Tool Support for Distributed Inspection

Filippo Lanubile

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

Teresa Mallardo

RCOST – Research Center on Software Technology
University of Sannio
Benevento, Italy
mallardo@unisannio.it

Abstract

Software inspection is one of the best practices for detecting and removing defects early in the software development process. We present a tool to support geographically distributed inspection teams. The tool adopts a reengineered inspection process to minimize synchronous activities and coordination problems, and a lightweight architecture to maximize easy of use and deployment.

1. Introduction

Software inspections were first developed by Michael Fagan at IBM [3] and then applied in many variants [9] with the main purpose of detecting and removing defects as early as possible in the software life cycle. Even if software inspections are considered a "best practice" for improving software quality, their widespread adoption is limited by clerical overhead and time bottlenecks due to the prevalence of paper-based activities and face-to-face meetings.

The conventional setting for software inspections also hinders their applicability in the context of global software development, where software engineering activities are spread across multiple sites and even multiple countries [7]. A case study at Alcatel's Switching and Routing Division shows that collocated inspection teams were more efficient and effective than distributed teams, where code inspections were conducted remotely [2].

Based on empirical studies of classical software inspections and lessons learned from open-source software projects, we present a tool, called the Internet-Based Inspection System (IBIS), that aims to support geographically distributed inspections. While in [1] IBIS was introduced at an initial stage of development, now it has become ready for use. The results of a preliminary empirical evaluation were reported in [10]. In this paper we describe the tool architecture and its use in the enacted distributed inspection process.

2. Tool Architecture

IBIS is an Internet application based on a lightweight architecture to achieve the maximum of simplicity of use and deployment. All structured and persistent data are stored as XML files, programmatically accessed via the DOM API, and automatically manipulated by XSL transformations. All required groupware features are developed from dynamic web pages on the basis of scripts and server-side components. Event notification is achieved through automatic generation of emails.

Analogously to open-source software projects, which provide the evidence of successfully distributed projects, the IBIS uses general-purpose media such as web browsers and email readers as the prevalent client-side communication and coordination tools. The reasons for this choice are the following:

(1) using the lowest common denominator for Internet-based communication significantly increases the chances for potential reviewers to participate at inspections;
(2) the geographical dispersion of inspectors across different countries and continents practically prevents the usage of synchronous communication;
(3) enforcing a prescriptive coordination technology on the client side might impair occasional participation of expert reviewers from different divisions or outside organizations.

Figure 1 shows the IBIS architecture, represented as an UML deployment diagram. The current configuration of the IBIS is the following:

- Modern browsers to render HTML 4.0 pages with presentation effects specified by CSS1 and CSS2 properties.
- Web and application servers based on Microsoft (MS) Internet Information Server (IIS), including
  - Active Server Pages (ASP) components with server-side scripts to implement the business logic and dynamically generate web client pages;
  - a dynamic link library to parse XML data, manage the tree representation via DOM, and execute XSL transformations;
  - a dynamic link library to generate notification mail messages;
  - a dynamic link library to upload new or changed documents through the HTPP POST method;
- Mail server based on MS SMTP Mail Service (an IIS feature for a virtual SMTP server).

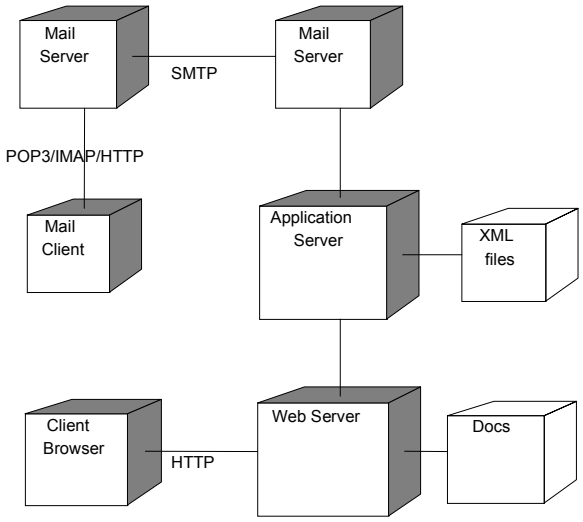

Figure 1. IBIS architecture

3. Tool Functionality

At the project startup, the functional requirements of the IBIS were mainly based upon a classical inspection process (shown in Figure 2), as defined by a NASA guidebook [12].

Other than providing an automated support for clerical activities, the major departure from the NASA guidebook was that inspections could not be collocated, and then face-to-face meetings should be replaced by various forms of data conferencing (synchronous or asynchronous) or removed at all, depending on circumstances.

However, other researchers have empirically assessed tool support for inspections, showing no differences from paper-based inspections with respect to both effectiveness and efficiency [11]. Internet-based support makes it possible to reach skilled reviewers everywhere but it does not provide a process itself for the effective interaction of geographically dispersed inspection teams. Because of its roots on manual activities and face-to-face meetings, the inspection process must be reengineered [8].

Based on many empirical studies on software inspections, a reorganization of the inspection process has been proposed [13]. The reengineered inspection process, shown in Figure 3, 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.

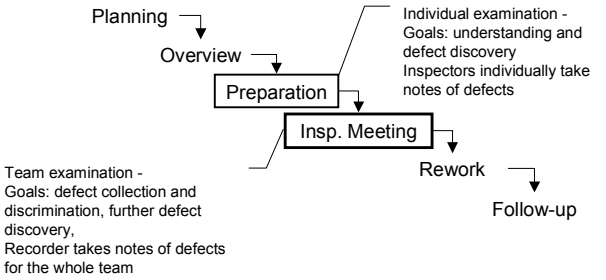

Figure 2. The classical inspection process

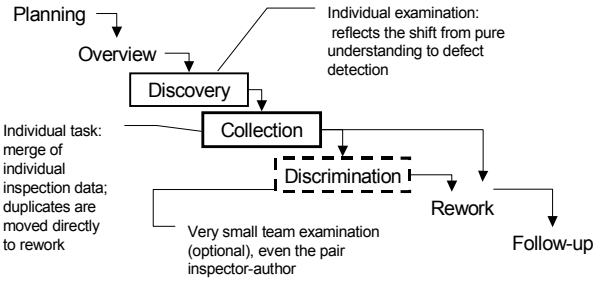

Figure 3. The reengineered inspection process

The name "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 new inspection phases are the result of having removed the goal for team activities of finding further defects and then separating the activities of defect

collection (putting together defects found by individual reviewers) from defect discrimination (removing false positives).

Collection independent of defect discrimination only requires either the moderator or even the author himself, with the addition of an automatic support for merging individual inspection data, thus eliminating the phenomenon of collection losses. Defect collection could also include identifying and marking for rework without further discussion any duplicate defects found, thus saving time. Defect discrimination may either be skipped, passing the collected defects directly to the author for rework, or the number of reviewers may be reduced to those that can be recognized as experts, on the basis of the analysis of individual findings.

The reengineered inspection process makes it possible to employ a large number of parallel inspectors focusing just on defect discovery, thus increasing the probability of detecting "hard to find defects". A high number of inspectors has an effect on cost but not on interval time, and will not pose coordination problems: defects found by more than one inspector are moved directly to the author and defects found by just one inspector are discussed by smaller groups, even the pair inspector-author.

In the following, we describe the use of the tool according to the seven stages that comprise 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 (Figure 4): 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 (Figure 5).

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.

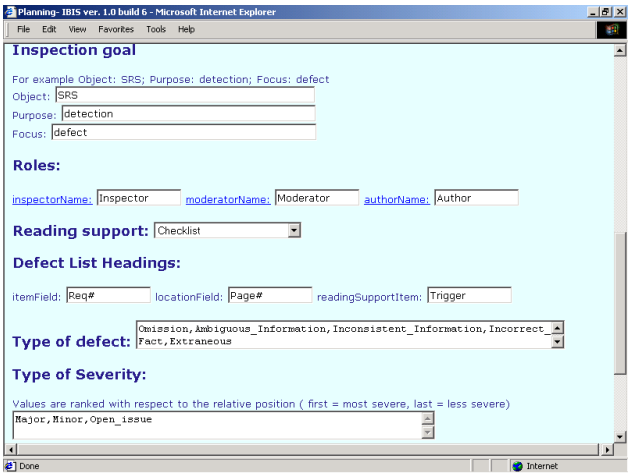

Figure 4. Customize inspection during Planning

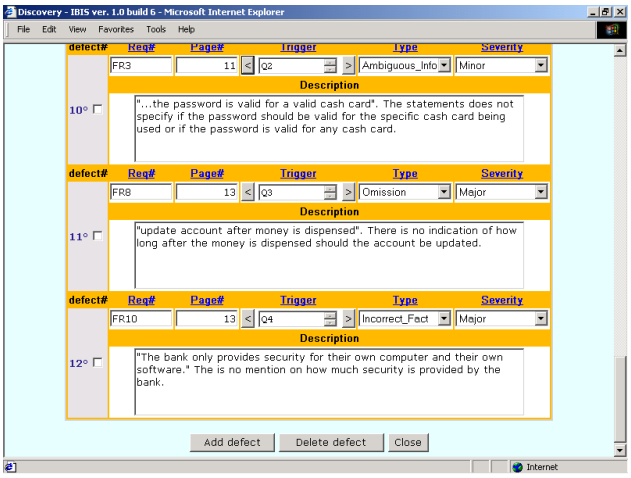

Figure 5. Defect logging during Discovery

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 (Figure 6) to determine whether team members have adequately performed the reviews, and eventually invite inspectors for additional discovery.

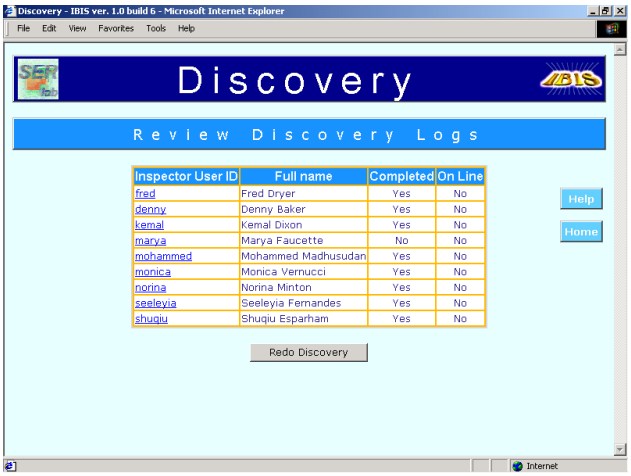

Figure 6. Review logs during Discovery

3.4 Collection

In the Collection stage, all the discovery logs are collapsed into a unique defect inspection list (Figure 7).

Either moderator or the author can set as duplicates identical defects from multiple inspectors (Figure 8). Looking for duplicates is helped by the ability to sort the merged defect list with respect to location fields and checklist questions. Duplicates can be excluded from the discrimination stage and be routed 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 (Figure 9). This decision is supported by the display of individual findings statistics, such as total number of reported defects and number of unique defects.

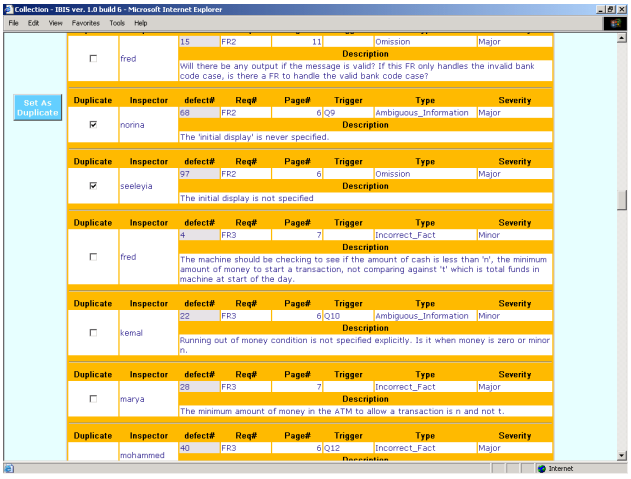

Figure 7. Merge discovery logs during Collection

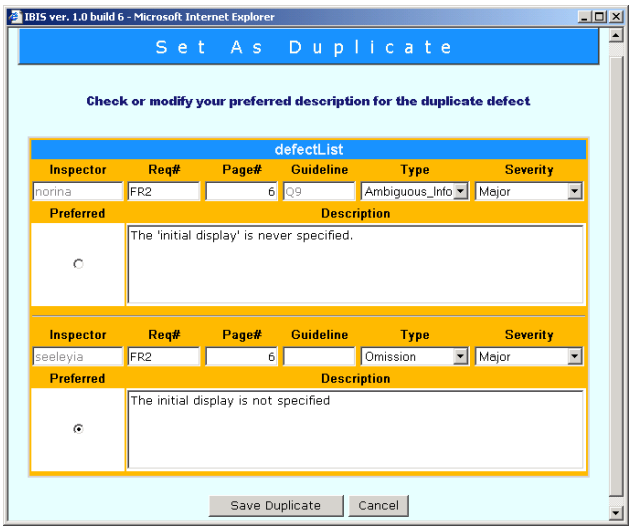

Figure 8. Duplicate setting during Collection

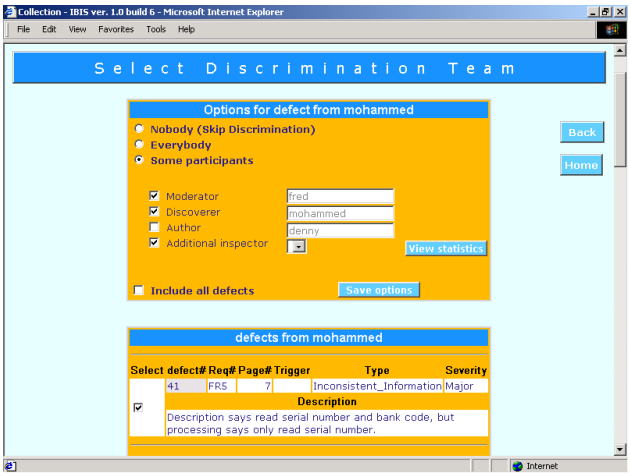

Figure 9. Discrimination planning during Collection

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 (Figure 10). 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 (Figure 11). At the end, the author can upload the revised document and a notification message is sent to the moderator.

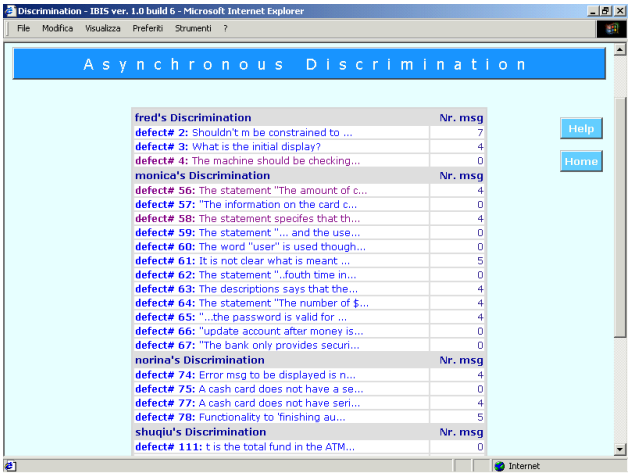

Figure 10. Threaded discussion during Discrimination

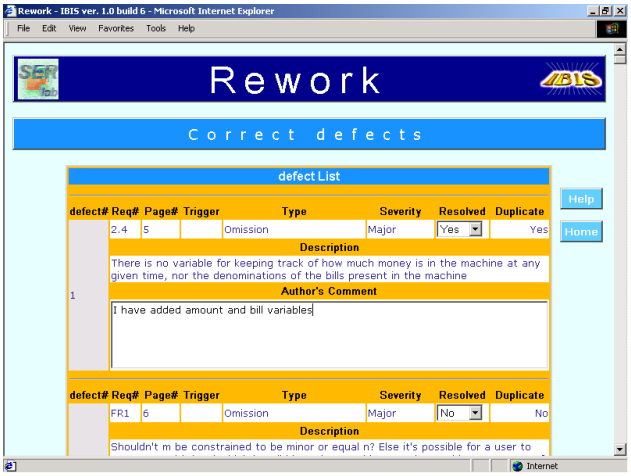

Figure 11. Defect correction during Rework

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.

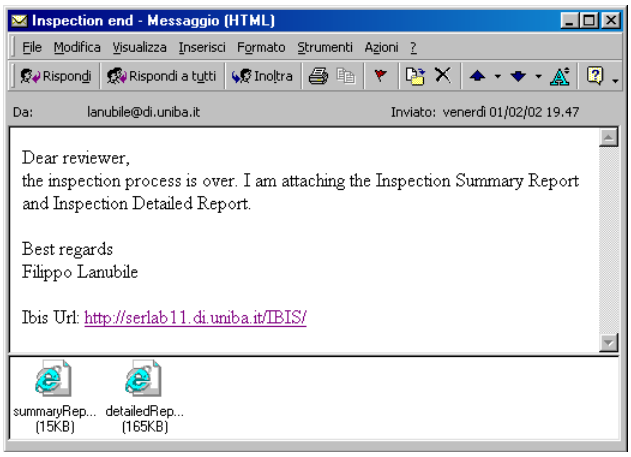

Figure 12. Notification of inspection closure during Follow-up

4. Conclusions

Software inspections have been adopted from many years by large industrial organizations because of their impact on product quality and cost of non-quality. However, software inspections have been limited in their adoption by a lack of automated support, the prevalence of synchronous communication between participants, and the overload of coordination on the moderator's shoulders.

In this paper we have described IBIS, a tool to support geographically distributed inspections. Based on lessons learned from open source software projects, IBIS is built upon a lightweight architecture to achieve the maximum of simplicity of use and deployment. Based on empirical studies of software inspections, IBIS adopts a reengineered inspection process to minimize coordination problems, without loosing the advantages of inspections (phased process, reading toolkits, roles, measurement) over less formal review processes.

The conformance to a specific reengineered inspection process and the lightweight architectural approach makes the IBIS different from other web-mediated inspection systems [4, 5, 6], and applications developed on the basis of groupware platforms [14, 15].

Currently, IBIS implements most of the functionality needed for distributed inspections. It can be tailored to support variations with respect to the reviewed work products, defect classifications, and defect detection support such as checklist questions. As future work, we intend to evaluate IBIS within industrial organizations to get user feedback.

5. References

[1] D. Caivano, F. Lanubile, and G. Visaggio, "Scaling Up Distributed Software Inspections", *Proc. of the Fourth ICSE Workshop on Software Engineering over the Internet*, Toronto, Canada, 2001.

[2] C. Ebert, C. H. Parro, R. Suttels, and H. Kolarczyk, "Improving Validation Activities in a Global Software Development", *Proc. of Int. Conf. on Software Engineering*, Toronto, Canada, 2001.

[3] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, 15(3):182–211, 1976.

[4] L. Harjumaa, and I. Tervonen, "A WWW-based Tool for Software Inspection", *Proc. of the 31st HICSS Conf.*, vol III, 1998.

[5] L. Harjumaa, and I. Tervonen, "Virtual Software Inspections over the Internet", *Proc. of the Third ICSE Workshop on Software Engineering over the Internet*, Limerick, Ireland, 2000.

[6] H. Hedberg, and L. Harjumaa, "Virtual Software Inspections for Distributed Software Engineering Projects", *Proc. of the ICSE Int. Workshop on Global Software Development*, Orlando, FL, USA, 2002.

[7] J. D. Herbsleb, and D. Moitra, "Global Software Development", *IEEE Software*, 18(2): 16-20, 2001.

[8] P.M. Johnson, "Reengineering Inspection", *Comm. of the ACM*, 41(2): 49-52, 1998.

[9] O. Laitenberger, and J.M. DeBaud, "An Encompassing Life Cycle Centric Survey of Software Inspection", *The Journal of Systems and Software*, 50: 5-31, 2000.

[10] F. Lanubile, and T. Mallardo, "Preliminary Evaluation of Tool-based Support for Distributed Inspection", *Proc. of the ICSE Int. Workshop on Global Software Development*, Orlando, FL, USA, 2002.

[11] J. Miller, and F. MacDonald, "An Empirical Incremental Approach to Tool Evaluation and Improvement", *The Journal of Systems and Software*, 51: 19-35, 2000.

[12] NASA Goddard Space Flight Center, Software Formal Inspections Guidebook, NASA-GB-A302, 1993, available at http://satc.gsfc.nasa.gov/fi/fipage.html

[13] C. Sauer, D. R Jeffery, L. Land, and P. Yetton, "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research", *IEEE Transactions on Software Engineering*, 26(1): 1–14, January 2000.

[14] Software Development Technology (SDT), ReviewPro, available at http://www.sdtcorp.com/reviewpr.htm

[15] M. van Genuchten, C. van Dijk, H. Scholten, and D. Vogel, "Using Group Support Systems for Software Inspections", *IEEE Software*, 18(3) : 60-65, 2001.