The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design — An Experiment

Lech Madeyski

Institute of Applied Informatics, Wroclaw University of Technology,
Wyb.Wyspianskiego 27, 50370 Wroclaw, Poland
Lech.Madeyski@pwr.wroc.pl
http://madeyski.e-informatyka.pl/

Abstract. Background: Test-driven development (TDD) and pair programming are software development practices popularized by eXtreme Programming methodology. The aim of the practices is to improve software quality.

Objective: Provide an empirical evidence of the impact of both practices on package dependencies playing a role of package level design quality indicators.

Method: An experiment with a hundred and eighty eight MSc students from Wroclaw University of Technology, who developed finance-accounting system in different ways (CS — classic solo, TS — TDD solo, CP — classic pairs, TP — TDD pairs).

Results: It appeared that package level design quality indicators (namely package dependencies in an object-oriented design) were not significantly affected by development method.

Limitations: Generalization of the results is limited due to the fact that MSc students participated in the study.

Conclusions: Previous research revealed that using test-driven development instead of classic (test-last) testing approach had statistically significant positive impact on some class level software quality indicators (namely CBO and RFC metrics) in case of solo programmers as well as pairs. Combined results suggest that the positive impact of test-driven development on software quality may be limited to class level.

1 Introduction

Test-driven development (TDD) [1] and pair programming (PP) [2] have recently gained a lot of attention as the key software development practices of eXtreme Programming (XP) methodology [3]. The main idea of test-driven development is that programmers write tests before production code. Pair programming is software development practice where two programmers work together, collaborating on the same development tasks. The basic aim of both practices, described

J. Münch and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034, pp. 278–289, 2006.

in section 3.5, is to improve software quality. The question is whether both practices (used separately or together) really improve software quality.

Researchers and practitioners have reported numerous, often anecdotal and favourable studies of XP practices and methodology. Empirical studies on pair programming often concern productivity [4, 5, 6, 7, 8]. A few studies have focused on pair programming or test-driven development as practices to remove defects [5, 6, 9, 10], influence external code quality (measured by the number of functional, blackbox test cases passed) [11, 12, 13] or reliability of programs (a fraction of the number of passed tests divided by the number of all tests) [14, 15, 16]. Janzen [17] has pointed out that there was no research on the broader efficacy of test-driven development, nor on its effects on internal design quality outside a small pilot study [18]. Recently, Madeyski [19] pointed out that using test-driven development instead of classic (test-last) development had significant positive impact on two Chidamber and Kemerer (CK) [20] class level software quality indicators — Response For a Class (RFC) and Coupling Between Object classes (CBO). Obtained results did not support similar, positive impact of pair programming practice [19]. Hulkko and Abrahamsson [21] also suggested that pair programming might not necessarily provide as extensive quality benefits as suggested in literature. The key findings from empirical studies concerning software quality are summarized below in table 1.

Table 1. Pair programming and test-driven development literature review

Study	Environment	Subjects	Key findings
PP studies:			
[5, 6]	Academic	41(14P/13S)	P had 15% less code defects than S
[15, 16]	Academic	37(10P/17S)	P did not produce more reliable code
[21]	Acad./Ind.	4x(4–6)	P did not provide extensive quality benefits
TDD studies:			
[14]	Academic	19(9CS/10TS)	T did not produce more reliable code
[11, 12]	Industrial	24(6CP/6TP)	TP products passed 18% more tests than CP
[9, 10]	Industrial	13(5CS/9TS)	Minimal/no difference in *LOC* per person-month T reduced defect rate by 40–50%
[18]	Academic	8(1Cx4/1Tx4)	No meaningful differences in package dependencies between T and C project
Combined study:			
[13, 19]	Academic	188 (28CS/28TS/ 31CP/35TP)	TS passed significantly less acc. tests than CS TP passed significantly less acc. tests than CP No difference between CS and CP as well as TS and TP in *NATP* (Number of Acc.Tests Passed) T had significant positive impact on *RFC* and *CBO* CK metrics in case of S and P

Abbreviations: S(Solo programmers), P(Pairs), x4(groups of four), T(TDD), C(Classic)

In spite of a wide range of empirical studies there is still limited evidence concerning the impact of pair programming and test-driven development on quality of an object-oriented design in terms of dependencies between packages