Fraunhofer Institut
Experimentelles
Software Engineering

Inspections and Pair Programming – competing or complementary?

Experiences from an Expert eWorkshop

Authors:
Ralf Carbon
Marcus Ciolkowski
Christian Denger
Mikael Lindvall
Forrest Shull
Patricia Costa
Dieter Rombach
Victor Basili

IESE-Report No. 029.05/E
Version 1.0
May 4, 2005

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Sauerwiesen 6
67661 Kaiserslautern

Abstract

The common goal of the two practices *Pair Programming* and *Inspections* is to produce high quality software. Even though they have a common goal, their approaches are different, and they are typically used in different situations. Pair programming is typically applied as a part of agile development methodologies, such as Extreme Programming, whereas Inspections are often applied in plan-driven or CMM-based methodologies. In order to gain a better understanding of these two practices and their strengths and weaknesses, we facilitated an expert eWorkshop. Our goals were to compare the two practices as well as to understand in which situations the two practices can be best applied. Our eWorkshop discussion highlighted several differences in the benefits that can be expected from the practices (along dimensions such as objectivity of review and achievable level of quality), indicating that the practices can be considered complementary in order to achieve the full range of effects.

Keywords: Pair Programming, Inspections, eWorkshop, Experience, Visek

Copyright © Fraunhofer IESE 2005

Table of Contents

1	**Introduction**	**1**
2	**Setting the Stage for Discussion**	**3**
3	**Effect on Quality**	**5**
4	**Third Party Perspective**	**6**
5	**Feedback Cycle**	**7**
6	**Using practices in foreign home grounds**	**8**
6.1	Applying Inspections in an agile context	8
6.2	Applying Pair Programming in a CMM context	9
7	**Conclusions**	**10**
References		**12**
Appendix: Definitions		**14**
Side bar: Pair Programming		14
Side bar: Software Inspections		14

Copyright © Fraunhofer IESE 2005

1 Introduction

Software developers striving to develop high quality software typically apply different practices to remove defects. Knowing that it is more expensive to remove defects late in the life cycle, a common approach is to detect defects as early as possible. There are several practices that detect defects early, theoretically leaving software developers with the need to pick the most efficient practice, or combinations of practices, for the task at hand. In practice, however, the environment in which they perform their work has a strong influence on the selection of the practice to be used. Inspections have, for example, been the natural choice in order to detect defects in plan-driven or CMM-oriented software development environments. Pair programming is more common in agile environments and form parts of agile methodologies such as eXtreme Programming (XP).

Although in many ways dissimilar, both practices, Pair Programming and Inspections, have the common goal of producing high quality software with minimal defects, through structured collaboration among developers and reviewers. Inspections and Pair Programming are both able to detect defects early, but in which environments and to what software development tasks are they best applied? And: In what ways are they complementary, and how can they be combined in order to maximize their defect detection capabilities? Questions like these cannot be satisfactorily answered until we can characterize these practices and their effects. For a more detailed definition of the techniques, please refer to the Appendix and the sources referenced there.

Performing such characterizations based on empirical observation is one of our research goals. Evidence from actual use of these and other practices forms part of our experience base with the goal of helping software developers select and tailor software processes for the current task at hand. Defect detection has long been a major component of our research and is one of the reasons why we strive to gain a better understanding of the common and different characteristics of Pair Programming and Inspections in their capacity to detect defects.

While each of the two practices is relatively well understood, we felt there was a lack of a systematic characterization of their respective strengths and weaknesses. Such characterization is hard, takes time and requires feedback from many different experts. This paper is just the first step of that characterization.

Copyright © Fraunhofer IESE 2005

In order to start the work on characterizing these practices, the Center for Empirically Based Software Engineering (CeBASE) in the U.S.[1] and the Virtual Software Engineering Competence Center (Visek) in Germany[2] collaborated to conduct a joint eWorkshop [2].

An eWorkshop is a means for discussing a specific topic via the world wide web. It allows people at different locations to exchange ideas and arguments in a virtual meeting room. The discussants need nothing more than a web browser to participate. Thus, this form of a discussion is an inexpensive and efficient way of capturing and synthesizing new knowledge from a group of experts. To obtain meaningful results, an eWorkshop requires a thorough preparation to focus and to direct the discussion. After their invitation the participants' input to the discussion is requested through a pre-meeting questionnaire. The organizers analyse this pre-meeting information in order to prepare the concrete issues to be discussed in the virtual meeting. A summary of the pre-meeting information is distributed to the participants to allow individual preparation. To ensure the discussion goes smoothly and yields information of value, a number of support roles are required: The moderator has the responsibility to monitor and focus the discussion. The moderator can call for a vote in order to measure consensus regarding a specific issue. The moderator is supported by a lead discussant who steers the discussion by proposing the issues to be discussed next. The scribe summarizes and displays the results online during the discussion. After the online discussion the eWorkshop is analyzed. The analysts scrutinize the chat log and the scribe's summary to extract knowledge from the discussion. This web-based chat application has been successfully applied to discuss other topics in the past [13].

Tom Gilb characterized well the overall goals of the discussion when he stated: "My position is that they (Inspections and Pair Programming) are two different and complementary practices. We need to understand their costs and benefits quantitatively, and their best practice modes." We cannot agree more because if we can gain this understanding, we would be able to bridge the gap between the agile and the CMM communities and make use of the best practices of each methodology to achieve the joint goal: high quality software by balancing traditional and agile methodologies.

In order to achieve this, we were happy to have the input of a very lively set of over twenty participants from five different countries and six different time zones

[1] www.CeBASE.org

[2] www.Visek.de

Copyright © Fraunhofer IESE 2005

2 Setting the Stage for Discussion

Many associate Pair Programming and Inspections with the coding phase of a project. While this is not necessarily true, we still decided to view Pair Programming as a coding activity and compare it to solo programming and Inspections.

We focused the discussion on the home grounds for the two practices. The term "home ground" is often used to characterize the application context for which a specific software development approach is best suited. According to Barry Boehm and Richard Turner, the home ground of a development approach is "a set of conditions under which it is most likely to succeed" [5][6]. Application, management, technical, and personnel characteristics define the home ground of a software development approach [5]. Boehm and Turner describe two possible home grounds: The agile home ground (primary goals are rapid value and responding to change) and the plan-driven or CMM-based home ground (primary goals are predictability, stability, and high assurance). Specific development practices, such as Pair Programming or Inspections can be assigned to a specific home ground: Pair Programming is assigned to the home ground of agile development approaches and Inspections to the home ground of CMM-based approaches. However, this may limit the applicability of different practices, as they may have strengths that can be of use even if they are applied in a context different from their home ground. Therefore, we also wanted to elicit experiences about the benefits of applying one or the other practice in "foreign" home grounds, for instance, applying Pair Programming in plan-driven approaches, or applying Inspections in agile development.

To elicit relevant experiences about the strengths and weaknesses of Pair Programming and Inspections, we proposed a set of initial attributes for the comparison (like quality and cost) and asked participants to propose others of importance based on their experiences. In this paper, we focus on three of the attributes that were shown during the discussion to best illustrate features of the use of both practices outside their traditional home ground. These attributes are: effect on quality (i.e., number of defects remaining in the system after using one of the practices), third party perspective (i.e., the ease of incorporating additional perspectives, resulting in more objectivity), and the feedback-cycle (i.e., period of time from appearance of a defect to its removal).

The following table summarizes the main points of discussion with respect to these attributes. The main outcome of the inspection with respect to the constructs is then outlined in detail in the following section.

Copyright © Fraunhofer IESE 2005

Table 1: Summary of the discussion by comparison attribute

Attribute	Discussion results
Effect on quality	• Both practices have one common goal: producing higher quality products • Both techniques reduce number of defects that slip to the next phase • More empirical evidence needed on the benefits of pair programming
Third party perspective	• Important benefit of Inspections is having a third party perspective • Enables a focus on those aspects that might have the highest impact on quality • Pair Programming lacks a third party perspective • Collective code ownership and rotation of development teams are hypothesized to overcome this lack in Pair Programming
Feedback cycle	• The shorter the feedback cycle the better • Pair Programming has a very short feedback cycle as defects are immediately detected • Inspections have a longer feedback cycle, as the product under inspection needs to be in a somehow stable state

The full results of the eWorkshop related to all of the attributes discussed can be found in the online summary of the eWorkshop[3].

[3] http://www.cebase.org/www/Resources/eWorkshops/PP-insp-summary.html

Copyright © Fraunhofer IESE 2005

3 Effect on Quality

During the discussion, the experts agreed that one basic commonality of Pair Programming and Inspections is their common goal: producing higher quality products. The discussion about quality was centered around the number of defects that might slip through later development phases to field use when Inspections or Pair Programming are applied. In this context, the participants discussed which practice could achieve a higher defect reduction. The pre-meeting feedback and the discussion seems to indicate that with Inspections it is possible to achieve a very high level of quality (i.e., low rates of defect slippage), although at high cost, but with Pair Programming it is possible to achieve a lower quality at less cost. Bernhard Rumpe stated that the reason for this is "that if people are pairing 100%, then pairing is at its limit. The resulting defect rate cannot be reduced through further pairing alone—but through additional Inspections, as they are done post-construction." However, the participants agreed (this was confirmed by a vote – 18 out of 18 respondents agreed) that the benefits of Inspections, especially with respect to reduced defect slippage are well documented. In contrast, even though Pair Programming is hypothesized to lead to lower defect slippage, this is not yet well documented.

Other definitions of quality besides defect slippage were also discussed. For example, Erik Arisholm mentioned an experiment that indicates that applying Pair Programming leads to better maintainable code. For more detail, the interested reader may refer to the full summary.

Copyright © Fraunhofer IESE 2005

4 Third Party Perspective

A second important point of comparison between the two practices is the question of third party perspective. The third party perspective is an additional view on the document under Inspection; that is, people who are not directly related to the construction of the code under Inspection provide additional feedback on the code. The pre-meeting feedback indicated that a perceived strength of Inspections is that they can be more objective because they provide a third-party perspective.

The assumption behind having a third party perspective is that it increases the defect detection potential as you get a "fresh" mind checking the quality. Karl Wiegers summarized this by saying "When you are too close to a work product, you tend to believe it's correct and to trust all the assumptions you made; external reviewers are less biased, albeit less knowledgeable about the specific work product." Another issue regarding third party perspectives is that it is possible to choose those perspectives that might have the highest impact on the quality. Barry Boehm said that "one advantage of Inspections is that you can work on multiple qualities. Perspective-based Inspections enable artifacts to be reviewed by experts in safety, usability, performance."

There was some agreement among the participants that Pair Programming itself lacks the external third party perspective. There was a broad discussion on other agile practices that help to compensate this lack of external perspectives. Two of the experts stated that some of the benefits of outside, objective, or focused reviewers can be achieved via pair rotation and collective code ownership. There was some discussion on how serious this lack of objectivity is in Pair Programming. Barry Boehm felt that without allowing the involvement of multiple quality viewpoints, converging on system requirements is likely to be problematic: "From a stakeholder win-win perspective, just getting two people to determine the correctness of requirements is very risky, as it excludes success-critical stakeholders from the process." Again, other participants agreed but said that this is exactly the reason why Pair Programming should always be implemented with more than one pair on the team, collective code ownership, and the rotation of people through work pairs/groups.

Copyright © Fraunhofer IESE 2005

5 Feedback Cycle

A third point that was discussed with respect to achievable quality is the feedback cycle of Pair Programming and Inspections. The feedback cycle is the amount of time between committing a defect during the software development process and detecting and removing that defect. All discussants agreed that a short feedback cycle is most valuable for software development. Most of the experts agreed that the extremely short feedback cycle of Pair Programming is a clear advantage of that practice compared to Inspections where the feedback cycle is longer. Some experts mentioned that the feedback of Pair Programming has a greater "present value." That is, its feedback cycle is supposed to be more cost effective as the defect is corrected as soon as it enters the system, or is even prevented; thus, it cannot cause follow-up defects. In contrast, Inspections have to wait until the product that is to be inspected is somehow in a stable state. Several participants also felt that "if people think the work product is done, they can be psychologically resistant to making changes suggested by Inspection." This indicates that a short feedback cycle may have more advantages than just early defect removal.

The experts discussed whether it is possible to overcome or at least to minimize the drawbacks of Inspections resulting from a longer feedback cycle. Two participants stated that Inspections should be performed iteratively; that is, the Inspection should start as soon as some parts of the work product under Inspection are available. Karl Wiegers stated that "a good heuristic is to start Inspections as soon as 10% of the document is available, rather than waiting until the whole document is done at which point it may be more costly to repair all the defects."

Copyright © Fraunhofer IESE 2005

6 Using practices in foreign home grounds

Based on the strengths and weaknesses of the two practices, we present the results of the eWorkshop discussion regarding the usage of the practices in the respective other home ground; that is, we identify under which circumstances it might be valuable to apply Inspections in an agile project, and in which cases it may be valuable to apply Pair Programming in a CMM context. In the table below, the main statements of the experts with respect to this question are summarized

Table 2: Using techniques in foreign home grounds

Context	Statements
Inspections in an agile context	• Involvement of various perspectives beneficial to overcome • Inspections should be used in agile context when high quality is desired, i.e. the system is safety critical
Pair Programming in an CMMI context	• Short feedback cycle of Pair Programming might be beneficial • Usage of Pair Programming should be driven by level of quality, experience of developers and the application domain

6.1 Applying Inspections in an agile context

During the pre-meeting and the eWorkshop, the experts were asked to consider the following scenario: "Assume you are applying XP on a project, including Pair Programming and other practices. The team lead wants to schedule a code Inspection of a key module before the delivery date of an important increment, but many of the developers feel that Inspections are always redundant if they are using Pair Programming. Whose side would you be on?" With this scenario the experts discussed particular circumstances that would change the relevance of extra Inspections in an agile context.

There was a controversial discussion about the defect detection potential and the level of quality that can be achieved with Pair Programming. Inspections can add a third party perspective and therefore can help detect defects missed by Pair Programming. However, some experts stated that if domain experts perform Pair Programming correctly, there should be no need for an extra Inspec-

Copyright © Fraunhofer IESE 2005

tion. In contrast, most of the experts agreed that there might be special cases when it is valuable to perform extra Inspections after Pair Programming. The decision whether to do so depends on the desired level of quality of the product. William Krebs said he feels "that neither Pair Programming, nor unit testing, nor formal Inspections catch 100% of the errors." Thus, a combination of the different practices may help to further reduce the chance of still having an error in a document. Most of the experts agreed that especially key modules and critical aspects of the system should be inspected additionally to have a higher quality guarantee. Barry Boehm stated that the decision whether to apply extra Inspections depends on the risk exposure of having defects in the delivered product. If the risk exposure due to unfound defects is high, doing the Inspection is worthwhile although it may not always be. Thus, some experts agreed that the development of safety critical systems with agile methods is an application area where extra Inspections may particularly be needed.

6.2 Applying Pair Programming in a CMM context

To focus the discussion of CMM-based home grounds, the participants used the following scenario: "Assume you are developing software following a plan driven approach that includes Inspections on different documents. The team lead wants to substitute solo programming and code Inspections with Pair Programming. Under which circumstances would you find this appropriate?"

Including the pre-meeting results and the eWorkshop discussion, most of the participants agreed that the decision to substitute solo programming and Inspections with Pair Programming should be driven by factors such as the desired level of quality, the application domain, and the experience of the developers. One expert stated that by applying Pair Programming, only a specific level of quality can be reached which cannot be expanded without using additional activities such as Inspections. Another suggestion was to substitute solo programming and Inspections with Pair Programming, but only in the case of low complexity modules. If we develop in the domain of safety-critical or dependable systems, a substitution of solo programming and Inspections with Pair Programming does not seem to be appropriate. Furthermore, the experience of the developers should be taken into account. Only if the developers are experienced, can solo programming and Inspections be substituted with Pair Programming.

One strength of Pair Programming mentioned in Section 3 is its short feedback-cycle; that is, defects do not stay in a software system for a long time. Thus, we reach a certain level of quality in a shorter period of time compared to applying solo programming and Inspections, because Inspections have to wait until a product is somehow stable to be applied, as Bernhard Rumpe mentioned. Thus, if time to market is more important than quality, this may be one reason to substitute solo programming and Inspections with Pair Programming.

Copyright © Fraunhofer IESE 2005

7 Conclusions

Inspections and Pair Programming are practices that share a common goal of increasing quality, although they are otherwise very different in nature. Thus, one important research goal is to characterize the different practices and to identify in what situations they are best applied. In this paper, we presented the results of an eWorkshop that represents a first step in answering this question.

The comparison was centered around effect on quality, the value of a third party perspective, and their impact on the feedback cycle. With respect to effect on quality, the participants agreed that both practices help to reduce the number of defects that slip into the next phase, but that we need more evidence for Pair Programming. With respect to third party perspective, the participants agreed that Inspections might be more objective because they explicitly involve a third party. In Pair Programming, the code is the subject of in-depth examination by only two people at a time. Neither pair rotation nor collective ownership fully compensates the lack of third party perspectives. The short feedback cycle was identified as a strength of Pair Programming, as defects are prevented or detected early. Inspections have to wait until the product is somehow stable; that is, defects may stay longer in the system.

One outcome of the eWorkshop is that all participants agree that it may make sense to combine the practices under certain conditions. For example, in the case of developing safety critical systems or the need for high quality end-products Inspections may be valuable to apply in agile home grounds, as their effectiveness and the possibility to involve various perspectives may help compensate for some shortcomings of Pair Programming. On the other hand, the short feedback cycle of Pair Programming may add value in CMMI-based home grounds, especially when the developed system is of lower complexity and the developers have a high domain experience.

In addition, during the pre-meeting feedback and the discussion, the participants raised several research questions that should be addressed in the future. For example, we need more information about the impact of the techniques on correctness, reliability, maintainability, etc. Furthermore, we would like to know what quality level can be reached with which practice, and which of the practices is useful for which type of defect, and for which type of documents.

All in all, the application of Pair Programming and Inspections in each other's respective home ground seems to be valuable under specific circumstances. More research is needed to develop a detailed understanding of the two practices to be able to find a trade-off between them in a concrete development

Copyright © Fraunhofer IESE 2005

context. The goal should be to define a process that gives explicit guidance on when to apply which practice, taking into account the desired level of quality, costs, benefits, criticality, and complexity of modules as well as the risks of a software failure after delivery.

Acknowledgements

We would like to thank all the participants of the eWorkshop for their valuable contributions, in alphabetical order: Scott Ambler, Erik Arisholm, Victor Basili, Barry Boehm, Winsor Brown, Tom Gilb, Philip Johnson, Bill Krebs, Oliver Laitenberger, Filippo Lanubile, John Manzo, Frank Maurer, Steve McConnell, Dieter Rombach, Bernhard Rumpe, Barbara Russo, Gunjan Sharman, Alberto Sillitti, Karl Wiegers, Laurie Williams. This work is sponsored by ViSEK (funded by the German Federal Ministry of Education and Research under grant 01ISA02) and CeBASE (funded by the National Science Foundation under grant CCR0086078).

Copyright © Fraunhofer IESE 2005

References

[1] Basili, Victor R.; Evolving and Packaging Reading Techniques; Journal of Systems and Software 38 (1); 1997.

[2] Basili, V., Tesoriero, R., Costa, P., Lindvall, M., Rus, I., Shull, F., Zelkowitz, M.: Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop. PROFES 2001, pp 110-125.

[3] Fagan; Michael E.; Design and Code Inspections to Reduce Errors in Program Development; IBM System Journal, 15 (3); 1976.

[4] Beck, K.: Extreme Programming Explained - Embrace Change. Addison Wesley, Reading, Massachusetts, 2000.

[5] Boehm, B., Turner, R.: Balancing Agility and Discipline – A Guide for the Perplexed. Addison Wesley, 2003.

[6] Boehm, B.; Turner, R.: Balancing your Organization's Agility and Discipline, XP/Agile Universe 2003, LNCS 2753, pp. 1-8, 2003.

[7] Cockburn, A., Williams, L.: The Costs and Benefits of Pair Programming. In Extreme Programming Explained, Addison-Wesley, pp. 223-243, 2001.

[8] Constantine, L.L.: Constantine on Peopleware. Yourdon Press, Englewood Cliffs, N.J., 1995.

[9] Fagan; Michael E.; Design and Code Inspections to Reduce Errors in Program Development; IBM System Journal, 15 (3); 1976.

[10] Gilb, Thomas; Graham, Dorothy; Software Inspections; Addison-Wesley Publishing Company; 1993.

[11] Laitenberger, Oliver; Cost-effective Detection of Software Defects through Perspective-based Inspections; PhD Thesis in Experimental Software Engineering; Fraunhofer IRB Verlag; 2000.

[12] Laitenberger, O., DeBaud, J: .An Encompassing Life-Cycle Centric Survey of Software Inspection, Journal of Systems and Software, 50 (1), 2000.

[13] Lindvall, M.; Basili, V.; Boehm, B.; Costa P.; Dangle K.; Shull, F.; Tesoriero, R.; Williams, L.; Zelkowitz, M.: Empirical Findings in Agile Methods, Proceedings of XP/Agile Universe 2002, pp. 197-207.

Copyright © Fraunhofer IESE 2005

[14] Strauss, S.H.; Ebenau, R. G.; Software Inspection Process; McGraw Hill Systems Design&Implementation Series; 1993.

[15] Travassos, H. Guilherme; Shull, Forrest; Fredericks Michael; Basili, Victor; Detecting Defects in Object Oriented Designs: Using Reading Techniques to Improve Software Quality; In the Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA); Denver, Colorado; 1999.

[16] Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R.: Strengthening the Case for Pair Programming. IEEE Software, July/August 2000.

[17] Wiegers, Karl: Peer Reviews in Software – A practical Guide, Addison-Wesley, 2002.

Copyright © Fraunhofer IESE 2005

Appendix: Definitions

Side bar: Pair Programming

One item of discussion in our eWorkshop was Pair Programming and its strengths and weaknesses. In Pair Programming, two programmers are working together, side by side, sharing one computer [16][7]. One of them takes the role of the *driver* who operates the keyboard and the mouse and writes the code. The other one, the *observer*, watches the driver's actions, tries to find errors and plans ahead; that is, he continuously reads through the code written by the driver and checks its quality. After a certain time the developers switch the roles, or the teams can rotate, which means new pairs are composed. Pair rotation aims at distributing knowledge in a software development team.
Pair Programming has been described several times in the last decades as an alternative to solo programming [8]. With the rise of agile software development Pair Programming gained in importance because it enforces quick feedback. Pair Programming is a key practice in several agile development approaches, for instance, Extreme Programming (XP) [4]. Pair Programming is a means of analyzing, designing, implementing, and testing a software system [16]. In this discussion, we focus on Pair Programming as an implementation practice to compare it to the combination of solo programming and code Inspections.

Side bar: Software Inspections

Software Inspections are an industrial-strength quality assurance technique that is widely used in many industrial domains. The Inspection approach was first published by Fagan [9]. Fagan Inspections were focused on detecting defects in code documents. During the last decades, the Inspection approach was tailored to other software engineering artifacts; for example, to requirements documents, test cases, and design documents [10][17][11][14][15][1]. The benefits of Inspections, especially their effectiveness and efficiency in reducing defects, are well documented in many empirical studies [12].
In this discussion, we focus on code Inspections to better compare it with Pair Programming. Thus, in our context, Inspections are defined as a static verification technique of code documents, where a set of inspectors reads a code document to ensure that certain quality criteria are fulfilled. Inspections are performed according to a defined process and follow the principle of getting many eyes on a document; that is, people with relevant technical knowledge verify its quality.

Copyright © Fraunhofer IESE 2005

Document Information

Title:	Inspections and Pair Programming – competing or complementary? Experiences from an Expert eWorkshop
Date:	May 4, 2005
Report:	IESE-029.05/E
Status:	Final
Distribution:	Public

Copyright 2005, Fraunhofer IESE.
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means including, without limitation, photocopying, recording, or otherwise, without the prior written permission of the publisher. Written permission is not needed if this publication is distributed for non-commercial purposes.