Preliminary Analysis of the Effects of Pair Programming and Test-Driven Development on the External Code Quality

Lech Madeyski

Wroclaw University of Technology, Institute of Applied Informatics,
Wyb.Wyspianskiego 27, 50-370 Wroclaw, POLAND
Lech.Madeyski@pwr.wroc.pl,
http://madeyski.e-informatyka.pl

Abstract. Test-driven development (TDD) and pair programming (PP) are the key practices of eXtreme Programming methodology that have caught the attention of software engineers and researchers worldwide. One of the aims of the large experiment performed at Wroclaw University of Technology was to investigate the difference between test-driven development and the traditional, test-last development as well as pair programming and solo programming with respect to the external code quality. It appeared that the external code quality was lower when test-driven development was used instead of the classic, test-last software development approach in case of solo programmers ($p = 0.028$) and pairs ($p = 0.013$). There was no difference in the external code quality when pair programming was used instead of solo programming.

1 Introduction

Test-driven development (TDD) [1] and pair programming (PP) [15] have gained a lot of attention recently as the key software development practices of eXtreme Programming (XP) methodology [2].

Researchers and practitioners have reported numerous, often anecdotal and favourable studies of XP practices and XP methodology. Only some of them concern the external code quality (measured by the number of functional, black-box test cases passed) [3], [4] or reliability of programs (the fraction of the number of passed tests divided by the number of all tests) [8], [9], [7]. Systematic review of empirical studies concerning, for example, PP or TDD productivity is out of the scope of this paper but some of these studies are also included in table 1.

Key findings from empirical studies:

FP1 — Prediction that it takes less time on average to solve the problem by pairs than by individuals working alone was not statistically supported, however the average time for completion was more than 12 minutes (41%) longer for individuals than for pairs [11].

FP2 — Pairs completed their assignments 40–50% faster than individuals [13], [14].

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

STUDY	ENVIRONMENT	SUBJECTS	FINDINGS
Nosek [11]	Industrial	15(5Pairs/5Solo)	FP1
Williams [13][14]	Academic	41(14Pairs/13Solo)	FP2
Nawrocki [10]	Academic	21(5Pairs/5+6Solo)	FP3
Müller [8][9]	Academic	37(10Pairs/17Solo)	FP4
Müller [7]	Academic	19(9Classic/10TDD)	FT1
George [3][4]	Industrial	24(6Classic/6TDD) in 3 trials	FT2
Williams [16][6]	Industrial	13(5Classic/9TDD)	FT3

FP3 — Almost no difference in development time between XP-like pairs and solo programmers [10]. PP appeared less efficient than it was reported in [11], [13].

FP4 — A pair of developers does not produce more reliable code than a single developer whose code was reviewed. Although pairs of developers tend to cost more than single developers equipped with reviews, the increased cost is too small to be seen in practice [8], [9].

FT1 — TDD does not accelerate the implementation (working time in minutes) and the resulting programs are not more reliable, but TDD seems to support better program understanding [7].

FT2 — TDD developers produced higher quality code, which passed 18% more functional black box test cases. However, TDD developer pairs took 16% more time for development [3], [4].

FT3 — The productivity (LOC per person-month) of the team was not impacted by the additional focus on producing automated test cases [16] or the impact was minimal [6].

Results of the existing empirical work on PP and TDD are contradictory. This may be explained by the differences in the context in which the studies were conducted. Therefore the context of our experiment is expressed in details to support the development of cause-effect theories and to enable meta-analysis.

In 2004 a large experiment was conducted at Wroclaw University of Technology to study the impact of TDD and PP practices on different aspects of the software development process. This paper examines the impact of TDD and PP on the external code quality.

2 Experiment definition

The following definition determines the foundation for the experiment:
Object of study. The object of study is the software development process.
Purpose. The purpose is to evaluate the impact of TDD and PP practices on the software development process.
Quality focus. The quality focus is the external code quality.

Perspective. The perspective is from the researcher's point of view.
Context. The experiment is run using MSc students as subjects and finance-accounting system as an object.

Summary: Analyze *the software development process* for the purpose of *evaluation of the TDD and PP practices impact on the software development process* with respect to *external code quality* from the point of view of *the researcher* in the context of *finance-accounting system development by MSc students.*

3 Experiment Planning

The planning phase of the experiment can be divided into seven steps: context selection, hypotheses formulation, variables selection, selection of subjects, experiment design, instrumentation and validity evaluation.

3.1 Context Selection

The context of the experiment was the Programming in Java (PIJ) course, and hence the experiment was run off-line. Java was the programming language, Eclipse was the IDE (Integrated Development Environment). All subjects had experience at least in C and C++ programming (using object-oriented approach). The course consisted of seven lectures and fifteen laboratory sessions. The course introduced Java programming language using TDD and PP as the key XP practices. The subjects' skills were evaluated during the first seven laboratory sessions. The experiment took place during the last eight laboratory sessions (90 minutes per each). The problem (development of the finance-accounting system) was close to the real problem (not toy size). The requirements specification consisted of 27 user stories. In total 188 students were involved in the experiment. The subjects participating in the study were mainly second and third-year (and a few fourth and fifth-year) computer science master's students at Wroclaw University of Technology. A few people were involved in the experiment planning, operation and analysis.

3.2 Quantifiable Hypotheses Formulation

The crucial aspect of the experiment is to know and formally state what we intend to evaluate in the experiment. This leads us to the formulation of the following quantifiable hypothesis to be tested:

- $H_{0\ CS/TS/CP/TP}$ — There is no difference in the external code quality between the software development teams using any combination of classic (test last) / TDD (test first) development and solo / pair programming approach (CS, TS, CP and TP are used to denote approaches).
- $H_{A\ CS/TS/CP/TP}$ — There is a difference in the external code quality between the software development teams using any combination of classic (test last) / TDD (test first) development and solo / pair programming approach (CS, TS, CP and TP).

- $H_{0\ CS/TS}$ — There is no difference in the external code quality between solo programmers using classic and TDD testing approach (CS, TS).
- $H_{A\ CS/TS}$ — There is a difference in the external code quality between solo programmers using classic and TDD testing approach (CS, TS).
- $H_{0\ CP/TP}$ — There is no difference in the external code quality between pairs using classic and TDD testing approach (CP, TP).
- $H_{A\ CP/TP}$ — There is a difference in the external code quality between pairs using classic and TDD testing approach (CP, TP).
- $H_{0\ CS/CP}$ — There is no difference in the external code quality between solo programmers and pairs when classic testing approach is used (CS, CP).
- $H_{A\ CS/CP}$ — There is a difference in the external code quality between solo programmers and pairs when classic testing approach is used (CS, CP).
- $H_{0\ TS/TP}$ — There is no difference in the external code quality between solo and pairs when TDD testing approach is used (TS, TP).
- $H_{A\ TS/TP}$ — There is a difference in the external code quality between solo and pairs when TDD testing approach is used (TS, TP).

If we reject the null hypothesis $H_{0\ CS/TS/CP/TP}$ we can try to investigate more specific hypotheses.

NATP (Number of Acceptance Tests Passed) was used as a measure of external code quality. The same measure was used by George and Williams [3], [4]. In contrast to some productivity measures, e.g. source lines of code (SLOC) per person-month, NATP takes into account functionality and quality of software development products. SLOC per unit of effort tend to emphasize longer rather than efficient or high-quality programs. Refactoring effort may even results in negative productivity measured by SLOC.

3.3 Variables Selection

The independent variable is the software development method used. The experiment groups used CS, TS, CP or TP approach.

The dependent (or response) variable is characteristic of the software products on which the factors under examination are expected to have an effect. In our case the dependent variable is NATP.

3.4 Selection of Subjects

The subjects are chosen based on convenience — the subjects are students taking the PIJ course. Prior to the experiment, the students filled in a questionnaire. The aim of the questionnaire was to get a picture of the students' background, see table 2. The ability to generalize from this context is further elaborated when discussing threats to the experiment. The use of the course as an experimental context provides other researchers opportunities to replicate the experiment.

Table 2. The context of the experiment

CONTEXT FACTOR	*ALL*	*CS*	*TS*	*CP*	*TP*
Number of:					
- MSc students:	188	28	28	62	70
— 2nd year students	108	13	16	40	39
— 3rd year students	68	12	11	18	27
— 4th year students	10	3	0	3	4
— 5th year students	2	0	1	1	0
- Students with not only academic but also industry experience	33	4	6	8	15
Mean value of:					
- Programming experience [in years]	3.8	4.1	3.7	3.6	3.9
- Java programming experience [in months]	3.9	7.1	2.8	3.4	3.5
- Programming experience in another OO language than Java [in months]	20.5	21.8	20.9	19.2	21.1

3.5 Design of the Experiment

The design is one factor (the software development method) with four treatments (alternatives):

- Solo programming using classic testing approach — tests after implementation (CS).
- Solo programming using test-driven development (TS).
- Pair programming using classic testing approach — tests after implementation (CP).
- Pair programming using test-driven development (TP).

The students were divided into groups based on their skills (measured by graders on an ordinal scale) and then randomized within TDD or classic testing approach groups. However the assignment to pair programming teams took into account the people preferences (as it seemed to be more natural and close to the real world practice). Students who did not complete the experiment were removed from the analysis. The design resulted in an unbalanced design, with 28 solo programmers and 31 pairs using classic testing approach, 28 solo programmers and 35 pairs using TDD practice.

3.6 Instrumentation

The instrumentation of the experiment consisted of requirements specification (user stories), pre-test and post-test questionnaires, Eclipse project framework, detailed description of software development approaches (CS, TS, CP, TP), duties of subjects, instructions how to use the experiment infrastructure (e.g. CVS Version Management System) and examples (e.g. sample applications developed using TDD approach).

3.7 Validity Evaluation

The fundamental question concerning results from an experiment is how valid the results are. When conducting an experiment, there is always a set of threats to the validity of the results. Shadish, Cook and Campbell [12] defined four types of threats: *statistical conclusion*, *internal*, *construct* and *external validity*.

Threats to the *statistical conclusion* validity are concerned with issues that effect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment. Threats to the *statistical conclusion* validity are considered to be under control. Robust statistical techniques, tools (e.g. Statistica) and large sample sizes to increase statistical power are used. Measures and treatment implementation are considered reliable. However the risk in the treatment implementation is that the experiment was spread across laboratory sessions. To avoid the risk access to the CVS repository was restricted to the specific laboratory sessions (access hours and IP addresses). The validity of the experiment is highly dependent on the reliability of the measures. The measure used in the experiment is considered reliable because it can be repeated with the same outcome. Heterogeneity of the subjects is blocked, based on their grades.

Threats to *internal* validity are influences that can affect the independent variable with respect to causality, without the researcher's knowledge. Concerning the *internal* validity, the risk of rivalry between groups must be considered. The group using the traditional method may do their very best to show that the old method is competitive. On the other hand subjects receiving less desirable treatments may not perform as well as they generally does. However, the subjects were informed that the goal of the experiment was to measure different approaches to software development not the subjects skills. Possible diffusion or imitation of treatments were under control of the graders.

Construct validity concerns generalizing the results of the experiment to the concepts behind the experiment. Threats to the *construct* validity are not considered very harmful. Inadequate explication of constructs does not seem to be a threat as the constructs were defined, before they were translated into measures or treatments — it was clearly stated what having higher external code quality means. The mono-operation bias is a threat as the experiment was conducted on a single software development project, however the size of the project was not toy size. Using a single type of measures is a mono-method bias threat, however the measure used in the experiment was rather objective.

Threats to *external* validity are conditions that limit our ability to generalize the results of our experiment to industrial practice. The largest threat is that students (who had short experience in PP and TDD) were used as subjects. Especially TDD possesses a fairly steep learning curve that must be surmounted before the benefits begin to accrue. However, study by Höst [5] suggest that students may provide an adequate model of the professional population. Furthermore, some of the subjects had also industry experience, see table 2. In summary, the threats are not considered large in this experiment.

4 Experiment Operation

The experiment was run at Wroclaw University of Technology in 2004 during eight laboratory sessions. The data was primarily collected by automated experiment infrastructure. Additionally the subjects filled in pre-test and post-test questionnaires, primarily to evaluate their experience and preferences. The package for the experiment was prepared in advance and is described in section 3.6.

5 Analysis of the Experiment

The experiment data are analyzed with descriptive analysis and statistical tests.

5.1 Descriptive Statistics

A good way to display the results of the experiment is by using a box and whisker diagram or box plot shown in figure 1. The box represents the range within which 50% of the data fall. The point within the box is the median. The 'I' shape shows us the limits within which all of the data fall. The first impression is that classic approaches performed better than TDD approaches.

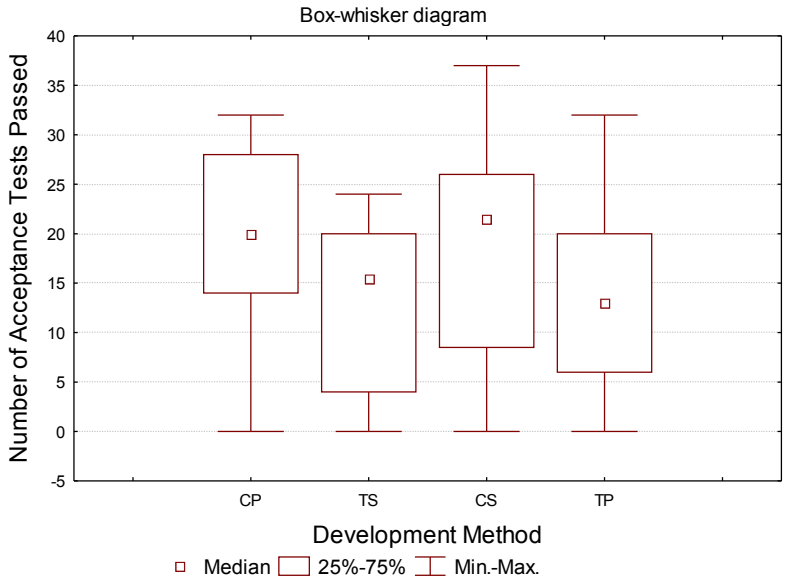

Fig. 1. Box-whisker plot for the number of acceptance tests passed in different development methods

5.2 Hypotheses Testing

Experimental data are analysed using models that relate the dependent variable and factor under consideration. The use of these models involves making assumptions about the data that need to be validated. Therefore we run some exploratory analysis on the collected data to check whether they follow the assumptions of the parametric tests:

- Interval or ratio scale — the collected data must be measured at an interval or ratio level (since parametric tests work on the arithmetic mean).
- Homogeneity of variance — roughly the same variances between groups or treatments (if we use different subjects).
- Normal distribution — the collected data come from a population that has a normal distribution.

The first assumption is met. The second assumption of homogeneity of variance is tested using Levene's test, see table 3. The Levene test is non-significant ($p > 0.05$) so we accept the null hypothesis that the difference between the variances is roughly zero — the variances are more or less equal.

Table 3. Test of Homogeneity of Variances

Levene Statistic	Significance
1.199	0.313

Having checked the two assumptions we have to test the third one — normality assumption using the Kolmogorov-Smirnov test as well as the Shapiro-Wilk test.

Table 4. Tests of Normality

Approach	Kolmogorov-Smirnov[1]			Shapiro-Wilk		
	Statistic	df[2]	Significance	Statistic	df[2]	Significance
CS	0.182	28	0.018	0.931	28	0.066
TS	0.157	28	0.075	0.893	28	0.008
CP	0.116	31	0.200[3]	0.912	31	0.014
TP	0.111	35	0.200[3]	0.960	35	0.222

[1] Lilliefors Significance Correction.
[2] Degrees of freedom.
[3] This is a lower bound of the true significance.

We find that the data are not normally distributed in case of CS approach (according to the Kolmogorov-Smirnov statistic), TS and CP (according to the Shaprio-Wilk statistic), see table 4. This finding alerts us to the fact that a nonparametric test should be used.

The hypothesis regarding the difference in external code quality between the software development teams using CS, TS, CP and TP approach is evaluated using the Kruskal-Wallis one way analysis of variance by ranks. The Kruskal-Wallis test is a non-parametric alternative to the parametric ANOVA and can always be used instead of the ANOVA if it is not sure that the assumptions of ANOVA are met. The Kruskal-Wallis test is used for testing differences between the four experimental groups (CS, TS, CP, TP) when different subjects are used in each group.

The Kruskal-Wallis test analyses the ranked data. Table 5 shows a summary of these ranked data and tells us the mean rank in each treatment. The test statistic is a function of these ranks.

Table 5. Ranks

Treatment	N	Mean Rank
CS	28	69.46
TS	28	50.79
CP	31	74.58
TP	35	52.11
Total	122	

Table 6 shows this test statistic and its associated degrees of freedom (in this case we had 4 groups so $4 - 1$ degrees of freedom), and the significance.

Table 6. Kruskal-Wallis Test Statistics [grouping variable: Approach(CS,TS,CP,TP)]

	NATP
Chi-Square	10.714
df	3
Asymp. Significance	0.013

We can conclude that the software development approach used by the subjects significantly affected the external code quality (measured by NATP). This test tells us only that a difference exists, however it does not tell us exactly where the difference lies.

One way to see which groups differ is to look at the box plot diagram of the groups (see figure 1). The first thing to note is that classic approaches (CS, CP) achieved better results (higher numbers of acceptance tests passed) than TDD approaches (TS, TP). However, this conclusion is subjective. We need to perform planned comparisons (contrasts) or multiple independent comparisons (Mann-Whitney tests) for specific hypotheses from section 3.2. It is justified as we identified the comparisons (specific hypotheses) as valid at the design stage of our investigation. The planned comparisons, instead of comparing everything with everything else, have the advantage that we can conduct fewer tests, and therefore, we don't have to be quite strict to control the type I error rate (the probability that a true null hypothesis is incorrectly rejected).

Tables 7, 8, 9 and 10 show the test statistics of Mann-Whitney tests on the four focused comparisons. When CP and TP approaches are compared the observed significance value is 0.013, see table 8. When CS and TS approaches are compared the observed significance value is 0.028, see table 7. The planned contrasts also suggest that using TDD instead of classic testing approach decreases the external code quality in case of pairs as well as solo programmers ($p < 0.05$).

Table 7. Mann-Whitney Test Statistics (CS vs. TS)

	NATP
Mann-Whitney U	258.000
Wilcoxon W	664.000
Z	−2.198
Asymp. Significance (2-tailed)	0.028

Table 8. Mann-Whitney Test Statistics (CP vs. TP)

	NATP
Mann-Whitney U	348.500
Wilcoxon W	978.500
Z	−2.495
Asymp. Significance (2-tailed)	0.013

Table 9. Mann-Whitney Test Statistics (CS vs. CP)

	NATP
Mann-Whitney U	393.500
Wilcoxon W	799.500
Z	−0.615
Asymp. Significance (2-tailed)	0.538

Table 10. Mann-Whitney Test Statistics (TS vs. TP)

	NATP
Mann-Whitney U	485.000
Wilcoxon W	1115.000
Z	−0.069
Asymp. Significance (2-tailed)	0.945

6 Summary and Conclusions

The external code quality (measured by a number of acceptance tests passed) was significantly affected by the software development approach (the Kruskal-Wallis test statistics: $H(3) = 10.71, p < 0.05$ where H is the test statistic function with 3 degrees of freedom and p is the significance). This means that there is a difference in the external code quality between the software development teams using CS, TS, CP and TP approach. Mann-Whitney tests were used to follow-up this finding. It appeared that the external code quality was lower when TDD was used instead of the classic, test-last software development approach in case of solo programmers (Mann-Whitney CS vs. TS test significance value $p = 0.028$) and pairs (Mann-Whitney CP vs. TP test significance value $p = 0.013$). There was no difference in the external code quality when pair programming was used instead of solo programming (Mann-Whitney CS vs. CP test significance value $p = 0.538$, Mann-Whitney TS vs. TP test significance value $p = 0.945$). The validity of the results must be considered within the context of the limitations discussed in the validity evaluation section.

Future research is needed to evaluate other properties than the external code quality as well as to evaluate PP and TDD in other contexts (e.g. in industry).

7 Acknowledgments

The author would like to thank the students for participating in the investigation, the graders for their help and the members of the e-Informatyka team (Wojciech Gdela, Tomasz Poradowski, Jacek Owocki, Grzegorz Makosa, Mariusz Sadal and

Michał Stochmiałek) for support and preparation of the infrastructure for the experiment to automate measurements which appeared extremely helpful.

The author also wants to thank prof. Zbigniew Huzar and dr Malgorzata Bogdan for helpful suggestions.

References

1. Beck, K.: Test Driven Development: By Example. Addison-Wesley (2002)
2. Beck, K.: Extreme Programming Explained: Embrace Change. 2nd edn. Addison-Wesley (2004)
3. George, B., Williams, L.A.: An initial investigation of test driven development in industry. In: Proceedings of the 2003 ACM Symposium on Applied Computing (SAC), ACM (2003) 1135–1139
4. George, B., Williams, L.A.: A structured experiment of test-driven development. Information & Software Technology **46** (2004) 337–342
5. Höst, M., Regnell, B., Wohlin, C.: Using students as subjects — a comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering **5** (2000) 201–214
6. Maximilien, E.M., Williams, L.A.: Assessing Test-Driven Development at IBM. In: Proceedings of the 25th International Conference on Software Engineering (ICSE), IEEE Computer Society (2003) 564–569
7. Müller, M.M., Hagner, O.: Experiment about test-first programming. IEE Proceedings - Software **149** (2002) 131–136
8. Müller, M.M.: Are reviews an alternative to pair programming? In: Proceedings of the Conference on Empirical Assessment In Software Engineering (EASE). (2003)
9. Müller, M.M.: Are reviews an alternative to pair programming? Empirical Software Engineering **9** (2004) 335–351
10. Nawrocki, J.R., Wojciechowski, A.: Experimental evaluation of pair programming. In: Proceedings of the European Software Control and Metrics Conference (ESCOM) (2001) 269–276
11. Nosek, J.T.: The case for collaborative programming. Communications of the ACM **41** (1998) 105–108
12. Shadish, W.R., Cook, T.D., Campbell, D.T.: Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflin (2002)
13. Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R.: Strengthening the case for pair programming. IEEE Software **17** (2000) 19–25
14. Williams, L.: The Collaborative Software Process. PhD thesis, University of Utah (2000)
15. Williams, L., Kessler, R.: Pair Programming Illuminated. Addison-Wesley (2002)
16. Williams, L.A., Maximilien, E.M., Vouk, M.: Test-driven development as a defect-reduction practice. In: Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE 2003), IEEE Computer Society (2003) 34–48