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