INTEGRATING A DISTRIBUTED INSPECTION TOOL WITHIN AN ARTEFACT MANAGEMENT SYSTEM[1]

Andrea De Lucia, Fausto Fasano, Genoveffa Tortora
Dipartimento di Matematica e Informatica, University of Salerno, Via Ponte Don Melillo, Fisciano (SA), ITALY

Giuseppe Scanniello
Dipartimento di Matematica e Informatica, University of Basilicata, Viale Dell'Ateneo, Macchia Romana, Potenza, ITALY

Keywords: Software Inspection, Synchronous communication, Global Software Engineering, Process Support Systems.

Abstract: We propose a web based inspection tool addressing the problem of software inspection within a distributed development environment. This tool implements an inspection method that tries to minimise the synchronous collaboration among team members using an asynchronous discussion to resolve the conflicts before the traditional synchronous meeting. The tool also provides automatic merging and conflict highlighting functionalities to support the reviewers during the pre-meeting refinement phase. Information about the inspection progress, which can be a valuable support to make inspection process related decisions is also provided. The inspection tool has been integrated within an artefact management system, thus allowing the planning, scheduling, and enactment of the inspection within the development process and integrating the review phase within the overall artefact lifecycle.

1 INTRODUCTION

Software inspection is a widely accepted and effective practice to discover defects within software artefacts. The success of inspection is due to early defect detection, when the cost of defect removal is less (Macdonald and Miller, 1998).

In a distributed setting, as the global software engineering, geographical distance becomes a critical factor for the enactment of a traditional inspection process (Fagan, 1976). Traditional inspection practices consider the enactment of a synchronous meeting as essential, while more recent findings question its utility. Indeed, studies demonstrating that an asynchronous discussion can be as effective as a face-to-face meeting have been proposed (Johnson and Tjahjono, 1998) (Lanubile *et al*., 2003) (Macdonald and Miller, 1998) (Mashayekhi *et al*., 1994).

In this paper we present an inspection process and a Web-based Artefact Inspection Tool (WAIT), where a preliminary asynchronous discussion is performed after the preparation phase and before the synchronous meeting. Concerning the proposed process we modified the structured inspection process proposed by Fagan (1976) according to the results presented in Damian *et al.* (2006), Johnson and Tjahjono (1998), and Lanubile *et al.* (2003). In particular, we introduce the refinement phase to prune the conflicts and duplication arising from an automatic merge of the defect logs after the detection phase. The tool highlights the conflicts and guides the involved software engineers in a refinement of the merged defect log to be discussed during the meeting. The refinement phase aims at resolving all the conflicts in the defect log and at removing the majority of the false defects. Based on the result of this phase the moderator can decide whether to perform the synchronous meeting, force a further analysis of the individual defect logs, or conclude the inspection and send the defect log to the author for the Follow Up phase.

WAIT has been integrated within ADAMS (De Lucia *et al.*, 2004) (ADvanced Artefact Management

[1] This research has been supported by the project METAMORPHOS (MEthods and Tools for migrAting software systeMs towards web and service Oriented aRchitectures: exPerimental evaluation, usability, and technology tranSfer), funded by MiUR (Ministero dell'Università e della Ricerca) under grant PRIN-2006-2006098097.

De Lucia A., Fasano F., Tortora G. and Scanniello G. (2007).
INTEGRATING A DISTRIBUTED INSPECTION TOOL WITHIN AN ARTEFACT MANAGEMENT SYSTEM.
In *Proceedings of the Second International Conference on Software and Data Technologies - SE*, pages 184-189
DOI: 10.5220/0001332901840189

System), an artefact-based process support system for the management of human resources, projects, and software artefacts. The integration of the inspection process tool within ADAMS aims at integrating quality management functionalities within the software process support system, thus allowing the planning, scheduling, and enactment of the inspection within the development process and integrating the review phase within the overall lifecycle of software artefacts.

The remainder of the paper is organised as follows: Section 2 discusses the related work while Section 3 presents an overview of ADAMS. Section 4 illustrates the inspection process and tool while Section 5 concludes the paper.

2 RELATED WORK

The usefulness and acceptance of software inspection in academic and industrial settings, have contributed to the development of several tools supporting the inspection process. For instance, ICICLE (Brothers *et al.*, 1990) addresses the inspections of C and C++ code, making use of specific knowledge on the programming language to assist during the defect discovery phase. However, this approach does not results appropriated to review software artefacts of different types.

Knight and Meyer (1991) propose InspeQ (Inspecting software in phases to ensure Quality), a toolset implementing the phased inspection technique (Knight and Meyers, 1993). This technique aims at improving the rigour, efficiency, and repeatability of the inspection of software products by examining the artefacts in a series of small inspections termed phases. Each phase aims at ascertaining if the artefact possesses some desirable property. InspeQ does not support distributed meeting and decision support is not provided.

Scrutiny (Gintell *et al.*, 1993) is a collaborative and distributed system based on an inspection process similar to the Fagan model. In particular, Scrutiny adds the verifier role, to ensure that all the defects founded during the inspection are addressed by the author. However, Scrutiny does not provide support for checklist based inspections and the meeting support is only synchronous.

CSI (Mashayekhi, 1993) (Collaborative Software Inspection) adopts the Humphrey's inspection process (Humphrey, 1989). CSI supports annotations by creating hyperlinks between the document and the reviewers' annotations. The inspection manager reviews the annotations and makes them into one list, which is successively discussed during the meeting. The discussion phase is synchronous, while the decision support is not provided.

The Humphrey's process is also implemented in AISA (Stein *et al.*, 1997) (Asynchronous Inspection of Software Artifacts) to inspect graphical documents. AISA uses a web client to visualise documents that are prepared as clickable image maps. The reviewers annotate documents using the coordinates of the image portion clicked. The only support provided by the tool is the notification of inspection completion by means of a message sent to the inspection team when all reviewers have finished the collection phase. The drawback is that annotations are available to all the participants, thus influencing the inspection members. To overcome this drawback, InspectA (Murphy and Miller, 1997) replaces the web publishing approach with e-mail massages that are sent only when all the detection phases are completed. In InspectA, the moderator does not have any progress information of the detection phase. Thus, the only way to know that a reviewer has completed the inspection is the receiving of the email generated at the end of the phase. This does not support the moderator in case an inspector has not completed the inspection.

The tool CSRS (Johnson, 1994) (Collaborative Software Review System) supports different checklist based inspection processes by using a process modelling language. CSRS distinguishes between a private review phase, where individual review of the artefact is performed and annotations are hidden to the other inspectors, and a public review phase, which represents an asynchronous meeting. Differently from our approach, support for synchronous meeting to address unresolved conflicts is not provided. Even ASSIST (Macdonald and Miller, 1998) (Asynchronous/Synchronous Software Inspection Support Tool) is designed to support different inspection processes. To reduce the effort to inspect a software artefact, ASSIST provides a checklist browser implementing active checklists, which records answers, monitors the checklist usage, and visualises cross-references. Meeting support is provided by a whiteboard and video and audio tools. ASSIST also provides a facility to merge multiple lists of defects using their similarity.

Lanubile *et al.* (2003) propose a web-based tool, called IBIS (Internet-Based Inspection System), which implements a reengineered Fagan's inspection process. In particular, the adopted process replaces the preparation and meeting phases of the Fagan's process with three new sequential phases: discovery, collection, and discrimination. The last of these

phases can be skipped to save time and coordination overhead. IBIS presents several similarities with our approach. The main differences with our approach rely on the discrimination phase that is based on a forum with a voting mechanism. Moreover, we provide the moderator information to support him/her during the inspection process decisions.

None of these tools are integrated within a process support system. As a consequence they do not manage the inspection process within the software development process, providing functionalities to link the review phase to the software artefact life cycle and maintain and easily recover inspection data during software evolution.

3 ADAMS OVERVIEW

ADAMS is an artefact-based process support system. It enables the definition of a software development process in terms of the artefacts to be produced and the relations among them. To this aim, ADAMS emphasises the artefact life cycle by associating software engineers with the different operations they can perform on each artefact. This feature, together with the resource permissions definition and management, represents a first level of process support and allows the project manager to focus on practical problems involved in the process and to avoid getting lost in the complexity of process modelling. ADAMS also enables process management in a flexible way, giving managers the possibility of changing the state of an artefact, the associated resources and the related permissions.

ADAMS has a web-based architecture and includes several subsystems. The Artefact Management Subsystem is the core subsystem of ADAMS and provides functionalities to manage the artefacts of a project and concurrent development of software engineers. This subsystem is used to create, modify, and delete artefacts, as well as to manage the artefact lifecycle and versioning. The Resource Management Subsystem manages human resources that can be allocated on projects as well as on single software artefacts. The Traceability Management Subsystem enables the management of traceability links between related software artefacts, which can be also visualised and used for impact analysis during software evolution. The Cooperative Development Subsystem provides synchronous collaborative modelling functionalities, such as allowing distributed team members to discuss, model, and modify the same UML diagram. The Quality Management Subsystem is responsible for quality assurance within the software project, such as the inspection process management functionality described in this paper. Finally, the Event & Notification Management Subsystem is used by ADAMS to generate and deliver notifications concerning the project. To enhance the context awareness level, notifications are also propagated across the traceability graph, that is the graph having artefacts as nodes and traceability links as edges.

4 INSPECTION PROCESS AND TOOL

The inspection process proposed in this paper is shown in Figure 1. The phases *Overview*, *Refinement*, and *Inspection Meeting* are optional and are performed considering the software artefact and the aim of the inspection. The inspection process has been implemented in WAIT (Web Based Inspection Tool). In the following, we detail the phases of the proposed inspection process and describe an example of application of WAIT on a Java class.

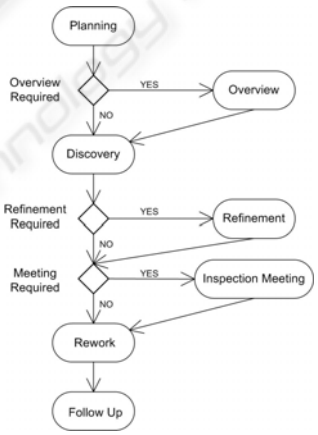

Figure 1: Reengineered Inspection Process.

4.1 Planning

During the Planning phase the quality manager specifies which artefact version must undergo a formal review process, defines or selects an existing checklist, and compose the inspection team. All these functionalities are accomplished by using the process support, human resources, and quality management functionalities of ADAMS. After this phase all the inspection participants receive a notification containing the details of the inspection and a new task appears in their to-do-lists.

In our example, the inspection moderator selects three members and creates the inspection team. The quality manager specifies three check items for the checklist to be used during the inspection of the Java

class. In particular, the checks address the use of naming conventions for the identifiers, the code indentation, and the identifier meaningfulness.

4.2 Overview

In the Overview phase, the artefact author explains the design and the logic of the software artefact to the inspectors. To this aim, he/she produces a document that briefly describes the purpose and the scope of the artefact to be inspected and then deploys it in ADAMS. The inspection mailing list managed by ADAMS is used to notify all the inspection participants. In our example, this results in a notification sent to the reviewers containing the document produced by the artefact's author, the schedule of the inspection, and information about the Java class to be inspected.

4.3 Discovery

During the Discovery phase, the inspectors analyse the artefact and take note of the candidate defects by highlighting all the cases where the artefact does not comply with the control checklist. WAIT supports the inspector during this phase by recording the identified defect, its severity, a comment, its location within the software artefact in terms of page and line numbers, or picture/table sequential number (see Figure 2).

Anytime in this phase, the moderator can visualise the inspector's defect log, the check items processed or not processed as well as a preview of the merged defect logs containing the inspection output produced by the inspection team. This information can be used to decide whether to stop the detection phase and start the next phase. However, even in case the moderator does not access this information, ADAMS notifies him/her by sending an event as soon as all the inspectors have completed the discovery phase.

In our example, the three inspectors fill in the defect logs by specifying each line in the Java class that does not respect a checklist item. In particular, regarding the first check item (i.e., respect of the naming conventions), the inspectors discover a different set of defects, while they agree on three defects for the second check item (i.e., the code indentation). None of the inspectors discovers defects for the third check item (i.e., meaningfulness of the identifiers). This situation is available to the inspection moderator. In fact, the tool notifies him/her about the conflict for the first check item and reports the number of discovered defects for the remaining checks.

4.4 Refinement

When the detection phase is completed, the inspection moderator accesses the defect log, containing all the defects identified by the inspectors. In case the inspectors do not agree on the defects for a checklist item, WAIT highlights it to the moderator (see Figure 3), who is allowed to decide if the refinement phase must be enacted. In this case, the tool sends an email containing the conflict list to the inspection members. This email also aims at notifying that the conflicts can be analysed in order to get an agreement. The main goal of this phase is to remove false defects and to build consensus on the true defects. As suggested by the empirical study presented by Lanubile *et al.* (2003) we consider a defect as a true defect when at least two reviewers recognise it. In this case WAIT does not highlight the defect. However, the minimum number of reviews required to automatically get an agreement can be customised according to the inspection process constraints.

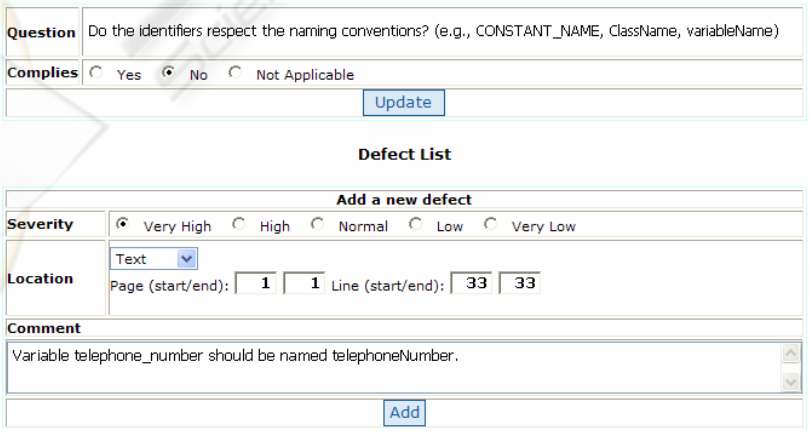

Figure 2: Defect identification.

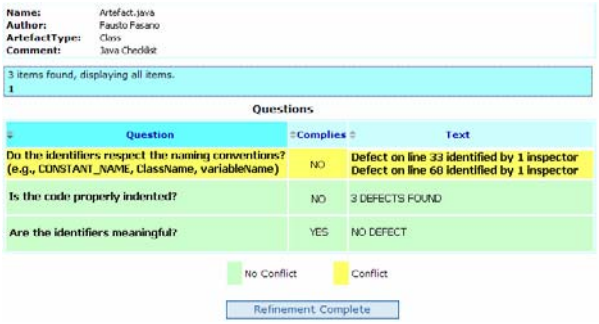

Figure 3: Merged defect list (inspector's view).

During the Refinement phase, the inspector accesses the merged defect list and selects one of the defects that caused the conflict. To assist the inspector during this phase, WAIT highlights the conflicts using a different colour for the conflicts, and provides hypertextual links to the defect details. By accessing the defect details the inspector decides whether it is an actual defect or a false positive. To this end, the inspector may decide to further analyse the considered software artefact. Note that the merged defect list is shared among the members of the inspection team. Hence, when a conflict is solved by an inspector, it is removed even from the lists of the remaining inspectors. When all the conflicts for a checklist item are solved, the corresponding highlighting is removed too. This phase is not mandatory, so the moderator can decide to skip it and directly resolves the conflicts on the identified defects (e.g., low number of defects).

In our example, the inspectors focus on the first checklist item (naming conventions). However, despite agreeing on most of the defects, two conflicts are not solved. To this aim, the inspection moderator decides to schedule a synchronous meeting to resolve the conflict.

4.5 Inspection Meeting

The synchronous inspection meeting component is implemented as a Java Applet and is eventually used to discuss on unsolved conflicts. When an inspector accesses the inspection meeting tool (see Figure 4), the result of the inspection process of each inspector is shown as well as the moderator decision. Inspectors can access the defect logs of the other inspectors grouped by checklist item. However, only the column regarding the logged inspector is editable. The checkbox in the inspector's column is used to indicate whether the artefact is compliant with the checklist item. The tool automatically deselects this checkbox when defects are discovered. Communication among the inspectors is further supported by a chat. Note that the moderator has also the possibility to fill in his/her own checklist and conclude the inspection process specifying whether the artefact can be baselined or not. The revision passes when all the checklist items are checked. Whether an agreement is not reached the moderator response is considered. This phase can be skipped in case the number of conflicts is directly manageable by the moderator or the time distance does not permit a meeting.

In our example, the three inspectors access the meeting and discuss on the two unsolved conflicts, classifying them as true defects.

4.6 Rework and Follow up

In case defects have been identified, the author has to fix them during the Rework phase. Once the defects have been addressed, the author creates a new version of the artefact that is validated by the moderator during the Follow Up phase. As a result a new baseline of the artefact can be created or a further re-inspection can be required. It is worth noting that the system maintains the defect logs for each of the artefact version. Thus, in case the artefact undergoes several inspections, it is possible to access the defect logs for each artefact version.

5 FINAL REMARKS

In this paper we have presented an inspection process whose aim is to minimise synchronous collaboration among geographical dispersed reviewers during the identification of defects in software artefacts. To this end, we modified the Fagan's method with an asynchronous discussion before performing a meeting. The asynchronous discussion aims at removing false defects resolving conflicts on the identified defects. The process has been then implemented in a Web-based Artefact Inspection Tool (WAIT), which has been integrated within a process support system, namely ADAMS (De Lucia *et al.*, 2004).

In order to assess the usefulness of the proposed tool and process, we carried out a controlled experiment (details can be found at http://www.scienzemfn.unisa.it/scanniello/CExp_2/. The experiment aimed at comparing the proposed inspection process and the Fagan's method in terms of time required to accomplish the inspection process as well as the number of identified true and false defects. The experiment revealed that the synchronous meeting is generally not necessary as the number of unsolved conflicts at the end of the Refinement phase is very low and can be generally managed by the inspection moderator.

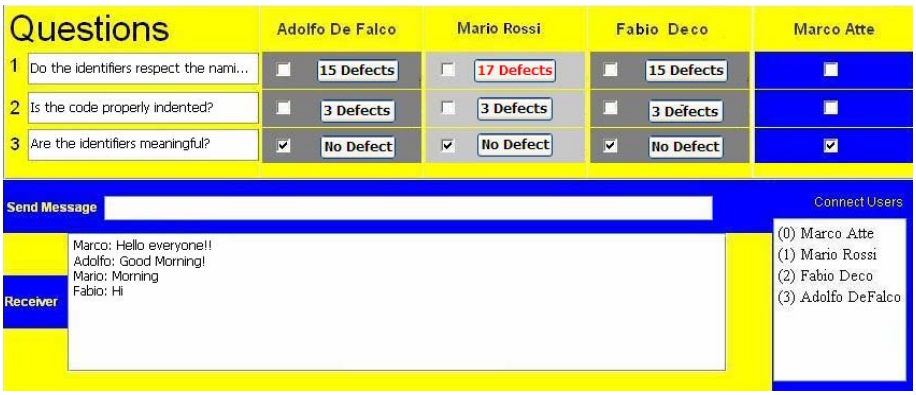

Figure 4: Synchronous meeting.

We also observed that reviewers adopting our process and tool spent less time to inspect a software artefact with respect to the Fagan's method. Moreover, the inspection teams were able to identify more true defects and more false positives when our inspection process and tool were adopted. A larger number of false defects might be acceptable in a distributed setting where the effort required to discard false positives can be repaid by the possibility to avoid a meeting.

Future work will be devoted to add features to further simplify the defect localisation within the software artefact. A second direction should aim at adding new features to better support optional synchronous meetings. Moreover, we are investigating the impact of face-to-face versus tool mediated synchronous meeting with respect to the number of true defects and false positive. This can be useful in case time distance is not an issue, but moving people might be a problem.

REFERENCES

Brothers L.R., Sembugamoorthy V., and Muller M., 1990. "ICICLE: Groupware for code inspections", *Proc. of ACM Conf. on Computer Supported Cooperative Work*, Los Angeles, CA, USA, pp.169-181.

Damian D., Lanubile F., and Mallardo T., 2006. "An Empirical Study of the Impact of Asynchronous Discussions on Remote Synchronous Requirements Meetings", *Lecture Notes in Computer Science*, vol. 3922, pp.155-169.

De Lucia A., Fasano F., Francese R., and Tortora G., "ADAMS: an Artefact-based Process Support System", *Proc. of 16th International Conference on Software Engineering and Knowledge Engineering*, Banff, Alberta, Canada, 2004, pp. 31-36

Fagan M.E., 1976. "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, vol. 15, no. 3, pp.182-211.

Gintell J.W., Arnold J., Houde M., Kruszelnicki J., McKenney R., and Memmi G., 1993. "Scrutiny: a collaborative inspection and review system", *Proc. of the 4th European Conf. on Software Engineering*, pp.344-360.

Humphrey W.S., 1989. "Managing the Software Process", *SEI Series In Software Engineering*, Addison-Wesley Longman Publishing, Boston, MA, USA.

Johnson P.M. and Tjahjono D., 1998. "Does Every Inspection Really Need a Meeting?", *Empirical Software Engineering*, vol.3, no.1, pp.9-35

Knight J.C., and Meyers E.A., 1991. "Phased inspections and their implementation", *Software Engineering Notes*, vol. 16, no.3, pp.29-35.

Knight J.C. and Meyers E.A., 1993. "An improved inspection technique", *Communications of the ACM*, vol.36, no.11, 1993, pp.51-61.

Lanubile F., Mallardo T., and Calefato F., 2003. "Tool Support for Geographically Dispersed Inspection Teams", *Software Process: Improvement and Practice*, vol.8, no.4, Wiley InterScience, pp.217-231.

Macdonald F. and Miller J., 1998. "A Comparison of Tool-Based and Paper-Based Software Inspection", *Empirical Software Engineering*, vol.3, no.3, pp.233-253

Mashayekhi V., Drake J.M., Tsai W.T., and Reidl J., 1993. "Distributed, collaborative software inspection", *IEEE Software*, vol.10, no.5, pp.66-75.

Mashayekhi V., Feulner C., and Reidl J., 1994. "CAIS: collaborative asynchronous inspection of software", *Proc. of the 2nd ACM Symposium on the Foundations of Software Engineering*, ACM Press, pp. 21-34.

Murphy P. and Miller J., 1997. "A process for asynchronous software inspection", *Proc. of the 8th Int. Workshop on Software Technology and Engineering Practice*, London, UK, pp.96-104.

Stein M., Riedl J., Harner S.J., and Mashayekhi V., 1997. "A case study of distributed, asynchronous software inspection", *Proc. of the 19th Int. Conf. on Software Engineering*, Boston, MA, USA, pp.107-117.