SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2003; **8**: 217–231 (DOI: 10.1002/spip.184)

Tool Support for Geographically Dispersed Inspection Teams

Filippo Lanubile*,†, Teresa Mallardo and Fabio Calefato
Dipartimento di Informatica, University of Bari, Italy

Research Section

Software inspection is one of software engineering's best practices for detecting and removing defects early in the development process. However, the prevalence of manual activities and face-to-face meetings within software inspections hinder their applicability in the context of global software development, where software engineering activities are spread across multiple sites and even multiple countries.

In this article, we describe a web-based tool, called the *Internet-Based Inspection System* (IBIS), that aims to support geographically dispersed inspection teams. On the basis of findings from empirical studies of software inspections, the IBIS tool adopts a reengineered inspection process to minimize synchronous activities and coordination problems. We present the underlying process model, how the tool is used within the inspection stages, and experiences using the IBIS tool as the enabling infrastructure for distributed software inspections. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: software inspection; distributed teams; tool support; empirical study

1. INTRODUCTION

Over the last few years, enterprise organizations have embraced global software development, where software projects are distributed over multiple geographical sites, separated by a national boundary (Carmel 1999, Herbsleb and Moitra 2001). Searching for low-cost but well-trained developers in emerging countries, and integrating groups from mergers and acquisitions are two of the main forces behind global software development. However, distance causes overheads for management and affects teamwork cooperation, thus resulting in longer development time (Herbsleb *et al*. 2001).

The focus of our research is on software inspection, an industry-proven type of peer review for detecting and removing defects as early as possible in the software development cycle, then reducing avoidable rework (Ebenau and Strauss 1994, Freedman and Weinberg 1990, Gilb and Graham 1993, Wiegers 2001). Even if software inspection is considered as the 'best practice' for improving software quality, the prevalence of paper-based activities and face-to-face meetings hinders its applicability in the context of global software development (Johnson 1998). 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 (Ebert *et al*. 2001).

Software inspection is an excellent example of a traditional software process that might be globally deployed, on condition that it is reorganized to reduce synchronization and coordination,

* Correspondence to: Filippo Lanubile, Dipartimento di Informatica, University of Bari, via Orabona 4, 70126, Bari, Italy
† E-mail: lanubile@di.uniba.it

and then supported by some Internet-mediated environment.

In this article, we describe a web-based tool, called the *Internet-Based Inspection System* (IBIS), that supports geographically dispersed inspection teams. Analogously to Perry *et al*. (2002), we used findings from empirical studies to determine how to restructure the inspection process in the context of geographical separation of reviewers.

Section 2 of the article summarizes and discusses software inspection, including the process variant we have adopted for our tool. Section 3 presents the IBIS tool and shows its use according to the underlying inspection process. Section 4 describes experiences where the IBIS tool was used as the enabling infrastructure for distributed software inspections. In Section 5, we review related work. In Section 6, we discuss the impact of synchronous and asynchronous activities on the software inspection process. Finally, in Section 7, we provide conclusions.

2. SOFTWARE INSPECTION

Software inspection is an industry's best practice for delivering high-quality software. The main benefit of software inspections derives from detecting defects early during software development and then reducing avoidable rework. Software inspections are distinguished from other types of peer reviews in that they rigorously define:

- a phased process to follow;
- roles performed by peers during review;
- a reading toolset to guide individual defect detection activity;
- forms and report templates to collect product and process data.

From the seminal work of Fagan (1976) to its many variants (Laitenberger and DeBaud 2000), the software inspection process is essentially made up of six consecutive steps, as shown in Figure 1.

During Planning, the moderator selects the inspection team, arranges the inspection material and sends it to the rest of the team, and makes a schedule for the next steps. During Overview, the moderator can optionally present process and product-related information for newcomers, if any. During Preparation, each inspector analyzes the document to become familiar with it and individually find potential defects. During the

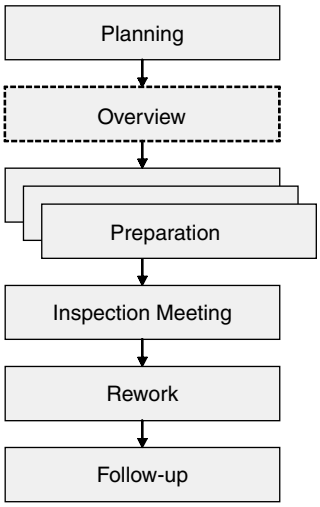

Figure 1. Conventional inspection process

Inspection Meeting, all the inspectors meet to collect and discuss the defects from the individual reviews, and further review the document to find further defects. During Rework, the author revises the document to fix the defects. Finally, during Follow-up, the moderator verifies author's fixes, gives a final recommendation, and collects process and product data for quality improvement.

The main changes from the original Fagan's inspection have been a shift of primary goals for the Preparation and Inspection Meeting stages. The main goal for Preparation has changed from pure understanding to defect detection, and so inspectors have to individually take notes of defects. Consequently, the main goal of the Inspection Meeting has been reduced from defect discovery to defect collection and discrimination, including the discussion of defects individually found during Preparation.

In the attempt to shorten the overall cost and total time of the inspection process, the need for a meeting of the whole inspection team has been debated among researchers and practitioners. Parnas and Weiss (1987) first dropped the team meeting in their Active Design Reviews. Then Votta (1993) showed how defect collection meetings lengthened the elapsed time of software inspections at Lucent Technologies by almost one-third, with defects discovered at the meeting (*meeting gains*), matched by defects not recorded at the meeting although found during Preparation (*meeting losses*). Further studies (Bianchi *et al*. 2001, Ciolkowksi *et al*. 1997,

Fusaro *et al*. 1997, Land *et al*. 1997, Miller *et al*. 1998, Porter *et al*. 1995, Porter and Votta 1998) have also observed that the *net meetings improvement* (difference between meeting gains and meeting losses) was not positive, and then *nominal teams* (teams that do not interact in a face-to-face meeting) are at least equivalent to *real teams*, at a lower cost and time. However, meetings have been found useful for filtering out *false positives* (defects erroneously reported as such by inspectors), training novices, and increasing self-confidence (Johnson and Tjahjono 1998, Land *et al*. 1997).

On the basis of the above empirical studies that argue the need for traditional meetings and on the behavioral theory of group performance, Sauer *et al*. (2000) have proposed a reorganization of the inspection process to shorten the overall cost and total time of the inspection process. The alternative design for software inspections mainly consists of replacing the Preparation and Inspection Meeting phases of the classical inspection process with three new sequential phases: Discovery, Collection, and Discrimination (see Figure 2).

The Discovery phase reflects the shift of goal for the Preparation phase that has changed from pure understanding to defect detection, and so inspectors are asked to individually take note of defects.

The other two inspection phases are the result of separating the activities of defect collection (i.e. putting together defects reported by individual reviewers) from defect discrimination (i.e. removing false positives), having removed the goal for team activities of finding further defects. The Collection phase is an individual task and requires either the moderator or the author himself. The Discrimination phase is the only phase in which inspectors interact in a meeting. Sauer *et al*. (2000) suggest that the participation of the entire inspection team is not required; the number of discussants can be reduced to a minimal set, even a single expert reviewer paired with the author.

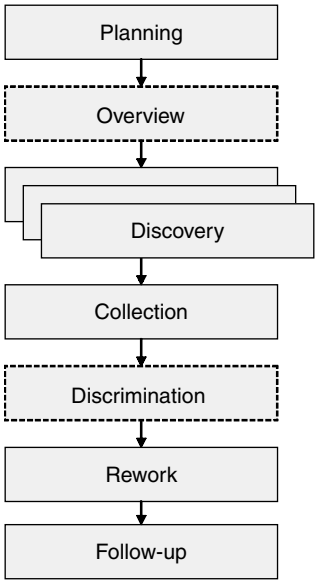

Figure 2. Reengineered inspection process

Another change for saving time and diminishing coordination overhead is introduced by skipping the Discrimination phase either entirely, passing all the collated defects directly to the author for rework, or partially, excluding from the discussion any potential defects (found by inspectors during the Discovery phase and merged in the Collection phase) that are considered to have high chances to be true defects. Sauer *et al*. (2000) suggest selecting for the Discrimination phase only *unique defects*, that is, defects that were found by only one inspector during the Discovery phase, while excluding *duplicates*, that is, defects that were discovered by multiple inspectors and were merged during the Collection phase.

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

In order to provide an Internet-based infrastructure for geographically distributed inspection teams, we developed the Internet-Based Inspection System.

IBIS is mainly a web-based application to achieve the maximum of simplicity of use and deployment. All structured and persistent data are stored as XML documents (Bray *et al*. 2000), programmatically accessed via the DOM API (Wood *et al*. 1998), and automatically manipulated by XSL transformations (Adler *et al*. 2001, Clark 1999). Most of the 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 e-mails.

In the following sections, we describe the use of the tool according to the seven stages that comprise the reengineered inspection process.

3.1. Planning

A new inspection starts when an author has sent a product to be reviewed and a moderator (who also acts as the coordinator of the inspection team) has determined that the product meets the entry criteria for inspection. The moderator selects a template for the inspection process, which may vary according to the inspection goal (e.g. analyze software requirements documents for the purpose of defect detection) and the reading technique, which inspectors will use for individual analysis (e.g. a checklist for software requirements documents). Then reviewers are invited to be part of the inspection team, and a specific reading technique is assigned to each inspector (see Figure 3).

Finally, the moderator can optionally schedule an overview meeting when the inspection team is unfamiliar with either the inspection process or the product.

3.2. Overview

The Overview meeting is optionally held as a virtual meeting to provide background information on the inspection process or the product being inspected.

The meeting is run using a helper application, called *P2PConference* (2004), which is launched from the browser. P2PConference is a peer-to-peer remote-conferencing tool, based on the JXTA framework (Wilson 2002). During a meeting, the speaker delivers his/her own text-based speech and the other participants can ask questions, after having 'raised their hands' (the moderator manages a queue of the pending questions). The moderator can also forbid participants to type and send statements. Figure 4 shows a screenshot of the conferencing tool.

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).

In order to support defect detection, inspectors can answer to checklist questions or, if they need more guidance, they can follow a reading scenario (Basili *et al*. 1996, Porter *et al*. 1995), which

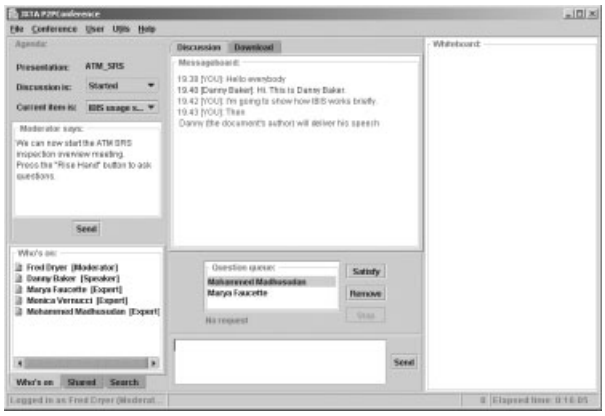

Figure 4. Remote conferencing during Overview

Figure 3. Team selection during Planning

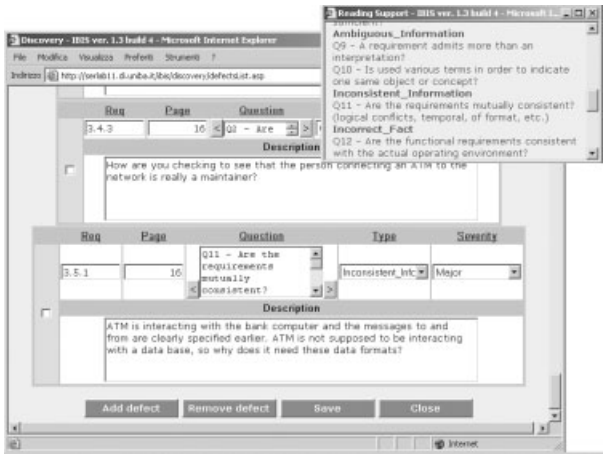

Figure 5. Defect logging during Discovery

gives a step-by-step description of the activities an inspector should perform while reading a document for the purpose of defect detection.

Defect logging can be performed in multiple sessions. When the Discovery task is finished, a notification message is sent to the moderator. The moderator can also browse discovery logs to check whether inspectors have adequately performed their task, and send invitations to complete the assignment.

3.4. Collection

In the Collection stage, all the Discovery logs are collapsed into a unique defect inspection list. Then either the moderator or the author looks for duplicate defects within the merged collection of discovery logs.

This is an iterative task that can be performed over multiple sessions too. For each iteration, the merged defect list is sorted with respect to location fields (e.g. document page number or requirement number) or questions related to the assigned reading technique (i.e. which question in a checklist or a reading scenario was helpful for defect discovery). When two or more defects are recognized as identical or almost identical (Figure 6), they are collapsed into a single defect entry and set as a duplicate (Figure 7). Duplicates can be excluded from the Discrimination stage and be routed directly to the Rework stage.

Looking at the merged defect list the moderator may plan the Discrimination stage by selecting

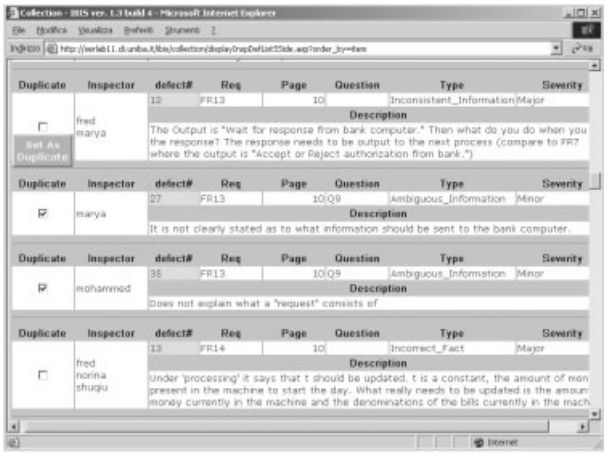

Figure 6. Duplicate search during Collection

Figure 7. Duplicate setting during Collection

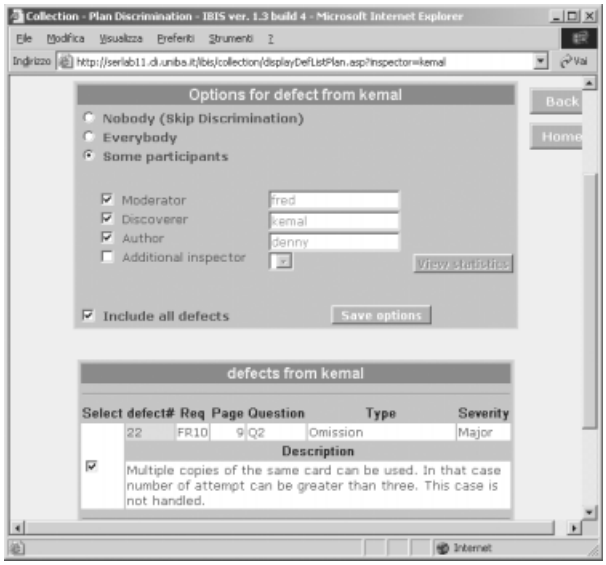

Figure 8. Discrimination planning during Collection

which defects are worth being discussed and which inspectors need to participate in the discussion (Figure 8). This decision can be supported by the display of inspectors' performance 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, where

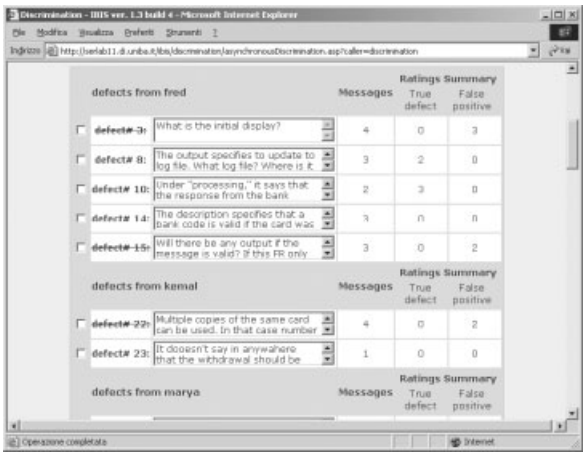

Figure 9. Defects as discussion threads during Discrimination

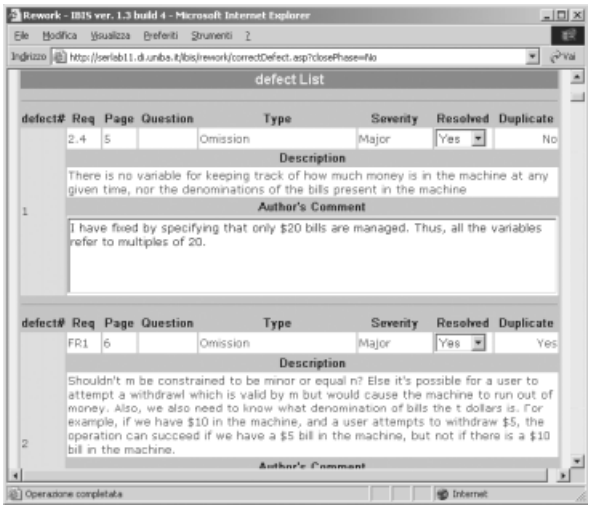

Figure 10. Defect correction during Rework

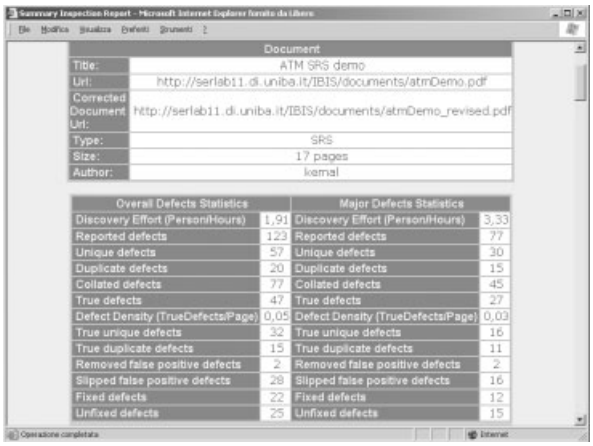

Figure 11. Final report sent to the inspection team

each defect is mapped to a threaded discussion (Figure 9).

Discussants can add comments within a thread and express votes by rating any potential defect as a true defect or a false positive. When a consensus has been reached, the moderator can mark potential defects as false positives. False positives appear as strikethrough in the discussion forum and are excluded from rework.

3.6. Rework

In the Rework stage, the author is invited to correct the defects found. He fills in defect resolution entries including information on how the defect has been fixed or, if not, the explanation (Figure 10). 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 the defects that have been 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 entire team, including a summary and a detailed report, as a feedback for participants (Figure 11). The detailed report mainly consists of the final defect list, together with the author's comments, and the list of defects that have been reported by each inspector. Both the summary and detailed reports include process and product data that have been collected during the inspection (the list of measures is shown in Table 1). Collected measures can be backed up to a project repository and analyzed for the purpose of process assessment and improvement.

4. EXPERIENCES WITH IBIS

We have conducted various distributed inspections as part of three consecutive editions of a web

Table 1. Product and process data measured in IBIS

Measure	Description
Document size	Number of pages (for requirements or design documents) or LOC (for source code).
Discovery effort	Person/hours spent during the Discovery stage.
Reported defects	Number of defects from the Discovery stage.
Unique defects	Number of defects that were reported by only one inspector during the Discovery stage.
Duplicate defects	Number of defects that were reported by multiple inspectors during the Discovery stage.
Collated defects	Number of defects resulting from merging unique and duplicate defects during the Collection stage.
True defects	Number of defects for which a general consensus is achieved during the inspection process.
Defect density	True defects/document size.
True unique defects	Number of true defects that were reported by only one inspector during the Discovery stage.
True duplicate defects	Number of true defects that were reported by multiple inspectors during the Discovery stage.
Removed false positives	Number of defects that are removed by the moderator at the end of the Discrimination stage because they are not considered as defects.
Slipped false positives	Number of defects from the Discrimination stage that are not acknowledged as true defects by the author during the Rework stage.
Fixed defects	Number of true defects for which the author provides a fix during the Rework stage.
Unfixed defects	Number of true defects for which the author does not provide a fix during the Rework stage.

engineering course at the University of Bari. Participants were graduate students interacting with the IBIS tool from university laboratories or home, thus simulating the conditions of geographically dispersed inspection teams.

4.1. First Experience

In the first experience with IBIS, we conducted two distributed inspections with the goal of testing the tool, gathering a first feedback from users, and observing the behavior of unusually large inspection teams (Lanubile and Mallardo 2002). 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. Using the IBIS tool itself, we measured inspection performance both at the individual and team level.

Figure 12 shows the box plots 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 who reported 24 defects each, with very few duplicates (respectively one and four). On the other hand, there were also poor reviewers who contributed with only three reported defects or no unique defects. In the second inspection (checking for style conformance), there were more defects reported by

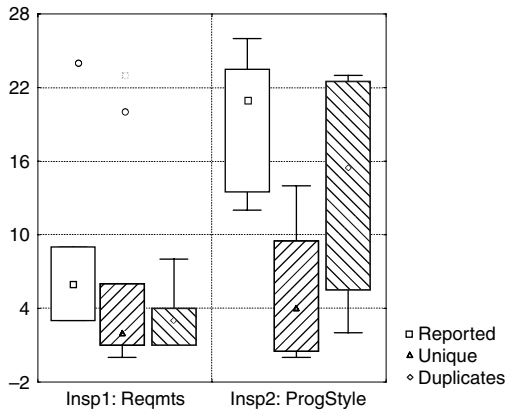

Figure 12. Individual reviewers' performance in the first experience

each reviewer but most defects overlapped rather than being unique contributions. Then, there were many more reviewers discovering independently the same programming style violation than detecting individually the same requirements defect.

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

- Defects worth discussing, and then selected for the Discrimination stage, were much more for the requirements document (discussion filtering = 0.47) than for the dynamic web document (discussion filtering = 0.13).

Table 2. Inspection team performance in the first experience

	Inspection 1 (requirements document)	Inspection 2 (programming style)
Collated defects	72	68
Defects selected for discrimination	34	9
Discussion filtering (selected defs/collated)	0.47	0.13
Messages	78	16
Discussion intensity (messages/selected defs)	2.3	1.8
Removed false positives	21	6
Slipped false positives	7	8
Discrimination efficacy (Removed FP/all FP)	0.75	0.43
True defects	44	54

- Discussants were more active when raising comments about requirements defects (discussion intensity = 2.3) than programming style deviations (discussion intensity = 1.8).
- Discriminating true defects from false positives was much more effective for requirements defects (discrimination efficacy = 0.75) than for style violations (discrimination efficacy = 0.43).

We also collected qualitative data 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 straightforward to use and convenient for the purpose because they could work at a time and a place of their favor.

All students who had been invited into the Discrimination stage found the discussion with respect to learning purposes useful, but most of them acknowledged that discussing about programming style deviations was not so worthwhile because the style was explicitly specified in a supplementary document (available to all the inspectors through IBIS).

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 prefer 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 to be part of an asynchronous discussion because, not being in a hurry, they had time to think about each other's messages and could write better comments.

4.2. Second Experience

In the second experience, IBIS was used as an instrument for an empirical study on software inspections (Lanubile and Mallardo 2003). The goal of the study was to find out how to reduce meeting effort by restricting the scope of discussions and the number of discussants. For this purpose, we aimed to identify: (1) which collated defects are worth a discussion, and (2) which discussants actively contribute to the decision of removing false positives. The former factor allows the moderator to shorten the discussion agenda while the latter makes it possible to shorten the size of the discussion group. Both factors have an effect on the meeting effort.

The second edition of the web engineering course required, as a student project work, to develop a web application, including documentation. The requirements documents of nine student projects (ranging from 7 to 23 pages) were submitted for inspection and a member of the development team was selected to act as the author in the inspection of his/her requirement document. In order to have a trained moderator, one of the researchers played the role of moderator for all the nine inspections. The rest of the inspection team was formed by two or three expert reviewers, external to the class, plus a student who was randomly selected from the class.

The Discrimination stage was planned to include all the collated defects (both unique and duplicate defects) and to invite the entire inspection team to the discussion.

We specifically tested the hypothesis that defects individually found by multiple inspectors (duplicates) are accepted as true defects in a group discussion, and can then skip the Discrimination stage, as suggested by Sauer *et al*. (2000).

We found that unique defects had higher chances than duplicates of being identified as false positives to be removed (conversely, duplicates had higher chances to be accepted as true defects). Figure 13 shows multiple bar plots for the two variables as measured in the nine inspections. We also found that decisions about false positives were actually supported by group consensus, because negative acknowledgments (votes as false positives) were proportionally higher for duplicates than for unique defects (as shown in Figure 14). These findings are consistent with the hypothesis of considering duplicates unworthy of a group discussion for the purpose of discrimination.

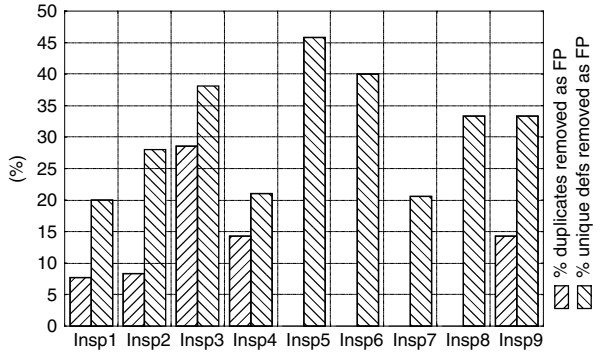

Figure 13. Duplicates and unique defects removed as false positives

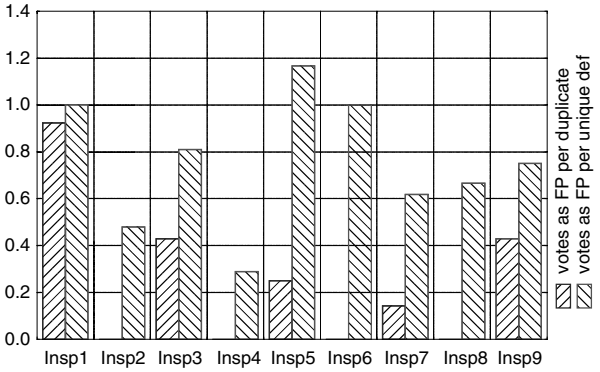

Figure 14. Votes as false positive per duplicate and unique defect

We also tested another hypothesis in the work of Sauer *et al*. (2000), stating that an expert pair performs the Discrimination task as well as any larger group. Figure 15 shows multiple bar plots for three variables, messages from moderator, author, and the most active reviewer respectively.

We found that most of the group discussions consisted of messages sent by either the moderator or the document's author. The other inspectors were less active in the discussion and mainly expressed their judgments by means of electronic votes without complementary messages. Then, these findings confirm the indication of limiting discrimination meetings to a couple of discussants (in this case, the moderator and the author).

At the end of each inspection, we collected qualitative data from interviews to the participants. Students, when acting in the role of author of a requirements document, declared to have found the discussion useful, because they had a chance to

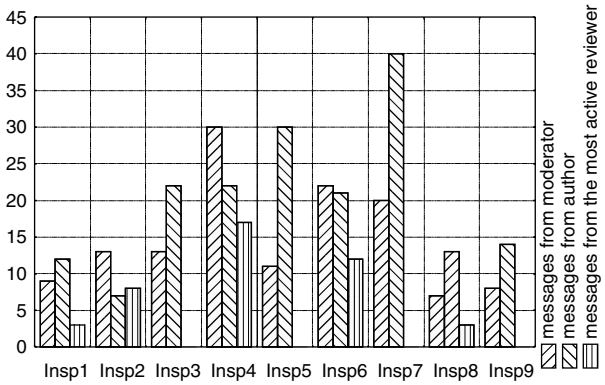

Figure 15. Posted messages per sender

learn from experts how to write a good requirement document.

When asked how to improve discussions in the Discrimination stage, most participants complained about late answers to their messages. However, they did not ask for a synchronous discussion or a face-to-face meeting but they pointed out the lack of features such as subscription to discussion threads or presentation of the most recent/active discussions.

4.3. Third Experience

In the third experience with IBIS, we ran a controlled experiment with the goal to find out if remote asynchronous discussions can replace face-to-face (F2F) meetings in the Discrimination stage without losses in effectiveness.

As in the second edition, students were required to develop a web application as the final course assignment. Again, the requirements documents of twelve student projects were peer reviewed with the help of the IBIS tool. However, this time six inspection teams performed the Discrimination task by discussing from a distance in an asynchronous mode, while a control group of other six inspection teams met in a university lab for the same purpose.

Participants were randomly assigned as reviewers to inspection teams. The Discrimination stage was planned to include the whole inspection team (three reviewers plus the author) and all the potential defects, which had been individually reported during the Discovery stage and merged by the author in the Collection stage. One of the inspectors was randomly selected for playing the role of

moderator. The moderator was assigned the responsibility of managing the discussion and marking defects as false positives, provided that there was the consensus of the other inspectors.

We invited participants to F2F meetings two days in advance. In order to guarantee similar conditions between the two types of inspections, two days were made available for asynchronous discussions.

At the team level, we collected the same measures as in the first experience, with the exception of the number of posted messages and discussion intensity, which could not be measured for F2F meetings.

The results are summarized in Table 3. Asynchronous discussions were slightly more effective (average discrimination efficacy = 0.895) than F2F meetings (average discrimination efficacy = 0.888). Only in two cases (Insp8 and Insp10) is the effectiveness of the asynchronous discussion lower than the minimum effectiveness of F2F meetings (Insp4). These low values seem to depend on poor discussion intensity.

Although the sample size is too small to run a test for differences, we can at least say that asynchronous discussions were as effective as F2F meetings to discriminate between false positives and true defects.

5. RELATED WORK

Conventional software inspection activities are paper-based and require face-to-face meetings but none of these two characteristics can live through geographical separation of inspectors.

There have been various research prototypes that provide an infrastructure technology for distributed software inspections. In a survey of computer support systems for software inspections, MacDonald and Miller (1999) reviewed 14 tools that had moved the entire inspection process online, then eliminating the need for paper. However, the first generation of tools did not provide support for distributed inspection teams. For example, ICICLE (Brothers *et al*. 1990) was designed to completely support the inspection of C and C++ code. The inspection meeting was intended to be collocated in the same room, with synchronized inspectors' screens. Another example is InspeQ (Knight and Myers 1993), which was developed to support only individual analysis with no support for group activities.

Conversely, CSI (Mashayekhi *et al*. 1993), Scrutiny (Gintell *et al*. 1993) and ASSIST (MacDonald and Miller 1997) focus on supporting a distributed environment by emulating face-to-face inspection meetings. CSI supports synchronous activities by means of shared displays and audio-conferencing, while Scrutiny uses teleconferencing facilities or simple text-based messages between participants. ASSIST is a flexible inspection tool supporting different inspection processes. It adds a shared whiteboard to text/audio/videoconferencing and threads of discussions.

CSI, Scrutiny, and ASSIST can be considered the precursors of inspection tools based on group support systems (GSS), which is a form of groupware. A

Table 3. Inspection team performance in the third experience

	F2F meeting						Asynchronous discussion					
	Insp1	Insp2	Insp3	Insp4	Insp5	Insp6	Insp7	Insp8	Insp9	Insp10	Insp11	Insp12
Collated defects	19	25	43	27	35	17	31	28	45	60	17	35
Defects selected for discrimination	19	25	43	27	35	17	31	28	45	60	17	35
Discussion filtering (selected defs/collated)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Messages	–	–	–	–	–	–	72	48	124	151	38	85
Discussion intensity (messages/selected defs)	–	–	–	–	–	–	2.32	1.71	2.76	2.52	2.24	2.43
Removed false positives	12	8	21	5	13	5	22	13	31	14	10	6
Slipped false positives	3	1	0	1	3	0	0	7	2	4	0	0
Discrimination efficacy (Removed FP/all FP)	0.80	0.89	1.00	0.83	0.81	1.00	1.00	0.65	0.94	0.78	1.00	1.00
True defects	4	16	21	21	19	12	9	8	12	42	7	29

GSS tool supports synchronous meetings by means of a shared list of tools, including categorizer, group outliner, topic commenter, vote, and so on. Modern GSS tools are Internet-based and then they can also support group tasks in distributed organizational contexts.

The GRIP tool (Halling *et al*. 2001) has been developed for inspecting requirements documents by tailoring a COTS GSS. GRIP exploits the GSS built-in tools to run collaborative synchronous inspection meetings. Experiments performed in an academic environment have shown that GRIP-based inspections outperform traditional manual inspections. Inspectors were interacting at the same time and the same place, through laboratory computers (Grunbacher *et al*. 2003, Halling *et al*. 2003).

Another GSS tool has been successfully applied in an industrial environment to support code inspections (van Genuchten *et al*. 2001). Inspectors worked from their office desks but they were located at the same organizational site.

GSS-based tools attempt to preserve the meeting-based characteristic of the classical inspection process. IBIS also supports synchronous remote meetings, through the P2PConference tool, but only in the Overview stage, where novices learn from the author about the artifact to inspect and from the moderator about the process to follow.

Another approach is to restructure the inspection process to eliminate the need for the inspection meeting or to replace it with asynchronous discussion forums. The goal is to remove the main bottleneck of the inspection process, without reducing inspection effectiveness. Furthermore, overlapping time windows for synchronous communication can dramatically increase the time to complete an inspection, when the inspection team works from multiple time zones.

CSRS, combined with the FTArm inspection method (Johnson 1994), and CAIS (Mashayekhi *et al*. 1994) are the first tools that depart from conventional meeting-based inspections by focusing on message-based asynchronous communication. AISA (Stein *et al*. 1997) evolved from CAIS to provide inspectors with a web browser as a front-end for performing inspection. The University of Oulu has developed a number of web-based inspection tools, starting from WiP (Harjumaa and Tervonen 1998), then WiT (Harjumaa and Tervonen 2000) and finally XATI (Hedberg and Harjumaa 2002). These tools support asynchronous inspections while improving user interaction and interoperability, thanks to the evolution of web technologies (from Java applets to XML).

HyperCode (Perry *et al*. 2002) is a web-based tool, which eliminates the inspection meeting and reduces the number of stages in the inspection process. Findings from individual defect discovery are shared among the inspection team as soon as they are detected. The Discovery stage and the Collection stage are performed concurrently, thus dramatically reducing synchronizing tasks. HyperCode has been used at Lucent Technologies to support geographically distributed reviewers and has resulted in a reduction of about 25% of the time needed to complete code inspections. However, it is still unknown how shared knowledge of defects affects effectiveness of inspections, also considering that HyperCode does not provide any guidance to inspectors while reading.

As shown in Table 4, IBIS has many similar features to those proposed in asynchronous software inspection tools, which organize defects to form threaded discussions.

However, discussions in IBIS are an option (as in WiT) and are limited to discriminating true defects from false positives, according to the redefined process beneath the tool. When discussions for discrimination purposes are conducted, IBIS makes it possible to reduce the number of participants (even a pair of discussants) and the number of defects that are worth a discussion (e.g. only unique defects). The selection is performed by the moderator before entering the Discrimination stage.

Also, GRIP makes it possible to reduce the number of reported defects, which are worth a discussion (Grunbacher *et al*. 2003). However, the selection criterion adopted by GRIP is based on votes expressed by inspectors at the beginning of the Discrimination stage. Only reported defects with diverging votes are discussed but, unlike IBIS, in a synchronous mode.

Furthermore, IBIS improves support for individual analysis by providing procedural reading scenarios other than checklists, and allowing the moderator to assign a specific reading support on an individual basis. These features are in common only with GRIP.

Table 4. Comparison of the features in tool support for software inspection

Features Tools	Document types	Meeting	Voting	Reading support	Detection responsibility	Data collection and measurement	Defect classification	Enabling infrastructure
ICICLE (Brothers *et al.* 1990)	Source code	F2F	No	Ad hoc	Identical	Yes	Yes	Client-server
InspeQ (Knight and Myers 1993)	Any text document	F2F	No	Checklists	Identical	No	No	Client-server
CSI (Mashayekhi *et al.* 1993)	Any document	Sync.	No	Checklists	Identical	Yes	Yes	Client-server
Scrutiny (Gintell *et al.* 1993)	Any text document	Sync.	Yes	Ad hoc	Identical	Yes	Yes	Client-server
ASSIST (MacDonald and Miller 1997)	Any document	Async./ sync.	Yes	Checklists	Identical	Yes	Yes	Client-server
GRIP (Halling *et al.* 2001)	Any document	Sync.	Yes	Checklists/scenarios	Identical/distinct	Yes	Yes	GSS-based
GSS at Baan (van Genuchten *et al.* 2001)	Any document	Sync.	Yes	Ad hoc	Identical	No	No	GSS-based
CSRS (Johnson 1994)	Any text document	No	Yes	Checklists	Identical	Yes	Yes	Client-server
CAIS (Mashayekhi *et al.* 1994)	Any text document	Sync.	Yes	Checklists	Identical	No	Yes	Client-server
AISA (Stein *et al.* 1997)	Any document	Async.	yes	Ad hoc	Identical	No	Yes	Web-based
WiP (Harjumaa and Tervonen 1998)	Any text document	No	No	Checklists	Identical	No	Yes	Web-based
WiT (Harjumaa and Tervonen 2000)	Any text document	Async. (optional)	No	Checklists	Identical	No	Yes	Web-based
XATI (Hedberg H. and Harjumaa 2002)	Any html document	No	No	Checklists	Identical	Yes	Yes	Web-based
HyperCode (Perry *et al.* 2002)	Source code	No	No	Ad hoc	Identical	No	No	Web-based
IBIS (Lanubile and Mallardo 2003)	Any document	Async. (optional)	Yes	Checklists/scenarios	Identical/distinct	Yes	Yes	Web-based

6. DISCUSSION

Software inspections have been adopted from many years by large industrial organizations because of their impact on product quality and cost of nonquality. First generation of computer support for software inspection involved computers being used as replacements of clerical and labor-intensive tasks to increase inspection effectiveness and reduce cost of inspection. The ever-growing phenomenon of global software development has posed a new challenge to computer support software inspection: how to bridge the gap in physical distance among inspection team members.

In the previous section, we have presented various attempts, which have exploited Internet technology to let inspections be conducted online. One major approach has been to provide software inspection tools that incorporate synchronous meeting support systems as an alternative to face-to-face meetings. This approach (different-place, same-time) still preserves the classical inspection process, which is built around a structured encounter of technical people. However, the blocking effects inherent in synchronous meetings, due to the divergent schedules of participants, augments the idle time and introduces delays that negatively affect the inspection interval time (Perry *et al*. 2002), and then the interval time of the overall software development process. For this reason, asynchronous mechanisms are recommended for the software inspection process while meetings are considered as 'the phase of last resort' (Johnson 1998).

Idle premeeting time can still be longer for global software inspections when time zones differ between team members. Time zone disparity between distant sites can provide few or no overlapping work hours for synchronous activities such as meetings. Even a time difference of a few hours can be a challenge in coordinating personal schedules because of dissimilar time slots for lunch and different national holidays.

Another problem that hinders synchronous communication in global software development is language differences. English is the common cross-border language for software business. However, nonnative English-speaking people can find it difficult to understand, write or speak at the pace imposed by real-time communication with English-native partners.

7. CONCLUSIONS

In this paper, we have presented IBIS, a web-based support tool for geographically distributed inspection. On the basis of lessons learned from empirical studies of software inspections, IBIS follows a different-place, different-time approach, with a prevalence of asynchronous communication among inspectors. IBIS adopts a reengineered inspection process to minimize coordination problems, without loosing the advantages of inspections (phased process, reading techniques, roles, measurement) over less formal review processes. The conformance to a specific but flexible redesigned inspection process, and the support for different reading techniques in the individual defect discovery activity, make the IBIS different from other asynchronous web-mediated inspection tools.

We can identify three lessons that we have drawn from our experiences in running distributed inspections through the IBIS tool. First we have learnt that, having removed the need for the inspection team to meet in the same place and in the same time, the usual team size limit of five to seven members can be exceeded without incurring an unbearable overhead. Second, we have learnt how to optimize the redesigned inspection process by focusing the discussion on those defects that have been reported by only one reviewer, and restricting the number of discussants to a subset of the whole inspection team. Finally, we have learnt that asynchronous discussions can be as effective as F2F meetings when discriminating between true defects and false positives.

Further work will be to empirically assess the effects of synchronous and asynchronous-based approaches on inspection effectiveness, cost and interval time. To pursue these investigations further will require both controlled experiments and field studies, and the availability of a global software development context, with differences in distance, time zone, and national culture.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful improvement suggestions. We are also grateful to the students at the University of Bari who performed distributed inspections and provided valuable feedback to enhance the tool.

REFERENCES

Adler S, Berglund A, Caruso J, Deach S, Graham T, Grosso P, Gutentag E, Milowski A, Parnell S, Richman J, Zilles S. 2001. Extensible Stylesheet Language (XSL) 1.0, W3C Recommendation, World Wide Web Consortium. http://www.w3.org/TR/xsl/.

Basili VR, Green S, Laitenberger O, Lanubile F, Shull F, Sorumgard S, Zelkowitz M. 1996. The empirical investigation of perspective-based reading. *Empirical Software Engineering* **1**(2): 133–164.

Bianchi A, Lanubile F, Visaggio G. 2001. A controlled experiment to assess the effectiveness of inspection meetings. *Proceedings of 7th METRICS* (London, England, April, 2001). IEEE Computer Society Press: Los Alamitos, CA, 42–50.

Bray T, Paoli J, Sperberg-McQueen CM, Maler E. 2000. Extensible Markup Language (XML) 2.0, W3C Recommendation, World Wide Web Consortium. http://www.w3.org/TR/REC-xml.

Brothers L, Sembugamoorthy V, Muller M. 1990. ICICLE: groupware for code inspection. Proceedings of Conference on Computer Supported Cooperative Work, Los Angeles, CA, October 1990, 169–181.

Carmel E. 1999.. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall PTR: San Francisco, CA.

Ciolkowksi M, Differding C, Laitenberger O, Munch J. 1997. Empirical Investigation of Perspective-based Reading: A Replicated Experiment. ISERN Report 97-13.

Clark J. 1999. Extensible Stylesheet Language Transformations (XSLT) 1.0, W3C Recommendation, World Wide Web Consortium. http://www.w3.org/TR/xslt.

Ebenau RG, Strauss SH. 1994. *Software Inspection Process*. McGraw Hill, NewYork, USA.

Ebert C, Parro CH, Suttels R, Kolarczyk H. 2001. Improving validation activities in a global software development. *Proceedings of ICSE* (Toronto, Canada, May 2001). IEEE Computer Society Press: Los Alamitos, CA, 545–554.

Fagan ME. 1976. Design and code inspections to reduce errors in program development. *IBM Systems Journal* **15**(3): 182–211.

Freedman DP, Weinberg GM. 1990. *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Dorset House Publishing, NewYork, USA.

Fusaro P, Lanubile F, Visaggio G. 1997. A replicated experiment to assess requirements inspection techniques. *Empirical Software Engineering* **2**: 39–57.

Gilb T, Graham D. 1993. *Software Inspection*. Addison-Wesley, Reading, Massachusetts, USA.

Gintell J, Arnold J, Houde M, Kruszelnicki J, McKenney R, Memmi G. 1993. Scrutiny: a collaborative inspection and review system. Proceedings of 4th European Software Engineering Conference, Garmisch-Partenkirchen, Germany, September 1993.

Grunbacher P, Halling M, Biffl S. 2003. An empirical study on groupware support for software inspection meetings. *Proceedings of 18th Int. Conference on Automated Software Engineering* (Montreal, Canada, October 2003). IEEE Computer Society Press: Los Alamitos, CA, 4–11.

Halling M, Biffl S, Grunbacher P. 2003. An experiment family to investigate the defect detection effect of tool-support for requirements inspection. *Proceedings of 9th Int. Software Metrics Symposium* (Sydney, Australia, September 2003). IEEE Computer Society Press: Los Alamitos, CA, 278–285.

Halling M, Grunbacher P, Biffl S. 2001. Tailoring a COTS group support system for software requirements inspection. *Proceedings of 16th Int. Conference on Automated Software Engineering* (San Diego, CA, November 2001). IEEE Computer Society Press: Los Alamitos, CA, 201–210.

Harjumaa L, Tervonen I. 1998. A WWW-based tool for software inspection. *Proceedings of 31st HICSS Conference* (Hawaii, HI, January 1998), Vol. III. IEEE Computer Society Press: Los Alamitos, CA, 379–388.

Harjumaa L, Tervonen I. 2000. Virtual software inspections over the internet. Proceedings of 3rd ICSE Workshop on Software Engineering over the Internet, Limerick, Ireland, June 2000.

Hedberg H, Harjumaa L. 2002. Virtual software inspections for distributed software engineering projects. Proceedings of ICSE International Workshop on Global Software Development, Orlando, FL, May 2002.

Herbsleb JD, Mockus A, Finholt TA, Grinter RE. 2001. An empirical study of global software development: Distance and speed. *Proceedings of ICSE* (Toronto, Ontario, May 2001). IEEE Computer Society Press: Los Alamitos, CA, 81–90.

Herbsleb JD, Moitra D. 2001. Global software development. *IEEE Software* **18**(2): 16–20.

Johnson PM. 1994. An instrumented approach to improving software quality through formal technical review. *Proceedings of 16th International Conference on Software Engineering* (Sorrento, Italy, May 1994). IEEE Computer Society Press: Los Alamitos, CA, 113–122.

Johnson PM. 1998. Reengineering inspection. *Communication of the ACM* **41**(2): 49–52.

Johnson PM, Tjahjono D. 1998. Does every inspection really need a meeting? *Empirical Software Engineering* **3**: 9–35.

Knight JC, Myers EA. 1993. An improved inspection technique. *Communications of the ACM* **36**(11): 51–61.

Laitenberger O, DeBaud JM. 2000. An encompassing life cycle centric survey of software inspection. *The Journal of Systems and Software*. **50**(5-31).

Land LPW, Jeffery R, Sauer C. 1997. Validating the defect detection performance advantage of group designs for software reviews: report of a replicated experiment. Caesar Technical Report 97/2, University of New South Wales, Sydney, Australia.

Lanubile F, Mallardo T. 2002. Preliminary evaluation of tool-based support for distributed inspection. Proceedings of ICSE International Workshop on Global Software Development, Orlando, FL, May 2002.

Lanubile F, Mallardo T. 2003. An empirical study of web-based inspection meetings. *Proceedings of 2nd International Symposium on Empirical Software Engineering* (Roma, Italy, October 2003). IEEE Computer Society Press: Los Alamitos, CA, 244–251.

MacDonald F, Miller J. 1997. A Software inspection process definition language and prototype support tool. *Software Testing, Verification, and Reliability* **7**(2): 99–128.

MacDonald F, Miller J. 1999. A comparison of computer support systems for software inspection. *Automated Software Engineering* **6**: 291–313.

Mashayekhi V, Drake JM, Tsai W-T, Riedl J. 1993. Distributed, collaborative software inspection. *IEEE Software* **10**(5): 66–75.

Mashayekhi V, Feulner C, Reidl J. 1994. CAIS: collaborative asynchronous inspection of software. Proceedings of 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering, New Orleans, LA, December 1994.

Miller J, Wood M, Roper M. 1998. Further experiences with scenarios and checklists. *Empirical Software Engineering* **3**: 37–64.

Parnas DL, Weiss DM. 1987. Active design reviews: principles and practice. *Journal of Systems and Software* **7**: 259–265.

Perry DE, Porter A, Wade MW, Votta LG, Perpich J. 2002. Reducing inspection interval in large-scale software development. *IEEE Transaction on Software Engineering* **28**(7): 695–705.

Porter A, Votta LG. 1998. Comparing detection methods for software requirements specification: a replication using professional subjects. *Empirical Software Engineering* **3**: 355–379.

Porter A, Votta LG, Basili VR. 1995. Comparing detection methods for software requirements inspections: a replicated experiment. *IEEE Transactions on Software Engineering* **21**(6): 563–575.

P2PConference homepage. 2004. http://p2pconference.jxta.org (last accessed on 03/01/2004).

Sauer C, Jeffery DR, Land L, Yetton P. 2000. The effectiveness of software development technical reviews: a behaviorally motivated program of research. *IEEE Transactions on Software Engineering* **26**(1): 1–14.

Stein M, Riedl J, Harner SJ, Mashayekhi V. 1997. A case study of distributed, asynchronous software inspection. *Proceedings of 19th International Conference on Software Engineering* (Boston, MA, May 1997). IEEE Computer Society Press: Los Alamitos, CA, 107–117.

van Genuchten M, van Dijk C, Scholten H, Vogel D. 2001. Using group support systems for software inspections. *IEEE Software* **18**(3): 60–65.

Votta LG. 1993. Does every inspection need a meeting? *ACM Software Engineering Notes* **18**(5): 107–114.

Wiegers KE. 2001. *Peer Reviews in Software: A Practical Guide*. Addison-Wesley Reading, Massachusetts, USA.

Wilson BJ. 2002. JXTA. New Riders.

Wood L, Nicol G, Robie J, Byrne S, Le Hors A, Sutor R Apparao V, Champion M, Isaacs S, Jacobs I, Wilson C. 1998. Document Object Model (DOM) Level 1 Specification, W3C Recommendation, World Wide Web Consortium. http://www.w3.org/TR/REC-DOM-Level-1/.