Is External Code Quality Correlated with Programming Experience or Feelgood Factor?*

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. This paper is inspired by an article by Müller and Padberg who study the feelgood factor and programming experience, as candidate drivers for the pair programming performance. We not only reveal a possible threat to validity of empirical results presented by Müller and Padberg but also perform an independent research. Our objective is to provide empirical evidence whether external code quality is correlated with the feelgood factor, or with programming experience. Our empirical study is based on a controlled experiment with MSc students. It appeared that the external code quality is correlated with the feelgood factor, and programming experience, in the case of pairs using a classic (test-last) testing approach. The generalization of the results is limited due to the fact that MSc students participated in the study. The research revealed that both the feelgood factor and programming experience may be the external code quality drivers.

1 Introduction

Pair programming [1] has recently gained a lot of attention, as key software development practice of eXtreme Programming (XP) methodology [2]. The main idea of pair programming software development practice is that two programmers work together, collaborating on the same development tasks. The basic aim of pair programming, described in section 3.2, is to 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 [3, 4, 5, 6, 7]. A few studies have focused on pair programming, or test-driven development, as practices to remove defects [4, 5, 8, 9], to influence the external code quality (measured by the number of functional, blackbox test cases passed) [10, 11, 12] or reliability of programs (a fraction of the number of passed tests divided by the number of all tests) [13, 14, 15] and other quality benefits [16].

In spite of a wide range of studies, there is still limited evidence concerning the role of the feelgood factor (how comfortably the developers feel in a pair

* This work has been financially supported by the Ministry of Education and Science as a research grant 3 T11C 061 30 (years 2006-2007).

P. Abrahamsson, M. Marchesi, and G. Succi (Eds.): XP 2006, LNCS 4044, pp. 65–74, 2006.

session [17]) and the programming experience in pair programming. The aim of this paper is to fill this gap. So far, the results obtained by Müller and Padberg [17] indicate that the pair performance is uncorrelated with the programming experience whereas the feelgood factor is a candidate driver for the performance of a pair.

The results presented by Müller and Padberg were obtained by applying a special scheme for pairing the subjects. The most skilled subject had to pair off with the lowest skilled subject, the second best skilled subject with the second lowest skilled subject, and so on. The aim was to balance the skill level across the pairs but, this special scheme for pairing the subjects might have hidden a possible correlation of pair performance with the programming experience, as the latter was averaged across pairs. In the Müller and Padberg study, the performance of a pair was measured by the implementation time [17]. In our study the implementation time is constant (eight laboratory sessions) and the dependent variable is the external code quality, measured by the number of acceptance tests passed ($NATP$), as suggested by George and Williams [10, 11] and later used by Madeyski [12]. Therefore, the research question is whether the external code quality is correlated with the pair feelgood factor, or programming experience.

2 Problem Statement

The data for this study comes from a controlled experiment performed at Wroclaw University of Technology. The purpose of the experiment was to investigate the impact of test-driven development and pair programming practices on software development products [12].

The following definition determines a foundation for our study [18]:

Object of study. The objects of study are software development products — developed code.
Purpose. The purpose is to find whether the quality of software development products is correlated with the programming experience, or the feelgood factor of pair programming.
Quality focus. The quality focus is the external code quality (measured by $NATP$).
Perspective. The perspective is from the researcher's point of view.
Context. The study is run using MSc students as subjects and the finance-accounting system as an object.

Summary: Analyse *the software development products* for the purpose of *finding correlation between quality of software development products and the feelgood factor, or programming experience* with respect to the *external code quality*, from *the researcher's* point of view, in the context of the *finance-accounting system development by MSc students.*

3 Study Description

3.1 Context Selection

The context of the experiment was the Programming in Java (PIJ) course, and hence the experiment was run off-line (not industrial software development) [18]. Java was the programming language, and Eclipse 3.0 was the Integrated Development Environment (IDE). All subjects had prior experience in at least C and C++ programming (using object-oriented approach). The PIJ course consisted of seven 90 minute lectures and fifteen laboratory 90 minute sessions. The course introduced Java programming language, using test-driven development and pair programming as key XP practices. The subjects' practical skills in programming in Java, using pair programming, and test-driven development were evaluated during the first seven laboratory sessions. The experiment took place during the last eight laboratory sessions. The problem addressed the development of the finance-accounting system. The requirements specification consisted of 27 user stories. The subjects participating in the study were mainly second and third-year (and few fourth and fifth-year) computer science MSc students. MSc programme of Wroclaw University of Technology is a 5-year programme after high school. In total, 188 students were involved in the experiment, but only 132 students were working in pairs, see table 1.

3.2 Variables and Subjects Selection

The variables considered in this study are:

- The external code quality was measured by the number of acceptance tests passed ($NATP$). This measure was proposed by George and Williams [10], [11]. The number of acceptance tests passed was collected automatically by our measurement infrastructure. 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.
- The pair feelgood factor (PFF) was measured by the mean value of the individual feelgood factors, collected by means of a post-test questionnaire. The post-test questionnaire asked how comfortable the subject felt during the pair programming session. An even number of alternatives (0–bad, 1–sufficiently, 2–good, 3–very good) was chosen, because it forces the subjects to get off the fence, and to prevent large numbers of neutral answers. The answer ranges on an ordinal scale and this metric is called the individual feelgood factor of a developer. Since our questionnaire did not ask the pairs to specify a joint feelgood factor, the mean of the individual assessments was taken as a substitute. The resulting metric is called the pair feelgood factor. This approach to calculate the pair feelgood factor was used by Müller and Padberg [15]. It may be questionable, because the individual feelgood factor is an ordinal value, but we used it for compatibility reasons.
- The mean programming experience (MPE) was measured by the mean value of the individual programming experience of each pair programmers,

collected by means of questionnaires. Not only industrial but also school (university) experience was included.

The subjects are chosen based on convenience — the subjects are students taking the PIJ course. Prior to the experiment, the students filled in a pre-test questionnaire. The aim of the questionnaire was to get a description of the students' background, see table 1. The ability to generalize from this context is further elaborated when discussing threats, see section 3.4.

In this study we analysed pairs using test-driven development practice (denoted as TP) and classic (test-last) testing approach (denoted as CP).

Table 1. The context of the study

Context factors	CP	TP
Number of MSc students:	62	70
– in the 2nd year	40	39
– in the 3rd year	18	27
– in the 4th year	3	4
– in the 5th year	1	0
– with industry experience	8	15
Median of individual feelgood factor (0–bad...3–very good)	3	3
Mean of programming experience (years)	3.61	3.86

Pair programming is a practice in which two programmers (called the driver and navigator) work together at one computer, collaborating on the same development tasks (e.g. design, test, code). The driver, is typing at the computer or writing down a design. The navigator observes the driver's work, reviews the code, proposes test cases and considers the implementations strategic implications [4, 19].

Test-driven development (TDD) is a practice based on specifying a piece of functionality, as a low level test before writing production code, on implementing the functionality, so that the test passes, and on refactoring (e.g. removing duplication) and iterating the process. The tests are run frequently while writing production code. In case of classic (test-last) development, the tests are specified after writing production code and less frequently [20].

The assignment of subjects to groups was performed first by stratifying the subjects with respect to their skill level, measured by graders, and then assigning them at random to test-driven development, or classic (test-last) testing approach teams. However, the assignment to pair programming teams took into account the people's preferences (as it seemed to be more natural and close to the real world agile software development practice). The students who did not complete the projects (did not check in the project prerequisites the final

version of their program, or did not fill in questionnaires) were not included in the analysis. The outcome was an unbalanced design, with 35 pairs using TDD practice and 31 pairs using classic (test-last) testing approach.

3.3 Materials

The materials prepared for the experiment consisted of requirements specification (user stories), pre-test and post-test questionnaires, Eclipse project framework, a detailed description of software development methods, and of duties of the subjects, instructions how to use the experiment infrastructure (e.g. CVS Version Management System), and examples (e.g. sample source code of applications developed using TDD approach and JUnit tests). The number of acceptance tests passed was collected using automated infrastructure developed by e-Informatyka team members of Wroclaw University of Technology.

3.4 Validity Evaluation

The fundamental question concerning the results of each study is how valid the results are. Shadish, Cook and Campbell [21] defined four types of threats: *statistical conclusion*, *internal*, *construct* and *external validity*.

The 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. The risk in the treatment implementation is that the study was spread across laboratory sessions. To avoid the risk, the access to the CVS repository was restricted to the specific laboratory sessions (access hours and IP addresses). The validity of the study is highly dependent on the reliability of the measures. The basic principle is that when you measure a phenomenon twice, the outcome should be the same. The number of acceptance tests passed is considered reliable because it can be repeated with the same outcomes.

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, the subjects receiving less desirable treatments may not perform so well as they generally do. However, the subjects were informed that the goal of the study was to measure different development methods, and not the subjects' skills. A possible diffusion or imitation of treatments were under control of the graders.

Threats to the *construct* validity are not considered very harmful. The mono-operation bias is a threat, as the study was conducted on a single software development project; however, the the project addressed a similar to real-life situation problem (the development of the finance-accounting system). Using a single type of measure would be a mono-method bias threat; however, measures used in the study were rather objective.

The largest threat to the *external* validity is that students (who had short experience in pair programming and test-driven development) were used as subjects. Kitchenham et al.[22] states that students are the next generation of

software professionals, so, they are relatively close to the population of interest. Replicated experiments by Porter and Votta [23] and Höst et al. [24] also suggest that students may provide an adequate model of professional population. However, it is too optimistic when we evaluate experience.

In summary, the threats are not regarded as being critical.

4 Operation

The experiment was run at Wroclaw University of Technology 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 get a description of their experience and preferences. The package for the experiment was prepared in advance and is described in section 3.3. A few people were involved in the experiment planning, operation and analysis.

5 Analysis

The data are analysed with scatterplot and Spearman's correlation coefficient. Before conducting any correlational analysis, it is essential to plot a scatterplot to look at the general trend of the data.

5.1 Discovering General Trend

A scatterplot tells us whether there seems to be a relationship between the variables, what kind of relationship it is, and whether any cases differ substantially from the general trend of the data. We use an overlay scatterplot, as we want to look at the role of both the pair feelgood factor and the programming experience on external code quality (but not the relationship between the pair feelgood factor and the programming experience).

Scatterplot has been used to plot the relationship between the pair feelgood factor and external code quality and between the programming experience and external code quality simultaneously, see figure 1. From figure 1 it seems that both the pair feelgood factor and programming experience are positively related to the external code quality, at least in the case of classic (test-last) development method used by pairs (CP). Spearman's correlations were used to follow up these findings.

5.2 Discovering Correlations

Table 2 shows Spearman's correlations and significances for two experimental groups (CP and TP).

In case of classic (test-last) testing approach the external code quality (measured by $NATP$) achieved by pairs is correlated with the pair feelgood factor ($p = .022$) and mean programming experience of programmers in pairs

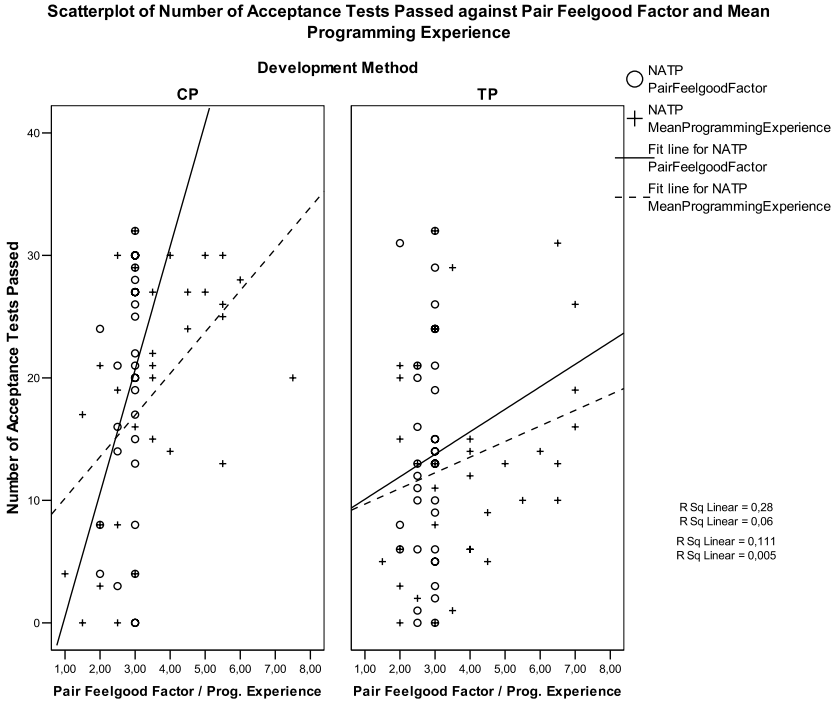

Fig. 1. Scatterplot of Number of Acceptance Tests Passed against Pair Feelgood Factor and Mean Programming Experience

Table 2. Nonparametric Correlations – Spearman's rho

		$NATP_{TP}$	$NATP_{CP}$
NATP	Correlation Coefficient	1.000	1.000
	N	35	31
Pair Feelgood Factor	Correlation Coefficient	0.121	0.364
	Sig.(1-tailed)	0.244	0.022
Mean Programming Experience [years]	Correlation Coefficient	0.222	0.512
	Sig.(1-tailed)	0.100	0.002

($p = .002$). The fact that a correlation exists is not sufficient to conclude that the feelgood factor, or programming experience, actually drives the external code quality in case of classic testing approach e.g. it is unclear whether a pair performs well because the feelgood factor is high, or, whether the developers feel comfortable because they have the impression that the number of acceptance tests passed is high.

In the case of pairs using test-driven development practice, the effect is smaller, and the results are not statistically significant ($p > .05$). A possible explanation is that the number of acceptance tests passed is significantly affected

by the software testing approach. It appeared that the number of acceptance tests passed was lower when test-driven development was used instead of the classic, test-last software development approach in case of solo programmers ($p = .028$) and pairs ($p = .013$) [12].

6 Summary and Conclusions

The previous research conducted by Müller and Padberg [17] revealed that pair performance may be uncorrelated with the programming experience, but correlated with the pair feelgood factor. A possible threat to validity of empirical results presented by Müller and Padberg is that they used a special scheme for pairing the subjects that averaged the programming experience.

The results obtained in our study suggest that both the pair feelgood factor and programming experience are correlated, in case of classic testing approach, with the number of acceptance tests passed, which is a measure of the external code quality, as suggested by George and Williams [10, 11]. Therefore, both the pair feelgood factor and programming experience may be external code quality drivers.

The existence of correlations should be considered as a basis for future research. From the correlation alone, one can not decide whether the number of acceptance tests passed is high because the pair feelgood factor or mean programming experience was high. To answer that question, further empirical studies are necessary. A further research (e.g. experiment with the pair feelgood factor in mind) is needed to establish evidence of the impact of the pair feelgood factor, and programming experience on the external code quality and to evaluate the impact of the pair feelgood factor and programming experience in other contexts (e.g. in industry).

The validity of the results must be considered within the context of the limitations discussed in the validity evaluation section.

Acknowledgments

The author expresses his gratitude to the students participating in the research, the graders and the members of the e-Informatyka team (Wojciech Gdela, Tomasz Poradowski, Jacek Owocki, Grzegorz Mąkosa, Mariusz Sadal and Michał Stochmiałek) for their help during preparations of the experiment infrastructure, and to anonymous reviewers for helpful suggestions.

References

1. Williams, L., Kessler, R.: Pair Programming Illuminated. Addison-Wesley (2002)
2. Beck, K.: Extreme Programming Explained: Embrace Change. 2nd edn. Addison-Wesley (2004)
3. Nosek, J.T.: The case for collaborative programming. Communications of the ACM **41**(3) (1998) 105–108

4. Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R.: Strengthening the case for pair programming. IEEE Software **17**(4) (2000) 19–25
5. Williams, L.: The Collaborative Software Process. PhD thesis, University of Utah (2000)
6. Nawrocki, J.R., Wojciechowski, A.: Experimental evaluation of pair programming. In: ESCOM '01: European Software Control and Metrics. (2001) 269–276
7. Nawrocki, J.R., Jasiński, M., Olek, L., Lange, B.: Pair Programming vs. Side-by-Side Programming. In Richardson, I., Abrahamsson, P., Messnarz, R., eds.: EuroSPI. Volume 3792 of Lecture Notes in Computer Science., Springer (2005) 28–38
8. Williams, L., Maximilien, E.M., Vouk, M.: Test-Driven Development as a Defect-Reduction Practice. In: ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering, Washington, DC, USA, IEEE Computer Society (2003) 34–48
9. Maximilien, E.M., Williams, L.A.: Assessing Test-Driven Development at IBM. In: ICSE '03: Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society (2003) 564–569
10. George, B., Williams, L.A.: An Initial Investigation of Test Driven Development in Industry. In: SAC '03: Proceedings of the 2003 ACM Symposium on Applied Computing, ACM (2003) 1135–1139
11. George, B., Williams, L.A.: A structured experiment of test-driven development. Information and Software Technology **46**(5) (2004) 337–342
12. Madeyski, L.: Preliminary Analysis of the Effects of Pair Programming and Test-Driven Development on the External Code Quality. In Zieliński, K., Szmuc, T., eds.: Software Engineering: Evolution and Emerging Technologies. Volume 130 of Frontiers in Artificial Intelligence and Applications. IOS Press (2005) 113–123
13. Müller, M.M., Hagner, O.: Experiment about test-first programming. IEE Proceedings - Software **149**(5) (2002) 131–136
14. Müller, M.M.: Are Reviews an Alternative to Pair Programming? In: EASE '03: Conference on Empirical Assessment In Software Engineering. (2003)
15. Müller, M.M.: Are Reviews an Alternative to Pair Programming? Empirical Software Engineering **9**(4) (2004) 335–351
16. Hulkko, H., Abrahamsson, P.: A Multiple Case Study on the Impact of Pair Programming on Product Quality. In: ICSE '05: Proceedings of the 27th International Conference on Software Engineering, New York, NY, USA, ACM Press (2005) 495–504
17. Müller, M.M., Padberg, F.: An empirical study about the feelgood factor in pair programming. In: METRICS '04: Proceedings of the Software Metrics, 10th International Symposium on (METRICS'04), Washington, DC, USA, IEEE Computer Society (2004) 151–158
18. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA (2000)
19. Williams, L.A., Kessler, R.R.: All I really need to know about pair programming I learned in kindergarten. Commun. ACM **43**(5) (2000) 108–114
20. Erdogmus, H., Morisio, M., Torchiano, M.: On the Effectiveness of the Test-First Approach to Programming. IEEE Transactions on Software Engineering **31**(3) (2005) 226–237
21. Shadish, W.R., Cook, T.D., Campbell, D.T.: Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflin (2002)

22. Kitchenham, B., Pfleeger, S.L., Pickard, L., Jones, P., Hoaglin, D.C., Emam, K.E., Rosenberg, J.: Preliminary Guidelines for Empirical Research in Software Engineering. IEEE Transactions on Software Engineering **28**(8) (2002) 721–734
23. Porter, A., Votta, L.: Comparing detection methods for software requirements inspections: A replication using professional subjects. Empirical Softw. Engg. **3**(4) (1998) 355–379
24. Höst, M., , Wohlin, C., Thelin, T.: Experimental context classification: incentives and experience of subjects. In: ICSE '05: Proceedings of the 27th International Conference on Software Engineering, New York, NY, USA, ACM Press (2005) 470–478