An Empirical Study on Design Quality Improvement from Best-Practice Inspection and Pair Programming

Dietmar Winkler and Stefan Biffl

Vienna University of Technology, Institute of Software Technology
Karlsplatz 13, A-1040 Vienna, Austria
{Dietmar.Winkler, Stefan.Biffl}@qse.ifs.tuwien.ac.at

Abstract. The quality of the software design often has a major impact on the quality of the final product and the effort for development and evolution. A number of quality assurance (QA) approaches for inspection of early-life-cycle documents have been empirically evaluated. An implicit assumption of these studies was: an investment into early defect detection and removal saves higher rework cost. The concept of pair programming combines software construction with implicit QA in the development team. For planning QA activities, an important research question is how effective inspectors can be expected to be at detecting defects in software (design and code) documents compared to programmers who find defects as by-product of their usual construction activities.

In this paper we present an initial empirical study that compares the defect detection effectiveness of a best-practice inspection technique with defect detection as by-product of constructive software evolution tasks during pair programming. Surprisingly, in the study context pair programmers were more effective to find defects in design documents than inspectors. However, when building a larger team for defect detection, a mix of inspection and pair programming can be expected to work better than any single technique.

Keywords: Verification & Validation, Inspection, Pair-Programming, Nominal Teams, Empirical Software Engineering.

1 Introduction

Traditional software development life cycle processes typically consist of five steps: requirements definition, system design, implementation, integration, and operation. Quality assurance (QA) activities embedded within the software development process are a key to achieve a higher level of software quality. In industrial practice a low number of defects in a software product are a measure for product quality. Product improvement also leads to a reduction of repair effort due to defects in artifacts, e.g., specification documents, requirements definitions, or code. Rework effort for defect repair can increase rapidly the later a defect is detected and removed [18]. Thus, early removal of defects in design documents and the requirements specification is expected to lead to better-quality products and to improve project performance, i.e., result in overall lower effort and cost.

J. Münch and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034, pp. 319 – 333, 2006.

Inspection and testing are common analytical methods for software product improvement along the life cycle to minimize the number of remaining defects. Obviously, both techniques can require considerable effort in the face of scarce resources in a project. To reduce this effort, an option is to strengthen constructive QA approaches in order to detect and remove defects shortly after they occur. Pair Programming (PP) aims at supporting the construction of higher-quality software products using a small-team approach that is expected to reduce rework effort.

An important parameter for planning QA activities is how effective a best-practice analytical QA approach (inspection) is compared to just using a constructive approach with implicit QA, such as PP. In this paper we report on an initial empirical study in an academic environment [6] that compares the performance of a best-practice constructive approach (PP) and a best-practice QA technique. We applied usage-based reading (UBR), a well-investigated reading technique for software inspection, as representative approach for best-practice defect detection. Further, we investigate the impact of team size and team composition involving PP teams and UBR individuals for improving software product quality, i.e., detecting defects. The comparison of the effort of the different methods is more difficult because of the different emphasis, proceedings, and outcomes of the two approaches. However, even the comparison of quality measures provides an interesting initial baseline and deeper understanding of the defect reduction characteristics of the investigated approaches.

The reminder of this paper is structured as follows. Section 2 describes related work on pair programming, software inspection, and team composition. Section 3 summarizes the research hypotheses; Section 4 outlines the experiment setting. Section 5 presents the results of the empirical study and Section 6 discusses these results. Finally, Section 7 concludes and sketches directions for further research.

2 QA Aspects of Usage-Based Inspection and Pair Programming

Software processes structure development to systematically achieve higher-quality software products. In this context we define software quality based on the number of (important) defects in a software product. In most software processes reviews (or inspections) are performed at milestones to check artifacts for quality compliance and correctness. An important goal is to reduce the number of defects effectively.

Recently, agile programming practices have been introduced that foster rapid QA feedback during software construction activities. A particularly promising approach is pair programming, where two persons jointly conduct construction tasks; one team member implicitly supports QA by questioning unclear work results of the other partner and by pointing out defect candidates as they occur. However, there are very few studies on the QA effect of PP in direct comparison to software inspections. In this initial study we pay special attention to the benefits of usage-based inspection and constructive software development approaches, as proposed in [6].

2.1 Defect Reduction with Usage-Based Reading During Design Inspection

Software inspection is a well-known, team-oriented, and empirically evaluated static verification and validation approach for the improvement of software artifacts [15][23]. The nature of inspection makes the method applicable to all types of

software documents, i.e., design documents and source code documents [9][10], because inspection does not require executable code.

In industrial practice, inspection processes consist of four major steps: inspection planning, defect detection, defect collection, and defect repair [23]. The inspection process includes a defect detection task, where individual inspectors (within an inspection team) traverse the document under inspection following a reading technique for guidance. A reading technique is a structured process for defect detection that uses predefined roles, scenarios, checklists, etc. Inspectors follow a set of guidelines and/or checklists defined by the reading technique. Several reading technique approaches have been discussed and investigated experimentally [1][3][15][17].

Requirements and use cases are the units of interest for usage-based reading (UBR) [15][16]. Use cases describe business cases on a defined level of detail to achieve full coverage of requirements in design specification and source code.

In this paper we use UBR as baseline for an active approach for defect detection [22] in design and code documents. UBR is a best-practice approach that focuses on scenarios and a pre-defined order of use cases (ranked according to their business value contribution) [2], which are understandable for customers and developers as well, using graphical representation and additional textual information for scenario and business case description. Experienced domain experts prioritize the use cases according to their business importance. UBR inspectors apply prioritized uses cases sequentially to the inspection artifacts starting with the most important use case and report candidate defects. Candidate defects are subjectively raised issues, recognized by the inspectors during the individual reading process. Note that inspectors do not fix defects during defect detection. Defect candidates are labeled as real defects or false positives after individual inspection by a group of experts in the project context. Use case prioritization is part of the preliminary inspection preparation phase and not in the scope of this paper. In the empirical study environment UBR inspectors perform the following sequence of steps [23]:

1. Choose the use case with the highest priority.
2. Apply use case to the documents under inspection and record candidate defects.
3. Continue with the next use case until time is up or all use cases are covered.

The third step of the traditional software inspection process is defect collection. Because of the team-oriented approach of inspection, a set of individual defect detection lists exist after individual inspection. Team meetings are one option for defect detection list aggregation. An alternative is a nominal team, i.e., a non-communicating team for defect collection after independent individual inspection [4][5]. The application of nominal teams ignores the impact of real team meetings and real interaction between team members [3]. Some studies [4][5] doubt the net benefit of team meetings. Nominal teams can be built from a permutation of the available individual inspectors.

2.2 Defect Reduction with Pair Programming in Design Artifacts

Pair programming (PP) is a constructive development technique that includes QA tasks for agile software development. Agile software processes [8] use smaller and

more frequent iterations for better cooperation between customers and developers and to foster communication in the software engineering team. These smaller steps support the project team in constructing software products that better meet the needs of the customers, partly due to smaller entities for QA [8].

A PP team consists of two persons, sharing one keyboard and one monitor. While the first person implements tasks according to pre-defined use cases and scenarios, the other person looks over her shoulder to raise issues in the new work results or defect candidates in the specification and code documents. The roles of team members can change frequently. Due to the involvement of "two brains" [19][20], the team can achieve a twofold benefit: (a) higher-quality development of the software product because of the effective involvement of two persons who work together, and (b) improvement of software quality within all parts of the project during software construction. The latter benefit is also a twofold one: (a) increasing quality because of a lower number of defects in the new software code due to continuous reviews [7] and (b) increasing quality of existing software artifacts; thus defect reduction is a by-product of PP. The key question of this paper focuses on the comparability of defect detection capabilities of traditional inspection approaches and constructive approaches, where defect detection is considered as a by-product.

In this paper we use a PP variant, proposed in [6], to accelerate agile processes by adding QA tasks as part of the constructive phase. The pair programmers follow the prioritized list of expert-ranked use cases and perform their construction tasks. In these tasks, they have to use existing material (requirements specification, existing code fragments, etc) to understand their working environment and also check the plausibility and consistency (analytical tasks); then they evolve the existing source code following the predefined order of use cases (constructive tasks).

While one part of the pair implements and completes open programming tasks, defined within the work package, the other person looks over her shoulder to check the code for correctness. In more detail, this process consists of four steps:

1. Select the use case with the highest priority.
2. Compare requirements and use cases to the design specification and already implemented code fragments. Report candidate defects in case of deviations.
3. Work on requirements (including previous check for correctness).
4. Continue with the next use case until all use cases are covered or time is up.

To our knowledge, there is an increasing number of empirical studies on PP [12][13][14][21], however, very few investigate the quality improvement effect on existing software products [11]. Müller et al. [11] propose additional review and testing phases after PP tasks to achieve internal validity in an experiment environment. In the empirical study reported in this paper, we did not perform additional QA tasks, but asked pair programmers to note defect candidates as by-product of construction. PP's primary task is the implementation of new software code fragments and defect detection as a by-product with focus on software code (using the usual development environment, i.e., computer, compiler, and tool support).

3 Research Issues

The main focus of this paper is the comparison of the defect reduction effect as by-product of a constructive software development to a focused defect detection technique. To achieve comparability of defect detection rates, we focus on finding defects in input documents to inspection and evolution, i.e., design and code.

3.1 Variable Definition

We define dependent and independent variables. The independent variable is the technique applied: PP (with 2-person teams) and UBR (with individual inspectors). We controlled the influence of participant capability by randomly assigning them to techniques (PP and UBR) and PP teams.

Dependent variables capture the performance of the individual techniques applied in the experiment. We focus on defect detection effectiveness, i.e., the share of defects found in relation to all defects seeded, as performance measure to investigate the influence of each approach on defect detection capability of existing software artifacts. In addition to the overall number of defects we also apply defect severity classes: critical defects (class A), major defects (class B), and minor defects (class C). For evaluation purposes we focus on important defects, i.e., the summary of critical and major defects (class A+B).

Furthermore we provide data on the effort of technique application for background information. Because of a different focus of the individual approaches, a comparison of effort would not seem useful. Note that we cover experiment participation duration, not the duration of experiment setup and artifact preparation.

In addition to measuring the effectiveness of PP teams and individual inspectors we measure the performance of teams with the parameters team size and team composition (the techniques participants in a team used) to investigate the impact of nominal teams on defect reduction.

3.2 Hypotheses

In the experiment we first observed effectiveness of techniques according to important defects (classes A+B) for PP teams and UBR individuals. In a second round we build nominal teams [4] from the performance of average experiment participants to investigate the effectiveness contributions of the two techniques.

In this paper we investigate hypotheses: on the effectiveness of work units (H1); on the effectiveness of 2 persons in a nominal UBR team or a PP team (H2); and on the effectiveness of technique variations in nominal teams of different sizes (H3).

- *H1.1: Effectiveness (PP) > Effectiveness (UBR) for source code documents.* To analyze the benefits of "natural work units" we compare PP teams with one UBR inspector. Note that PP involves higher effort and more persons, but also has a focus that is different from finding defects. PP focuses primary on implementation including defect detection as a by-product. Therefore, we expect a higher effectiveness of PP teams for source code documents.
- *H1.2: Effectiveness (PP) < Effectiveness (UBR) for natural-language text documents.* In contrast, the main scope of UBR inspectors is defect detection with

focus on written text documents, e.g., design documents (DD). Therefore, we expect a higher effectiveness of UBR regarding written text documents.

To improve comparability among the techniques according to team size, we build nominal 2-person UBR teams (so-called minimal teams, MT) randomly (complete permutation). The hypotheses are similar to H1.

- *H2.1: Effectiveness (PP) > Effectiveness (UBR-MT) for source code documents and minimal teams.* The argument is similar to H1.1 because of the different main focus of both approaches. Because of an additional UBR inspector we expect a somewhat higher effectiveness.
- *H2.2: Effectiveness (PP) < Effectiveness (UBR-MT) for natural-language text documents and minimal teams.* Because of an additional UBR inspector we expect strengthened results similar to H1.2 with higher deviation of design document effectiveness.

Because both approaches focus on different issues, we assume a team consisting of members from both techniques to combine their benefits.

- *H3.1: Effectiveness (PP+) > Effectiveness (UBR+) for source code documents and nominal teams.* We expect a higher effectiveness in a team from adding a PP team (PP+) rather than compared to adding a similar number of UBR inspectors (UBR+).
- *H3.2: Effectiveness (PP+) < Effectiveness (UBR+) for design documents and nominal teams.* Additionally we expect a higher defect detection rate for written text documents from additional UBR individuals (UBR+) in a team.

3.3 Experiment Description

The study material is based on a taxi management system, provided by Thelin et al. [15][16], who investigated different reading technique approaches. We proposed an empirical investigation of analytical QA activities (best-practice inspection) with constructive approaches (PP) in [6]. First empirical results were published in [23].

The main task of UBR inspectors was defect detection in software documents. PP teams used similar artifacts including a constructive task, i.e., extending the source code according to prioritized use cases. The main purpose of this paper is the investigation of the performance of defect detection, i.e., the number of found defects in relation to the total number of seeded defects (effectiveness) for important defects and depending on defect locations in documents (source code documents and written text documents). All participants applied a pre-defined set of prioritized use cases, a business case description, a requirements document, a design specification, and a set of guidelines for method application. The design specification and the source code documents contained seeded defects.

The experiment was conducted in three steps: (a) experiment preparation, (b) experiment execution, and (c) data evaluation.

During *experiment preparation* experts prepared the software artifacts, i.e., the requirements definition including use cases description and prioritization according to business process importance. Selected parts of the software code were provided by the experiment preparation team. The design document and source code documents were

seeded with a pre-defined set of defects. The other documents were improved in extensive review cycles until they were found to be correct.

The *execution phase* consisted of 3 steps: (a) tutorials where participants got an overview on the software system and the application domain (45 min) as well as training on the application of their technique (45 min), (b) the individual application of the technique (up to 300 min for inspection and up to 600 min for PP individuals), and (c) data submission to a database. The participants were supported by experiment supervisors, to clarify upcoming questions and to check the produced products for accuracy and usability. Concerning team evaluation, we counted multiple matched defects (within a team) only once. During individual application, the inspectors/pair programmers found candidate defects in the software artifacts and submitted them to the defect database.

Data evaluation: Members of the experiment team mapped defect candidates to seeded defects ("real defects"). Candidate defects that matched to seeded defects were classified as matched defects. Note that multiple candidate defects, which matched to one specific seeded defect, were counted only once, i.e., at the first time of detection and reporting. We built nominal teams from UBR individuals and PP teams by applying full permutation of all available data for the desired team size. For UBR-MT, the team size was 2 UBR persons. For statistical testing we applied the Mann-Whitney test at alpha level 0.05.

3.4 Software Artifacts

The system describes a taxi management system, presented by Thelin et al. [17]. We extended the experiment package to UML notation and included additional constructive approaches, i.e., PP [6]. First empirical results on the replication part (without constructive approaches) were published in [23].

The taxi management system consists of 2 parts, a central and a taxi. In our evaluation context we investigated defect detection capability in both parts (view on systems level).

Fig. 1. Taxi management system – overview according to [17]

The experiment setup consists of (a) a textual description of the requirements and the use cases in users view, (b) the design document as well as the source code containing seeded defects and (c) the guidelines for the techniques applied as well as questionnaires for determining inspector capability and feedback.

- The textual requirements document consists of 8 pages including 2 UML2 component diagrams. The textual requirements document describes the basic functionality of the system in a user-friendly way.

- The design document spans 8 pages (including about 2,400 words, 2 component diagrams and 2 UML diagrams). We describe an overview of the software modules as well as their context including internal (relationships between two or more modules) and an external representation (relationships between the user and the system). Furthermore, we provide prioritized use case descriptions containing 24 use cases from user point of view and an overall number of 23 sequence diagrams. This artifact describes the technical dimension of the taxi management system.
- We provide source code fragments (some 1,500 lines of code) written in Java2, seeded with defined defects, and a method description of about 9 pages. These source code fragments were used in a twofold way: (a) to investigate the inspection effectiveness for source code, and (b) to investigate the QA effectiveness of PP and when extending the existing modules.
- The participants use guidelines for application of the assigned technique. Furthermore we provide questionnaires to measure inspector/programmer experience, capability indicators, and feedback on the individual techniques.

The design specification and the source code documents were seeded with a pre-defined number of defects by highly experienced experts.

Table 1. Distribution of Seeded Defects

Defect class	Design	Source	*Sum*
A (critical)	10 (17%)	19 (32%)	*29 (49%)*
B (major)	12 (20%)	12 (20%)	*24 (40%)*
C (minor)	5 (8%)	2 (3%)	*7 (11%)*
Sum	*27 (45%)*	*33 (55%)*	*60 (100%)*

Critical defects (class A) would have a severe and frequent impact on important functionality. Class B (major) defects are rarely occurring important defects or less important frequent defects (medium risk). Minor defects appear seldom and have little influence on functionality and quality. The document package (design specification and source code documents) contains overall 60 seeded defects according in three defect severity classes and two defect locations. Table 1 presents the nominal numbers of seeded defects according to defect severity classes and document types. In this paper we focus on important, i.e., critical and major (classes A and B), defects.

3.5 Subjects

The subjects in this initial study were 41 graduate software engineering students. We used a PP qualification test for candidate participants to ensure sufficient implementation skills. All participants were assigned randomly to the techniques, PP and UBR, to control the influence of inspector capability and to achieve better external validity. The experiment was integrated in a practical part of a software engineering and quality assurance workshop. We assigned 15 inspectors (54%) to UBR and 26 persons (46%) to 13 PP teams.

3.6 Threats to Validity

We controlled the external validity by randomly assigning participants to UBR and PP. Additionally, all candidates had to pass a PP qualification test to ensure their sufficient programming skills. 41 subjects (of about 60 candidates) passed this test. This qualification test was used to enable comparability to an industrial setting and to minimize the influence of variance of inspector capability. We did not perform a qualification test for inspectors; however, all participants had had classes on the skills needed for inspection in their regular curriculum.

We seeded representative defects in the design specification and source code documents according to different types of defects and defect locations. The seeded defects were representative of defects found during the development of the documents under study. For achieving higher external validity with defects in specific industry setting, replicated studies would be appropriate that consider the typical range of defects in the target context.

The correctness of the requirements document was achieved with extensive review cycles. To achieve better internal validity the experiment preparation team set up the experiment package under guidance of experienced researchers including several reviews. To achieve comparability to previous studies [15][16], we used approved material modified to fit the experiment context.

Note different focus of inspection and PP. Inspection focuses on defect detection as primary task. PP focuses on the construction of software code and defect detection as a by-product.

4 Experiment Results

In this section we present results of the initial empirical study concerning effectiveness of work units, minimal teams, and some preliminary results of nominal team composition to investigate the influence of mixtures of both approaches.

4.1 Effort

We report effort to illustrate the background of the study. In the study context, we define effort as the overall session duration, including individual preparation (reading the documents and getting familiar with the technique applied) and technique application time. We do not consider experiment preparation time which was done by experts as preliminary work packages before the experiment started.

PP teams needed 1,030 min (about 17 person-hours (ph)) on average and a standard deviation of 120min (2ph). Inspectors required 273 min (about 4.5 ph) with a standard deviation of 38 min (0.6 ph). Note that PP involves 2 persons and took much longer than UBR because of the main focus on the implementation of additional software code. Defect detection is considered as by-product of PP in this study.

4.2 Effectiveness of Work Units

Effectiveness is the number of defects found in relation to the number of seeded defects according to defect class and defect location in a document (design document and source code documents). The evaluation covers important defects (class A+B), which should be supported by prioritized use cases.

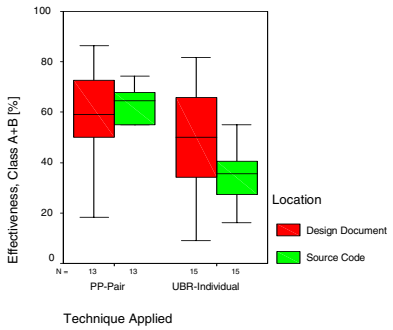

Fig. 2. Effectiveness of important defects acc. to defect location

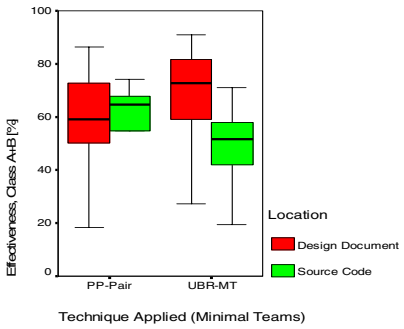

Fig. 3. Effectiveness of important defects from minimal teams

*Effectiveness of Work Units***.** We concern "work units" as the original configuration of participants applying each technique: PP team and UBR inspector. The results presented in Figure 2 show best effectiveness for important defects according to all defect locations for PP teams. We observe a smaller difference (9 defects on average) for design document effectiveness, but a higher difference (21 defects on average) for source code effectiveness.

Table 2. Effectiveness, important defects

	Location	PP-Pair	UBR-Individuals	P-value
Mean	DD+SC	56.3	40.3	0.013 (S)
	DD	56.3	47.3	0.212 (-)
	SC	56.3	35.3	0.004 (S)
Std.Dev	DD+SC	20.6	13.6	-
	DD	26.7	20.6	-
	SC	17.9	11.4	-

Nevertheless, PP outperforms UBR for all defect severity classes. Note also a higher standard deviation for design defects found by UBR inspectors. Table 2 presents an overview of effectiveness according to technique applied and defect location. Applying the Mann-Whitney-Test to investigate significant differences, we observe significant differences for all documents (DD+SC) and source code (SC), but no significant difference with respect to design documents (DD). Obviously, the investigated implementation approach (PP) outperforms the individual inspection approach (UBR) for SC documents.

A more detailed investigation of this defect type shows the benefits of the PP approach more clearly. A subset of the seeded source code defects (4 defects) focus on logic, dataflow, and visibility defects, e.g., private and public statements, which may be found more easily applying an implementation approach rather than a paper-based inspection approach. Additionally, PP uses a computer and a compiler to find defects, which supports defect detection in source code documents. The results show

an effectiveness of about 79% according to these defect types for PP and 23% for UBR a significant difference.

4.3 Effectiveness of Minimal Teams (MT)

To achieve comparability of team size, we compare the original work unit of PP teams to a nominal team of 2 UBR inspectors. Figure 3 displays the results of minimal team effectiveness for important defects according to defect locations. Table 3 depicts mean value and standard deviation of effectiveness according to important defects (class A+B) with respect to minimal teams, i.e., PP team and randomly assigned 2 person UBR nominal teams (UBR-MT).

Table 3. Effectiveness of Minimal Teams (MT)

	Defect	PP-Pair	UBR-MT	p-value
Mean	DD+SC	56.3	57.8	0.292 (-)
	DD	56.3	68.6	0.680 (-)
	SC	56.3	50.1	0.014 (S)
Std.Dev	DD+SC	20.6	10.5	-
	DD	26.7	15.8	-
	SC	17.9	9.9	-

The UBR-MT approach achieves a notably, but not significantly, higher average effectiveness with respect to the all documents and design documents, but there is a difference for PP teams. Concerning defect detection regarding source code defects, PP teams outperform UBR-MT significantly.

4.4 Effectiveness of Nominal Teams

Because the two techniques focus on different aspects of a software document, we expect a higher defect detection effectiveness of teams, consisting of participants of UBR and PP. Team composition can achieve a twofold benefit: (a) more efficient handling of source code documents with tool support (naturally available for PP) and (b) higher defect detection rate in written text documents due to the application of reading techniques (in our study: UBR). Table 4 depicts the team composition regarding PP-teams and UBR individuals to a team size up to 5 team members. Note that PP teams consist of 2 persons and the table shows individual persons (we did not split PP teams). The "team size" describes the overall number of participants (persons) within a nominal team. The participants were assigned to a nominal team randomly, performing full permutation. Regarding team notation, *P* defines the integration of a number of PP-teams in the team, while *R* indicates the number of UBR individuals included.

Figure 4 displays the mean of effectiveness according to the nominal teams with respect to important defects (class A+B). Note that the figure includes also the overall effectiveness according to all matched defects as control value for nominal team evaluation. The analysis shows an increasing effectiveness, regarding defect detection rates for design documents, independent of the technique applied. A closer view

shows somewhat smaller gain including another PP-Pair within the nominal team. Concerning source code location, we observe an increasing effectiveness for PP-teams and a constant value (team size 4) and decreasing value (team size 5) when including UBR individuals.

Table 4. Nominal team composition (individuals)

	Team-Members	PP Ind.	UBR Ind.	Team Size (number of individuals per team)
Teams	PR	2	1	3
	PP	4	0	4
	PRR	2	2	4
	PPR	4	1	5
	PRRR	2	3	5

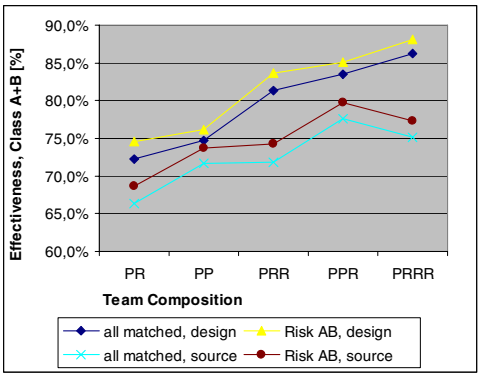

Fig. 4. Effectiveness of nominal teams

Table 5. Team effectiveness, class A+B

	A+B / Team Size	PR 3	PP 4	PRR 4	PPR 5	PRRR 5
Mean	DD+SC	71.2	74.8	78.1	**82.0**	81.8
	DD	74.6	76.2	83.6	85.1	**88.2**
	SC	68.7	73.7	74.2	**79.7**	77.4
Std.Dev	DD+SC	12.6	13.3	9.1	8.6	7.2
	DD	16.4	17.2	11.3	10.9	8.1
	SC	11.8	11.6	9.6	8.5	8.5

We observe a similar trend for all matched defects and for important defects (class A+B). Table 5 displays mean value and standard deviation for effectiveness of important defects according to defect location.

5 Discussion

In Section 3.2 we presented hypotheses on the expected effectiveness of individuals, real teams, and nominal teams regarding the defects in different documents. This section discusses the hypotheses with the experiment results.

Effectiveness of Work Units (H1). Concerning the effectiveness of work units (PP team and UBR individuals) according to defect location and important (classes A+B) defects, we expected a higher effectiveness of PP for source code documents and a higher effectiveness of UBR for text documents. The results show a higher effectiveness of a PP team for both aspects. PP significantly outperforms UBR concerning defect in the whole experiment package (DD+SC). We did not recognize a significant difference for design documents. The results support hypothesis (H1.1), that effectiveness of PP outperforms UBR according to source code documents. Hypothesis (1.2) that UBR performs better than PP for design documents could not be supported in the study context. We assume two possible reasons for this effect: (a) PP requires additional effort for implementation and defect detection (therefore more defects were found), and (b) implementation tasks support defect detection in design specifications because of immediate observation of the corresponding impact of defects. While implementing, defects in the design specifications might be detected easier than during reading a document.

Effectiveness of Minimal Teams (H2). Because PP involves 2 persons and UBR is a single person defect detection approach, we set up minimal nominal UBR teams to compare effectiveness according to team size, defect location, and defect severity. Again, we assume a better performance of PP for source code defects and a better performance of UBR for defects in the design document.

Hypothesis (H2.1), PP outperforms UBR-MT for source code defects, was confirmed. We observe a significant difference for SC defects. Again we assumed that UBR-MTs perform better (H2.2) than PP teams for design documents. UBR-MTs find more defects than PP. The Mann-Whitney test does not flag significant differences. Therefore the results did not agree to our assumptions.

Effectiveness Team Composition (H3). The combination of PP and UBR approaches promises to deliver better results, summarizing the benefits of PP for the detection of source code defect and UBR for defect detection in written design documents. We investigate teams up to a size of 5 persons, regarding PP and UBR. The results describe an increasing effectiveness concerning an increasing team size. We also record, that additional PP participants improve defect detection in source code documents and additional participants of UBR inspectors support defect detection in design documents. Therefore, both assumptions agree to the results.

Our expectations, that additional PP teams improve defect detection in source code (H3.1) were confirmed by the observed results. Additionally, the expectations, that additional UBR inspectors improve defect detection capability regarding design specifications (H3.2), were also confirmed by the observed results. These observations might be used in decision support processes for combining analytical and constructive QA approaches in a given project context to achieve a certain level of product quality, e.g., building a QA team to find 70% of important defects.

6 Conclusion and Further Work

Software product improvement is an important issue in software development. Defect prevention, defect detection, and defect removal should be established as early as possible during development. Software inspection is a defect detection approach applicable to written text documents early in the development process. The main task of inspection is defect detection, in best practice applying reading techniques, e.g., UBR based on expert-prioritized use cases. As inspections do not require executable code, they may be performed in early software development phases.

Pair programming (PP) is a method, applicable to agile software processes and focuses on implementation of code fragments, involving two persons. While one person implements the software code, the other performs continuous reviews of code documents and specification documents. Therefore, PP performs defect detection as a by-product of software implementation, also rather early in low-level development. Note that code implementation is still the main focus of PP.

The results of our initial empirical study showed that PP is a promising method for both implementation and defect detection regarding the seeded defects in our study context. However, PP requires higher effort because of the involvement of two persons and the focus on software construction. Nevertheless, PP performed better than best-practice inspection concerning defect detection in code documents, which may partly be attributed to tool support for some defect detection tasks. UBR as a paper-based defect detection approach performed better for design documents. Regarding design documents, we do not recognize any significant difference between PP and UBR. Combinations of both approaches lead to overall best results.

Further work is necessary to validate the results of this study, including more participants to achieve a higher level of external validity. The replication and extension of the initial study enabled a closer look on agile software development methods, like pair programming, including quality assurance approaches. Further work is necessary to investigate the impact of inspector capability on defect detection effectiveness and on defect profiles for typical target contexts in practice.

References

[1] Basili V., Caldiera G., Lanubile F., Shull F.: "Studies on Reading Techniques", 21st Annual Software Engineering Workshop, NASA/Goddard Software Engineering Laboratory Series, SEL-96-002, pp 59-65, College Park, Maryland, 1997.

[2] Biffl S., Aurum A., Boehm B.: "Value-Based Software Engineering", Springer, 2005.

[3] Biffl S.: "Software Inspection Techniques to support Project and Quality Management", Shaker Verlag, 2001.

[4] Biffl S., Halling M.: "Investigating the Defect Detection Effectiveness and Cost Benefit of Nominal Inspection Teams", IEEE Transactions on Software Engineering 29 (5), pp.385-397, 2003.

[5] Biffl S., Gutjahr W.: "Influence of Team Size and Defect Detection Methods on Inspection Effectiveness", Proc. of IEEE Int. Software Metrics Symposium, London, 2001.

[6] Biffl S., Winkler D., Thelin T., Höst M., Russo B., Succi G., “Investigating the Effect of V&V and Modern Construction Techniques on Improving Software Quality”, Poster presented at ISERN 2004, Los Angeles.

[7] Cockburn A., Williams L.: “The Costs and Benefits of Pair Programming in Extreme Programming Examined”, Addison Wesley, 2001.
[8] Cockburn A.: “Agile Software Development”, Addison Wesley, 2002.
[9] Fagan M.: “Design and Code Inspections To Reduce Errors In Program Development”, IBM Systems J., vol. 15, no. 3, pp. 182-211, 1976.
[10] Gilb T., Graham D.: "Software Inspection", Addison-Wesley, 1993.
[11] Müller M.: "Are Reviews an Alternative to Pair Programming?", Conference on Empirical Assessment In Software Engineering (EASE), pages 3- 12, UK, 2003.
[12] Nawrocki J., Wojciechowski A.: "Experimental Evaluation of Pair Programming", Proceedings of the 12th European Software Control and Metrics Conference, pages 269-276, April 2001.
[13] Padberg F., Müller M.: "Analyzing the Cost and Benefit of Pair Programming", International Symposium on Software Metrics METRICS 9, 2003.
[14] Parrish A., Smith R., Hale D., Hale J.: "A field study of developer pairs: Productivity impacts and implications", IEEE Software, 21(2), Pages 76-79, 2004.
[15] Thelin T, Andersson C., Runeson P., Dzamashvili-Fogelström N.: „A Replicated Experiment of Usage-Based and Checklist-Based Reading“, 10th IEEE International Symposium on Software Metrics, pp. 246-256, 2004.
[16] Thelin T., Runeson, P., Regnell B.: “Usage-Based Reading - An Experiment to Guide Reviewers with Use Cases,” Information and Software Technology, vol. 43, no. 15, pp. 925-938, 2001.
[17] Thelin T., Runeson, P., Wohlin, C.: “An Experimental Comparison of Usage-Based and Checklist-Based Reading, IEEE Trans on Software Engineering, 29(8), pp. 687-704, 2003.
[18] Westland J. C.: “The cost of errors in software development: evidence from industry”, Journal of Systems and Software 62, 1-9., 2002.
[19] Williams L., Kessler R., Cunningham W., Jeffies R.: "Strengthening the Case for Pair Programming", IEEE Software 17(4):19-25, 2000.
[20] Williams L. Kessler, R.: “All I really need to know about pair programming I learned in Kindergarten”, Communication of the ACM, Volume 43, Issue 5 (May 2000), ACM Press, New York, pp. 108-114, 1999.
[21] Williams L., McDowell C., Nagappan N., Fernald J., Werner L.: “Building Pair Programming Knowledge through a Family of Experiments”, IEEE, Proceeding of the 2003 International Symposium on Empirical Software Engineering, ISESE, 2003.
[22] Winkler D., Biffl S., Thurnher B.: “Investigating the Impact of Active Guidance on Design Inspection”, Proc. of Profes 05, 2005.
[23] Winkler D., Biffl S., Riedl B.: „Improvement of Design Specifications with Inspection and Testing”, Proc. Of Euromicro 05, 2005.