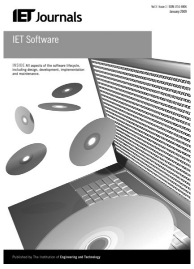
www.ietdl.org

Published in IET Software
Received on 6th September 2010
Revised on 4th July 2011
doi: 10.1049/iet-sen.2010.0108



ISSN 1751-8806

Improving artefact quality management in advanced artefact management system with distributed inspection

A. De Lucia[1] *F. Fasano*[2] *G. Scanniello*[3] *G. Tortora*[1]

[1]*Dipartimento di Matematica e Informatica, University of Salerno, Via Ponte Don Melillo, Fisciano, 84084 SA, Italy*
[2]*Dipartimento di Scienze e Tecnologie per l'Ambiente e il Territorio, University of Molise, Pesche, Italy*
[3]*Dipartimento di Matematica e Informatica, University of Basilicata, Viale Dell'Ateneo, Macchia Romana, Potenza 85100, Italy*
E-mail: giuseppe.scanniello@unibar.it

Abstract: Advanced artefact management system (ADAMS) is an artefact-based process support system for the management of human resources, projects and software artefacts. This system puts great emphasis on the artefact life cycle by associating software engineers with the different operations that can be performed on a given artefact. Managing the quality of software artefacts is considered as one of the main issues. To this end, ADAMS integrates web-based artefact inspection tool (WAIT), a web-based system implementing a distributed inspection process. A case study has been accomplished to evaluate both the integration of WAIT in ADAMS and the provided quality management support. The main result of this empirical investigation is that the integrated system provides an effective support for the management of the quality of software artefacts.

1 Introduction

Many software companies are moving their business to Q1 distributed virtual organisation models [1–3]. However, such a globalisation creates software engineering challenges (e.g. cooperation and collaboration for system design) due to the impact of time zones, distance or diversity of culture and communication. In global software development, a combination of traditional and novel methodologies and practices are required to overcome these challenges and to take advantage of the opportunities that such development entails [4–6].

A lot of research effort has been devoted to the development of methodologies and technologies supporting coordination and collaboration of distributed software engineering teams [7–9]. Typical examples are systems for the management and version control of software artefacts [10–13]. These systems help to coordinate the activities of developers by providing capabilities that either avoid parallel development altogether or assist in resolving conflicts. These systems also enable software engineers to work on the same artefact either through a lock-based policy or concurrently, if branch versions are allowed. Some of these systems provide feature to deal with some of the most common problems faced by cooperative environments, such as context awareness. Nevertheless, these systems marginally support quality management and the inspection of software artefacts, in particular.

Software inspection is a software engineering practice aiming at identifying defects, reducing rework and producing high-quality software systems [14, 15]. This practice is not new. In fact, Michael Fagan in 1976 [16] proposed the first structured inspection process, where team members individually analyse the software artefact according to its purpose and scope as well as the inspection goals. Team members successively perform a face-to-face meeting to produce a log containing details on the identified defects. One of the open issues related to the Fagan's inspection process concerns to the enactment of a face-to-face meeting. Research findings question the usefulness of a meeting [17, 18], since it requires adequate skills and experience. In fact, to be effective a meeting requires adequate preparation, efficient moderation, readiness of the work product for review and cooperation among group members, besides the simultaneous attendance of all the team members [19]. In global software development, geographical distance becomes an augmenting factor for the costs of face-to-face meetings and the time distance can even create barriers to the enactment of distributed virtual meetings conducted using information and communication technologies (e.g. text-based chat, virtual blackboards, web platforms or virtual environments like Second Life). In order to overcome the issues related to enactment of inspection processes in distributed settings, asynchronous discussions could be adopted before a face-to-face or virtual synchronous meeting [20].

In this paper we present a geographically dispersed inspection process that modifies the Fagan method to encourage inspection members to perform a preliminary asynchronous discussion after a preparation phase and before an optional meeting. We have implemented this process in a web-based artefact inspection tool (WAIT), which has been integrated in advanced artefact management system (ADAMS), a web-based fine-grained artefact management system [21]. ADAMS integrates project management features, such as work-breakdown structure definition, resource allocation and schedule management, as well as artefact management features, such as artefact versioning, traceability management and artefact quality management. Originally, the quality management in ADAMS was limited to the possibility of associating each artefact type with a checklist, while the inspection process was externally managed. A more effective support for the quality management is provided in the system resulting from the integration of WAIT in ADAMS. In particular, the resulting system allows planning, scheduling and enactment of the artefact inspection process, thus integrating the review phase within the artefact life cycle and the baseline production process. Furthermore, the integrated system allows the software engineer to (i) trace the whole inspection process and effectively manage the different versions of the artefacts under inspection; (ii) associate the results of the inspection process to the specific artefact version; (iii) allow the software engineer to trace each change to the defect that originated it; (iv) support the inspection of any type of artefact managed by ADAMS, including high-level (e.g. analysis and design documents) and low-level (e.g. source code) artefacts.

To evaluate the effectiveness of integrating WAIT in ADAMS, we have conducted a case study. The context of this study constituted Master and Bachelor students in Computer Science at the University of Salerno, who used the integrated system as an infrastructure for the development of their projects and the inspection of the most important software artefacts. At the end of the project, the students were asked to fill in a questionnaire to get information about the perceived usefulness of the integration of WAIT in ADAMS and the quality management support, in particular. The main result of the case study indicated that the subjects appreciated the quality management support provided by the integrated system. The data analysis also revealed that a preliminary asynchronous discussion is considered very useful to identify the greater part of the actual defects, although the subjects also considered the meeting enactment as necessary.

This paper is an extension of the work presented in [22], where a preliminary description of WAIT is provided. In [23] controlled experiments have also been conducted to compare our distributed inspections process with Fagan's method. With respect to these previous papers, we provide here the following main new contributions:

A deeper description of the inspection process and of the integration of WAIT in ADAMS. We also provide here a running example to explain how the inspection process is supported by the proposed integrated environment.
A case study conducted with Master and Bachelor students in 12 projects over two academic years to empirically assess the validity of both the inspection process and the integrated environment.

The remainder of this paper is organised as follows: Section 2 discusses related work on distributed software inspection. An overview of ADAMS and its integration with WAIT and the quality management in ADAMS are presented in Sections 3 and 4, respectively. Section 5 describes the inspection process and a running example to explain how our system supports software engineers. The case study is presented in Section 6, while final remarks and future work conclude the paper.

2 Related work

The first inspection process was proposed by Michael Fagan [16], who defines software inspection as a formal, efficient and economical method to find errors in design and code. This method is based on a formal and structured process, composed of five sequential phases: (i) overview, (ii) preparation, (iii) inspection, (iv) rework and (v) follow-up. In the overview phase the artefact author first describes the overall domain area of the software artefact and then provides details about it. Documentation concerning the software artefact to inspect is distributed to all the participants of the inspection team in the preparation phase. During the inspection phase a meeting is carried out to identify defects. In order to address the defects identified in the rework phase, the moderator should produce a written report that is provided to the author of the artefact. In the follow-up phase the moderator checks the quality of the rework and determines whether a re-inspection is required.

One of the open discussions regarding the Fagan's inspection process refers to the meeting. Traditional inspection practices consider the enactment of a meeting as essential, while some researchers question its usefulness [17, 18, 24]. For example, Votta [24] asserts that meetings may be useless and resource-consuming. However, he also highlights that a meeting is more appropriate to remove most of the false defects. A false defect is a defect identified by an inspector that is not an actual defect. Q2

Porter and Johnson [25], Miller *et al.* [26] and Sabaliauskaite *et al.* [27] confirm that meetings do not improve defect finding process, and recommend replacing meetings with some other practices, for example, asynchronous discussions [19] or nominal teams [28]. In particular, Johnson and Tjahjono [19] present a controlled experiment in which they demonstrate that the cost of a meeting is higher than the cost of an asynchronous discussion. However, the effectiveness of the inspection meeting is still an open issue. On the other hand, Bianchi *et al.* [29] compares the effectiveness of real teams and nominal teams, that is, individual inspectors who do not communicate in a face-to-face meeting. The obtained results show that nominal teams outperform real teams. On the other hand, Biffl and Halling [30] analyse the costs and benefits of nominal inspection teams.

In global software engineering, geographical distance becomes an augmenting factor for the costs needed to perform code inspection and the time distance can even create barriers to the enactment of a meeting. To overcome these issues, a number of tools have been proposed for document tracking and inspection planning [31], comment preparation [32] and for both the individual preparation and the group meeting [33]. Meyer [34] proposes to run the design and code review entirely on the web and desktop-sharing solutions. This approach is similar to the online inspection tools, which specifically manage the entire inspection process online.

A number of online inspection tools have been proposed in the past, for example [35–37]. Among these tools several differences can be observed. For example, ICICLE [35] addresses the inspections of C and C++ code, making use of specific knowledge on the programming language to assist the discovery of defects. Knight and Meyer [38, 39] propose an inspection technique that examines the artefacts in a series of small checklist-based inspection phases. This technique is implemented in the InspeQ (inspecting software in phases to ensure quality) toolset. Scrutiny [36] is a collaborative and distributed system for the inspection and review of textual software artefacts. It implements a process that is similar to the Fagan's process. Collaborative software inspection (CSI) [40] adopts the Humphrey's inspection model [41]. This tool supports both synchronous and asynchronous discussions. Decision support for the moderator is not provided, while process awareness is provided by e-mail notifications. The asynchronous inspection of software artefacts (AISA) prototype [37] also implements the Humphrey's model. Furthermore, it addresses the problem of inspecting graphical artefacts without adopting a checklist. A web-based client is used to visualise documents that are prepared as clickable image maps. The approach can lead to a greater number of false defects as that annotations are made immediately available to all the participants. To overcome this drawback, InspectA is proposed in [42].

Collaborative software review system (CSRS) [43] is a flexible tool supporting different inspection processes. This is achieved by using a process modelling language for defining the process phases, the participant roles and the artefacts to inspect. Asynchronous/synchronous software inspection support tool (ASSIST) [44] like CSRS is designed to support any inspection process and any kind of software artefacts. To this aim, it uses an inspection process definition language. ASSIST also provides an auto-collation facility to merge multiple lists of issues or defects by using their similarity in terms of position, content and classification. E-mail notifications are used to support process awareness.

Yamashita [45] proposes Jupiter, an inspection support tool developed as an eclipse plug-in. This tool only supports asynchronous discussion among inspectors, addresses the inspection of source code only and does not support geographical dispersed reviews. Process awareness is implemented through e-mail notifications. Differently, Hedberg and Harjumaa [46] proposed to perform 'virtual software inspections' by implementing a new XML-capable annotation tool, XATI (XML annotation tool for inspection) that uses Mozilla as a generic application environment and to view the artefact under inspection.

Perpich *et al.* [18] presented a web-based tool, named Hypercode, to asynchronously support geographically distributed teams in the inspection of HTML documents. No support for synchronous and asynchronous discussions is provided to solve possible conflicts. Similarly, Lanubile *et al.* [17] propose a web-based tool, called IBIS (internet-based inspection system), that adopts a variation of the Fagan's inspection process. In particular, starting from the reorganisation of the inspection process proposed by Sauer *et al.* [47], the authors replace the preparation and meeting phases of the process proposed by Fagan with three new sequential phases: discovery, collection and discrimination.

None of the highlighted inspection tools are integrated within an artefact management system. As a consequence, they do not integrate an inspection process in the software artefact life cycle and do not provide functionalities to link the reviews to software artefact versions and maintain and easily recover inspection data during software evolution. On the contrary, our proposal provides a large number of advantages that mainly derive from the support that the used artefact management system (i.e. ADAMS) provides to the inspection process implemented in WAIT and the management of the inspection teams. These advantages can be summarised as follows:

- integrating the review phase within the artefact life cycle and the baseline production process;
- tracing the whole checklist-based inspection process;
- identifying defects within any kind of software artefact since ADAMS is able to manage both high (e.g. UML models) and low (e.g. source code) level artefacts;
- extending the to-do list with activities concerning the inspections to perform;
- providing asynchronous communication facilities between team members;
- including features for classifying the identified defects and for decision support;
- using e-mail notifications to enhance context-awareness within an inspection process.

A more detailed comparison between WAIT and the above-discussed tools can be found in [23].

3 Enhancing ADAMS with distributed inspection: an overview

ADAMS [21] is a web-based system that integrates project management features, such as work-breakdown structure definition, resource allocation and schedule management as well as artefact management features, such as artefact versioning, traceability management and artefact quality management. ADAMS has been implemented using J2EE technologies, in particular Java Server Pages (JSP) and Java Servlets; the web server is Apache Tomcat 6.0, while the database management system is MySql 5.0.

Fig. 1 shows a UML package diagram illustrating the subsystem decomposition of ADAMS. In the following, we provide a brief description of the main functionalities provided by each subsystem of ADAMS. Further details are available at the ADAMS website [http://adams.dmi.unisa.it/demo]. Fig. 1 also highlights how the integration of WAIT in ADAMS has impacted on different subsystems of ADAMS, in particular on the quality management subsystem, as described in Section 4.

The resource management subsystem provides administrative functionalities for human resource and account management, and allocation of resources on projects. ADAMS adopts a role-based access control policy. Human resources can be allocated with different roles on a project (e.g. project manager, quality manager, architect, tester) or on an artefact (e.g. developer, auditor).

The project management subsystem is responsible of managing project definition and scheduling. ADAMS supports a general and customisable software development process that is based on the artefacts to be produced and the relations among them. This allows the project manager to focus on practical problems involved in the process and avoids getting lost in the complexity of process modelling, like in workflow management systems. Functionalities are provided to define the schedule and allocation of human resources to software artefacts. The definition of the schedule is performed during the definition of the artefacts by specifying the starting and the due date for it. This information is used to determine the schedule status of each artefact. Indeed, when a software engineer accesses his/her

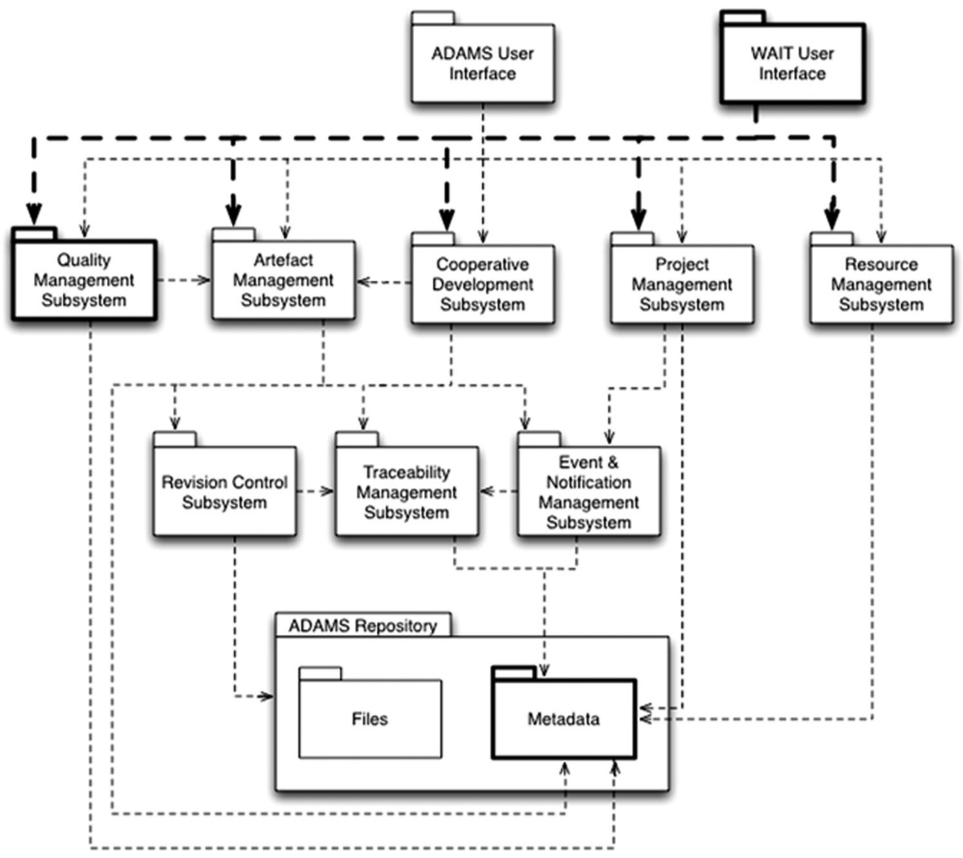
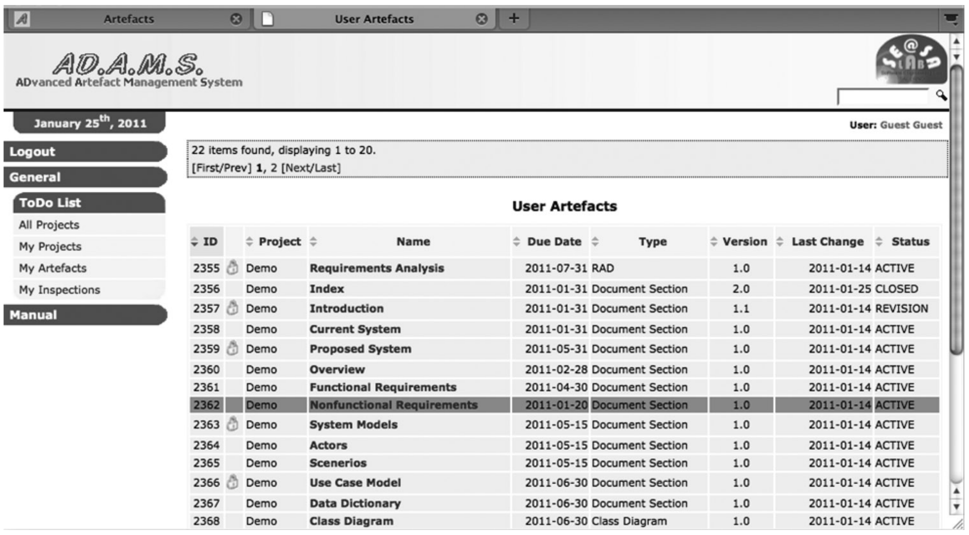

ADAMS User Interface
WAIT User Interface
Quality Management Subsystem
Artefact Management Subsystem
Cooperative Development Subsystem
Project Management Subsystem
Resource Management Subsystem
Revision Control Subsystem
Traceability Management Subsystem
Event & Notification Management Subsystem
ADAMS Repository
Files
Metadata

Fig. 1 *Integrated system architecture*

to-do list (containing the set of artefacts he/she has to work on), the system highlights late and overdue artefacts (see Fig. 2).

The artefact management subsystem is responsible for the management of the artefact lifecycle and the fine-grained management of software artefacts. Any intermediate product of the software process can be managed in ADAMS as an artefact. In fact, in ADAMS artefacts can either be atomic entities or they can be composed of an arbitrary number of atomic or further-composed artefacts. The fine-grained management of artefacts proposed in ADAMS allows the software engineer to choose the granularity level of this decomposition according to the artefact type, the concurrency level needed for the human resources allocated on them, the number of software engineers that are likely accessing them at the same time, the necessity to specify responsibilities for specific unit of work and the need for the definition of traceability links between them. Functionality for the automatic re-composition of decomposed artefacts is also provided. The hierarchical organisation of composite artefacts provides a product-oriented work-breakdown structure to the project manager as well as concurrent development functionality. Further details of the fine-grained management of software artefacts can be found in [21].



User: Guest Guest

22 items found, displaying 1 to 20.
[First/Prev] **1**, 2 [Next/Last]

User Artefacts

ID		Project	Name	Due Date	Type	Version	Last Change	Status
2355		Demo	**Requirements Analysis**	2011-07-31	RAD	1.0	2011-01-14	ACTIVE
2356		Demo	**Index**	2011-01-31	Document Section	2.0	2011-01-25	CLOSED
2357		Demo	**Introduction**	2011-01-31	Document Section	1.1	2011-01-14	REVISION
2358		Demo	**Current System**	2011-01-31	Document Section	1.0	2011-01-14	ACTIVE
2359		Demo	**Proposed System**	2011-05-31	Document Section	1.0	2011-01-14	ACTIVE
2360		Demo	**Overview**	2011-02-28	Document Section	1.0	2011-01-14	ACTIVE
2361		Demo	**Functional Requirements**	2011-04-30	Document Section	1.0	2011-01-14	ACTIVE
2362		Demo	**Nonfunctional Requirements**	2011-01-20	Document Section	1.0	2011-01-14	ACTIVE
2363		Demo	**System Models**	2011-05-15	Document Section	1.0	2011-01-14	ACTIVE
2364		Demo	**Actors**	2011-05-15	Document Section	1.0	2011-01-14	ACTIVE
2365		Demo	**Scenerios**	2011-05-15	Document Section	1.0	2011-01-14	ACTIVE
2366		Demo	**Use Case Model**	2011-06-30	Document Section	1.0	2011-01-14	ACTIVE
2367		Demo	**Data Dictionary**	2011-06-30	Document Section	1.0	2011-01-14	ACTIVE
2368		Demo	**Class Diagram**	2011-06-30	Class Diagram	1.0	2011-01-14	ACTIVE

Fig. 2 *Artefacts to-do list*

A revision control subsystem has been implemented to create, modify and delete artefacts, as well as to manage the artefact state and versions. However, in case there is a need of using an existing repository, the integration can be executed by replacing the revision control subsystem with the existing versioning management tool. In particular, it is possible to interface the revision control subsystem with a wrapper for other versioning systems, for example CVS [10] or subversion [48], as well as to implement additional features such as caching and delta versioning.

Traceability links between related artefacts can also be inserted and visualised through the traceability management subsystem and used for impact analysis during software evolution. In fact, traceability links can be visualised and browsed to look at the state of previously developed artefacts, to download latest artefact versions or to subscribe events on them and to receive notifications on the state of their development. An example of event could be the creation of a new version for a software artefact. Event subscription and notifications are managed by the event and notification management subsystem, which is used to enhance context-awareness within the software project [21].

The quality management subsystem is the subsystem that is more impacted by the integration of WAIT in ADAMS. This subsystem provides functionalities to manage the quality within a project. This entails the definition of the artefact types managed by the system and the standard templates and review checklists associated with them. More details are provided in Section 4.

Finally, the cooperative development subsystem includes tools used to enhance synchronous collaboration within ADAMS, such as an internal chat and an UML collaborative editor that enables developers to access and modify the same diagram concurrently [49], and asynchronous collaboration tools, for example, the internal e-mail, comments and feedbacks that can be attached to a specific artefact (or to a specific artefact version), and the rationale management tool that enables the software engineers to address open issues that need to be investigated by the team, using an argumentation-based approach.

WAIT mainly relies on three of these subsystems, namely the project management, resource management, and artefact management subsystems, as it uses many of the above described functionalities. Moreover, WAIT extends the quality management subsystems and the metadata. In particular, the quality management subsystem has been extended with the distributed inspection functionality, whereas the metadata has been enriched with inspection process and defect information.

4 Artefact lifecycle and quality management in ADAMS

The integration of WAIT in ADAMS entailed the extension of the artefact lifecycle. Regarding the artefact lifecycle, a software artefact can assume different states as shown by the UML statechart diagram of Fig. 3. The status of the artefact within its lifecycle is shown on the artefact details view as well as on the to-do list view (see the right-hand side of Fig. 2).

The scheduled state is the initial state of a newly created artefact. As shown in Fig. 4, in this state a number of information can be defined, in particular the start and end date of the artefact, whether branches are allowed to enable software engineers to work concurrently on the artefact, and whether the artefact is subject to inspection, and in this case whether the inspection has to be externally managed (this was the inspection functionality provided by ADAM before the integration of WAIT) or the integrated inspection functionality provided by WAIT has to be used. In both cases, the quality manager can associate a checklist to the artefact.

The ‘external’ inspection process allows the quality manager to associate an artefact with a checklist template file that can be later downloaded and used for the artefact inspection. In case the ‘integrated’ inspection process has been selected for the artefact, an existing web-based checklist can be used or a new one can be created (see Fig. 5). Checklists can be defined for each artefact type at the organisation level (template checklists) and can be

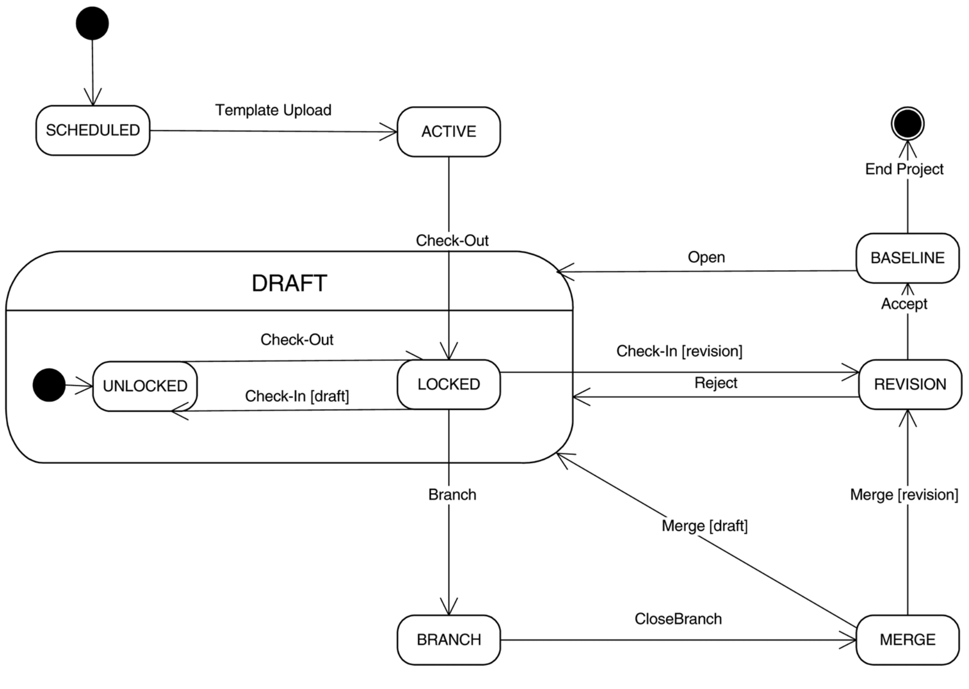
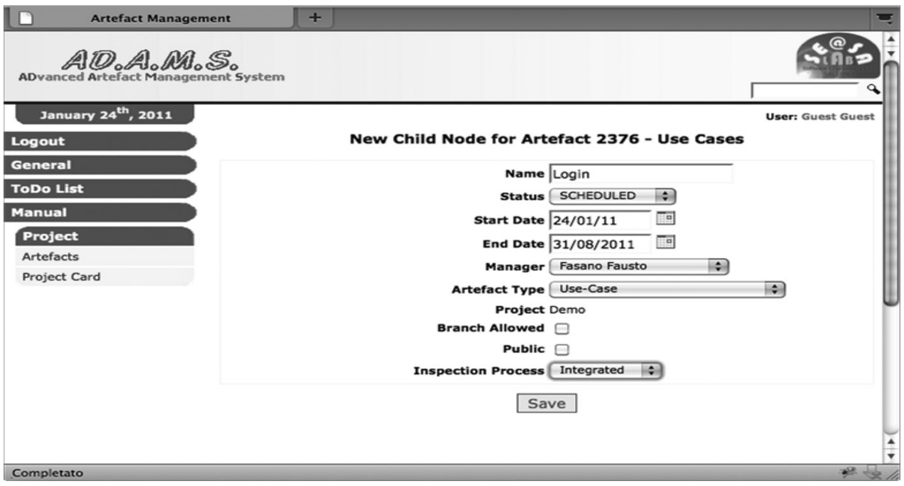


Fig. 3 *Artefact lifecycle in ADAMS*

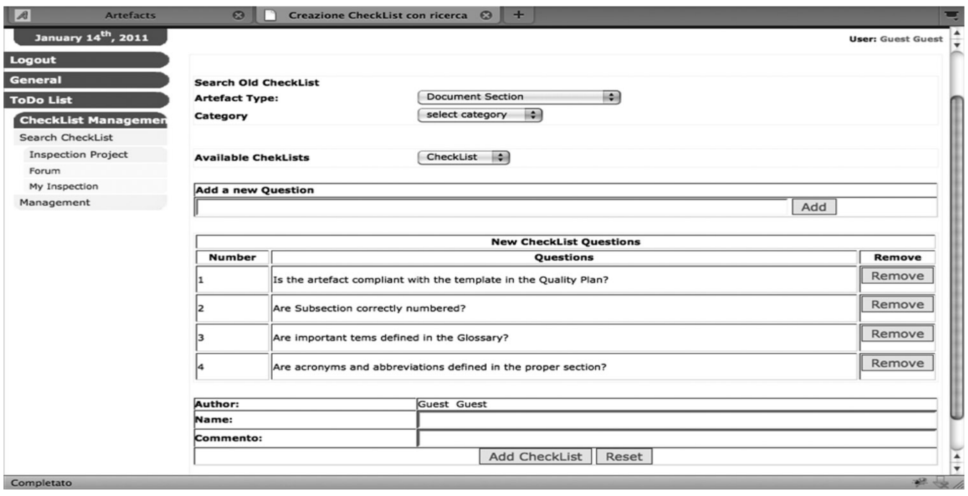


User: Guest Guest

New Child Node for Artefact 2376 - Use Cases

Name	Login
Status	SCHEDULED
Start Date	24/01/11
End Date	31/08/2011
Manager	Fasano Fausto
Artefact Type	Use-Case
Project	Demo
Branch Allowed	☐
Public	☐
Inspection Process	Integrated

Fig. 4 *Artefact creation*



User: Guest Guest

Search Old CheckList

Artefact Type:	Document Section
Category	select category
Available ChekLists	CheckList

Add a new Question

New CheckList Questions		
Number	Questions	Remove
1	Is the artefact compliant with the template in the Quality Plan?	Remove
2	Are Subsection correctly numbered?	Remove
3	Are important tems defined in the Glossary?	Remove
4	Are acronyms and abbreviations defined in the proper section?	Remove

Author:	Guest Guest
Name:	
Commento:	

Fig. 5 *New checklist definition*

customised at the project level or for a specific artefact. When a checklist has to be associated to an artefact subject to inspection, all available checklists already defined for the corresponding artefact type are proposed by the system. Then, the quality manager can decide to use or customise one of them or even to create a new checklist, by defining all the check-items (see Fig. 5). It is worth noting that the selection/definition of the checklist for an artefact to be inspected is one of the activities of the planning phase of the inspection process described in Section 5.1, that also includes other activities, such as the definition of the inspection team and inspection schedule.

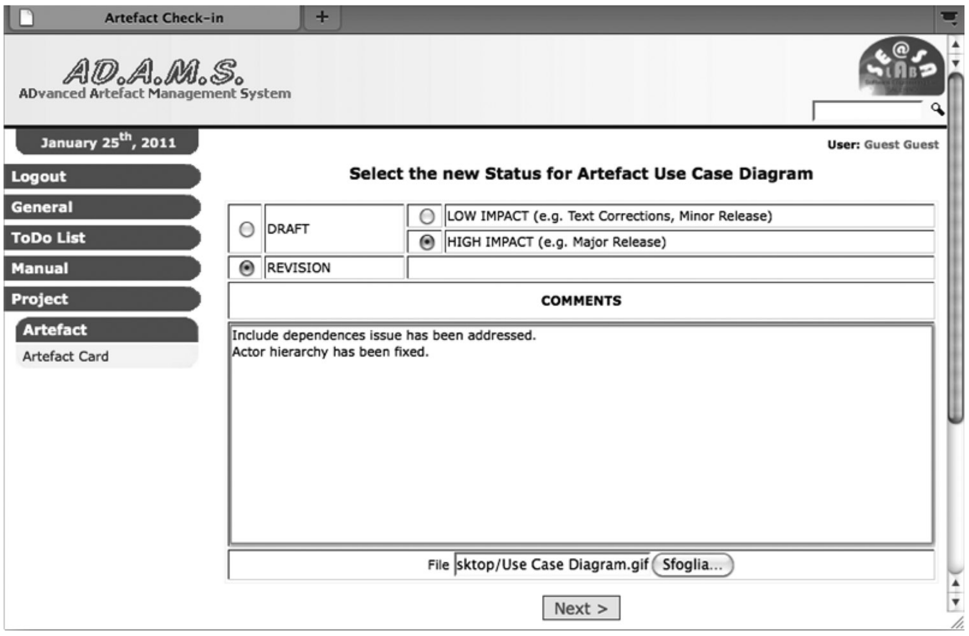
Once the artefact is activated (active state in Fig. 3), several draft versions can be created and maintained by ADAMS. An artefact that is in the draft state can be either in the locked state or in the unlocked state. Each time a new version is produced, the software engineer can tag the artefact as draft, in case he/she still needs to work on the artefact, or as revision, in case his/her work is completed and he/she wants to submit the artefact to an inspection (see Fig. 6). In the latter case, the system notifies the artefact manager and all members of the inspection team (see Section 5.1) by using the event notification mechanism of ADAMS (see Fig. 7).

Note that software engineers can also work concurrently creating different branches and working independently on them. Once an artefact is in the branch state, software engineers cannot submit the artefact to an inspection until the work on all the branches is completed and a merge operation is performed (the artefact is in the merge state).

When an artefact is in the revision state, the inspection process described in Section 5 starts and at the end of the process the artefact is either approved and closed (baseline) or sent back to the draft state, in case some rework is needed. WAIT relies on the event notification mechanism of ADAMS to keep the artefact manager, as well as all the members of the inspection team, aware. Baselines can be reworked only after the artefact manager has explicitly and formally reopened the artefact, for example, due to an accepted change request: this causes the transitions of the artefact back to the draft state.

5 Distributed inspection process

The distributed inspection process implemented in WAIT modifies the Fagan's method according to the findings discussed by Damian *et al.* [20]. As illustrated in Fig. 8, the process is composed of seven subsequent phases, namely planning, overview, discovery and detection, refinement, inspection meeting, rework and follow-up. The phases overview, refinement and inspection meeting are



User: Guest Guest

Select the new Status for Artefact Use Case Diagram

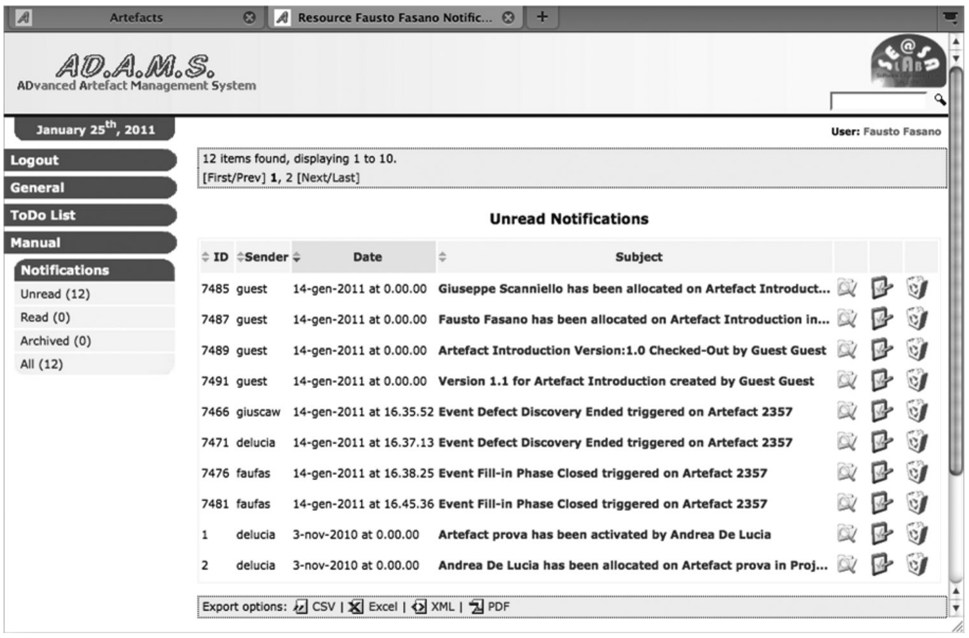
○	DRAFT	○	LOW IMPACT (e.g. Text Corrections, Minor Release)
		●	HIGH IMPACT (e.g. Major Release)
●	REVISION		

COMMENTS

Include dependences issue has been addressed.
Actor hierarchy has been fixed.

File sktop/Use Case Diagram.gif

Fig. 6 *Artefact check-in*



User: Fausto Fasano

12 items found, displaying 1 to 10.
[First/Prev] **1**, 2 [Next/Last]

Unread Notifications

ID	Sender	Date	Subject
7485	guest	14-gen-2011 at 0.00.00	Giuseppe Scanniello has been allocated on Artefact Introduct...
7487	guest	14-gen-2011 at 0.00.00	Fausto Fasano has been allocated on Artefact Introduction in...
7489	guest	14-gen-2011 at 0.00.00	Artefact Introduction Version:1.0 Checked-Out by Guest Guest
7491	guest	14-gen-2011 at 0.00.00	Version 1.1 for Artefact Introduction created by Guest Guest
7466	giuscaw	14-gen-2011 at 16.35.52	Event Defect Discovery Ended triggered on Artefact 2357
7471	delucia	14-gen-2011 at 16.37.13	Event Defect Discovery Ended triggered on Artefact 2357
7476	faufas	14-gen-2011 at 16.38.25	Event Fill-in Phase Closed triggered on Artefact 2357
7481	faufas	14-gen-2011 at 16.45.36	Event Fill-in Phase Closed triggered on Artefact 2357
1	delucia	3-nov-2010 at 0.00.00	Artefact prova has been activated by Andrea De Lucia
2	delucia	3-nov-2010 at 0.00.00	Andrea De Lucia has been allocated on Artefact prova in Proj...

Fig. 7 *Inspection related notifications integrated in ADAMS*

optional and are performed depending on the software artefact and the aim of the inspection. In the following, we detail the different phases of the inspection process and describe an example of application to the inspection of a Java class.

5.1 Planning

In this phase the quality manager specifies which artefact version has to undergo a formal review process, defines a new checklist or modifies an existing one, and selects the inspection team members (see Fig. 9). All these features are accomplished by using the functionalities provided by 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-list (see Fig. 7).

Fig. 9 shows how the quality manager selects three members of the project team and creates the inspection team. In our example, the quality manager also specifies three check items for the checklist to be used during the inspection of the Java class addressing the use of identifier naming conventions, code indentation and meaningfulness of the identifiers.

5.2 Overview

In the overview, the artefact author explains the design and the logic of the software artefact under inspection to the

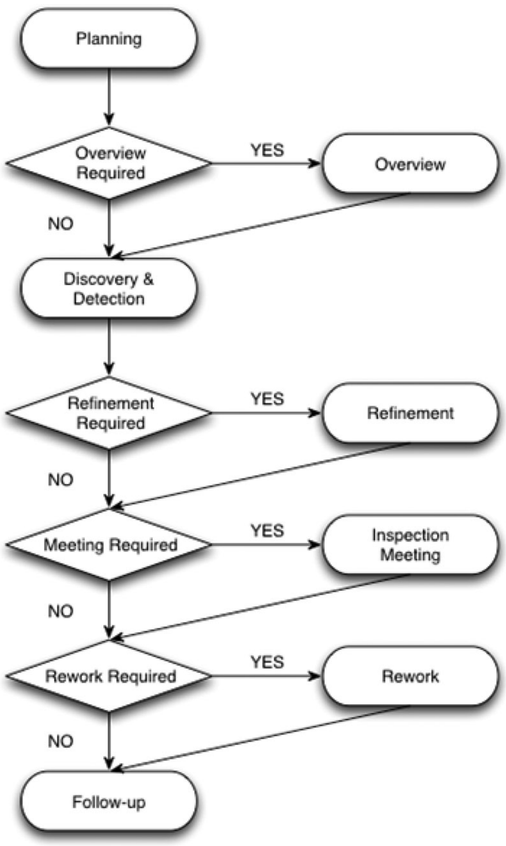


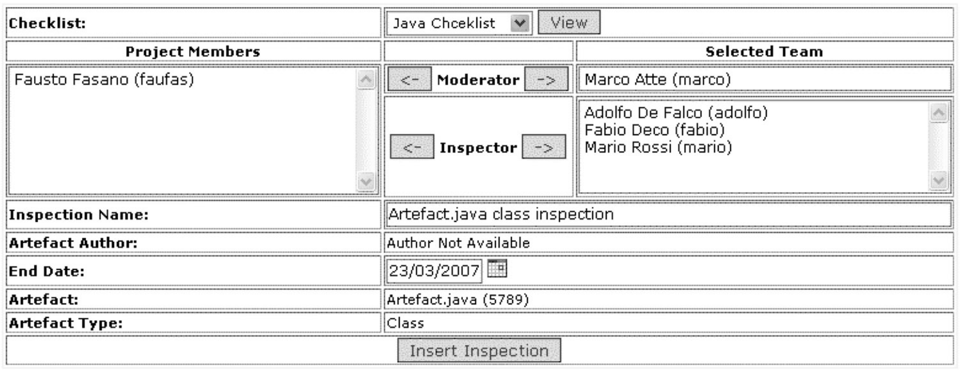
Fig. 8 *WAIT inspection process*

other members of the inspection team. To this aim, he/she produces a document that briefly describes the purpose and the scope of the artefact and then deploys it in ADAMS. The event notification management subsystem of ADAMS is used to notify all the inspection participants. In particular, the system sends an e-mail containing the reference to the document produced by the artefact's author, the schedule of the inspection, and the information about the Java class to be inspected.

5.3 Discovery and detection

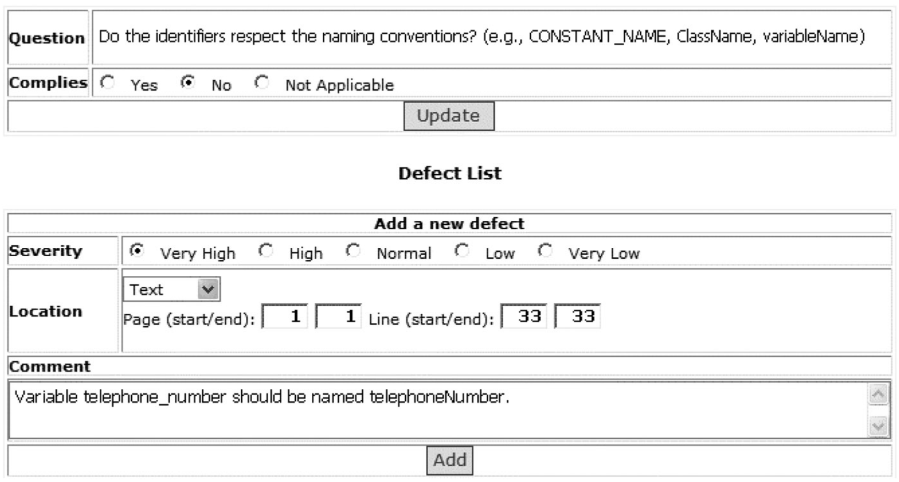
The inspectors analyse the artefact using the previously defined checklist (see Fig. 5) and take note of the candidate defects by highlighting all the cases where the artefact does not comply with the control checklist. The system records the identified defects, its location within the software artefact in terms of page and line numbers or picture/table sequential number (see Fig. 10). The inspector can also indicate the severity of a candidate defect and give a brief comment describing the reason why it contrasts with the check item.

Anytime, the moderator can visualise the inspector's defect log, the check items as well as a preview of the merged defect logs (see Fig. 11). This information can be used to decide whether the detection phase can be concluded and the next phase can start. However, even in case the moderator does not access this information, ADAMS notifies him/her as soon as all the inspectors have completed the discovery and detection phase.



Checklist:	Java Chceklist View	
Project Members		**Selected Team**
Fausto Fasano (faufas)	<- **Moderator** ->	Marco Atte (marco)
	<- **Inspector** ->	Adolfo De Falco (adolfo) Fabio Deco (fabio) Mario Rossi (mario)
Inspection Name:	Artefact.java class inspection	
Artefact Author:	Author Not Available	
End Date:	23/03/2007	
Artefact:	Artefact.java (5789)	
Artefact Type:	Class	
	Insert Inspection	

Fig. 9 *Inspection planning*

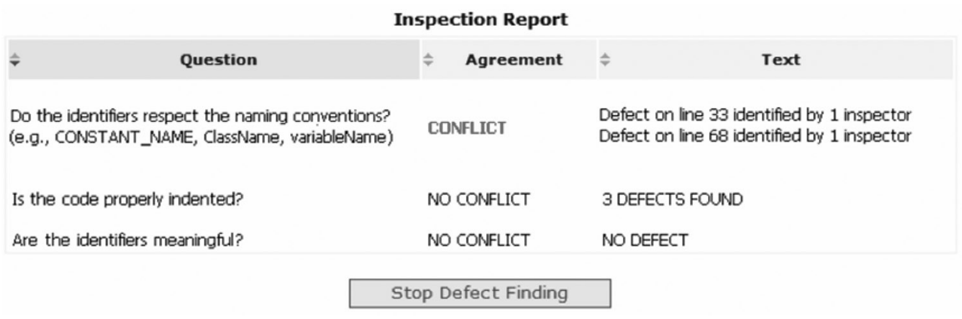


Question	Do the identifiers respect the naming conventions? (e.g., CONSTANT_NAME, ClassName, variableName)
Complies	Yes [✓ No] Not Applicable
	Update

Defect List

Add a new defect	
Severity	[✓ Very High] High Normal Low Very Low
Location	Text Page (start/end): 1 1 Line (start/end): 33 33
Comment	
Variable telephone_number should be named telephoneNumber.	
	Add

Fig. 10 *Defect identification*


Inspection Report

Question	Agreement	Text
Do the identifiers respect the naming conventions? (e.g., CONSTANT_NAME, ClassName, variableName)	CONFLICT	Defect on line 33 identified by 1 inspector Defect on line 68 identified by 1 inspector
Is the code properly indented?	NO CONFLICT	3 DEFECTS FOUND
Are the identifiers meaningful?	NO CONFLICT	NO DEFECT

Fig. 11 *Merged defect list (moderator's view)*

In our example, the three inspectors fill in the defect logs by specifying each line in the Java class that does not respect a check item. In particular, regarding the first check item, that is, respecting the naming conventions, the inspectors discover a different set of defects: one inspector identified a defect at line 33, whereas a second inspector identified the same type of defect at line 68, thus generating a conflict. A defect is marked as a conflict when only one inspector identifies it as defect. The inspectors agree on three defects for the second check item, that is, the code indentation. None of the inspectors discovers defects for the third check item, that is, meaningfulness of the identifiers. This situation is available to the moderator as the tool notifies him/her about the conflict for the first check item and reports on the number of discovered defects for the remaining check items (see Fig. 11).

5.4 Refinement

When the detection phase is concluded, the moderator accesses the defect log containing all the defects identified by the inspectors. In case the inspectors disagree on the defects for a check item, the system highlights it to the moderator (see Fig. 11), who decides whether an asynchronous discussion is needed in the refinement phase. In this case, the tool sends an e-mail containing the conflict list to the members of the inspection team. This e-mail also aims at notifying the team members that the conflicts can be analysed. The main goals of this phase are to get an agreement among the team members and to encourage an asynchronous discussion among the inspectors to remove false defects and to build the consensus on the true defects. There is no time constraint for the accomplishment of this phase. Indeed, the manager decides when this phase can be concluded according to the project constraints and the state of the discussion.

According to Lanubile *et al.* [17], we consider a defect as a true defect when at least two inspectors recognise it; otherwise it is marked as a conflict. However, the minimum number of reviews required to automatically get an agreement can be specified by the quality manager, for example, according to the project quality plan.

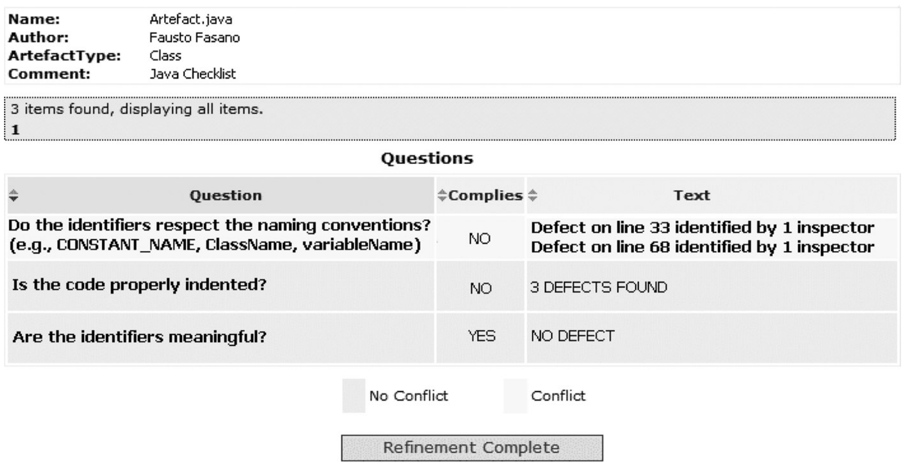
In this phase, the inspector accesses the merged defect list and selects one of defects that caused the conflict (see Fig. 12). To assist the inspector, the system highlights the conflicts using a different colour for the conflicts and provides hypertextual links to the defect details. By accessing the defect details, the inspector decides whether it is a true or false defect.

The merged defect list is shared among the members of the team. Hence, when an inspector solves a conflict, it is also removed from the list of the remaining inspectors. When all the conflicts for a check item are solved, the highlighting is removed as well. As this phase is not mandatory, the moderator can decide to skip it and directly resolve conflicts on the identified defects.

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

5.5 Inspection meeting

Unsolved conflicts can be synchronously discussed using an inspection meeting implemented as a chat. As shown in


Name:	Artefact.java
Author:	Fausto Fasano
ArtefactType:	Class
Comment:	Java Checklist

3 items found, displaying all items.
1

Questions

Question	Complies	Text
Do the identifiers respect the naming conventions? (e.g., CONSTANT_NAME, ClassName, variableName)	NO	**Defect on line 33 identified by 1 inspector** **Defect on line 68 identified by 1 inspector**
Is the code properly indented?	NO	3 DEFECTS FOUND
Are the identifiers meaningful?	YES	NO DEFECT

Fig. 12 *Merged defect list (inspector's view)*

Fig. 13, when an inspector accesses the chat, he/she visualises the checklist and the output of the inspection process for each inspector. One column for each inspector is shown. The column associated to the current inspector (Mario Rossi in the example) is highlighted (the column background colour is light grey), while the last column visualises the moderator decision. An inspector can access the defect logs of the other inspectors grouped by check item. However, only the column regarding the logged inspector is editable. It is worth noting that conflicts – as well as the inspector that caused them – can be identified by looking at the text indicating the number of conflicts (red/light grey). Further details about the conflict (check item, location, author and comments) for each defect are accessible by pressing the defect count button. The checkbox in the inspector's column is used to indicate whether the artefact is compliant with the check item. The tool automatically deselects this checkbox when defects are discovered. Communication among the inspectors is supported by instant messaging (on the bottom left hand side). The instant messaging tool also visualises the connected members (on the bottom right hand side).

Besides coordinating the meeting and supporting the inspector during the conflict resolution, the moderator has the possibility to fill in his/her own checklist and conclude the inspection process specifying whether the artefact can be baselined or not. In the latter case the author has to fix the defects during the follow-up. The revision is concluded when all the check items are satisfied. In case an agreement is not reached on a check item, the moderator response is considered. The discussion of the unsolved conflicts can start even in case not all the participants join the inspection meeting. This enables the enactment of meetings to resolve conflicts that do not involve some inspectors.

In our example, the three inspectors access the meeting and discuss on the two unsolved conflicts. At the end of the discussion the moderator decided to classify them as true defects.

5.6 Rework and follow-up

In case defects are identified, the author has to fix them during the rework phase. Once defects are addressed, the author creates a new version of the artefact that is validated by the moderator during the follow-up phase. As a result of this phase, a new baseline of the artefact can be created or a further re-inspection is required. The system maintains the defect logs for each artefact version. Thus, in case the artefact undergoes several inspections, it is possible to access the defect logs of each version.

6 Case study

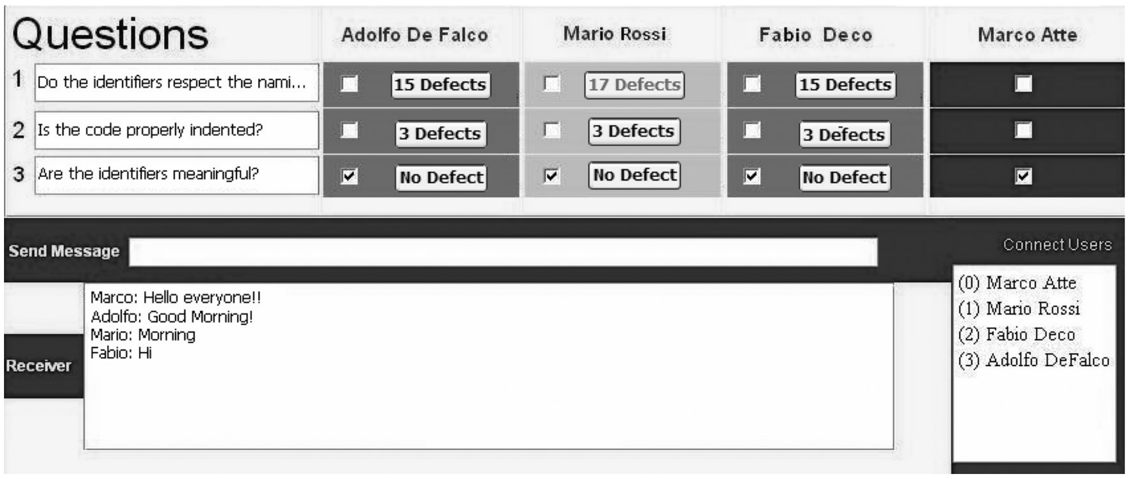
We previously compared the distributed inspection process implemented in WAIT with the Fagan's method through controlled experiments [23]. We report here on the results of a case study conducted within a University setting during a preliminary usage of the integrated WAIT–ADAMS system. Among the different kinds of empirical research strategies, case study represents a suitable methodology for software engineering research, because it studies phenomena in a natural context that are hard to study in isolation [50].

The presentation of this study follows the guidelines suggested by Yin [51]. The study has both a research perspective, aiming at qualitatively assessing the integration of a distributed inspection software system (i.e. WAIT, within an artefact management system), and an application perspective, trying to evaluate the advantages of adopting the integrated system in software projects.

6.1 Context, data set and planning

The system resulting from the integration of WAIT in ADAMS has been experimented within the projects conducted by the students of the Computer Science program at the University of Salerno. The case study was carried out over two sequential academic years from April 2007 to July 2008. Each project team included between 3 and 16 Bachelor students (2nd year B.Sc.) with software engineer and/or developer roles and one or two Master students (2nd year M.Sc.) with roles of project and quality managers. The Bachelor students were attending the software engineering course, whereas Master students were attending the software project management course. Bachelor students had previously attended programming and database management courses, while master students had also attended advanced courses of programming and software engineering. In the case study, the Bachelor students played the role of inspectors, while the Master students acted as moderators of the inspections.

The students were asked to use the integrated WAIT–ADAMS system to incrementally design and develop software systems with a distributed architecture (typically three-tier) using Java, web technologies and relational database management system. We randomly grouped the



Questions	Adolfo De Falco	Mario Rossi	Fabio Deco	Marco Atte
1 Do the identifiers respect the nami...	☐ 15 Defects	☐ 17 Defects	☐ 15 Defects	☐
2 Is the code properly indented?	☐ 3 Defects	☐ 3 Defects	☐ 3 Defects	☐
3 Are the identifiers meaningful?	☑ No Defect	☑ No Defect	☑ No Defect	☑

Send Message

Receiver

Marco: Hello everyone!!
Adolfo: Good Morning!
Mario: Morning
Fabio: Hi

Connect Users

(0) Marco Atte
(1) Mario Rossi
(2) Fabio Deco
(3) Adolfo DeFalco

Fig. 13 *Communication environment to support the inspection meeting phase*

students in teams and then we allocated each team on a software project. The team adopted an incremental development process similar to the one suggested in [52]. In particular, the teams performed a complete requirements analysis and high-level design and then proceeded with an incremental development of each identified subsystems. The Master students were responsible for coordinating the projects, defining the schedule, organising meetings, collecting process metrics and allocating human resources to tasks. They were also responsible for defining process and product standards of the project, collecting product metrics, managing the artefact reviews for quality control and acting as moderators during the inspection.

With regard to the case study presented in the paper, we asked the Master students (i.e. the quality managers) to identify the software artefacts to inspect and to create a checklist for each type of artefact that had to undergo an inspection (see Fig. 5). They were also responsible to compose the inspection teams (see Fig. 9) and to schedule the inspections (planning phase of the inspection process in Section 5.1).

The Bachelor students (i.e. the inspectors) performed the inspections planned by the quality manager according to the inspection process described in Section 5. To start an inspection the manager has to turn the state of an artefact into a revision state (see Fig. 6).

6.2 Research questions

To assess the integration of WAIT in ADAMS and to evaluate the advantages of adopting it, we have defined the following two research questions:

RQ1: Is the integration of WAIT in ADAMS useful to support the moderator within the inspection process?
RQ2: Is the integration of WAIT in ADAMS useful to support the inspector within the inspection process?

To address these questions, the moderators and the inspectors were required to fill in two different questionnaires to assess the effectiveness of the distributed inspection system and the support provided by the integrated system. Each student had the questionnaire to fill in his/her to-do list on ADAMS. Once filled in, the questionnaires were uploaded in ADAMS.

The questionnaires for the moderators and the inspectors are shown in Tables 1 and 2, respectively. In both the questionnaires the answers are based on a five-point Likert scale [28]: (1) strongly agree; (2) agree; (3) neither agree nor disagree; (4) disagree; (5) strongly disagree.

These questionnaires were designed to (i) minimise comprehension problems (e.g. reducing as much as possible the use of unfamiliar terms), (ii) reduce complexity and memory overload and (iii) increase respondent attention (e.g. reducing the number of questions to the essential). The subjects were also asked to annotate problems and observations useful to improve the integrated system.

All the questions except Q1 are different in the questionnaires of the moderators and the inspectors. The rationale for using two different questionnaires relies on the fact that the moderators and the inspectors use different functionalities of the integrated system since their roles in the enactment of the inspection process is completely different. Therefore with respect to the moderator, we formulated the questions thus getting information on the

Table 1 Moderators' questionnaire

Id	Question
Q1	I found useful that the to-do list allows accessing my inspections and also shows the progress status of my inspections
Q2	I found useful the possibility of accessing an artefact under inspection together with the list of identified defects
Q3	I found useful that the software system notifies the team member allocated to an inspection
Q4	I found useful that the software system notifies me when the defect detection starts
Q5	I found useful that the software system notifies me when the asynchronous discussion on the possible defects starts
Q6	I found useful that the software system notifies the developers of a given artefact when the inspection process is accomplished and defects are detected
Q7	I found useful the way to access the identified defects (during the rework phase)
Q8	During the creation of a new version of a revised artefact, I found useful that the software system allows developers to specify which defect has been addressed and which defect needs to be addressed
Q9	I found useful that the software system allows associating checklists to each type of artefact
Q10	During the definition of the inspection team, I found useful the possibility of accessing information about who worked on an artefact, how the project team is composed, who already inspected similar artefacts, etc
Q11	I found useful the traceability layer to schedule inspections for artefacts depending on previously inspected related artefacts

Table 2 Inspectors' questionnaire

Id	Question
Q1	I found useful that the to-do list allows accessing my inspections and also shows the progress status of my inspections
Q2	I found useful the notification received when the moderator allocates me on an inspection
Q3	I found useful the notification received when the defect detection of an inspection assigned to me starts
Q4	I found useful the notification received when the asynchronous discussion is started on the candidate defects
Q5	I found useful the notification received when defects are identified on an artefact I have produced
Q6	I found useful the filtering of the list of artefacts (I am working on) containing defects
Q7	I found useful the list of the defects of the artefact (I am working on)
Q8	During the creation of a new version of a revised artefact, I found useful that the software system allows developers to specify which defect has been addressed and which defect needs to be addressed

support the integrated tool provides to manage the teams and the inspection process. On the other hand, the questionnaire of the inspectors aimed at gathering information on the support the environment provides to inspect software artefacts. Further on, differences between the questionnaires also concern the questions formulated to get information on the usefulness perceived by moderators and inspectors on the context-awareness within the defined inspection process supported by the notification mechanisms of ADAMS.

To analyse the data of the post-experiment survey questionnaires, we used a Mann–Whitney test in order to find out whether the median answer is significantly different from the neutral mid option (i.e. 'uncertain'). We also graphically represent the results of the post-experiment survey questionnaires by using bar plots, since they provide a quick visual representation to summarise ordinal variables. The values for the question, whose answers that are significantly different from the mid option, are highlighted in red (dark grey). For replication purposes the raw data and the R script of the bar plots can be downloaded from http://sesa.dmi.unisa.it/tr/wait.rar.

6.3 Results

Table 3 reports on some descriptive statistics of the students' projects accomplished in the case study. In particular, the first column shows the ID of the project, whereas the second and third columns summarise the composition of each team. The subsequent two columns report on the number of high-level artefacts and source code files produced by the teams. The total number of artefacts and the total number of created versions of these artefacts are shown in the sixth and seventh columns, respectively. A brief description of the developed software systems is provided in the following:

- eTour is an application for Smartphone that supports tourists during the discovery of artistic and cultural heritage;
- SMEG is a SCORM compliant engine able to trace the fruition of e-learning contents;
- Web Agency is a web-based system that provides functionalities form the management of websites development and deployment companies;
- Holiday Heaven is a web-based hotel reservation and customer management system.
- Nova is a distributed information system to support the technical assistance centres in small and medium-sized companies;
- EasyUse is a web application for the management of tourist and beach resorts;
- H-@-Commerce is an e-commerce platform that provides features for the auction sale, marketing, inventory and supply of products or services over the Internet;
- ELEION is a system for the electronic vote for Italian political elections;
- Nomadblue is a client/server application that handles information messages/advertisements via Bluetooth;
- 4Gym is software to automate the management of gym and sport centres;
- Planet Video is a web application to coordinate and optimise the management of video rental companies;
- Supermarket is a client/server application to manage the inventory of a supermarket.

The study involved 94 Bachelor students and 13 Master students allocated on 12 software projects. In particular, a team was composed of 16 Bachelor students and two Master students sharing management responsibilities. Five teams were composed of nine Bachelor students and one Master student, while three teams were composed of eight Bachelor students and one Master student. The remaining three teams were composed of three Bachelor students and one Master student.

Table 3 also shows that the average number of produced artefacts was about 540 (9 artefacts were produced within the smallest project and 1120 artefacts were produced within the largest one). The low number of artefacts produced within the smallest project was due to the fact that the team members did not produce fine-grained software artefacts during the software development process. For instance, the requirements analysis and design documents were produced as single software artefacts. The average number of produced versions was about 1416. For six projects the number of source code artefacts is one. This is due to the fact that some teams decided to use the versioning management system integrated in the development environment to manage source code artefacts. A compressed archive containing the source code was

Table 3 Projects statistics

ID	Project name	# students		# artefacts			# of versions	Document pages			CC	LOC
		BS	MS	HLA	LLA	Tot		RAD	SDD	ODD		
1	eTour	16	2	886	214	1100	2954	219	28	66	392	48 427
2	SMEG	9	1	166	1	167	361	93	23	22	23	5414
3	Web Agency	9	1	776	62	838	1901	108	41	61	171	17 325
4	Holiday Heaven	9	1	582	96	678	1591	188	42	472	162	16 263
5	Nova	9	1	257	1	258	701	115	25	27	81	11 982
6	EasyUse	9	1	444	1	445	652	84	20	30	31	2739
7	H-@-Commerce	8	1	718	67	785	2416	262	36	24	103	24 878
8	ELEION	8	1	840	211	1051	2676	462	31	40	169	11939
9	Nomadblue	8	1	1051	69	1120	3612	179	31	94	318	35 292
10	4Gym	3	1	8	1	9	42	108	21	41	65	8474
11	Planet Video	3	1	14	1	15	45	83	37	20	79	9023
12	Supermarket	3	1	17	1	18	37	89	29	32	70	9456
average values			479.9	60.4	540.3	1415.7	165.8	30.3	77.4	138.7	16768	

BS = Bachelor students; MS = Master students; HLA = high-level artefacts; LLA = low-level artefacts; CC = code classes; LOC = lines of code uploaded in ADAMS. In this case no inspection was enacted in the integrated system. Finally, some metrics about both high-level and low-level artefacts are reported. In particular, the number of pages for three of the main documents produced, namely the requirements analysis document (RAD), the system design document (SDD) and the object design document (ODD), are provided. In this case, the RAD is the longest document (on average about 165 pages long), whereas the SDD is the shortest (on average about 30 pages). It is important to point out that the ODD of the project 4 is huge, as compared to the others, as the authors decided to include the entire documentation generated by the JavaDoc program to specify the application program interface. Concerning the source code, the number of code classes and the number of lines of code (LOC) for each project is reported. In this case, the average number of code classed is about 138 (318 classed were produced for the biggest project and 23 for the smallest one) and the average number of LOC is about 17K (about 48K for the biggest project and 2.7K for the smallest one).

Table 4 summarises some descriptive statistics on the inspections the teams accomplished within each project. In particular, for each project the total number of inspected artefacts is reported. Information on the number of performed inspections and the average number of defects for each inspection is provided as well. We got the raw data to compute the descriptive statistics in Table 4 by directly accessing to the database of the ADAMS system.

The average number of performed inspection was about 51 (see Table 4). In particular, the largest and smallest numbers of performed inspections were 94 and 19, respectively. The teams inspected high-level (e.g. requirements analysis and design documents) and low-level software artefacts (i.e. source code). Concerning the high-level software artefacts, the teams used the system to inspect the whole or a part of the following documents: RAD, SDD, ODD and test plan documents. All these artefacts were based on the templates suggested in [52].

RQ1: Support provided to the moderator: Fig. 14 summarises the answers provided by the moderators. In particular, they strongly agreed on the usefulness of accessing their inspections and their progress status (see bars of Q1). In fact, the strongly agree answer is significantly different (P-value $= 0.002$) from the mid value. Furthermore, one subject suggested removing from the to-do list the software artefacts that underwent the inspection process and were baselined. In fact, the to-do list is not automatically updated as ADAMS delegates the moderator to manually allocate/de-allocate a software engineer on a given artefact. We plan to address this issue in the new version of our system.

The subjects also found useful the feature that allows accessing an artefact under inspection together with the list of identified defects. All the moderators except one answered ‘I strongly agree’ on the question Q2 (P-value $= 0.001$).

The bars of the statements from Q3 to Q5 indicate that the moderators found the notification mechanism useful (P values are 0.002 for Q3 and Q4 and 0.006 for Q5). Despite the positive judgment, the managers found less useful the fact that the system sends a notification when an asynchronous discussion starts (see bars of Q5). Others would have preferred a notification concerning the accomplishment of the asynchronous discussion, that is, the refinement phase.

The moderators found very useful the notifications sent to the authors before starting the rework phase (P-value < 0.001) as shown by the bars of Q6. As shown by the bars of Q7, the moderators also found very useful that the software system shows the defects that have been addressed and defects that needs to be addressed (P-value $= 0.001$). However, some moderators suggested highlighting in green the defects that have been corrected and in red the remaining defects. In particular, 4 out of 13 were the subjects that asked for this modification.

The bars of the statement of Q8 indicates that the moderators generally found useful that during the creation of a new version of a revised artefact the integrated system allowed to specify which defect have been addressed and which defects need to be addressed (P-value $= 0.002$).

The bars of both Q9 and Q10 indicate that the moderators agree (P-value $= 0.005$) on the usefulness of both associating a checklist to an artefact type and accessing information on the inspection teams. As shown by the bars of Q11, the subjects strongly agreed (P-value $= 0.001$) on the usefulness of obtaining traceability links on the software artefacts that could be affected by the defects of a given inspected artefact. This shows another interesting advantage provided by the integration of WAIT within ADAMS.

According to the analysis of the answers provided by the subjects, we can positively address the research question RQ1. In particular, we can conclude that the system proposed here effectively supports the moderators.

Table 4 Statistics of the inspection performed

Project ID	Inspected artefacts	Inspected high-level artefacts	Inspected source code artefacts	Number of Inspections	Avg. number of identified defects
1	77	26	51	85	7.8
2	20	13	0	26	4.5
3	48	15	33	63	6.8
4	43	15	28	56	7.1
5	23	12	0	30	12.8
6	24	13	0	38	6.6
7	36	16	20	58	18.2
8	69	23	46	90	5.9
9	72	20	52	94	10.6
10	8	8	0	27	19.3
11	13	13	0	30	16.5
12	12	12	0	19	18.2
average values	37.17	15.58	21.58	51.33	11.2

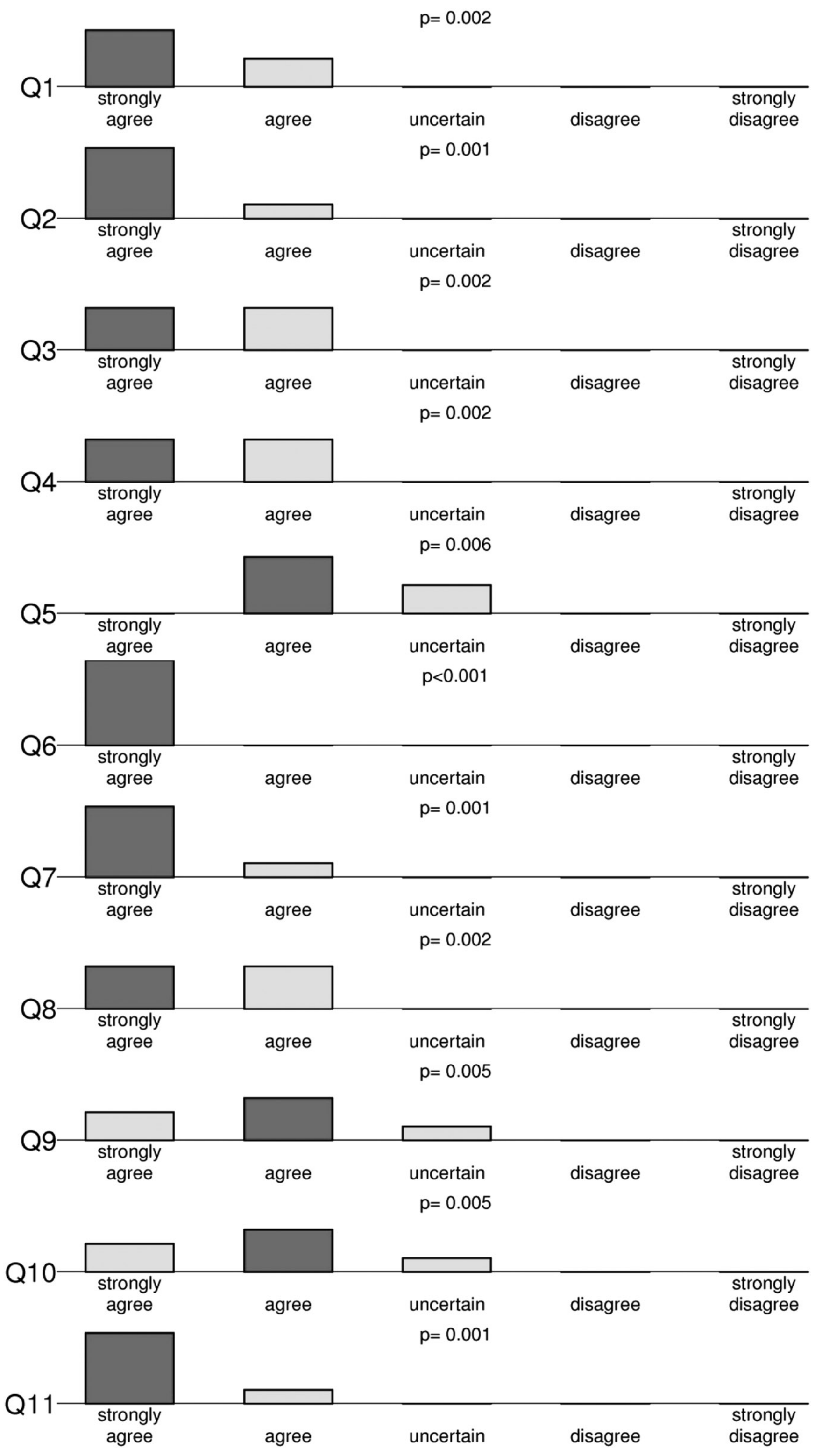


Fig. 14 *Bar plots of the moderators' questionnaires*

RQ2: Support provided to the inspector: The data of the questionnaire filled in by the inspectors are visually summarised in Fig. 15. This figure reveals that the agreement level of the subjects is concordant (*P*-value <0.001) about the possibility of accessing their on-going inspections and the corresponding progress status within their to-do list (see bars of Q1).

Similarly, the bars of the statements from Q2 to Q4 (*P*-values are all less than 0.001) show that the inspectors agreed on the usefulness of the notification mechanism provided by the integrated environment. Similarly to the results achieved with the moderators, the inspectors found less useful the notification regarding the start of the asynchronous discussion, that is, the refinement phase. In fact, the bar of the mid-value of Q4 indicates that the subjects neither agree nor disagree with respect to the usefulness of such a kind of notification. The subjects found more useful the notification received when defects

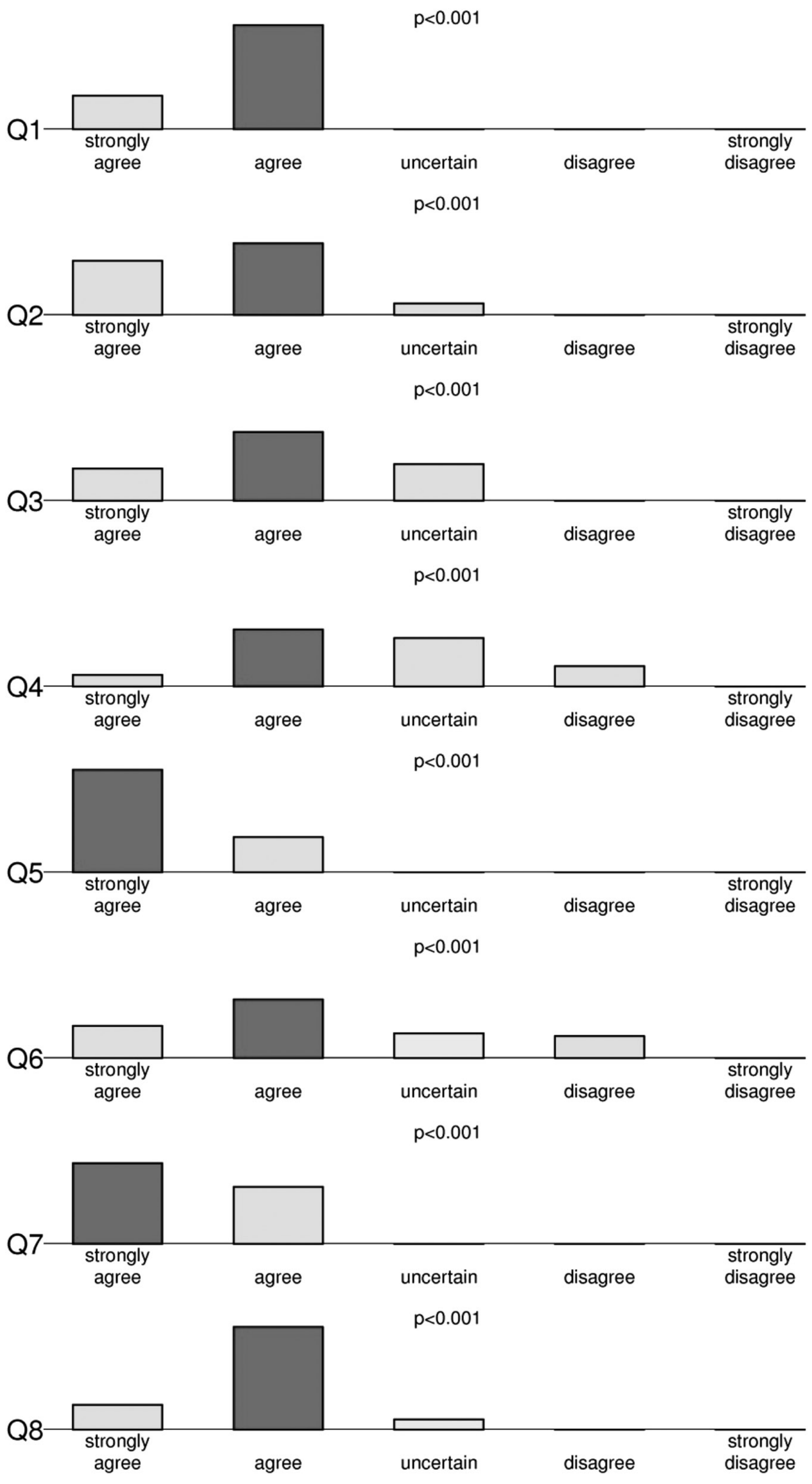


Fig. 15 *Bar plots of the inspectors' questionnaires*

are identified on an artefact they produced. In fact they strongly agree (P-value < 0.001) on this notification mechanism as the bars of Q5 show.

The bars of the question Q6 indicate that the inspectors expressed a positive judgment on the possibility of filtering the list of artefacts. Most subjects answered agree to the statement Q6 (P-value < 0.001). On the other hand, inspectors found useful the possibility of showing the defect identified for an artefact they are working on (see bars Q7). The agreement level was mainly strongly agree (P-value < 0.001).

As shown by the bars of Q8, the inspectors agreed on the fact that the software system distinguishes the defects that have been addressed and the defects that still need to be considered (P-value < 0.001). It is worth noting that the moderators found this feature more useful than the inspectors.

Concluding, the research question RQ2 can be positively answered, thanks to the analysis of the inspectors' questionnaires. In particular, most inspectors found useful the integration of WAIT in ADAMS to perform inspections in distributed settings.

6.4 Threats to validity

We describe here the threats to validity (i.e. internal, external, construct and conclusion) that may affect the results of the presented case study. Internal validity threats regard external factors that may affect the observed results. We attempted to simulate a real working environment with strict deadlines, although Bachelor and Master students were used as subjects. Another factor that may have influenced the internal validity is the difficulty to comprehend the statements, for example, ambiguous, not clear, not well formulated. However, we designed the questionnaire to minimise these problems. Furthermore, the subjects were asked to contact one of the authors in case of problems related to the comprehension of the statements. None of them asked for clarifications, thus enabling us to believe that the statements were clearly formulated.

External validity threats concern the generalisation of the results and are always present when empirical investigations are conducted with students. However, moderators have been selected among last-year Master students, so they have analysis, development and programming experience, and they are not far from junior industrial analysts. In fact, most of the involved Masters students had some working experience due to the internship they made in the industry as final part of their Bachelor degree. Further on, the use of students in empirical investigations like the one presented in this paper cannot be considered a major issue [53], since the cognitive processes to use our technology are more or less similar across students and industry professionals. To confirm or contradict the results, we have planned, however, to conduct experimentations in industrial research projects, to confirm or contradict the results presented in the paper. This part of our research is actually the most challenging and will possibly take place over the next years. Similarly to [54], the execution of these studies will also enable us to assess whether or not our technology is ready to be transferred to the software industry. A further threat that might affect the external validity concerns the fact that the software systems developed by the students are small/medium sized, but they represent a good benchmark as they cover many different projects with different goals. Furthermore, students used the proposed inspection process and system on different types of artefacts having different size, for example, RAD, scenarios, use cases, class models, sequence diagrams and source code. However, to better assess the developed technology, we aim to assess our technology on case studies conducted on larger projects developed in collaboration with industrial partners.

Construct validity threats concerns the questionnaire used in the study. To mitigate this validity the questionnaire was designed using standard ways and scales [28]. Furthermore, the statements are expressed in positive manner, so there is the risk that subjects answered them without paying attention. In investigations based on survey questionnaires, it is usually impossible to know whether the subjects answer truthfully. Also, the subjects' motivations could affect the answers and then the observed results. To avoid social threats caused by evaluation apprehension, the students were not evaluated on the number and quality of inspections they conducted.

Threats to conclusion validity concern the issues that affect the ability to draw a correct conclusion. In this study, threats to conclusion validity concern the selection of the subjects, the data collection and the validity of the statistical tests. With regard to the selection of the populations, we drew fair samples and conducted our experiments with subjects belonging to these samples. Non-parametric tests were used to analyse the answers provided by the subjects.

7 Final remarks and future work

To drive software development towards engineering-level precision, software companies have to use quality-enhancing techniques, methods and tools right from the early phases of the development process [48]. To this end, special conceived development systems are needed to support companies to develop and maintain reliable software products. As a first step towards this direction, in this paper we have reported the results of the integration of a web-based tool, that is, WAIT – web-based artefact inspection tool, implementing a distributed inspection process checklist based within an artefact-based process support system, namely ADAMS.

The validity and the effectiveness of the integrated system were assessed in a case study conducted with Master and Bachelor students in Computer Science of the University of Salerno. Master students acted as quality managers and inspection moderators, while Bachelor students acted as inspectors. A questionnaire was proposed to the subjects to get information about the quality management support provided by the system resulting from the integration of WAIT in ADAMS.

This study revealed that the subjects found the asynchronous discussion (i.e. the refinement phase) useful to solve conflicts on potential defects. The greater part of the subjects asserted that the meetings were rarely adopted. The study also revealed that the teams did not perform the meeting when the number of unsolved conflicts at the end of the asynchronous discussion is very low. However, to resolve the potential defects unsolved during the asynchronous discussion, the subjects considered necessary the enactment of a meeting. This result confirms the achievements obtained by the experimentation presented in [23], where the distributed inspection process (without considering the integration in ADAMS) was compared with the Fagan's method by means of controlled experiments. We also observed that in case meetings were required, they were carried out to discuss very few conflicts. The results obtained from our empirical investigations together with the ones presented in [17] confirm that a virtual meeting is more effective in case a preliminary asynchronous discussion is conducted before.

We also noticed that differences between the moderators and the inspectors exist on the perceived usefulness of the functionalities implemented in ADAMS. In particular, we observed that the moderators generally found the notifications more useful. This is a predictable result, as moderators need tools to monitor the project and inspection process, in particular. Therefore in the future we plan to further differentiate the notification mechanism according to the role of a software engineer within the inspection process.

Other improvement areas can be derived from the results of the case study. A first direction would be to add some features to further simplify the defect localisation within the software artefact. A second direction should aim at adding new features to better support synchronous discussion among inspectors and moderators. As a possible further direction for our research, we plan to support different reading techniques to identify defects within software artefacts, for example, *ad hoc* and scenario-based [55, 56]. We also plan to further assess our system using the technology acceptance model [57]. This will enable us to better investigate the relationships that exist between the usefulness of our system, its ease of use and its use.

Our research agenda also includes an empirical evaluation with practitioners to evaluate whether the technology implemented and experimented in our research laboratory can be transferred to the software industry [58, 59]. Therefore we have conducted a state of the practice industrial survey within Italian software companies/organisations [60]. This research strategy is typically conducted when the use of a technique or tool has already taken place or before it is introduced. One of the main findings of this investigation is that the software companies manifested great interest in the integration of distributed inspection methods within a software artefact management system. Despite the available number of distributed inspection processes and tools, the industrial practice is still far to adopt them since the management consider them non-effective. Another relevant point is the lack of integrated environments to support all the phases of a development process. Finally, although most of the software companies use software configuration management systems to access the level two of CMM [http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr012.pdf] and to get ISO 9000 certification, distributed inspection tools are not widely employed in the industrial practice despite they are being recognised useful to improve software quality.

According to the case study results and the main findings of a recent state of the practice industrial survey [60], we have conceived a research plan between academy and industry to further investigate the developed technology and possibly speed up its transfer to the industry. To this end, we are planning controlled experiments and case studies with some of the industries involved in the survey that manifested interest in the experimentation of distributed inspection methods. This part of our research is actually the most challenging and will take place over the next few years.

8 References

1 Dewan, P., Agarwal, P., Shroff, G., Hegde, R.: 'Distributed side-by-side programming'. Proc. 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering, Washington, DC, 2009, pp. 48–55

2 Herbsleb, J.D., Paulish, D.J., Bass, M.: 'Global software development at Siemens: experience from nine projects'. Proc. Int. Conf. Software Engineering, 2005, pp. 524–533

3 Lings, B., Lundell, B., Agerfalk, P.J., Fitzgerald, B.: 'A reference model for successful distributed development of software systems'. Proc. Int. Conf. Global Software Engineering, 2007, pp. 130–139

4 Lanubile, F., Damian, D., Oppenheimer, H.L.: 'Global software development: technical, organizational, and social challenges', *ACM SIGSOFT Softw. Engng. Notes*, 2003, **28**, (6), p. 2

5 Panjer, L.D., Damian, D., Storey, M.: 'Cooperation and coordination concerns in a distributed software development project'. Proc. 2008 Int. Workshop on Cooperative and Human Aspects of Software Engineering, Leipzig, Germany, 2008, pp. 77–80

6 Prikladnicki, R., Audy, J.L., Evaristo, R.: 'Global software development in practice lessons learned', *Softw. Process: Improve. Pract.*, 2003, **8**, (4), pp. 267–281

7 Aversano, L., De Lucia, A., Gaeta, M., Ritrovato, P., Stefanucci, S., Villani, M.L.: 'Managing coordination and cooperation in distributed software processes: the GENESIS environment', *Softw. Process Improve. Pract.*, 2004, **9**, (4), pp. 239–263

8 Booch, G., Brown, A.W.: 'Collaborative development environments', in', in 'Advances in computers' (Academic Press, 2003), vol. 59 Q3

9 Lanubile, F., Ebert, C., Prikladnicki, R., Vizcaino, A.: 'Collaboration tools for global software engineering', *IEEE Softw.*, 2010, **27**, pp. 52–55

10 Berliner, B.: 'CVS II: parallelizing software development'. Proc. USENIX Winter 1990 Technical Conf., Berkeley, CA, 1990, pp. 341–352

11 Estublier, J., Leblang, D., VanDerHoek, A., *et al.*: 'Impact of software engineering research on the practice of software configuration management', *Trans. Softw. Engng. Meth.*, 2005, **14**, (4), pp. 383–430

12 Heydon, A., Levin, R., Mann, T., Yu, Y.: 'Software configuration management using Vesta' (Springer, 2006)

13 Serban, A.: 'Visual sourcesafe 2005 Software configuration management in practice: best practice management and development of visual studio. NET 2005 applications with this easy-to-use SCM tool from Microsoft' (Packt Publishing, 2007)

14 Freedman, D.P., Weinberg, G.M.: 'Handbook of walkthroughs, inspections, and technical reviews: evaluating programs, projects, and products' (Little Brown & Co., 1982, 3rd edn.)

15 Laitenberger, O., DeBaud, J.M.: 'An encompassing life cycle centric survey of software inspection', *J. Syst. Softw.*, 2000, **50**, (1), pp. 5–31

16 Fagan, M.E.: 'Design and code inspections to reduce errors in program development', *IBM Syst. J.*, 1976, **15**, (3), pp. 182–211

17 Lanubile, F., Mallardo, T., Calefato, F.: 'Tool support for geographically dispersed inspection teams', *Softw. Process: Improve. Pract.*, 2003, **8**, (4), pp. 217–231

18 Perpich, J.M., Perry, D.E., Porter, A.A., Votta, L.G., Wade, M.W.: 'Anywhere, anytime code inspections: using the web to remove inspection bottlenecks in large-scale software development'. Proc. Int. Conf. Software Engineering, 1997, pp. 14–21

19 Johnson, P.M., Tjahjono, D.: 'Does every inspection really need a meeting?', *Empirical Softw. Engng.*, 1998, **3**, (1), pp. 9–35

20 Damian, D., Lanubile, F., Mallardo, T.: 'On the need for mixed media in distributed requirements negotiations', *IEEE Trans. Softw. Engng.*, 2008, **34**, (1), pp. 116–132

21 De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: 'Fine-grained management of software artefacts: the ADAMS system', *Softw. Pract. Exp.*, 2010, **40**, (11), pp. 1007–1034

22 De Lucia, A., Fasano, F., Scanniello, G., Tortora, G.: 'Integrating a distributed inspection tool within an artefact management system'. Proc. Int. Conf. Software and Data Technology, 2007, pp. 184–189

23 De Lucia, A., Fasano, F., Scanniello, G., Tortora, G.: 'Evaluating distributed inspection through controlled experiments', *IET Softw.*, 2009, **3**, (5), pp. 381–394

24 Votta, L.: 'Does every inspection need a meeting?', *ACM Softw. Engng. Notes*, 1993, **18**, (5), pp. 107–114

25 Porter, A.A., Johnson, P.M.: 'Assessing software review meetings: results of a comparative analysis of two experimental studies', *IEEE Trans. Softw. Engng.*, 1997, **23**, (3), pp. 129–145

26 Miller, J., Wood, M., Roper, M.: 'Further experiences with scenarios and checklists', *Empirical Softw. Engng.*, 1998, **3**, (1), pp. 37–64

27 Sabaliauskaite, G., Kusumoto, S., Inoue, K.K.: 'Assessing defect detection performance of interacting teams in object-oriented design inspection', *Inf. Softw. Technol.*, 2004, **46**, (13), pp. 875–886

28 Oppenheim, N.: 'Questionnaire design, interviewing and attitude measurement' (Pinter Publishers, 1992)

29 Bianchi, A., Lanubile, F., Visaggio, G.: 'A controlled experiment to assess the effectiveness of inspection meetings'. Proc. 7th Int. Symp. on Software Metrics, METRICS, Washington, DC, 2001, pp. 42–50

30 Biffl, S., Halling, M.: 'Investigating the defect detection effectiveness and cost benefit of nominal inspection teams', *IEEE Trans. Softw. Engng.*, 2003, **29**, (5), pp. 385–397

31 Aurum, A., Petersson, H., Wohlin, C.: 'State-of-the-art: software inspections after 25 years', *Softw. Test. Verif. Reliab.*, 2002, **12**, (3), pp. 133–154

32 Bull. Inspection Process Assistant: User Guide, 1997

33 Iniesta, J.B.: 'A tool and a set of metrics to support technical reviews', in Ross, M., *et al.* (Ed.): 'Software Quality Management II, Volume II: Building Quality into Software' (Computational Mechanics, Southampton, UK, 1994), pp. 579–594

34 Meyer, B.: 'Design and code reviews in the age of the internet', *Commun. ACM*, 2008, **51**, (9), pp. 67–71

35 Brothers, L.R., Sembugamoorthy, V., Muller, M.: 'ICICLE: Groupware for code inspections'. Proc. 1990 ACM Conf. Computer Supported Cooperative Work, 1990, pp. 169–181

36 Gintell, J.W., Arnold, J., Houde, M., Kruszelnicki, J., McKenney, R., Memmi, G.: 'Scrutiny: a collaborative inspection and review system'. Proc. European Conf. Software Engineering, 1993, pp. 344–360

37 Stein, M., Riedl, J., Harner, S.J., Mashayekhi, V.: 'A case study of distributed, asynchronous software inspection'. Proc. Int. Conf. Software Engineering, 1997, pp. 107–117

38 Knight, J.C., Meyers, E.A.: 'Phased inspections and their implementation', *Softw. Engng. Notes*, 1991, **16**, (3), pp. 29–35

39 Knight, J.C., Meyers, E.A.: 'An improved inspection technique', *Commun. ACM*, 1993, **36**, (11), pp. 51–61

40 Mashayekhi, V., Drake, J.M., Tsai, W.T., Reidl, J.: 'Distributed, collaborative software inspection', *IEEE Softw.*, 1993, **10**, (5), pp. 66–75

41 Humphrey, W.S.: 'Managing the software process. SEI Series in Software Engineering' (Addison-Wesley Longman Publishing, Boston, MA, USA, 1989)

42 Murphy, P., Miller, J.: 'A process for asynchronous software inspection'. Proc. Int. Workshop on Software Technology and Engineering Practice, 1997, pp. 96–104

43 Johnson, P.M.: 'An instrumented approach to improving software quality through formal technical review'. Proc. Int. Conf. Software Engineering, 1994, pp. 113–122

44 Macdonald, F., Miller, J.: 'A comparison of tool-based and paper-based software inspection', *Empirical Softw. Engng.*, 1998, **3**, (3), pp. 233–253

45 Yamashita, T.: 'Evaluation of Jupiter: a lightweight code review framework'. MS thesis, University of Hawaii, 2006, drafted from csdl.ics.hawaii.edu/techreports/06-09/06-09.pdf

46 Hedberg, H., Harjumaa, L.L.: 'Virtual software inspections for distributed software engineering projects'. Proc. ICSE Int. Workshop on Global Software Development, Orlando, FL, May 2002

47 Sauer, C., Jeffery, D.R., Land, L., Yetton, P.: 'The effectiveness of software development technical reviews: a behaviorally motivated program of research', *IEEE Trans. Softw. Engng.*, 2000, **26**, (1), pp. 1–14

48 Collins-Sussman, B., Fitzpatrick, B., Pilato, C.: 'Version control with subversion' (O'Reilly, 2004), drafted from http://svnbook.red-bean.com/

49 De Lucia, A., Fasano, F., Scanniello, G., Tortora, G.: 'Enhancing collaborative synchronous UML modelling with fine-grained versioning of software artefacts', *Int. J. Visual Languages Comput.*, 2007, **18**, pp. 492–503

50 Runeson, P., Höst, M.: 'Guidelines for conducting and reporting case study research in software engineering', *Empirical Softw. Engng.*, 2009, **14**, (2), pp. 131–164

51 Yin, R.K.: 'Case study research: design and methods' (Sage, Thousand Oaks, CA, 1994)

52 Bruegge, B., Dutoit, A.: 'Object-oriented software engineering' (Prentice-Hall, 2003, 2nd edn.)

53 Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., *et al.*: 'Preliminary guidelines for empirical research in software engineering', *IEEE Trans. Softw. Engng.*, 2002, **28**, (8), pp. 721–734

54 Colosimo, M., De Lucia, A., Scanniello, G., Tortora, G.: 'Evaluating legacy system migration technologies through empirical studies', *Inf. Softw. Technol.*, 2009, **51**, (2), pp. 433–447

55 Cheng, B., Jeffrey, R.: 'Comparing inspection strategies for software requirements specifications'. Proc. 1996 Australian Software Engineering Conf., 1996, pp. 203–211

56 Porter, A.A., Votta, L.G., Basili, V.R.: 'Comparing detection methods for software requirements inspections: a replicated experiment', *IEEE Trans. Softw. Engng.*, 1995, **21**, (6), pp. 563–575

57 Davis, F.D.: 'Perceived usefulness, perceived ease of use, and user acceptance of information technology', *MIS Quart.*, 1989, **13**, (3), pp. 319–340

58 Pfleeger, S.L., Menezes, W.: 'Marketing technology to software practitioners', *IEEE Softw.*, 2000, **17**, (1), pp. 27–33

59 Redwine, S.T., Riddle, W.E.: 'Software technology maturation'. Proc. 8th Int. Conf. Software Engineering, London, UK, 1985, pp. 189–200

60 De Lucia, A., Fasano, F., Scanniello, G., Tortora, G.: 'Software quality assessment and software error/defect identification: preliminary results from a state of the practice survey'. Submitted to 12th Product Focused Software Development and Process Improvement, 2011

doi: 10.1049/iet-sen.2010.0108

SEN20100108

Author Queries

A. De Lucia, F. Fasano, G. Scanniello, G. Tortora

Q1	References are renumbered to be in numerical order as per the journal style.
Q2	Footnotes are moved into the text as per the journal style. Please confirm their placement.
Q3	Please provide the editor names in [8].