBIS configuration.

Participants were graduate students in the computer science department at the University of Bari. All participants had received one lecture on software inspection and had practiced with the tool by using the demo version. The elapsed time of inspection was unusually long (one month, including Christmas vacations), but this allowed participants to work not only from the university but also from home, thus simulating the conditions of geographically dispersed teams.

The following quantitative data were measured from the key inspection stages:

- from Discovery: for each reviewer, number of defects recorded, number of unique and duplicate defects;
- from Collection: number of defects recorded, number of unique defects, number of duplicates, number of merged defects, number of defects selected for discrimination;
- from Discrimination: number of discussants (other than moderator and author), number of messages posted, number of resolved false positives;
- from Rework and Follow-up: number of unresolved false positives, number of true defects.

From the above direct metrics, the following indirect metrics were derived:

- Duplication: ratio of recorded defects to merged defects; a ratio of one means no overlapping of collected defects, while a high ratio would mean that many defects were discovered by multiple inspectors;
- Discussion Filtering: ratio of defects selected for discrimination to merged defects: a ratio of one means that there were no filtering while a low ratio would mean that only few defects were considered of interest for discussion;
- Discussion Intensity: number of posted messages per discussion thread (a thread corresponds to a defect selected for discrimination);
- Discrimination Efficacy: ratio of resolved false positives to total number of false positives; a ratio of one means that the author may concentrate on fixing true defects in the rework stage.

Table 1 shows the main descriptive statistics of individual reviewers' performance. Individual findings varied a lot between reviewers, particularly for the first inspection. In the requirements inspection, there were a couple of outstanding reviewers which reported 24 defects both, with very few duplicates (respectively one and four). On the other hand there were also poor reviewers who contributed with only three recorded defects or no unique defects. In the second inspection (checking for style conformance), there were more defects recorded by each reviewer but most defects overlapped rather than being unique contributions.

Table 1. Descriptive statistics of individual findings

	Inspection 1 (requirements inspection)	Inspection 2 (programming style compliance)
Defects recorded by each reviewer		
<i>Mean</i>	9.1	19.2
<i>Min</i>	3	12
<i>Max</i>	24	26
<i>SD</i>	8.2	5.5
Unique defects by each reviewer		
<i>Mean</i>	6.0	5.2
<i>Min</i>	0	0
<i>Max</i>	23	14
<i>SD</i>	8.4	5.4
Duplicate defects by each reviewer		
<i>Mean</i>	3.1	14.0
<i>Min</i>	1	2
<i>Max</i>	8	23
<i>SD</i>	2.0	8.8

Table 2 summarizes the measures for both the inspections. Looking at the results we highlight the following differences between the two inspections:

- There were more much more reviewers discovering independently the same programming style violation than detecting individually the same requirements defect;
- The “interesting” defects (i.e., worth to be selected for discrimination) were much more for the requirements document than for the dynamic web document.
- Discussants were more active when raising comments about requirements defects than programming style deviations.
- Discriminating among potential requirements defects was much more effective than discriminating among potential style violations.

Finally, qualitative data were collected from a questionnaire submitted to the participants at the end of the two inspections. Apart from reporting some problems with the tool, students stated that the IBIS tool was easy to use and practical because they could work at a time and a place of their favor. A student complained about response time but he said that this happened only at times, when working at home, and cannot be considered a tool fault.

Table 2. Measures from the two inspections

	Inspection 1 (requirements inspection)	Inspection 2 (programming style compliance)
Inspectors	10	8
Defects recorded	91	154
Unique defects	60	42
Duplicate defects	12	26
Merged defects	72	68
Duplication (recorded / merged)	1.3	2.3
Defects selected for discrimination	34	9
Discussion filtering (selected / merged)	0.47	0.13
Discussants (excluded moderator-author)	6	4
Messages	78	16
Discussion intensity (messages / selected)	2.3	1.8
False positives resolved	21	6
False positives not resolved	7	8
Discrimination Efficacy (FP resolved / all FP)	0.75	0.43
True defects	44	54

All students who had been invited into the discrimination stage found useful the discussion with respect to learning purposes, but most acknowledged that discussing about programming style deviations was not so worthwhile. When asked if they would have preferred to discuss in a face-to-face meeting or in a chat, the answers were of two types. Some answered that they would choose a chat or a face-to-face meeting, because they would feel less alone and more active in the discussion. Others answered that they liked the asynchronous discussion because, not being in a hurry, they had time to think about each other messages and could write better comments.

5. CONCLUSIONS

We have developed IBIS, a tool to support distributed software inspections over the Internet. The underlying process model has been summarized, as it provides the basis to cut down synchronous communication among inspectors and the coordination overhead of the moderator. The design of the system has been outlined together with a description of its use within the inspection stages.

We have presented two distributed inspections enacted by the tool. In analyzing the results, we were interested to see how the inspection performance may vary with respect to a very distinct type of product being reviewed. While an inspection team of ten reviewers was cost-effective for requirements defect detection, the opposite was true when an eight-person team looked for violations of programming style. Furthermore, we observed that the

contribution of the discussion for discriminating between false positives and true defects was important for the inspection of the requirements document but not for the verification of style conformance. These results can be explained with the different depth of understanding required to perform a careful review of the two products.

Result of this preliminary evaluation of IBIS represents both a tool beta test and an empirical contribution to which inspection characteristics work best under certain project conditions. As IBIS will continue to be used and improved, we intend to run controlled experiments and field studies to investigate the mechanisms that determine costs and benefits of distributed software inspection.

6. ACKNOWLEDGMENTS

Our thanks to students who volunteered for using IBIS and providing feedback to improve the tool.

REFERENCES

- [1] Bianchi, A., Lanubile, F., and Visaggio, G. A controlled experiment to assess the effectiveness of inspection meetings. Proceedings of METRICS 2001 (London, United Kingdom, April 2001), 42-50.
- [2] Caivano, D., Lanubile, F., and Visaggio, G. Scaling up distributed software inspections. Proceedings of the ICSE Workshop on Software Engineering over the Internet (Toronto, Canada, May 2001).
- [3] Ebert, C., Parro, C. H., Suttels, R., and Kolarczyk, H. Improving validation activities in a global software development. Proceedings of ICSE 2001 (Toronto, Canada, May 2001), 545-554.
- [4] Fagan, M. E. Design and code inspections to reduce errors in program development. IBM Systems Journal, 15, 3 (1976), 182-211.
- [5] Herbsleb, J. D., Mockus, A., Finholt, T. A., and Grinter, R. E. An empirical study of global software development: Distance and speed. Proceedings of ICSE 2001 (Toronto, Canada, May 2001), 81-90.
- [6] Herbsleb, J. D., and Moitra, D. Global software development. IEEE Software, 18, 2 (2001), 16-20.
- [7] Laitenberger, O., and DeBaud, J.M. An encompassing life cycle centric survey of software inspection. The Journal of Systems and Software, 50 (2000), 5-31.
- [8] Porter, A., Siy, H., Mockus, A., and Votta, L. Understanding the sources of variation in software inspections. ACM Trans. on Software Engineering and Methodology, 7, 1 (1998), 41-79.
- [9] Sauer, C., Jeffery, D. R., Land, L., and Yetton, P. The effectiveness of software development technical reviews: A behaviorally motivated program of research. IEEE Trans. on Software Engineering, 26, 1 (2000), 1-14.