Published in IET Software
Received on 4th November 2008
Revised on 30th July 2009
doi: 10.1049/iet-sen.2008.0101

In Special Issue on EASE 2008

ISSN 1751-8806

Evaluating distributed inspection through controlled experiments

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 (SA), Italy*
[2]*Dipartimento di Scienze e Tecnologie per l'Ambiente e il Territorio, University of Molise, Italy*
[3]*Dipartimento di Matematica e Informatica, University of Basilicata, Viale Dell'Ateneo, Macchia Romana, Potenza, Italy*
E-mail: gscanniello@unisa.it

Abstract: Inspection methods can be classified according to their discipline and flexibility. The discipline concerns the formal aspect of an inspection method, whereas the flexibility is strongly related to the simplicity of organising and conducting a meeting. The majority of the available distributed inspection methods have a high level of discipline and flexibility as they are based on a well-defined process and the discussion among team members is easily organised and conducted. In this study the authors present two controlled experiments to evaluate the effectiveness and the efficacy of a distributed inspection process to discover defects within source code. In particular, the first experiment compares the distributed inspection method proposed to a disciplined but not flexible method (i.e. the Fagan's inspection process). In the second experiment the authors investigate differences between the same distributed inspection method and a flexible but not disciplined method (i.e. the pair inspection method). Data analysis reveals that more flexible methods require less time to inspect a software artefact, while the discipline level does not affect the inspection quality.

1 Introduction

Software inspection is a software engineering practice aiming at identifying defects, reducing rework and producing high-quality software systems [1]. During the last years formal inspection techniques have increased in popularity. This caused the proliferation of several inspection methods to identify defects within software artefacts [2–4].

Tervoven *et al.* [5] propose a classification of inspection methods based on two dimensions: discipline and flexibility. The former concerns the formal aspect of the inspection approach, the latter is strongly related to the simplicity of organising and conducting a meeting and is measured with respect to: place and/or time independence (i.e. the difficulty of having reviewers in the same place at the same time), network (i.e. the base solution for place and/or time independence), tailorable (i.e. how the approach may be adapted according to different inspection scenarios) and varying numbers of participants (what does it happen in case a lower number of participant is available). Nevertheless, this classification does not help software quality managers to select the most suitable inspection method to be adopted. Furthermore, the larger number of available methods can still generate confusion in the management of a software company.

In the last years, many software companies are moving their business to distributed virtual organisation models to remain competitive in the market. However, such a globalisation creates software engineering challenges because of the impact of time zones, distance or diversity of culture and communication. In the 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 global development entails.

Several distributed tools have been presented in the literature to inspect software artefacts [6–8]. The majority of them have a high level of discipline and flexibility as they are based on well-defined process and the discussion among the team members is easily organised and conducted. In [9] we have proposed a geographically

dispersed inspection process, which has been also implemented in WAIT (web-based artefact inspection tool) a web-based software system. The process extends the Fagan's method (FAG) [3] according to the findings discussed by Damian *et al.* [10] and encourages inspection team members to perform a preliminary asynchronous discussion after a preparation phase and before an optional meeting. Furthermore, to provide a more effective quality management support within the software development process, we also integrated WAIT within an artefact-based process supporting system (i.e. ADAMS [11]).

Recently, there is an increasing understanding that experimentations are needed to assess processes, methods and tools for software development and maintenance [12] and to understand how to increase the odds that the technology choice is the right one [13, 14]. To this end, various forms of research strategies are available: controlled experiments, case studies, archival analyses and surveys [10, 15, 16].

We conducted two controlled experiments to investigate the effect of using a geographical dispersed inspection process with respect to two well-known traditional inspection methods selected according to the classification proposed in [5]. In particular, in the first controlled experiment we compared the process implemented in WAIT, which has a high level of discipline and flexibility, to a method with a high level of discipline and a low level of flexibility (i.e. the FAG), whereas in the second experiment we compared WAIT to a method with a low level of discipline and a high level of flexibility [i.e. pair inspection (PI) [17]]. The context of both the experiments is constituted of Master students in Computer Science at the University of Salerno. In both the controlled experiments the subjects were asked to inspect two Java classes.

The remainder of the paper is organised as follows: Section 2 discusses work related to geographically dispersed inspection processes and tools. Differences and similarities with WAIT are presented and discussed as well. Section 3 highlights our inspection process and tool, whereas the design and the results of the controlled experiments are presented in Sections 4 and 5, respectively. The discussion of the results and the threats that could affect the validity of the experiments are delineated in Section 6. Final remarks and future work conclude the paper.

2 Related work

Since Michael Fagan proposed his inspection process, a number of prototype tools for document tracking and inspection planning [2], comment preparation [18], and for both individual preparation and group meetings [19] have been proposed. The most important area of inspection support tools is represented by the online inspection tools, which manage the entire process online. These tools provide support for document handling, individual preparation, meeting support, and data collection.

Among online inspection tools, ICICLE [6] addresses the inspections of C and C++ code, making use of specific knowledge on the programming language to assist during the defect discovery phase. Obviously, this approach is not suitable to review high level software artefacts. Moreover support to distributed inspection is not provided.

Knight and Meyer [20] propose an inspection technique that examines the artefacts in a series of small inspection phases. This technique is implemented in the InspeQ [21] (inspecting software in phases to ensure quality) toolset. A preliminary evaluation of InspeQ for the inspection of a source code written in C has been presented in [20]. InspeQ is different from WAIT, indeed it does not support distributed meeting. Furthermore, support to the inspection process management and awareness is poor.

Scrutiny [22] is a collaborative and distributed system for the inspection and the review of software artefacts. It implements a process that is similar to the Fagan's process, but adds the verifier role to check whether all the founded defects have been addressed by the author. Scrutiny is different from WAIT as it only supports source code artefacts and does not provide support for checklist-based inspections and asynchronous discussions.

CSI [23] (collaborative software inspection) adopts the Humphrey's inspection model [7], which is divided into four sequential phases: the initiation, the preparation, the meeting and the post-discussion. CSI supports annotations by creating hyperlinks between the document and the inspectors' annotations. The producer reviews and categorises the annotations and makes them into one list, which is discussed by the participants during the meeting. CSI supports both synchronous and asynchronous discussions; however, the meeting phase is always synchronous and the decision support for the moderator is not provided.

AISA [24] (asynchronous inspection of software artifacts) is another prototype implementing the Humphrey's model. It addresses the problem of inspecting graphical documents using a web client to visualise documents that are prepared as clickable image maps. This allows the inspectors to annotate the document by using the coordinates of the image portion clicked. Besides visualisation and annotation facilities, the only support the tool provides is the notification of the inspection completion by means of a message sent to the participants when all inspectors have finished the collection phase. The adopted approach can lead to a greater number of false positives as that annotations are made immediately available to all the participants. For such a reason in our process and tool defect annotations are available only when all the inspectors have accomplished the defect discovery. To overcome the

doi: 10.1049/iet-sen.2008.0101

false positive drawback, InspectA [25] sends a notification only when all the detection phases are completed. Differently from our approach, the moderator does not have any progress information about the detection phase. As a consequence, the only way to know that an inspector has completed the inspection is to wait for the email generated at the end of the phase.

CSRS [4] (collaborative software review system) 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. CSRS distinguishes between a private review phase, where individual review of the artefact is performed and annotations are hidden to the other inspectors, and public review phase, which represents an asynchronous discussion. Differently from us, no support for synchronous discussion is provided. Moreover, CSRS supports only plain text artefacts.

ASSIST [26] (asynchronous/synchronous software inspection support tool) like CRSS is designed to support any inspection process. To this aim it uses an inspection process definition language. Another goal of the author was to reduce the effort required during the inspection. To this aim, the tool provides a checklist browser implementing active checklists to record answers, monitor the checklist usage and visualise cross-references (e.g. to show the same word appearing in different documents). Meeting support is provided by means of video and audio tools, and a whiteboard for a textual discussion. 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. Differently from WAIT, ASSIST is not integrated within a software project management system, thus the inspection process must be managed separately from the development process.

CAIS [8] is a tool that supports only asynchronous discussions for a software inspection in distributed environment. The tool uses CSI to create defects lists and organises the inspection meeting as a sequence of contributions (comments and votes) to a discussion. Moreover, differently from WAIT, the awareness concerning the inspection process is not guaranteed by a notification mechanism.

Jupiter [27] is an inspection support tool developed as an Eclipse plug-in. A case study carried out with nine undergraduate students and 16 graduate students revealed that this tool is more usable and useful than the text-based code review. An evaluation Jupiter has been conducted and the results of its effectiveness have been reported in [27]. The prototype only addresses the inspection of source code and does not support geographical dispersed reviews.

Perpich *et al.* [28] present a web-based tool, named Hypercode, to support software inspections with geographically distributed inspectors. However, no support for synchronous and asynchronous discussions and inspection process support are provided.

Lanubile *et al.* [29] propose a web-based tool, called IBIS (internet-based inspection system), which adopts a reengineered inspection process to minimise synchronous activities and coordination problems and reduce the overall cost and time of the inspection process. In particular, starting from the reorganisation of the inspection process proposed by Sauer *et al.* [30], they replace the preparation and meeting phases of the process proposed by Fagan 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. Similarly to WAIT, IBIS has been evaluated through controlled experiments [29]. The main difference as compared to WAIT is the support provided to the moderator about the ongoing inspection and in the inspection process supported.

Moreover, none of the tools presented before 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.

Table 1 summarises the main functionalities provided by the discussed online inspection tools.

It is worth noting that, being all online computer supported inspection tools, none of the considered tool has a low level of flexibility, since they all allow distributed inspection teams. However, the level of flexibility for CSI, ICICLE and Scrutiny is lower as compared to the other tools, since they require all the inspectors be available at the same time. Moreover, despite being classified as a synchronous tool, InspeQ does not expressly provide support for group meeting, thus its level of flexibility is the lowest as compared to the other tools.

Finally, all the online inspection tools, except Hypercode, implement a software inspection process. Some of them are also checklist based. As a consequence, the level of flexibility is high for most of the considered tools.

3 Distributed inspection process and tool support

The inspection process implemented within WAIT is composed of seven phases, namely Planning, Overview, Discovery, Refinement, Inspection meeting, Rework and Follow up. During the Planning phase, the quality manager specifies which artefact version must undergo a formal review process, defines or selects an existing checklist, and

Table 1 Comparison of discussed online inspection tool features

	AISA	ASSIST	CAIS	CSI	CSRS	Hypercode	IBIS	ICICLE	InspectA	InspeQ	Jupiter	Scrutiny	WAIT
Supported artefact type													
graphical	•												
text/code					•			•	•		•	•	
any		•	•	•		•	•			•			•
Inspection process support													
checklists		•	•	•			•		•	•			•
defect classification	•	•	•	•	•		•	•			•	•	•
decision support	•	•	•		•						•	•	•
inspection data collection		•	•	•	•		•	•				•	•
customisable inspection process		•			•								
Meeting/discussion													
synchronous discussion		•		•			•	•		•		•	•
asynchronous discussion	•	•	•		•	•	•		•		•		•
Process awareness													
project management													•
email notification		•		•	•	•			•		•		•
evaluation						•	•			•	•		•
flexibility	high	high	high	high	high	high	high	high	high	high	high	high	high
discipline	high	high	high	high	high	low	high	high	high	high	high	high	high

composes the inspection team. When this phase is complete, the inspectors receive a notification containing the inspection details and a new task appears in their to-do-list.

In the Overview phase, the author of the artefact 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. The inspection mailing list is used to notify all the inspection participants. Please note that this phase is not mandatory, so the moderator can decide to skip it in case no details are required to perform the subsequent phase.

During the Discovery phase, each inspector analyses the artefact and takes note of the candidate defects by highlighting all the cases where the artefact does not comply with the check items in the checklist. WAIT supports the inspectors during this phase by recording the identified defect, its severity and its location within the software artefact in terms of page and line numbers, or picture/table number. The inspector can also include a brief comment describing the reason why it contrasts with the check item. Anytime during 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 useful to decide whether to stop the detection phase and start the next phase.

When the Discovery phase is completed, the moderator accesses the defect log, containing all the defects identified by the inspectors. In case the inspectors disagree with the

IET Softw., 2009, Vol. 3, Iss. 5, pp. 381–394

doi: 10.1049/iet-sen.2008.0101

defects for a check item, WAIT highlights it, allowing the moderator to decide if the Refinement phase should be enacted. In this case, the tool sends a notification message containing the list of conflicts to the inspection participants. This message also aims at notifying the team members that the conflicts can be analysed in order to obtain an agreement. The main goal of this phase is to remove false defects and to build the consensus on true defects. Similarly to Lanubile *et al.* [29], we consider a defect as a true defect when at least two inspectors identify it. In this case the defect is not highlighted in the defect log. It is worth noting that the minimum number of reviews required to automatically obtain an agreement could be differently chosen by the inspection manager according to the inspection process constraints and the number of inspectors.

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 different types of conflicts and provides hypertextual links to the defect details. By using the defect details, the inspector can decide 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 all the members of the inspection team. Hence, when an inspector solves a conflict, it is removed even from the list of all the other inspectors. When all the conflicts for a check item are solved, the corresponding highlighting is also removed. Note that this phase is not mandatory, so the moderator can decide to skip it and directly resolve the conflicts on the identified defects (e.g. in case of a low number of defects).

A synchronous inspection meeting can be performed in the Inspection meeting phase to discuss about unsolved conflicts. Even this phase can be skipped (e.g. in case the number of conflicts is manageable by the moderator or the time distance does not permit a synchronous discussion). It is worth noting that our tool does not require that all the participants join the inspection meeting, thus enabling the enactment of sub-teams meeting to resolve conflicts not involving other inspectors.

The author can use the produced defect log to fix the artefact during the Rework phase. Let us note that a different defect log is maintained for each version of a software artefact. Thus, in case the artefact undergoes several revisions, it is possible to access the defect logs for each version.

Similarly to the Fagan's process, during the Follow-up phase the moderator checks the quality of the revised artefact and determines whether a new inspection is needed.

4 Controlled experiments

In this section we describe the design of both the controlled experiments according to the guidelines proposed by Wohlin *et al.* [31]. In the following, these controlled experiments are referred to as Experiment I and Experiment II, respectively.

4.1 Experiment definition and context

To investigate whether the choice of different approaches significantly affects the time and quality of the inspection, two controlled experiments involving second year Master students in Computer Science were conducted in the Software Engineering research laboratory at the University of Salerno. Experiment I aimed at comparing the efficiency and the effectiveness of WAIT and the structured inspection process proposed by Michael Fagan [3]. In particular, the Fagan's inspection process is a formal, efficient and economical method to find errors in software artefacts produced both in the design and coding phases. To this end, Fagan proposes a structured process, which consists of five sequential phases: Overview, Preparation, Inspection, Rework and Follow up. In the Overview phase the designer first describes the overall domain area being addressed and then provides details about the specific area he/she has designed. Once this phase is concluded, documentation concerning the software artefact to inspect is distributed to all the inspection participants. Participants, using this documentation, do their 'homework' trying to understand the design, as well as its intent and logic. The checklist adopted in the following phase of the process has to be studied and deeply understood. During the Inspection phase, the moderator chooses a reader (usually the author) and a face-to-face meeting is carried out. Within one day of the conclusion of the inspection, the moderator should produce a written report, which is provided as input to the Rework phase. During the Rework phase, the designer or coder/implementer addresses the defects identified during the Inspection phase. Finally, during the Follow-up phase the moderator checks the quality of the rework and determines whether a re-inspection is needed.

WAIT and the PI methods have been compared in the second experiment, namely Experiment II. PI is a very flexible and undisciplined way to review software artefacts. Only two people are required to inspect a software artefact, that is the author and an inspector who reviews the author's artefact. In some cases the author could lack. In this case two inspectors conduct the inspection. PI requires continuous iterations, so any strict rules can be formulated to guide the inspection. Owing to the iterative nature of the process, the recording of defects found in informal meetings is not required. A form or a template is adopted to annotate the defects of the software artefact under review. In some cases, the defects are annotated on paper documents and corrected during the next inspection cycle. A checklist can be optionally used to drive the inspectors

during the review of a software artefact. In addition, the author and inspector together can also write down comments for the acceptance review.

The main goal of the experimentation presented here concerns the investigation of how the discipline and flexibility dimensions affect the effectiveness and efficacy of the inspection of low-level software artefacts. To this end, two are the defined research questions that we aim at addressing:

Q1. Is the distributed inspection process implemented within WAIT more efficient and effective than the FAG to inspect a software artefact?

Q2. Is the distributed inspection process implemented within WAIT more efficient and effective than the PI method to inspect a software artefact?

The research question Q1 will be addressed in Experiment I, whereas the research question Q2 will be investigated in Experiment II.

Both the experiments involved 18 students. The subjects were volunteers and were grouped into six inspection teams. All the involved subjects conducted a controlled experiment as a series of optional laboratory sessions conducted within a Software Engineering course. Note that each subject performed only one experiment. This was possible as the students involved in the experiments attended the Software Engineering course within two subsequent academic years.

It is worth to point out that we posed great care to ensure that ethical requirements imposed by the Italian legislation were met. In fact, the subjects were asked to sign a document where the following sentence was written: 'Data collected will be used only for research purposes and they will be revealed only in aggregated form'. The subjects were also informed that: (i) the experiment had a pedagogical purpose, (ii) for privacy reasons the results were made anonymous and (iii) students were not evaluated on their performance.

4.2 Hypotheses

To address the research questions presented above, for each experiment two null hypotheses have been defined. These hypotheses aim at assessing the efficacy and the effectiveness of our distributed process, respectively. To assess the efficacy of the distributed process and the supporting tool, the following null hypothesis has been defined:

- H_{n1}: the use of WAIT does not significantly affect the time to inspect the software artefact.

The alternative hypothesis is

- H_{a1}: the use of WAIT significantly affects the time to inspect the software artefact.

On the other hand, we also investigated the effectiveness of the distributed process implemented in WAIT with respect to the other two inspection methods (i.e. Fagan's process and PI method). To this end, the following null hypothesis has been formulated:

- H_{n2}: the use of WAIT does not significantly affect the inspection quality.

The related alternative hypothesis is:

- H_{a2}: the use of WAIT significantly affects the inspection quality.

4.3 Selected variables

In order to properly design the experiment and analyse the results, we considered the following independent variables:

- Method: this variable indicates the factor the studies are focused on, namely the investigated inspection methods, WIT (WAIT inspection tool), FAG and PI.

- Lab: the experiments are organised in two subsequent laboratory sessions, also named runs (i.e. Lab1 and Lab2). The runs were subsequently performed to reduce the possibility that subjects exchange information between Lab1 and Lab2.

- Task: this variable indicates the inspection tasks to perform in both the experiments.

In particular, the inspection tasks the subjects were asked to perform in both the experiments are:

- T_1: inspecting a Java class implementing a binary tree data structure and the algorithms to traverse and modify it;

- T_2: inspecting a Java class enabling the construction and the execution of queries on a relational database.

Some statistics of the tasks are presented in Table 2. It is worth mentioning that the defects within the tasks were uniformly distributed, thus reducing their effect on the experimental results.

Table 2 Statistics of the tasks

	T1	T2
lines of code	166	145
lines of comments	27	31
number of methods	12	7
number of global variables	4	13
number of inner classes	3	0
number of defects	78	56

doi: 10.1049/iet-sen.2008.0101

The use of source code within the two tasks is due to the fact that the subjects were familiar with procedural and object oriented programming languages. Therefore they can be considered not far from junior industry programmers, thus improving the external validity of both the experiments. Note that the defined inspection tasks were expected to be accomplished within 2 h.

To verify the null hypotheses H_{n1}, we considered the following dependent variable:

- Time: the sum of the times that the team members spent to perform the task.

The quality of an inspection was assessed using two well-known measures, namely precision and recall. In our case the recall is defined as the ratio between the number of actual defects identified by the team and the total number of actual defects, whereas the precision is ratio between the number of actual defects identified by the team and the total number of identified defects. These measures range between 0 and 100%. If the recall is 100%, it means that all the true defects have been identified, even if there could be identified defects that are false defects. If the precision is 100%, it means that all the identified defects are correct, even if there could be correct defects that were not identified. This indicates that the precision and recall measures abstract different and specific concerns. Hence, to obtain a balance between these measures the harmonic mean (i.e. *F*-measure [32]) was adopted. This measure is defined as follows

$$F\text{-measure} = 2^* \frac{\text{precision}^* \text{recall}}{\text{precision} + \text{recall}}$$

As a result, to test the null hypothesis H_{n2} the following dependent variable has been considered:

- *F*-measure: the harmonic mean of the precision and recall values.

4.4 Experiment design

Although we designed the experiments to avoid the task effect on the considered dependent variables, the Task factor has to be tested to ensure that it does not influence the results. Hence, the possible combinations of Method and Task factors represent the considered treatments. To avoid results to be biased by group ability, each group experienced both the inspection methods and both the tasks over two laboratory sessions.

4.4.1 Experiment I: The design of Experiment I is summarised in Table 3. We randomly assigned inspection teams to the groups A–D. Since the number of teams was six and the number of groups was four, two groups had two teams assigned namely A and B, and two groups had one team assigned, namely C and D. A researcher, who played the role of moderator within the team, supported each team. Moderators were asked to restrict themselves to coordinate the inspections. All combinations of the Method (FAG and WIT) and Task (T1 and T2) factors represent the considered treatments. Also, to minimise the learning effect, we needed to have teams starting to work in Lab1 both with and without the tool on both the tasks.

Table 3 Design of Experiment I

Subject's group	Method	
	FAG	WIT
A	T_1, Lab1	T_2, Lab2
B	T_2, Lab2	T_1, Lab1
C	T_2, Lab1	T_1, Lab2
D	T_1, Lab2	T_2, Lab1

4.4.2 Experiment II: Table 4 summarises the design of Experiment II. We randomly assigned inspection teams to groups A and B. These groups had three teams assigned. In the first laboratory session, each team was composed of three students and a researcher. Similarly to Experiment I, the researchers played the role of moderator without participating in the identification of the defects within the software artefact of the tasks. When the first laboratory session was concluded we removed the moderator and one subject from each team belonging to groups A and B. The removed subject was randomly selected. This was necessary because the PI needs only two inspectors for each team. Similarly to Experiment I, all the combinations of the Method and Task factors represent the considered treatments.

4.5 Preparation

All the subjects attended an introductory training session on the usefulness of the inspection methods to detect and remove defects within software artefacts produced in the whole development process of a software system. Furthermore, the subjects of Experiment I attended a training session where the Fagan's inspection method was deeply presented and discussed. Examples (not related to the tasks to avoid biasing the experiment) were also proposed. On the other hand, details on the PI method were provided to the subjects of Experiment II. Even in

Table 4 Design of Experiment II

Subject's group	Method	
	WIT	PI
A	T_1, Lab1	T_2, Lab2
B	T_2, Lab1	T_1, Lab2

this case we discussed examples not related to the experiments. Note that the subjects of both the experiments also attended a training session on our distributed inspection process and tool. To give subjects more confidence with the tool, some examples were also presented. The training sessions were concluded presenting detailed instructions on the tasks.

The attended training sessions aimed at providing all the subjects an equal prior knowledge and to deeply describe the software application and the investigated inspection methods. At the end of each laboratory session a post-experiment survey questionnaire was submitted to the subjects (see Table 5). This questionnaire has been mainly employed to assess the overall quality of the provided material and the clearness of the laboratory session goals and of the inspection tasks. The questions expected closed answers according to a five-point Likert scale [33]: 1 (strongly agree); 2 (agree), 3 (neither agree nor disagree), 4 (disagree) and 5 (strongly disagree).

4.6 Material, execution and data analysis

To perform the experiment each subject was provided with a folder containing a pencil, white sheets and the following hard copy material:

1. the introductory presentation of the training session;

2. the guidelines to perform the assigned tasks according to our inspection method and the FAG;

3. the source code of the class to be inspected in the two tasks;

4. a checklist to be used to perform the inspection tasks. The checklist (see Table 6) was the same for each treatment within the experiments;

5. an inspection defect log template to be used to perform the tasks when the subjected used FAG and PI; and

6. the post-experiment survey questionnaires to be filled in at the end of each laboratory session.

Table 5 Post-experiment survey questionnaire

Id	Question
Q1	I had enough time to perform the inspection task
Q2	the task objectives were perfectly clear
Q3	performing the task was easy
Q4	the supporting material to perform the task provided enough information
Q5	the checklist was clear and well structured
Q6	the defect identification was easy

Table 6 Checklist used within the experiments

the class is properly commented
there is a comment for each declared variable
the comments do not negatively influence source code readability and are always useful
there is correspondence between the methods signature and the comments
the scope and behaviour of the classes are properly clarified by comments
the comments properly describe the source code in close proximity
the comments are not banal
the names of the constants, variables, classes, and packages are compliant with the established naming convention (e.g. CONSTANT_NAME, variableName, ClassName, packageName.subpackageName)
the names of the classes and variables are meaningful
the information hiding is respected
the output of each method is consistent with the name
before using an object that could be null its state is always verified
the code is properly indented
each method implements a specific functionality

The inspection teams accomplished each laboratory session of the experiment without time limit. When the experiments were concluded the supervisors collected the post-experiment survey questionnaires, the log files containing the information traced by WAIT as well as the defect reports. The supervisors also gathered information (i.e. time spent and defect logs) about the inspection tasks performed by using the FAG and PI methods. The defect logs have been also analysed in order to compute the precision and recall values for each treatment. To this end, two experts not involved in the development of WAIT analysed the defect logs. These worked independently and iterated until they reached an agreement. The experts adopted the same checklist used by the inspection teams during the task execution.

Owing to the low number of observations, we used parametric tests in the data analysis of both the experiments. In particular, to reject the defined null hypotheses we used the Wilcoxon test [34], whereas to verify the absence of learning or tiring effects we used the Kruskal–Wallis test. In the first experiment the Kruskal–Wallis test was employed to analyse the influence of the Task and Lab factors on the selected dependent variables. On the other hand, in Experiment II this test was used to investigate the influence of the Task factor.

doi: 10.1049/iet-sen.2008.0101

Owing to the design of Experiment I (i.e. factorial design with confounded interaction), the order the subjects performed the tasks within the two subsequent laboratory sections (i.e. order of methods) needed to be controlled [35]. In fact, the order of methods may produce learning effects, which may bias the results in terms of time to accomplish the tasks and inspection quality. Accordingly, for each subject x and for the time and F-measure dependent variables we computed:

$$\text{Diff}_x = \text{observation}_x(\text{FAG}) - \text{observation}_x(\text{WIT})$$

Diff(WIT) represents the performance difference of the subjects, who used WIT first and then FAG, whereas Diff(FAG) indicates the performance difference of the subjects, who used FAG first and WIT second. To test whether Diff(FAG) is larger than Diff(WIT), we used the non-parametric two-sample Kolmogorov–Smirnov test. We expected Diff(FAG) to be larger than Diff(WIT) on both the dependent variables. This is because the subjects' ability to use WIT may improve through laboratory sessions. As a result, we needed to verify the null hypothesis H_{0t}: Diff(FAG) = Diff(WIT) on the time dependent variable. The alternative hypothesis is as follows: H_{at}: Diff(WIT) $<>$ Diff(FAG). Similarly, we defined the null hypothesis H_{0f} on F-measure. The alternative hypothesis H_{af} can be easily derived as well.

Let us note that the results of the tests used here are intended as statistically significant at $\alpha = 0.05$.

5 Results

In this section we present the results of the experiments including the results of the post-experiment survey questionnaire.

5.1 Experiment I

Table 7 shows some descriptive statistics (i.e. minimum, maximum, mean and standard deviation) – grouped by Method – on the time and F-measure dependent variables.

Table 7 Descriptive statistics of Experiment I

	Minimum	Maximum	Mean	Std. deviation
FAG				
Time	147	259	202	47.24
F-measure	55	74	65.66	6.31
WIT				
Time	127	235	171	39.80
F-measure	52	77	67	10

The subjects spent on average less time to accomplish the task using WIT. On average a slight difference in terms of the F-measure values was observed between the teams using or not our inspection process.

The Wilcoxon test revealed that the null hypotheses H_{n1} and H_{n2} cannot be rejected as the p-values were 0.463 and 0.600, respectively. To further investigate this concern, we analysed the needed time to inspect the software artefact within the tasks and the F-measure values achieved by the subjects. In particular, Table 8 shows the number of subjects that using WIT spent less time to accomplish the tasks (WIT < FAG) and the number of subjects that using WIT spent more time (WIT > FAG). Similarly, the results on the F-measure dependent variable are presented as well.

5.1.1 Influence of Task, Lab and Order of method:

The Kruskal–Wallis test revealed that the influence of the Task factor was not statistically significant concerning the Time-(p-value = 0.199) and F-measure (p-value = 0.748) dependent variables. Even, the influence of Lab was not statistically significant on Time (p-value = 0.05). However, the significance level of the test is very close to 5%. A further analysis revealed that the subjects on average spent less time to accomplish the second laboratory session. This could be because of the fact that in the first session they spent time to get familiar with the inspection methods (i.e. FAG and WIT). On the other hand, the the Kruskal–Wallis test revealed that the influence of Lab was not statistically significant on the F-measure dependent variable (p-value = 0.810).

Finally, the two-sample Kolmogorov–Smirnov test revealed that the null hypothesis H_{0t} could not be rejected (p-value = 0.518). This means that the second laboratory session does not bring significantly larger performance differences in terms of time to accomplish the tasks. Even, the null hypothesis H_{0f} cannot be rejected (p-value = 0.996), thus indicating that the second laboratory session does not bring significantly better results with respect to F-measure.

5.1.2 Post-experiment survey questionnaire for Experiment I:

Fig. 1 visually summarises the data collected from the survey questionnaire of Experiment I. In particular, the answers corresponding to FAG and WIT are summarised by the bars on the left- and right-hand sides, respectively. The analyses of the collected data showed that the time to perform the inspection tasks was

Table 8 Performance differences according to WIT and FAG

Dependent variable	WIT < FAG	WIT > FAG
Time	4/6	2/6
F-measure	3/6	3/6

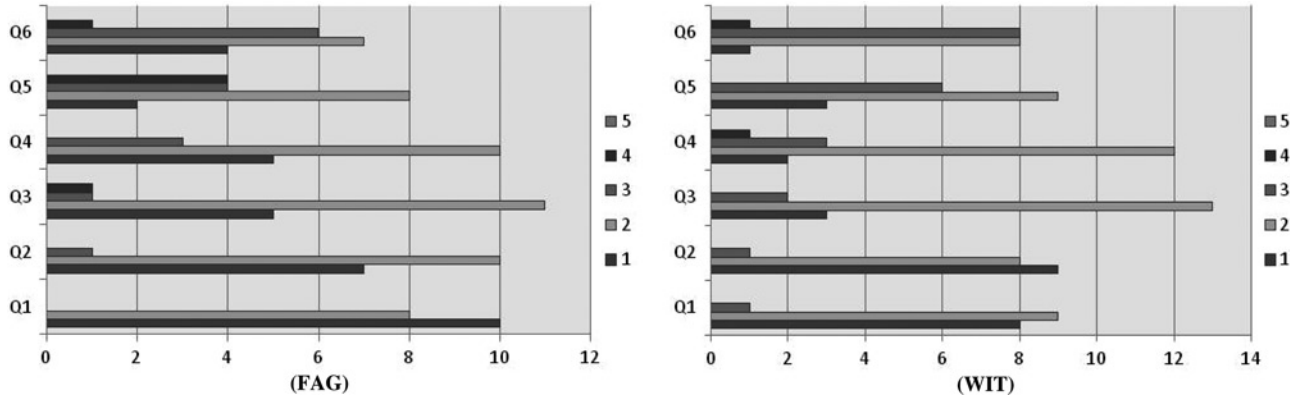

Figure 1 *Results of post-experiment survey questionnaire of Experiment II*

considered appropriate as well as their objectives (see questions Q1 and Q2). The bars of Q3 and Q4 concerning the FAG and WIT methods present some differences. Despite this, the general judgment of the subjects is satisfactory. Therefore the subjects expressed on the average a positive judgment on the inspection task clarity as well as on the hard copy material. Moreover, the checklist was considered clear and well structured (see Q5). The subjects used the FAG and WIT methods generally provided different answers on Q6. In particular, the subjects that performed the task using the FAG method, on the average, found the defect identification simpler than the subjects that used the tool. This is the case where a controlled experiment provides an insight into the difference between the perceived usefulness of a given method and the effective advantage of using it. To better address this point replications on a larger dataset are however needed.

5.2 Experiment II

The descriptive statistics of Experiment II – grouped by Method – on the Time and *F*-measure dependent variables are shown in Table 9. On the average the subjects spent more time to accomplish the tasks using the WIT, while a slight difference was observed in terms of the *F*-measure value.

Table 9 Descriptive statistics of Experiment II

	Minimum	Maximum	Mean	Std. deviation
PI				
Time	77	104	85.5	10.7
F-measure	62	83	69	8.02
WIT				
Time	127	237	186.33	45.47
F-measure	55	85	71.83	12.10

The Wilcoxon test revealed that the null hypothesis H_{n1} can be rejected as the p-value was 0.028. This indicates that use of WAIT significantly affects the time to perform the tasks. In particular, we observed that the subject on average spent more time when they used WAIT to accomplish the task. This is an expected result as the PI method has a low level of discipline with respect to the distributed inspection process implemented in WAIT. The null hypothesis H_{n2} cannot be rejected as the Wilcoxon test revealed (p-value = 0.462). This indicates that the investigated method does not significantly affect the inspection quality.

As for Experiment I, we further analysed the time and the harmonic mean of precision and recall values that the subjects achieved in the two subsequent laboratory sessions (see Table 10). We can observe that the inspection teams obtained better *F*-measure in case the WIT method was used. In particular, four teams out of six benefit more using WIT.

5.2.1 Influence of Task: The influence of Task was not statistically significant on the Time dependent variable as the Kruskal–Wallis test revealed that (p-value = 0.470). This indicates that there is no a significant difference in the time to perform the inspection tasks T1 and T2 in the two laboratory sessions. The Kruskal–Wallis test also indicated that the Task factor is not significant on the *F*-measure dependent variable (p-value = 0.172).

5.2.2 Post-experiment survey questionnaire for Experiment II: The data collected from the survey questionnaire of Experiment II are visually summarised in Fig. 2. In particular, the bars on the left and right hand sides summarise the answers of the post-experiment survey

Table 10 Performance differences according to WIT and PI

Dependent variable	WIT < PI	WIT > PI
Time	0/6	6/6
F-measure	2/6	4/6

doi: 10.1049/iet-sen.2008.0101

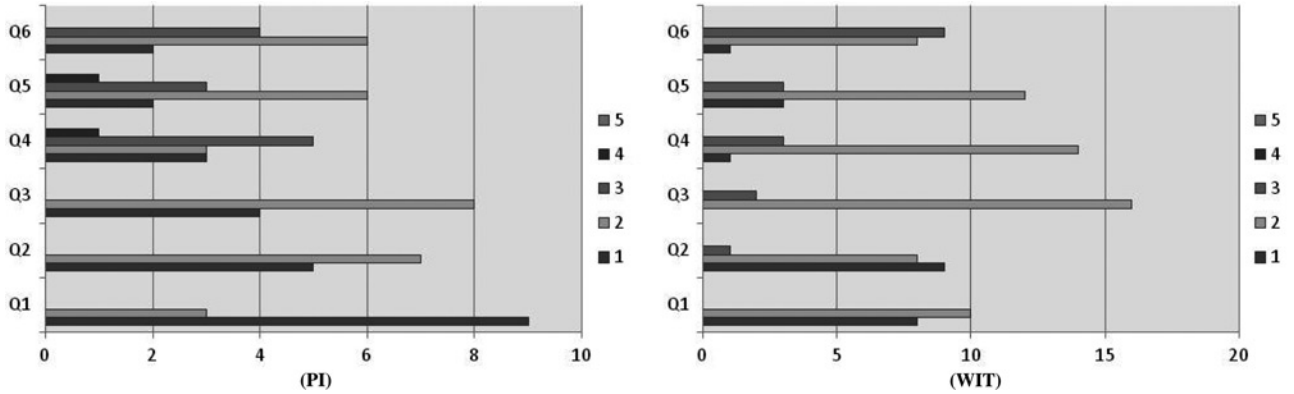

Figure 2 *Results of the post-experiment survey questionnaire of Experiment II*

questionnaire regarding PI and WIT, respectively. The time to perform each inspection task was considered appropriate (see question Q1). However, the subjects that performed the task using PI provided a better judgment. Moreover, the tasks objectives were considered clear as the bars for question Q2 show. The answers of the questions from Q3 to Q5 concerning the FAG and WIT methods present some differences. In particular, the subjects using WIT provided a better judgment. However, all the subjects expressed on the average a positive judgment on the inspection task clarity, on the provided hard copy material and on the checklist. Regarding the question Q6, the subjects that performed the task using WIT generally found the defect identification more complex than the subjects that used PI.

6 Discussion

The controlled experiments showed that the only dimension that influences the inspection is the discipline. In particular, the time needed to inspect the software artefact is the only factor influenced by the discipline dimension. In fact, the data analysis revealed that there is not a statistical significant difference on the time needed to accomplish the task when using the methods with a high level of discipline (i.e. the WIT method and the FAG). On the other hand, a significant difference was observed in case we compared the WIT method to the method with a low level of discipline (i.e. the PI method). Differently, the discipline dimension does not significantly influence the inspection quality.

Moreover, considering methods according to the flexibility dimension we observed that both Time and *F*-measure are not statistically significant. In fact, Experiment I showed that Time and *F*-measure were not significantly affected by the use of the WIT method (high level of flexibility) or the Fagan's process (low level of flexibility). This result was confirmed in Experiment II, where the compared methods had a comparable flexibility level.

Despite the differences when using or not our tool are not significant with respect to FAG, the experiment revealed that the subjects spent less time as compared to the FAG, while the overall quality of the results is comparable. To further investigate the inspection results we analysed the actual and false defects that the subjects identified using FAG and WIT. In particular, we observed that they identified a larger number of actual defects and false positives when using our tool. This could be motivated with the fact that the inspectors performed a more rigorous detection phase when our tool was used. This emerged by the analysis of the description of the defects provided with the defect logs. In fact, when the FAG was used, we observed that the description of the obvious defects was poor and informal. Differently, a more precise and formal description was provided when our tool was used. Such a better defect description could also have affected the larger number of false positives. Indeed, the description of the defect identified by the inspector could have convinced the remaining team members, thus inducing them to erroneously remove the conflict. Note that this represents an acceptable drawback in a distributed setting, where the possibility of reducing or avoiding face-to-face meetings or synchronous discussions is always welcomed.

Similarly to Experiment I, we analysed the actual and false defects that the subjects identified using WIT and PI in Experiment II. This analysis showed that the total number of defects is nearly the same for all the investigated methods. On the other hand, the teams identified a larger number of false defects when using PI. This could be because of the low number of inspectors involved in the discussion of the possible defects. In fact, with a lower number of inspectors an agreement is easier to be achieved.

The results of the post-experiment survey questionnaires showed that the subjects that performed the inspection tasks using FAG and PI found, on the average, the defect identification simpler than the subjects that used WAIT. This aspect could be because of the fact that the subjects had to interact with each team member through a tool in order to accomplish each phase of the inspection process. Probably, the definition of a distributed approach with a lower disciplined level could improve the subjects'

satisfaction to inspect software artefacts. Therefore a possible direction for future work could be the definition of a distributed inspection process with high flexibility and discipline.

The distribution of the identified defects within the software artefacts (in both the experiments) is nearly the same whatever inspection method is used. Nevertheless, this is an interesting point that could be further investigated in the future. In fact, it could be useful to understand whether an inspection method is more suitable than others to identify a specific kind of defect.

In order to comprehend the strengths and limitations of our empirical investigation, threats that could affect its internal, construct, external and conclusion validity need to be discussed. Internal validity threats are relevant in studies that try to establish a causal relationship. In particular, the presented experiments aimed at concluding that our approach produces different outcomes with respect to the FAG and the PI. Generally, the key question is whether the observed differences can be attributed to the learning effect and not to other possible causes. The internal validity threats are mitigated by the designs of both the experiments, since each group worked, over the two Labs, on different tasks and with two different methods. Additional considerations are due about the second experiment. In fact, despite its careful design there is still the risk that subjects might have learned how to improve their performances in the second laboratory session. However, this could only positively affect the needed time and the inspection quality of the subjects when experimenting PI. It is worth noting that the subjects found clear everything regarding the tasks both in Experiment I and Experiment II.

The construct validity threats (i.e. the interactions between different treatments) were mitigated by a proper design of the experiments that allowed separating the analysis of the different factors and of their interactions. Moreover, depending on Method, the measurements of the dependent variables were performed either analysing the log files produced by the tool or considering the times gathered by the experiment supervisors (one for each team). The questionnaire was designed using standard ways and scales [33]. Let us note that the construct validity could be affected by the true and false positive defects that experts manually identified to compute precision, recall and *F*-measure. Even the measure (i.e. *F*-measure) used to get a quantitative evaluation of the inspections could condition the achieved results. Social threats (e.g. evaluation apprehension) could also affect the observed results. To mitigate this effect, the subjects were not evaluated on the results they obtained and were not aware of the experimental hypotheses.

External validity threats are always present when experiments are performed with students. Generally, last-year Master students have a good analysis, development and programming experience, and they are not far from junior industry programmers. Another threat that could affect the external validity concerns the type of software artefacts used within the experiments (i.e. the source code) and the Java classes to be inspected in the tasks (e.g. the subjects could be very familiar with the class implementing a binary tree). The complexity and the size of the used artefacts could influence the external validity as well. To confirm or contradict the achieved results replications with subjects of different academic and industrial contexts should be performed on larger, different and more complex software artefacts. For example, replications are needed on software artefacts produced in the early phases of the software development process (e.g. analysis and design).

Conclusion validity threats concerns the issues that affect the ability of drawing a correct conclusion. Even, the conclusion validity threats were mitigated by the design of the experiments. Regarding the recruited subjects, we drew a fair sample from that population and conducted the experiments with subjects belonging to this sample. Moreover, proper tests were performed to statistically reject null hypotheses. In cases where differences were present but not significant, this was explicitly mentioned and analysed. Non-parametric tests were used in place of parametric tests because of the low number of observations.

7 Conclusion and further work

This paper presented two controlled experiments aimed at comparing three software inspection methods, in terms of efficacy and efficiency. These experiments involved Master students in Computer Science at the University of Salerno. The students had to discover defects within software artefacts (i.e. Java classes) using inspection methods that differ in terms of discipline and flexibility. In particular, in the first controlled experiment we compared the disciplined and flexible inspection process implemented in WAIT to a disciplined but not flexible method (i.e. the Fagan's process). In the second experiment WAIT was compared to a flexible but not disciplined method (i.e. the PI). The experiments revealed that the only significant factor is the time. In particular, less disciplined methods need less time to inspect the software artefact. One of the main objectives here is to provide a software quality manager with some insight into choosing the inspection method to adopt according to the projects needs and constraints.

As mentioned above, a further work we have performed concerns the integration of WAIT within an artefact-based process support system (i.e. ADAMS [11, 36]), thus providing a more effective support of the quality management within the development process of a software system. The advantages of integrating WAIT within ADAMS are manifold. First of all, the possibility of allocating human resources on the software artefact to review, planning inspection-related tasks within the project schedule, capturing the minute resulting from synchronous/asynchronous discussions and associating it to the

IET Softw., 2009, Vol. 3, Iss. 5, pp. 381–394

doi: 10.1049/iet-sen.2008.0101

corresponding artefact. A further advantage of this integration concerns the possibility of using functionalities to notify the accomplishment of each phase of the inspection process. Finally, this integration also enables to maintain information regarding the inspections of all the versions of a given software artefact and to understand how the identified defects evolve during the different inspections. More details can be found in [9].

The support provided by the system resulting from the integration of WAIT within ADAMS has been preliminary assessed within the projects conducted by some students of the Computer Science programme at the University of Salerno. The students were allocated on software projects including between three and sixteen Bachelor students (2nd year B.Sc.) with development role. On the other hand, one or two Master students (2nd year M.Sc.) were involved with roles of project and quality management. Each project manager was responsible for coordinating the project, defining the project schedule, organising project meetings, collecting process metrics and allocating human resources to tasks. Quality managers were responsible for defining process and product standards of the project, collecting product metrics and managing the artefact reviews for quality control.

This investigation confirmed the main results of the empirical experimentation presented here and provided a number of directions to improve the usefulness of the inspection tool in the quality management of ADAMS. A first direction is 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.

Future work will be also devoted to conduct controlled experiments and case studies within a software industrial context to increase the body of knowledge about the efficacy and effectiveness of inspection methods in the global software development and to provide feedback on whether the technologies produced in research laboratories fulfil the industry needs. We also plan to investigate the impact of face-to-face against tool-mediated meeting as compared to the number of true defects and false positives. This can be useful in case time distance is not an issue, but moving people might be a problem. Finally, future work will be devoted to the investigation of over-simplification problems in formal experiments [37]. In particular, we will concentrate on different types of benefit, for example the development of team spirit and the transferring of technology among the inspection participants.

8 References

[1] FREEDMAN D.P., WEINBERG G.M.: 'Handbook of walkthroughs, inspections, and technical reviews: evaluating programs, projects, and products' (Little Brown & Co., 1982, 3rd edn.)

[2] 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

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

[4] JOHNSON P.M.: 'An instrumented approach to improving software quality through formal technical review'. Proc. 16th Int. Conf. on Software Engineering, Sorrento, Italy, 1994, pp. 113–122

[5] TERVONEN I., IISAKKA J., HARJUMAA L.: 'Software inspection – a blend of discipline and flexibility'. Proc. ENCRESS-98, 1998, pp. 157–166

[6] BROTHERS L.R., SEMBUGAMOORTHY V., MULLER M.: 'ICICLE: Groupware for code inspections'. Proc. 1990 ACM Conf. on Computer Supported Cooperative Work, Los Angeles, CA, USA, 1990, pp. 169–181

[7] HUMPHREY W.S.: 'Managing the software process' (SEI series in software engineering, Addison-Wesley Longman Publishing, Boston, MA, USA, 1989)

[8] MASHAYEKHI V., FEULNER C., RIEDL J.: 'CAIS: collaborative asynchronous inspection of software', *SIGSOFT Softw. Eng. Notes*, 1994, **19**, (5), pp. 21–34

[9] DE LUCIA A., FASANO F., SCANNIELLO G., TORTORA G.: 'Integrating a distributed inspection tool within an artefact management system'. Proc. Second Int. Conf. Softw. and Data Technologies, Barcelona, Spain, 22–25 July 2007, pp. 184–189

[10] DAMIAN D., LANUBILE F., MALLARDO T.: 'On the need for mixed media in distributed requirements negotiations', *IEEE Trans. Softw. Eng.*, 2008, **34**, (1), pp. 116–132

[11] DE LUCIA A., FASANO F., FRANCESE R., TORTORA G.: 'ADAMS: an artefact-based process support system'. Proc. 16th Int. Conf. on Software Engineering and Knowledge Engineering, Banff, Alberta, Canada, 2004, pp. 31–36

[12] BASILI V.R., SELBY R.W., HUTCHENS D.H.: 'Experimentation in software engineering', *IEEE Trans. Softw. Eng.*, 1986, **12**, (7), pp. 733–743

[13] PFLEEGER S.L., MENEZES W.: 'Marketing technology to software practitioners', *IEEE Softw.*, 2000, **17**, (1), pp. 27–33

[14] REDWINE S.T., RIDDLE W.E.: 'Software technology maturation'. Proc. Eighth Int. Conf. on Software Engineering, London, UK, 1985, pp. 189–200

[15] KOLLANUS S., KOSKINEN J.: 'Survey of software inspection research: 1991–2005'. Computer Science and Information

Systems Reports, Working Papers WP-40, Dept. of Computer Science and Information Systems, Univ. of Jyväskylä. Jyväskylä University Printing House, Jyväskylä, 2007

[16] LAITENBERGER O., DEBAUD J.M.: 'An encompassing life cycle centric survey of software inspection', *J. Syst. Softw.*, 2000, **50**, (1), pp. 5–31

[17] IISAKKA J., TERVONEN I., HARJUMAA L.: 'Experiences of painless improvements in software inspection'. Project Control for Software Quality, ESCOM-SCOPE'99, Shaker Publishing B.V, 1999, pp. 321–327

[18] BULL S.A.: 'Inspection process assistant: user guide v 3.0', 1997

[19] INIESTA J.B.: 'A tool and a set of metrics to support technical reviews', in ROSS M. (EDS.): 'Software quality management II, volume II: building quality into software' (Computational Mechanics, Southampton, UK, 1994), pp. 579–594

[20] KNIGHT J.C., MEYERS E.A.: 'An improved inspection technique', *Commun. ACM*, 1993, **36**, (11), pp. 51–61

[21] KNIGHT J.C., MEYERS E.A.: 'Phased inspections and their implementation', *Softw. Eng. Notes*, 1991, **16**, (3), pp. 29–35

[22] GINTELL J.W., ARNOLD J., HOUDE M., KRUSZELNICKI J., MCKENNEY R., MEMMI G.: 'Scrutiny: a collaborative inspection and review system'. Proc. Fourth European Conf. on Software Engineering, 1993, pp. 344–360

[23] MASHAYEKHI V., DRAKE J.M., TSAI W.T., REIDL J.: 'Distributed, collaborative software inspection', *IEEE Softw.*, 1993, **10**, (5), pp. 66–75

[24] STEIN M., RIEDL J., HARNER S.J., MASHAYEKHI V.: 'A case study of distributed, asynchronous software inspection'. Proc. 19th Int. Conf. on Software Engineering, Boston, MA, USA, 1997, pp. 107–117

[25] MURPHY P., MILLER J.: 'A process for asynchronous software inspection'. Proc. Eighth Int. Workshop on Software Technology and Engineering Practice, London, UK, 1997, pp. 96–104

[26] MACDONALD F., MILLER J.: 'A comparison of tool-based and paper-based software inspection', *Empir. Softw. Eng.*, 1998, **3**, (3), pp. 233–253

[27] YAMASHITA T.: 'Evaluation of Jupiter: a lightweight code review framework'. M.S. thesis, University of Hawaii, Honolulu, Hawaii, Number CSDL-06-09, December, 2006, available at http://csdl.ics.hawaii.edu/techreports/06-09/06-09.pdf

[28] 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. 19th Int. Conf. on Software Engineering, Boston, Massachusetts, USA, pp. 14–21

[29] LANUBILE F., MALLARDO T., CALEFATO F.: 'Tool support for geographically dispersed inspection teams', *Softw. Process: Improv. Pract.*, 2003, **8**, (4), pp. 217–231

[30] 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. Eng.*, 2000, **26**, (1), pp. 1–14

[31] WOHLIN C., RUNESON P., HÖST M., OHLSSON M.C., REGNELL B., WESSLEN A.: 'Experimentation in software engineering: an introduction' (The Kluwer International Series in Software Engineering, 2000)

[32] BAEZA-YATES R., RIBEIRO-NETO B.: 'Modern information retrieval' (Addison-Wesley, 1999)

[33] OPPENHEIM N.: 'Questionnaire design, interviewing and attitude measurement' (Pinter Publishers, 1992)

[34] DEVORE J.L., FARNUM N.: 'Applied statistics for engineers and scientists' (Duxbury, 1999)

[35] BRIAND L., LABICHE Y., DI PENTA M., YAN-BONDOC H.: 'An experimental investigation of formality in UMLbased development', *IEEE Trans. Softw. Eng.*, 2005, **31**, (10), pp. 833–849

[36] BRUEGGE B., DE LUCIA A., FASANO F., TORTORA G.: 'Supporting distributed software development with fine-grained artefact management'. Proc. Int. Conf. on Global Software Engineering, Costão do Santinho, Florianópolis, Brazil, 2006

[37] KITCHENHAM B.A., PFLEEGER S.L., PICKARD L.M., JONES P.W., HOAGLIN D.C., EL EMAM K., ROSENBERG J.: 'Preliminary guidelines for empirical research in software engineering', *IEEE Trans. Softw. Eng.*, 2002, **28**, (8), pp. 721–734

doi: 10.1049/iet-sen.2008.0101