SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2002; **12**:133–154 (DOI: 10.1002/stvr.243)

State-of-the-art: software inspections after 25 years

Aybuke Aurum[1], Håkan Petersson[2] and Claes Wohlin[3,*,†]

[1]*School of Information Systems, Technology and Management, University of New South Wales, Sydney, NSW 2052, Australia*
[2]*Department of Communication Systems, Lund University, Box 118, SE-221 00 Lund, Sweden*
[3]*Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden*

SUMMARY

Software inspections, which were originally developed by Michael Fagan in 1976, are an important means to verify and achieve sufficient quality in many software projects today. Since Fagan's initial work, the importance of software inspections has been long recognized by software developers and many organizations. Various proposals have been made by researchers in the hope of improving Fagan's inspection method. The proposals include structural changes to the process and several types of support for the inspection process. Most of the proposals have been empirically investigated in different studies.

This is a review paper focusing on the software inspection process in the light of Fagan's inspection method and it summarizes and reviews other types of software inspection processes that have emerged in the last 25 years. This paper also addresses important issues related to the inspection process and examines experimental studies and their findings that are of interest with the purpose of identifying future avenues of research in software inspection. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: software inspections; Fagan inspection method; survey; software reviews; reading techniques

1. INTRODUCTION

Software inspection is an important means of verifying and achieving sufficient quality in software projects today. It is a static verification technique and one of its main benefits is that it can be applied to any artefact produced during software development. It is probably one of the few methods that people actually agree will help in improving software quality, although it is not always applied.

*Correspondence to: Claes Wohlin, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden.
†E-mail: claes.wohlin@bth.se

Contract/grant sponsor: Swedish Institute Research Grant
Contract/grant sponsor: Department of Communication Systems, Lund University

Received 13 December 2000
Accepted 21 November 2001

Published online 5 February 2002

The objective of this paper is to provide a survey of software inspections given that this technique celebrated its 25th anniversary in 2001. Since the first description of software inspections, there have been further improvements; new ways of conducting inspections have been suggested and several methods supporting software inspections have been proposed. This paper summarizes and provides an overview of the software inspection processes that have emerged in the last 25 years. The paper identifies the main differences between commonly known inspection methods and Fagan's inspection method. It studies the evolution of inspections, highlights articles presenting original ideas and presents evaluations of new ideas that have since emerged. It also addresses important issues related to the inspection process that are of interest for the purpose of identifying future avenues of research in software inspection and it provides a list of concrete research questions.

Software inspection is a peer review process led by software developers who are trained in inspection techniques [1]. Michael Fagan originally developed the software inspection process '*out of sheer frustration*' [2]. It has been 25 years since Fagan published the inspection process in his famous article in 1976 [3]. Since then the importance of software inspections has been increasingly recognized by software developers and many organizations. The need to improve the inspection process has become a long-standing concern of software developers and researchers. Fagan's inspection method has been studied and presented by many researchers in various forms.

1.1. Terminology

Software inspection is a type of software review process. One common problem is that the terminology used for different types of software review processes is often imprecise and leads to confusion. In particular, there is no universal agreement on what a *software inspection process* is and how it is different from the other types of software review processes such as a walkthrough, a formal technical review or a management review. In practice, the inspections performed do not necessarily follow the original definition. In other words, an organization can conduct an inspection that may also fit under the category of a walkthrough or a technical review.

Fagan [3] strongly emphasizes that inspections have formal procedures and produce repeatable results, whereas walkthroughs are performed in varying regularity and thoroughness. He also remarks that in some cases walkthroughs may be identical to formal inspections, but in many cases they are informal and less efficient [4]. Wheeler *et al*. [2] point out some principal differences between review processes. Knight and Myers [5] suggest that walkthroughs are used to examine source code, formal reviews are the presentation of the work product to the rest of the team members and inspections are error detection techniques that ensure particular coding standards and issues are enforced. According to these authors, Fagan's inspection method is a combination of a walkthrough, formal review and inspection. IEEE Standard 1028-1997 [1] provides the following descriptions:

- an inspection is '*a visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications*';
- a walkthrough is '*a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems*';

- a review is '*a process or meeting during which a software product is presented to project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval*'.

According to [1], the objective of technical reviews and walkthroughs is to evaluate the software product, whereas the objective of an inspection is to detect and identify defects in the software but not to evaluate it. Thus, inspections are viewed as distinctively different from the other techniques, although the document type used is the same. This definition also parallels Fagan's original idea, i.e. the aim of the inspection process is to detect the defects and this does not involve searching for solutions and reaching consensus over the product implementation.

2. SOFTWARE INSPECTION PROCESS

A software inspection is a well-structured technique that originally began on hardware logic and moved to design and code, test plans and documentation [4]. The process itself can be characterized in terms of its objective, number of participants, preparation, participants' roles, meeting duration, work product size, work maturity, output products and the process discipline [2]. The essential criterion to start an inspection is to have a well-defined software process with an exit criterion and a software product that meets that criterion [6]. An inspection team who plays defined roles performs a software inspection. It is important that the people performing the inspection are familiar with the product and that they have a sound knowledge about the inspection process or otherwise they must be trained. The members of the inspection team examine the material individually to learn about the product. Then participants attend a meeting with the intended purpose of effectively and efficiently identifying defects early in the development process. Next, the list of defects is sent to the author of the documents to be repaired and removed in the later stage of the review process. Hence, software inspection provides a powerful way to improve the quality and productivity of the software process [7,8].

An effective software review process needs to address the relationships of all the required variables in terms of tasks involved, tools and methods used, and the skill, training and motivation of people. Various researchers have made proposals which attempt to improve upon the process of Fagan's inspection method. A literature review reveals two major areas of study, as illustrated in Figure 1. There has been considerable research carried out on the structure of the inspection process. Researchers have developed several new process models by restructuring the basic processes in Fagan's inspection method and these models have been evaluated and validated empirically. Another group of studies focuses on particular methods, tools and models that support the structure of the inspection process. Examples include support for the preparation stage, i.e. reading techniques, electronic support both for preparation and the inspection meeting and various models that support the reinspection process, which have again been empirically validated.

This paper focuses on the software inspection process in the light of Fagan's inspection method. The structure of this paper is drawn from Figure 1. This section addresses the review process in terms of its objectives and benefits, roles of the participants, documents, inspection pace and the team size. Section 3 describes Fagan's inspection process and Section 4 reviews the literature on structural changes made to this process and experimental findings. Section 5 focuses on the methods and models that support the structure of the inspection process and the experiments that validate these studies.

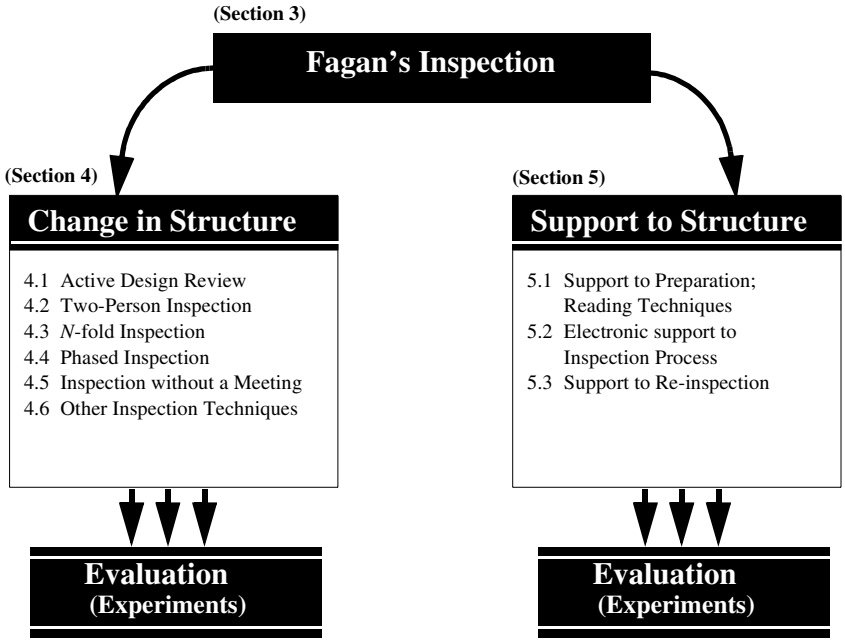

Figure 1. Evolution of the inspection process with two major areas: change in structure and support to structure.

2.1. Objectives and benefits

The basic objective of software inspections is to improve the quality of the product by analysing the product, detecting defects in time and removing them before the product is released. According to O'Neill [9], 42% of all defects result from a lack of traceability from the code to the design, to the requirements and to the business case. Researchers all agree that inspections appear to be an effective technique in error detection and productivity improvement [4,7,8]. First of all, inspections reduce the number of defects in the software throughout the development process. The more defects that can be found earlier, the easier and much less expensive they are to correct. Secondly, they can also uncover defects that would be difficult or impossible to discover in later stages of the life cycle. Thirdly, the inspection process improves learning and communication within the software team [10]. Overall, inspections reduce the software development cost, increase the quality of software and improve the productivity as well as the quality of the decision making process for management [4,7,11].

Despite the documented benefits of inspection, in many organizations productivity is still measured in terms of lines of code per effort [10]; therefore, the benefits of inspection are not perceived well by management and thus inspections are not used widely in the software industry. Shirey [12] points out that one of the problems in the inspection process is that the information collected from the process may not necessarily be used. Grady [13] reports that to get inspections incorporated into a large company,

such as Hewlett-Packard, may take a long time, e.g. 15 years. Since inspections are labour intensive, the return from this process is not immediate and clear to the management [12]. Therefore there is a perception that inspections cost more than they are worth [8].

2.2. Roles

Software inspections start with a request from the author of the work product. Following this request, a team of reviewers is formed and roles are assigned to each member of the team. The team members may play multiple roles. Each role requires specific skills and knowledge. There are roles related to the inspection process and to the product being inspected. The process roles defined by Fagan are author, moderator, reader and reviewer(s). According to some researchers, when reviewing requirements documents, each reviewer may be assigned a product role to represent the viewpoint of that role during the review process [6,14]. Researchers all seem to agree that the manager (project leader) should not be involved with the review process of technical documents. In some cases, the manager is the person who plans the inspection and assigns the resources and roles. However, on the other hand, a moderator (not being the manager) chairs the software inspection activities and facilitates the interaction between reviewers [9]. The moderator may also collect the defects from the reviewers and creates a list of defects [8]. In other cases, the moderator is a leader who is drawn from other development projects, having many responsibilities and may act as administrator, manager and supporter [11]. In each case, the moderator must be trained to carry out the review process. Although Russell [8] recommends that the author participates as a silent observer in the review process, Gilb and Graham [11] stress the active participation of the author; in fact, they claim that the author may be the best defect finder.

2.3. Documents

The document type can be a design, requirements specification, code, test plan or any type of document that is related to the product. One of the strengths of inspection is that a similar process can be applied to a wide range of documents. The document format can be in textual, graphical or tabular form or program code.

The size of a document or work product can vary in length, starting from a couple of pages to dozens of pages or thousands of lines of code. In order to prevent the information-overloading problem on reviewers, a document is generally divided into small chunks and then presented to reviewers. Hence, inspection of a document may take several cycles.

2.4. Inspection pace

Researchers agree that a slower review process yields more impressive results [8,11]. Russell [8] recommends that in code reading, the maximum reading speed should not exceed 150 lines per hour. Gilb and Graham [11] also support this idea by suggesting an optimum reviewing rate as one page per hour per participant. Laitenberger and DeBaud [15] report that 200–300 lines of C code is the right size of document to inspect within a two-hour time frame.

2.5. Team size in software inspection

Software inspections involve teamwork where a group of individuals work together to analyse the product in order to identify and remove defects. Teamwork is essential for software developers. There

are some major assumptions behind teamwork. First of all, the software products in today's market are too complex to be designed by individuals. The complexity of the products has become the driving force behind the creation of teams. It has been found that the quality of the product improves as it is inspected from multiple viewpoints [15]. Secondly, there is a belief that people are more committed to their work if they have a voice in the design of the product or use of the development process [16].

Inspection teams are formed from small groups of developers. Fagan [3], suggests that four people constitute a good-sized inspection team. This suggestion also ties in with the industrial work done by Weller [17], who reports that four person teams are more effective and efficient than three person teams. IEEE Standard 1028-1997 suggests teams of three to six people [1]. Teams of four or five people are common in practice [2]. Owens [18] reports that although it is expensive to have more reviewers, it is more effective to have more points of view for requirements inspection (e.g. five or six inspectors) than for design and more inspectors are needed for design than coding (e.g. one or two inspectors for coding). Bisant and Lyle [6] also support the idea of having two people in coding inspection. Their findings illustrate that overall programming speed increases significantly as a result of two-person inspections. Porter *et al*. [19] performed a detailed study on team size and found that the performance of one-person teams is significantly lower than two- or four-person teams. Porter *et al*. [19] also did not find a significant difference in performance between two and four person teams for code inspection.

3. FAGAN'S INSPECTION METHOD

Fagan [3], in his influential article describing an inspection process developed at IBM, defines inspections as '*a formal, efficient and economical method of finding errors in design and code*'. Fagan's inspection method is now known to be applicable to any phase of the software development lifecycle including requirements analysis, test documents or management documents, as well as to design and code. Below is a summary of the key elements from the 1976 paper.

Software inspections involve teamwork and are organized by a moderator. Essentially, a number of participants review a document with the aim of discovering defects. Fagan states that the inspection will be successful if all team members play out their roles fully in the process. In summary, the following inspection process roles are described.

- *Moderator*. This is the key person in the inspection who manages the team and coordinates the inspection process.
- *Author (designer or coder)*. The programmer who produces the program design and who may also translate the design into code.
- *Reader*. The programmer who paraphrases the design or code during the meeting.
- *Tester*. The programmer who reviews the product from the testing point of view.

Fagan's inspection method also consists of six major steps.

1. *Planning*. An inspection team is formed and the roles are assigned to team members.
2. *Overview*. An optional stage where the inspection team is informed about the product.
3. *Preparation*. Individual reviewers inspect the material independently. The inspection material may be a product of analysis, design (such as entity-relationship diagrams, data-flow diagrams, state-transition diagrams or a text specification) or coding. The aim of preparation is to learn about the material and to fulfil their assigned roles.

4. *Examination*. This is also called an *inspection meeting*. The aim of the meeting is not to discuss or evaluate the solution, but to find defects and pool them together. The moderator, who guides the meeting, takes notes and prepares a list of defects. The meeting should not last more than two hours. After the meeting the moderator produces a written report of the meeting to ensure that all issues identified in the meeting will be addressed in the rework and follow-up.
5. *Rework*. The defects, to be verified by the moderator, are corrected by the author. Some products must be reworked and reinspected several times if necessary.
6. *Follow-up*. The moderator checks and verifies each correction.

Although the main steps are preparation, examination and rework in the review process, Fagan states that all steps are necessary and skipping or combining steps is not recommended. One of the benefits of this approach is the feedback provided to the programmer that may take place within a few days of writing a program. Fagan also recommends that, since the inspection process from overview to follow-up may take up to 100 people hours, it is best to have a two hour session when performing an inspection meeting.

Fagan [4] points out that software inspections are less expensive if defects are detected and repaired at the stage where they are inserted, as opposed to when they are found and removed during the testing and operational stage. He presents a detailed cost-benefit analysis showing its cost-effectiveness and provides descriptions to allow software developers to understand the inspection process better. He describes a 1976 project which, when inspection was applied, resulted in 38% less defects in the final product than would have been accomplished if walkthroughs were used [3]. He also reports that IBM inspections found 90% of all defects detected over the lifecycle of a product and IBM had a 9% reduction in average project cost compared to when walkthroughs were applied [4].

4. CHANGES IN THE INSPECTION PROCESS

Figure 2 compares the seven well known inspection methods based on the main process steps involved. The grey and black parts on each bar illustrate the relative emphasis put on the various steps in each method. For instance, in the active design review [20] preparation and inspection meeting steps are handled differently from Fagan's inspection process. On the other hand, in Gilb inspections [11] an extra step is added to the inspection process to improve the software development process that initially produced the document. The following section further describes the seven methods and addresses experimental findings in relation to the inspection team size, the number of teams and the coordination strategy within the team or the multiple teams.

4.1. Active design review

Parnas and Weiss [20] presented active design reviews in the mid 1980s. According to the authors the conventional inspection methods fail to find defects in the product development because:

(1) reviewers are generally overloaded with information during the preparation stage and finding the relevant information is difficult;
(2) in most cases the reviewers are not familiar with the goals of the design and they may not be competent in the task;

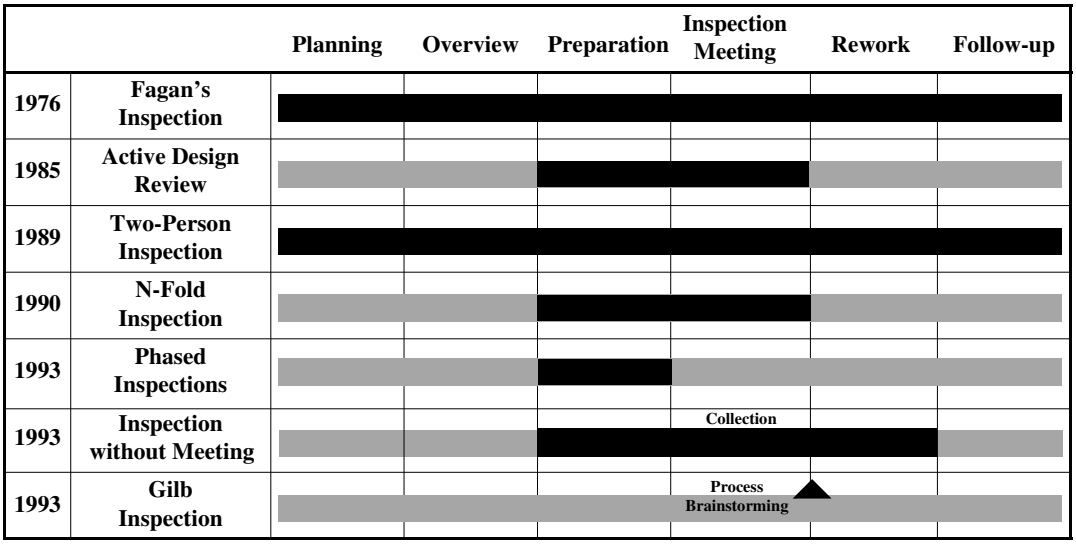

Figure 2. Commonly known inspection processes with comparison to Fagan's inspection method.

(3) social interaction in large meetings brings a few drawbacks such as production blocking and evaluation apprehension.

In order to achieve a full review process they redesigned the review process. The active design review is conducted as several brief reviews, as opposed to one large review, in which each review focuses on a certain part of the product. Different reviewers who have particular expertise and specific responsibilities perform each review. In order to guide the reviewers the errors are classified in terms of inconsistency, inefficiency and ambiguity. Reviewers are also provided with a list of questions to be used as a guideline during the review process. Reviews are designed according to the properties of the product that the reviewer is examining. Each review consists of three stages. In the first stage, the reviewers are presented with a brief description of the main product. In the next stage, reviewers study the material following the guideline. Then, in the third stage, the issues raised by the reviewers are discussed in small meetings where only the designer and the particular reviewers meet.

Parnas and Weiss successfully applied this approach to the design of a military flight navigation system, but did not provide any quantitative measurements.

4.2. Two-person inspection

This approach uses a formal method for software inspection. It removes the requirement of the initial resources of Fagan's inspection method by eliminating the role of the moderator. The inspection team consists of two persons: an author and a reviewer.

Bisant and Lyle [6] by applying this approach, conducted experiments to study the programmers' productivity to determine whether their productivity improves with this technique during the design or coding phase of product development. They found that two-person inspections had immediate benefits in the program quality and program productivity. They pointed out the significant improvements in individual productivity, in particular with novice and less productive programmers. They suggested using this method in small organizations where the team sizes are relatively small and as a transition to the larger team technique, which will have the extra benefits of establishing consistent data to be used in process control.

4.3. *N*-fold inspection

This approach uses N independent small efficient teams. The aim in using multiple teams is to identify defects that may not be found by a single team. It is also expected that different teams will detect different faults. Therefore, having a larger number of teams should result in the detection of more defects. The team size varies between three and four people. The participants of this approach are the author, reviewers and one single moderator. The moderator is assigned for coordinating the team and collecting data from it. Each team follows the six steps of the Fagan inspection method. In this approach, multiple teams work in parallel sessions. First, each team member individually reviews the document for about two hours. Then, they meet as a team to discuss the defects and this meeting may also last up to two hours. After the inspection only one single team becomes responsible for the rest of the development process.

Martin and Tsai [21], who originally developed this technique, suggested using N-fold inspection for the initial phase of software development for mission-critical systems. They conducted their experiments based on this approach and found the following.

(a) N-fold teams out-performed single teams. Even one pair of teams found more defects during the inspection than a single team discovered after inspection, specification and design. Tripp *et al*. [22], who applied multiple team inspection to improve the technical review and quality of a safety-critical software standard, have also reported favourable results. They found that there was a low redundancy in defects between teams. Schneider *et al*. [23] found that a single team was able to detect 35% of defects present whereas N teams ($N = 9$) was able to detect 78% of defects present.
(b) There was no significant overlapping between the faults found by different teams.

Martin and Tsai [21] raised a point that although increasing the number of teams results in finding more defects, this increment may not be significant. The best value for N depends on:

(a) the availability of teams;
(b) the cost of additional teams; and
(c) the potential cost for not finding a defect during the inspection.

Tripp *et al*. [22] also described that multiple team inspection is costly, but provides substantial benefits.

Kantorowitz *et al*. [24] who conducted experiments with both information and real-time systems, identified that N-fold inspection is not dependent on the type and size of the system. Their analysis suggested that the performance of N-fold inspection depends on the level of expertise of the team members and the number of teams. Based on their findings, they developed a probabilistic model for

N-fold inspections in which the model captures the performance of N-fold inspections and can also be used to determine the optimal number of inspection teams.

4.4. Phased inspection

Knight and Myers [5] developed phased inspection. Some of the ideas in active design reviews, Fagan inspection and N-fold inspection are adopted for phased inspection. In this approach the software product is reviewed in a series of partial inspections called *phases*. A simple inspection may have up to six phases and phases are conducted in sequential order. Each phase has a specific goal and is intellectually manageable. During each phase, reviewers fully examine and validate the product for fulfilment of a specific property. The inspection does not progress to the next phase until corrections are completed. To achieve a higher degree of confidence, the approach uses two types of phases, the *single-inspector* phase and the *multiple-inspector* phase. In the first case, a single person studies the product to determine whether it complies with a checklist. It is recommended to use a technical writer for this process. After this, in the multiple-inspector phase, several reviewers individually study the product by using a different checklist and then attend a meeting session to compare their findings. The multiple-inspector phase may take around 4 hours. According to Knight and Myers, phased inspections can be used not only for fault detection but also to examine other desirable characteristics of the product, e.g. portability, reusability or maintainability.

4.5. Inspection without a meeting

In Fagan's inspection method the main focus is on inspection meetings where the members of the inspection team meet to identify and discuss the defects. The goal of inspection meetings is to bring synergy to the software team. It is believed that the combination of different viewpoints, skills and knowledge from many inspectors creates this synergy (process gains). Synergy can overcome many difficulties encountered in organizational life, including those in the software development process. Fagan [3] strongly emphasized that the inspection meeting is crucial and reported that most defects will be found during the inspection meeting. He suggested that reviewers who have meetings will be more successful in finding a larger quantity of defects, as opposed to those who work individually. Johnson and Tjahjono [25] pointed out that the reason that teams who have meetings will be more successful than individuals working alone, is that in Fagan's inspection method defect detection only becomes an explicit goal during the meeting phase.

Recently, it has been reported that in many organizations the structure of Fagan's inspection method is radically changed and performed in three stages, preparation, collection (with or without an inspection meeting) and rework [26,27]. Furthermore, the expected outcomes from preparation and inspection meeting have also considerably changed, i.e. the preparation stage is used to identify defects, whereas the collection stage is used to pool the defects from the reviewers [28]. Experimental findings by Eick *et al*. [29] showed that when this structure was applied, reviewers were able to identify 90% of the defects during the preparation stage, whereas only 10% of the defects were found during the inspection meeting.

After observing several meetings and collecting data from the programmers of the work products, Votta [26] identified five reasons for holding meetings, synergy, education, schedule deadline, competition and requirement. His experiments at AT&T Bell Labs showed that on the positive side holding meetings tends to minimize 'false positives', but on the negative side they may not necessarily

bring the synergy. He claimed that holding meetings after the preparation stage does not have a significant effect on the inspection. This finding also ties in with the experimental work done by Johnson and Tjahjono [25]. These researchers reported that meeting-based software inspections are more costly and do not find significantly more defects than non-meeting-based methods; however, the former approach is significantly better at reducing the number of 'false positives' and the reviewers actually prefer inspection meetings over a non-meeting approach. Mashayekhi *et al*. [30] argued that face-to-face meetings can be quite expensive because they involve three to six people. It is labour intensive and takes many meetings to completely inspect a product, since each inspection covers only a small part of the product. Porter *et al*. [27] also indicated that holding meetings is time consuming, because bringing several people into one room on a specific day and time requires a significant amount of effort.

Votta [26] raised a point that in a meeting, although n number of reviewers participate, only two of the reviewers can actually interact at any time. This is because in face-to-face meetings communication is established in sequential order. He also found that, most of the time, only 30–80% of $n-2$ reviewers were actually listening to the conversation. Hence, in each meeting on average $n-4$ reviewer hours per meeting were wasted. Votta found that most defects were also identified before the meeting. He concluded that having a meeting is not cost-effective and made a few suggestions:

(1) using a nominal[‡] group approach;
(2) using correspondence instead of face-to-face meetings; or
(3) replacing meetings of all reviewers by two- or three-person meetings, called *depositions*, which consist of an author, a reviewer and a moderator (optional).

According to his experimental findings, in many cases depositions were more efficient than inspections.

4.6. Other inspection techniques

There are several more inspection techniques, similar to Fagan's inspection method, that have several distinct features in relation to the number of people involved in the process, the role of the reviewers and the goal of inspections. For instance, Yourdon [31] defines a specific walkthrough that is similar to Fagan's inspection method. He calls this approach 'structured walkthroughs', where a walkthrough is defined as being simply a peer group review of any product. In Yourdon's inspection, the preparation stage and the inspection meeting does not take more than 2 hours. In the preparation stage the reviewers are encouraged to note positive aspects of the document. Gilb and Graham [11] suggest adding an extra step to improve the inspection process itself, right after the inspection meeting, called a 'process brainstorming meeting'. The reader can find a literature review on various software inspection processes in Porter *et al*. [32] and a detailed taxonomy of review processes in Wheeler *et al*. [2] and Johnson [33]. Recently, Laitenberger and DeBaud [34] also provided a survey and developed a different taxonomy which characterized the nature of software inspection in various dimensions. Then they illustrated the differences between software inspection processes during different phases of the software development lifecycle.

[‡]A nominal group is an artificial group that is created based on combining the results of the performance of the individuals. This means that no group meeting is held.

5. SUPPORT TO STRUCTURE

There are a considerable number of studies that focus on methods and tools to support the structure of the inspection process. Section 5.1 reviews different reading techniques and the results of experimental findings. Section 5.2 presents the research on electronic support tools and the experimental studies. Finally, Section 5.3 is concerned with various methods and models that have been developed to support the reinspection process and the experiments that validate these studies.

5.1. Support for preparation: reading techniques

Multiple reviewers identifying potential defects in the material using a particular reading technique perform the preparation stage. There are few techniques available to support this activity that are proven to be more effective. Researchers all agree that the choice of reading techniques has a potential impact upon inspection performance. Reading techniques are classified as systematic techniques and non-systematic techniques [27,35]. The systematic reading techniques such as perspective-based reading, further described in Section 5.1.5, apply a highly explicit and structural approach to the process. It provides a set of instructions to reviewers and explains how to read the software document and what they should look for [36]. The non-systematic reading techniques, such as *ad hoc* reading or checklists, apply an intuitive approach and offer little or no support to the reviewer. A few empirical studies have also made comparisons between reading techniques by measuring the number of defects detected by each technique. The following sections describe the most commonly used forms of reading techniques and report the results of a few empirical studies.

5.1.1. Ad hoc reading

This technique takes a very general viewpoint of reviewers. There is no clear procedure to follow for the reviewer and no training is needed. Reviewers use their own knowledge and experience to identify defects in the documents. *Ad hoc* reading does not offer any support to the reviewer.

5.1.2. Checklist

Checklist reading is considered to be more systematic than *ad hoc* reading. The proposed procedure by Fagan [3] included the use of checklists. The reviewer answers a list of questions or ticks a number of predefined issues that need to be checked. It is expected that the questions will guide the reviewer throughout the inspection process. The aim is to define the responsibilities of the reviewers and provide guidance to them to help identify the defects. According to Gilb and Graham [11] checklists are developed from the project itself and they must be prepared for each individual type of documentation and for each different type of product or process role. A checklist may concentrate on questions that help reviewers identify major defects or prioritize different defects. A checklist should be no more than one single page for each type of documentation [11].

5.1.3. Stepwise abstraction

This reading technique was developed in the late 1970s as a code reading technique and has been experimentally validated [37]. In code reading by stepwise abstraction, each reviewer identifies subprograms in the software and determines their functions. Next, each reviewer determines the

function of the entire program by combining subfunctions together and constructing their own specification for the program. Then, the reviewer compares this derived specification with the official specification. Based on this comparison, inconsistencies are identified as defects and studied in the next stage.

Basili and Selby [38] conducted some experiments to compare code reading by stepwise abstraction with functional testing and structural testing in four small programs. They reported that code reading by stepwise abstraction detected more defects than the other two techniques. They pointed out that the number of defects observed, defect detection rate and the total effort in detection depended on the type of software tested.

Based on the idea that different reading techniques may identify different defects, Roper *et al*. [39] investigated the relative effectiveness of different reading techniques. They examined the effect of combining three techniques, code reading by stepwise abstraction, functional testing and structural testing. They found that individual techniques were similar to each other in terms of finding defects. They also pointed out that the reading techniques would be more effective when used in conjunction with other techniques.

5.1.4. Scenario-based reading (defect-based reading)

Scenario-based reading is proposed as a systematic reading technique with specific responsibilities for reviewers [27,35]. This technique was originally developed to identify defects from requirements documents. In this approach, defects are classified and a set of questions is developed for each defect class. Scenarios, which are a collection of procedures for detecting particular types of defects, are also developed and focused on a particular viewpoint. The reviewer answers the questions by following a specific scenario.

Several researchers have compared scenario-based reading with *ad hoc* and checklist reading techniques and studied the potential benefits of scenario-based reading on requirements documents by conducting multiple experiments [27,35]. In these experiments documents were written in both plain English and tabular form. For scenario-based reviewers, they developed a few scenarios that were derived from checklists and aimed to detect particular classes of defects. Then, each reviewer was assigned to a scenario and was asked to search for different classes of defects. For checklist reviewers, they developed a single checklist that contained several questions. *Ad hoc* reviewers did not receive any assistance. Experimental findings showed that scenario-based reading had a higher detection rate than *ad hoc* and checklist methods. In particular, the scenario-based approach found 35% more defects than the other two groups. Checklist reviewers performed no better than *ad hoc* reviewers.

Gough *et al*. [40] performed a large-scale study using scenarios in a market-driven industrial environment (a medical system) and the use of Fagan's inspection method with stakeholders, i.e. customers and developers. Each team consisted of a moderator, a scribe and an average of five reviewers who were assigned to particular roles, e.g. requirements engineer or customer representative. In this large case study reviewers typically identified two to four defects per hour. The authors pointed out that scenarios are the prime means of elicitation of requirements.

Fusaro *et al*. [41], who also conducted experiments using requirements documents on real-time systems and compared scenario-based reading with *ad hoc* and checklist techniques, discovered that the average defect detection rate for scenario-based reading was not significantly different from those obtained from the other two techniques.

5.1.5. *Perspective-based reading*

Perspective-based reading (PBR) was originally developed and experimentally validated at NASA [14]. PBR is an enhanced version of scenario-based reading. The technique focuses on the point of view or needs of the stakeholders. Each scenario consists of a set of questions. Scenarios are developed based on the viewpoint of stakeholders. During the review process, reviewers read the document from a particular viewpoint and produce a physical model that can be analysed to answer the questions based on the viewpoint. It is expected that this structural approach will reduce any existing gaps or overlaps between reviewers during the inspection process.

Basili *et al*. [14] conducted several experiments at NASA to study the effectiveness of PBR on requirements documents. They found no significant difference in the performance of reviewers who were using PBR and those using their own usual technique when finding defects. However, PBR reviewers performed significantly better on the generic documents. Laitenberger and DeBaud [15], also found no significant performance differences when they ran a more detailed experiment using PBR on code documents at Robert Bosch GmbH. Shull *et al*. [36] pointed out that PBR is suited to reviewers with a certain range of experience. According to these authors, reviewers who use PBR to inspect requirements documents tend to detect more defects than those who use less-structured approaches. They also emphasized that PBR has beneficial qualities because it is systematic, focused, goal-oriented, customizable and transferable via training.

5.2. Electronic support for the inspection process

In recent years, there have been a number of attempts to increase inspection efficiency further by the introduction of electronic tool support that has resulted in a number of prototype systems. These systems are designed to support either synchronous, asynchronous or both types of communications between team members. Some systems are designed to support the inspection process fully whereas others partially support the process, e.g. the preparation or the inspection meeting. The researchers who conducted experiments studied the effectiveness of these tools and the performance of meetings with and without electronic support. Below, several examples of these tools, which are evaluated in empirical studies, are presented.

Mashayekhi *et al*. [30] developed a tool for reviewers who are geographically distributed. The tool provided support for the preparation and the inspection meeting. This system was structured for software inspections on all types of products including requirements, design and coding and was designed to facilitate both asynchronous and synchronous activities in software inspection. The authors conducted experiments where each team consisted of six members: an author, a moderator, a recorder and three reviewers. Their findings indicated that meetings using electronic support were as effective as face-to-face meetings.

Johnson [42] developed a tool to support asynchronous communication. The inspection mainly consists of two stages, *private review* and *public review*. During the private review, reviewers study the material individually and cannot see other participants' comments and issues. During the public review, reviewers can access all comments and issues. Then they vote on whether they agree on issues. Next, the moderator analyses the private and public review and decides whether to go ahead with a face-to-face group review meeting or not.

Murphy and Miller [43] also developed a prototype tool to support asynchronous communication that used electronic mail as the basis of communication. In this approach some of the tasks of team

members were modified; the author did not participate in the process whereas the moderator was responsible for planning the inspection, monitoring the communication, broadcasting the list of defects to the inspection team to be reinspected and compiling the list of defects for the author.

Vermunt *et al*. [44], who conducted experiments using electronic support only for the inspection meeting, found that 95% of the defects were found during the preparation stage and groups using electronic support performed as well as the groups who did not have electronic support.

5.3. Support for reinspection

This section is concerned with methods and models that have been developed to support the reinspection process.

5.3.1. Inspection metrics

Inspection metrics provide important information for the software project manager. Gilb and Graham [11] state that '*The metrics themselves are the lifeblood of inspections*'. Inspection metrics are useful in many ways; they help the project manager to see the economics of the software review process, to judge the quality of inspection and decide whether to reinspect the material or not, and to understand the potential problems within the software process better. They also provide feedback for the review process.

Although Fagan [3] does not explicitly discuss the importance of collecting metrics from the inspection process, some metrics on the software inspection process are presented in the original paper. Fagan mentions the classification of defects and gives examples from the inspection reports and forms such as the number of defects found, date of inspection, size of product and the length of inspection. Ackerman *et al*. [7] list twelve basic measures that should be collected for each inspection, e.g. date of inspection, identity of product to be inspected, number of reviewers and number of defects. Barnard and Price [45] present nine derived key metrics that are used to answer questions concerning different factors such as inspection cost, product quality and the effectiveness of the inspection process. They also show various analyses and process control techniques that can be used in inspections. Detailed information about inspection metrics and descriptions of measures to be collected from an inspection process can be found in Gilb and Graham [11], and Ebenau and Strauss [46].

Inspection metrics measure various aspects of the inspection itself directly as well as indirectly [11]. There are many useful metrics that can be collected from inspections for analysis to monitor, control and improve the review process. Examples of these inspection metrics include total number of defects found, elapsed time for preparation and inspection meeting, and number of reviewers. By using the inspection metrics several statistics can be computed, e.g. inspection rate, the number of defects found per reviewer, and the number of remaining defects can be estimated. By collecting inspection metrics, developers can establish a baseline of measurement about their development process. This historical data may help them to determine the quality of the current inspection of the product and to control the process [11,47,48]. Chillarege *et al*. [49], introduced the concept of orthogonal defect classification in order to provide a cause–effect relationship that can be used to monitor the software process. In this approach, by keeping the same classification of defects throughout the phases of the development and computing the total number of defects found, it is possible to obtain information about the status of the product and the process.

Collecting metrics can be tedious, in particular when dealing with various sizes and types of products. Furthermore, the developers may avoid measurement because of a fear of data misuse. Shirey [12] reported from a survey at Hewlett-Packard that in most cases the data from the inspection process had never been used for any of the intended purposes.

5.3.2. Estimation of remaining defects

The main objective of an inspection is to find and correct defects. The correction of defects leads to increased quality, hence eliminating defects from the product is an important issue. One way of measuring the quality is estimating the number of remaining defects after an inspection. If the number of remaining defects is estimated, it becomes easier to calculate an estimation of the risk of failure. This would also give a hint of the amount of further effort that has to be spent on the project. Thus, knowledge of the number of remaining defects gives an estimation of the quality of the inspected artefact.

During the review process, the moderator records the number of defects the reviewers find. The list of defects is then sent to the author to be corrected. Meanwhile, by using various methods the number of remaining defects is calculated. Based on the results, the developers decide whether to reinspect the document. Reinspection is performed after the author revises the software product [1]. The most commonly used estimation techniques which are applied to software inspection data are summarized below [29,46,50]. Other methods also exist to estimate defect density; however, this is not within the scope of this paper. The focus here is solely on estimations from inspections.

(a) *Defect density*. This is expressed in terms of the defects detected per unit reviewed [46]. A reasonably high defect density may indicate that the review process is working well or that the document contains an unreasonably high number of defects. Since the aim is to eliminate the causes of defects, by reinspecting the document it is expected that defect density will continually become lower throughout the development. Using upper and lower thresholds on the number of defects per unit of size is also a common approach [51]. The drawback to this approach is that reviewers tend to reach the upper limit regardless of the document's quality [52].

(b) *Subjective assessment*. Reviewers make their own estimation at the end of the review process. The approach is not fully reliable. Thus, it is useful to combine this approach with an objective reinspection criterion [50].

(c) *Using historical data*. The remaining number of defects is estimated based on the historical data from the previous projects. The document is reinspected if the number of undetected defects is significantly different from the historical average. Too many defects may indicate that the document is poorly written. Too few defects may indicate that the document is poorly reviewed. According to Eick *et al*. [29] this approach assumes that the variation between reviewers is larger than the variation between the documents. The drawback in this approach is that, if this assumption is not true then a high-quality document may be reinspected and a poor-quality document may not be reinspected if the inspection is poorly performed. Gilb and Graham [11] also suggest using historical data as an estimation method.

(d) *Capture–recapture method*. This approach is based on the amount of overlap in defects found by various reviewers and uses estimation models to calculate the number of remaining defects in the product. The capture–recapture method was originally used to estimate the number of animals in the wild (animal population). Recently, researchers have started using the same method to estimate the number of defects in software products. The approach is based on applying statistical methods to collected defects. Three methods have been applied for this purpose: maximum likelihood estimator

(MLE), jackknife estimator (JE) and Chao's estimator. Capture–recapture models applied to software inspections make use of models based on assumptions of two factors: (1) the probability of a specific defect being found; (2) the ability of the reviewers.

The capture–recapture method for estimating the number of remaining defects has been empirically studied by many researchers during the last decade. Eick *et al*. [29] performed the first application of capture–recapture in software inspections. They applied this method to estimate the defect content in the design documents in AT&T's International Switching Division. Vander Wiel and Votta [51] used Monte Carlo simulation to investigate the assumptions made by capture–recapture methods, in particular MLE and JE. They found that if defects are classified into a small number of groups, MLE performs better than JE and if there is no grouping then MLE performs poorly. Based on this, Wohlin *et al*. [53] carried out the first controlled experiment on capture–recapture in software engineering. They proposed the application of a filtering process that grouped the defects into small classes. The classes are defined in such a way that some of the assumptions of the capture–recapture method are better fulfilled and hence the estimate is improved.

Briand *et al*. [52] studied the impact of the number of reviewers on the performance of capture–recapture models. They commented that the capture–recapture models generally underestimate the remaining defects. They suggested that the estimates will not be accurate if the group size is less than four people. They recommended using JE for a group of four and the Chao estimator for a group of five. The experimental findings of Miller [54] also supported this recommendation. Miller found that JE is the best estimator for a team of three to five people and recommended using Chao's estimator for larger group sizes or JE with a combination of other estimators. Briand *et al*. [50] conclude that models are strongly affected by the number of reviewers and, therefore, developers must consider this factor before using capture–recapture models.

(e) *Curve-fitting methods*. Wohlin and Runeson [55] proposed estimating the number of defects using a curve-fitting approach. They plotted the inspection data and fitted a curve to the plot to produce an estimate. They pointed out that curve fitting is more useful than capture–recapture estimates as the latter is based on statistical analysis with underlying assumptions that are not largely fulfilled. The curve-fitting approach was integrated with capture–recapture methods by Briand *et al*. [56], where they also applied a linear function to perform the estimation. The function applied by Wohlin and Runeson [55] was an exponentially decreasing function.

The estimation methods listed above, from (a) to (e), are validated in empirical studies. It is important to note that most of the evaluations are from laboratory environments (controlled experiments), whereas only a few of them are based on industrial case studies.

6. SUMMARY AND CONCLUSION

Software inspections are important for improving the quality of software. Although software inspections are widely used among developers, the application of the process varies. This paper has reviewed existing literature on software inspections and addressed issues related to the inspection process. Reviewing the literature in this field has highlighted the differences and overlaps between studies. Figure 1 illustrates the summary of the evolution of software inspections.

The following two major areas of study are identified in the literature.

- In the last 25 years several variations of Fagan's inspection method have been proposed to improve the performance of software inspections. There are some distinctive structural

differences between models. Restructuring the basic processes in Fagan's inspection method creates those different models. This includes changing the activities in each preparation or inspection meeting, emphasizing different goals in each stage, changing the number of participants in teams, employing single or multiple teams, changing the coordination strategy between teams or participants of the teams. A summary of these inspection methods is illustrated in Figure 2. Most of the methods in Figure 2 have been empirically evaluated.

- There is a considerable amount of research on various models, methods and tools that support the structure of software inspections. Support approaches include various reading techniques, electronic support tools and various models that support the decision making process in relation to reinspection.

Table I provides a brief summary of the two major areas of research and corresponding empirical studies in the literature. The first two columns show the main approaches or suggested changes in software inspections. The first column includes changes to the structure of software inspections and the second column presents new ideas of how to support software inspections. The objective, in the first two columns, has been to give credit to the authors that proposed the new ideas. Thus, the references provided in the first two columns are intended to reference the original source. The new ideas are annotated so that cross-references to empirical studies in columns three and four can easily be identified. The references in columns three and four are to papers where primarily a new idea has been evaluated, although some minor changes may also have been introduced. The table is divided into three rows to classify the research into the three major phases of software inspections, i.e. preparation, meeting and reinspection. The table is by no means exhaustive. The intention has been to capture some of the main contributions in the area.

Software inspections are widely used and recognized as a cost-effective way of removing defects. Despite this, there is still room for improvement in this field. The extensive research into software inspections has provided valuable knowledge and experience. There have been contributions to areas such as the inspection process, reading techniques and defect estimation techniques. It is not possible to identify any single study or proposal that has provided a major breakthrough in how software inspections are researched or practised. The studies have, however, jointly contributed to the evolution of software inspections. This means that they have contributed to the general body of knowledge in software inspections. It should however, be noted that no definitive answers have been found and that many research questions are still open, although inspections have matured considerably during the last 25 years. Some of the open research questions are as follows.

- What is an efficient reading technique? The research is inconclusive on this issue and more work is needed.
- Which reading technique should be used for which type of artefacts? The inability to find a superior way of reading may be due to different reading techniques being better suited to different artefacts or even applications.
- How should models, in for example UML, be inspected? Object orientation, UML and Java have been major changes in the last couple of years, but it is not clear how to inspect these models and notations efficiently.
- What is the best way to support inspectors in their work? There has been little work on how to develop checklists in an efficient way and also how best to support different scenarios when using scenario-based techniques.

Table I. A summary of the literature.

Change in structure	Support to structure	Evaluation of structure	Evaluation of support to structure
Preparation			
(A1) Fagan inspection method, Fagan [3] (A2) Active design review, Parnas and Weiss [20] (A3) Two-person inspection, Bisant and Lyle [6] (A4) *N*-fold inspection, Martin and Tsai [21] (A5) Phased inspection, Knight and Myers [5] (A6) Inspection without meeting, Votta [26] (A7) Gilb inspection, Gilb and Graham [11]	(B1) *Ad hoc* reading (B2) Checklists (B3) Stepwise abstraction, Linger *et al.* [37] (B4) Scenario-based reading, Porter and Votta [35] (B5) Perspective-based reading, Basili *et al.* [14] (B6) Electronic support	• Wheeler *et al.* [2] (A1) • Russell [8], (A1) • Tripp *et al.* [22] (A4) • Schneider *et al.* [23] (A4) • Kantorowitz *et al.* [24] (A4)	• Basili *et al.* [38] (B3) • Roper *et al.* [39] (B3) • Porter *et al.* [27] (B4) • Gough *et al.* [40] (B4) • Fusaro *et al.* [41] (B4) • Laitenberger and DeBaud [15] (B5) • Shull *et al.* [36] (B5) • Mashayekhi *et al.* [30] (B6) • Johnson [42] (B6) • Murphy and Miller [43] (B6)
Meeting			
(C1) Active design review, Parnas and Weiss [20] (C2) *N*-fold inspection, Martin and Tsai [21] (C3) Inspection without meeting, Votta [26] (C4) Gilb inspection, Gilb and Graham [11]	(D1) Electronic support		• Mashayekhi *et al.* [30] (D1) • Johnson [42] (D1) • Murphy and Miller [43] (D1) • Vermunt *et al.* [44] (D1)
Reinspection			
	(E1) Defect density (E2) Subjective assessment (E3) Historical data (E4) Capture–recapture, Eick *et al.* [29] (E5) Curve fitting, Wohlin and Runeson [55]		• Briand *et al.* [10] (E2) • Vander Wiel and Votta [51] (E4) • Wohlin *et al.* [53] (E4) • Briand *et al.* [52] (E4) • Briand *et al.* [50] (E4) • Miller [54] (E4) • Briand *et al.* [56] (E4) and (E5)

- What is the best way to plan and staff software inspections? Research is needed toward supporting the person who takes the decisions prior to an inspection. Support is required so that a person can determine the number of reviewers needed and the competence of the reviewers in a cost-effective and efficient way.
- Is it possible to develop a benchmark that can provide a general measure of the performance of software inspections which can also be further tailored for specific applications? Software inspections are carried out in a number of ways and at a large number of places, but so far it has not been possible to make comparisons between different ways of conducting software inspections. Research along this line should aim at establishing methods for benchmarking software inspection processes.
- In what way can the decision making process after an inspection be supported? Work has been conducted regarding estimation of remaining software defects, but little work has been carried out on how to use the information. When is it suitable to add reviewers, make a reinspection or continue with the development process?
- What is the relationship between software inspections and software testing? Both these techniques aim at identifying software defects, but which defects are best found by which technique? What is the best way to ensure that the techniques complement each other in the best possible way?
- What is the best way to pool the comments from individual reviewers? Some work has been done regarding inspection meetings, but using electronic support to view each others comments may be helpful and possibly it is more efficient to include only a sub-group of the reviewers for these meetings.
- Are the estimation methods good estimators of the actual number of defects? This research question includes evaluation and validation of defect estimation methods in industrial case studies, where the actual data may be compared with the estimated value.
- Is it possible to identify relationships between different parameters in software inspections? This includes empirical studies related to different parameters in software inspections, for example, inspection pace versus effectiveness and defect density versus inspection pace.
- Are inspections different between different application domains such as real-time systems versus information systems or web-based systems?

These open research questions illustrate that although a lot of research has been devoted to software inspections during the last 25 years, there are still many unresolved issues. The research has come far since 1976, but it is important to continue to increase the software community's understanding of software inspections to allow for improvements in the area. Given the potential, in terms of increasing software quality throughout the software development process, research into software inspections is still very relevant and important for the success of software development.

ACKNOWLEDGEMENTS

This research was partially conducted during a sabbatical visit by Dr Aurum to the Department of Communication Systems, Lund University. Dr Aurum was partially supported by the Swedish Institute Research Grant and a Research Grant from the Department of Communication Systems, Lund University.

REFERENCES

1. IEEE Standard. Standard for software reviews, 1028-1997, 1998.
2. Wheeler DA, Brykczynski B, Meeson RN. Peer review process similar to inspection. *Software Inspection: An Industry Best Practice*. IEEE Computer Society Press: Los Alamitos, CA, 1996.
3. Fagan ME. Design and code inspections to reduce errors in program development. *IBM Systems Journal* 1976; **15**(3):182–211.
4. Fagan ME. Advances in software inspections. *IEEE Transactions on Software Engineering* 1986; **12**(7):744–751.
5. Knight JC, Myers AE. An improved inspection technique. *Communications of ACM* 1993; **36**(11):50–69.
6. Bisant DB, Lyle JR. A two person inspection method to improve programming productivity. *IEEE Transactions on Software Engineering* 1989; **15**(10):1294–1304.
7. Ackerman FA, Buchwald LS, Lewski FH. Software inspections: An effective verification process. *IEEE Software* 1989; **6**(3):31–36.
8. Russell GW. Experience with inspection in ultralarge-scale developments. *IEEE Software* 1991; **8**(1):25–31.
9. O'Neill D. Issues in software inspection. *IEEE Software* 1997; **14**(1):18–19.
10. Briand LC, Freimut B, Vollei F. Assessing the cost-effectiveness of inspections by combining project data and expert opinion. *Proceedings 11th International Symposium on Software Reliability Engineering*. IEEE Computer Society Press: Los Alamitos, CA, 2000; 124–135.
11. Gilb T, Graham D. *Software Inspection*. Addison-Wesley: Wokingham, U.K.
12. Shirey GC. How inspections fail. *Proceedings 9th International Conference on Testing Computer Software, 1992*. *Software Inspection: An Industry Best Practice*, Wheeler DA, Brykczynski B, Meeson Jr RN (eds.). IEEE Computer Society Press: Los Alamitos, CA, 1996; 151–159.
13. Grady RB. Successfully applying software metrics. *IEEE Computer* 1994; **27**(9):18–25.
14. Basili VR, Green S, Laitenberger O, Lanubile F, Shull F, Sörumgård S, Zelkowitz M. The empirical investigation of perspective-based reading. *International Journal on Empirical Software Engineering* 1996; **1**(2):133–164.
15. Laitenberger O, DeBaud JM. Perspective-based reading of code documents at Robert Bosch GmbH. *Information and Software Technology* 1997; **39**(11):781–791.
16. Smart KL, Thompson M. Changing the way we work: Fundamentals of effective teams. *IPCC'98 Proceedings IEEE International Communication Conference*, vol. 2. IEEE Computer Society Press: Los Alamitos, CA, 1998; 383–390.
17. Weller EF. Lessons from three years of inspection data. *IEEE Software* 1993; **10**(5):38–45.
18. Owens K. Software detailed technical reviews: Findings and using defects. *Wescon'97, Conference Proceedings*. IEEE Computer Society Press: Los Alamitos, CA, 1997; 128–133.
19. Porter AA, Siy HP, Toman CA, Votta LG. An experiment to assess the cost–benefit of code inspections in large scale software development. *IEEE Transactions on Software Engineering* 1997; **23**(6):329–346.
20. Parnas DL, Weiss DM. Active design reviews: Principles and practices. *Proceedings 8th International Conference on Software Engineering*, London, U.K., August 1985. IEEE Computer Society: Los Alamitos, CA, 1985; 132–136.
21. Martin J, Tsai WT. *N*-fold inspection: A requirements analysis technique. *Communications of ACM* 1990; **33**(2):225–232.
22. Tripp LL, Struck WF, Pflug BK. The application of multiple team inspections on safety critical software standard. *Proceedings 4th Software Engineering Standards Application Workshop, 1991*. *Software Inspection: An Industry Best Practice*, Wheeler DA, Brykczynski B, Meeson Jr RN (eds.). IEEE Computer Society Press: Los Alamitos, CA, 1996; 106–111.
23. Schneider GM, Martin J, Tsai WT. An experimental study of fault detection in user requirements documents. *ACM Transactions on Software Engineering and Methodology* 1992; **1**(2):188–204.
24. Kantorowitz E, Guttmann A, Arzi L. The performance of the *N*-fold requirements inspection method. *Requirements Engineering Journal* 1997; **2**(3):152–164.
25. Johnson PM, Tjahjono D. Does every inspection really need a meeting? *International Journal on Empirical Software Engineering* 1998; **3**(1):9–35.
26. Votta LG. Does every inspection need a meeting? *Proceedings ACM Symposium on Software Development Engineering*. Reprinted in *Software Inspection: An Industry Best Practice*. IEEE Computer Society Press: Los Alamitos, CA, 1996.
27. Porter AA, Votta LG, Basili V. Comparing detection methods for software requirements inspection: A replicated experiment. *IEEE Transactions on Software Engineering* 1995; **21**(6):563–575.
28. McCarthy P, Porter R, Riedl J. An experiment to assess cost–benefit of inspection meetings and their alternatives. *Technical Report*, Computer Science Department, University of Maryland, 1995.
29. Eick SG, Loader CR, Long MD, Votta LG, Vander Wiel S. Estimating software fault content before coding. *Proceedings of the 14th International Conference on Software Engineering*, Melbourne, Australia, May 1992. ACM Press: New York, 1992; 59–65.
30. Mashayekhi V, Drake J, Tsai WT, Riedl J. Distributed, collaborative software inspection. *IEEE Software* 1993; **10**(5):66–75.

31. Yourdon E. *Structured Walkthroughs* (4th edn). Prentice-Hall: Englewood Cliffs, NJ, 1989.
32. Porter AA, Siy HP, Votta LG. A review of software inspections. *Advances in Computers* 1996; **42**:39–76.
33. Johnson PM. Reengineering inspection: The future of formal technical review. *Communications of the ACM* 1998; **41**(2):49–52.
34. Laitenberger O, DeBaud JM. An encompassing life cycle centric survey of software inspection. *Journal of Systems and Software* 2000; **50**(1):5–31.
35. Porter AA, Votta LG. An experiment to assess different defect detection methods for software requirements inspections. *Proceedings 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994. IEEE Computer Society Press: Los Alamitos, CA, 1994; 103–112.
36. Shull F, Rus I, Basili V. How perspective-based reading can improve requirements inspections. *IEEE Computer* 2000; **33**(7):73–79.
37. Linger RC, Mills HD, Witt BI. *Structured Programming: Theory and Practice*. Addison-Wesley: Reading, MA, 1979.
38. Basili VR, Selby RW. Comparing the effectiveness of software testing strategy. *IEEE Transactions on Software Engineering* 1987; **13**(12):1278–1296.
39. Roper M, Wood M, Miller J. An empirical evaluation of defect detection techniques. *Information and Software Technology* 1997; **39**(11):763–775.
40. Gough PA, Fodemski FT, Higgins SA, Ray SJ. Scenarios—an industrial case study and hypermedia enhancements. *Proceedings 2nd IEEE International Symposium on Requirements Engineering*. IEEE Computer Society Press: Los Alamitos, CA, 1995; 10–17.
41. Fusaro P, Lanubile F, Visaggio G. A replicated experiment to assess requirements inspection techniques. *International Journal on Empirical Software Engineering* 1997; **2**(1):39–57.
42. Johnson PM. An instrumented approach to improving software quality through formal technical review. *Proceedings 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994. IEEE Computer Society Press: Los Alamitos, CA, 1994; 113–122.
43. Murphy P, Miller J. A process for asynchronous software inspection. *Proceedings 8th IEEE International Workshop on Software Technology and Engineering Practice*, London, U.K., July 1997. IEEE Computer Society Press: Los Alamitos, CA, 1997; 96–104.
44. Vermunt A, Smits M, Van der Pijl G. Using GSS to support error detection in software specifications. *Proceedings 31st Annual Hawaii International Conference on System Sciences*. IEEE Computer Society Press: Los Alamitos, CA, 1998; 566–574.
45. Barnard J, Price A. Managing code inspection information. *IEEE Software* 1994; **11**(2):59–69.
46. Ebenau RG, Strauss SH. *Software Inspection Process* (*System Design and Implementation Series*). McGraw-Hill: New York, 1994.
47. Christenson DA, Huang ST, Lamparez AJ. Statistical quality control applied to code inspections. *IEEE Journal on Selected Areas in Communications* 1990; **8**(2):196–200.
48. Burr A, Owen M. *Statistical Methods for Software Quality*. Thomson Computer Press: London, U.K., 1996.
49. Chillarege R, Bhandari IS, Chaar JK, Halliday MJ, Moebus DS, Ray BK, Wong M. Orthogonal defect classification: A concept for in-process measurements. *IEEE Transactions on Software Engineering* 1992; **18**(11):943–956.
50. Briand LC, El Emam K, Freimut BG, Laitenberger O. A comprehensive evaluation of capture–recapture models for estimating software defect content. *IEEE Transactions on Software Engineering* 2000; **26**(6):518–539.
51. Vander Wiel S, Votta L. Assessing software designs using capture–recapture methods. *IEEE Transactions on Software Engineering* 1993; **19**(11):1045–1054.
52. Briand LC, El Emam K, Freimut BG, Laitenberger O. Quantitative evaluation of capture–recapture models to control software inspections. *Proceedings 8th International Symposium on Software Reliability Engineering*. IEEE Computer Society Press: Los Alamitos, CA, 1997; 234–244.
53. Wohlin C, Runeson P, Brantestam J. An experimental evaluation of capture–recapture in software inspection. *Software Testing, Verification and Reliability* 1995; **5**(4):213–232.
54. Miller J. Estimating the number of defects after inspection. *Software Testing, Verification and Reliability* 1999; **9**(4):167–189.
55. Wohlin C, Runeson P. Defect content estimations from review data. *Proceedings 20th International Conference on Software Engineering*, Kyoto, Japan, April 1998. IEEE Computer Society Press: Los Alamitos, CA, 1998; 400–409.
56. Briand LC, El Emam K, Freimut BG. A comparison and integration of capture–recapture models and the detection profile method. *Proceedings 9th International Symposium on Software Reliability Engineering*. IEEE Computer Society Press: Los Alamitos, CA, 1998; 32–41.

