Empirical Validation of Test-Driven Pair Programming in Game Development

Shaochun Xu
Department of Computer Science
Algoma University College, Laurentian University,
xu@auc.ca

Vaclav Rajlich
Department of Computer Science
Wayne State University
rajlich@wayne.edu

Abstract

This paper investigates the effects of some extreme programming practices in game development by conducting a case study with 12 students who were assigned to implement a simple game application either as pairs or as individuals. The pairs used some XP practices, such as pair programming, test-driven and refactoring, while the individuals applied the traditional waterfall-like approach. The results of the case study showed that paired students completed their tasks faster and with higher quality than individuals. The programs written by pairs pass more test cases than those developed by individuals. Paired programmers also wrote cleaner code with higher cohesion by creating more reasonable number of methods. Therefore, some XP practices, such as pair programming, test-driven and refactoring could be used in game development.

1. Introduction

Traditional software development process usually fails to deliver products in time because it requires a thorough analysis of the requirements and a detailed design before implementation. Software that takes years or so to design and to implement may lose the market because of the fast change of the requirements, high competition of software market, and increasing complexities of software. This also happens in game development.

In order to solve those problems, agile software development processes, like Scrum, FDD, Extreme Programming (XP) have been recently proposed [2, 5, 20]. XP is a lightweight process and thus, suitable for small to medium sized projects [2]. It is becoming pervasive in the world of software development. XP includes 12 practices, among which pair programming, test-driven and refactoring are the most commonly used.

Pair programming refers to the techniques where two programmers work on a programming problem using only one computer [2]. Each programmer has a distinct role (i.e., driver and observer), in which one is writing the code and the other is helping and doing immediate code review. The test-driven technique means that the programmers write the test before writing the code and it forces the programmers to define the exact functionality of each method and the system will be automatically tested when it is developed. Refactoring is often used in association with test-driven technique, which transforms the source code to be more readable and more elegant [2].

Pair programming has been used in industry and reported with some promising results [18]. Test-driven and refactoring have not been used much, but their benefits have also been reported [10]. However, those XP practices have scarcely been applied in game development. A survey shows that only 5% of people in game industry have been using XP in some aspects [12]. Although a few game companies, such as Coyote Development [7] and Sammy Studios [1], are using pair programming and test-driven, no experiment is known to have been conducted to evaluate the results of such usage.

In order to investigate the effects of some XP practices in game development, we designed a case study where the participants worked either in pairs or individually and implemented a simple game application. Paired programmers also used test-driven and refactoring practices, and individuals used the traditional waterfall approach. That allowed us to make a comparison between these two approaches. We also conducted a survey on those techniques.

Section 2 reviews the three XP practices in game development. Our case study design is described in section 3. The results of the case study and the survey are discussed in Section 4 and the conclusions and the future work are presented in Section 5.

2. Pair Programming, Test-Driven, and Refactoring in Game Development

In this section, we discuss the related work with pair programming, test-driven and refactoring in game development.

2.1. Pair Programming

Pair programming has been widely known with its fast development cycle and high quality code [6, 23].

Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06)
0-7695-2613-6/06 $20.00 © 2006 IEEE

Nosek [18] found that all pairs outperformed the individuals in terms of quality and time spent after having studied 15 professional programmers with pair programming and individual programming. Williams [23] conducted a survey on professional programmers and found that 100% agreed that they were more confident in their solutions using pair programming than when they worked alone. Williams and Upchurch [22] found that the programmers communicated with each other more effectively, appeared to learn faster, and were happier. Canfora [4] also noticed the positive effect of pair programming on knowledge sharing during program design.

However some negative effects of pair programming were also reported. Nawrocki and Wokciechowski [17] mentioned that pairs spend nearly twice as much total effort than individual programmers. Beck [2] stated that pair programming is not suitable for very large projects.

The big difference between game applications and other applications lies in the fact that game applications use heavy graphics and users mainly interact with the graphic interface. However, the actual process is in fact done behind the interface similar to other applications and the most part of code in game applications is just like code in other types of applications, object doing some processes and communicating with other objects. A lot of code in a game is more than simple functions that return a value or set states.

Game applications are usually implemented by a group of 4-10 programmers [13]. The collaboration in game design and development is also an important issue because it affects the quality of the games and production time [8]. However, the overhead communication in a team of more than 4 people was reported to be high [19]. With pair programming, it can be reduced greatly [2].

Game applications are often of medium-size in term of number of lines of code. According to the characteristics of pair programming [2], the game applications may be suitable to apply pair programming due to their sizes.

In industry, there are some game companies applying this methodology, such as Coyote Development [7] and Sammy Studios [1].

2.2. Test-Driven Development

In the traditional waterfall approach, test definitions are developed independently from code according to the requirements. Automated tests may be done independently of coding. The number of tests passing vs the total number of tests is a metric to show the quality of the products. In test-driven approach, programmers write the tests for the new functionality before writing the code. At first, the test case fails since the corresponding code has yet to be written. Then the code for the actual functionality is written and tested till the test case has been passed. This process continues until all functionalities have been completed.

In general, the test-driven approach means that the system developed does exactly what it needs to do. It is easy to modify to make it do more things in the future as they are driven out by more tests. In test-driven, once the tests are passed, they become regression tests for ongoing development, which improves the code quality.

Muller and Hagner [16] conducted an experiment in which they divided the participants into two groups, one with test-driven development, and the other with traditional programming. Although they found no obvious discrepancies between two groups in overall development time and quality of the code, the test-driven group had significant fewer errors when the code was reused. Test-driven is more than unit testing, since it helps keeping the design simple from the initial stage and easy to change.

Game is composed of a set of rules. By knowing the rules, we know what game we are playing and how to implement the game. The game application needs to follow those rules. In fact, game designers are formulizing the rules, changing the rules and testing the rules. Test-driven approach can help game developers to efficiently complete their tasks.

Using test-driven can avoid the effort spent on the un-requested functionality of games, since the code written should be no more than necessary to pass the tests. The tests, in effect, are the functional requirements specification of the game applications; therefore, they ensure that the requirements and only the requirements are developed. This also increases the understanding of the functionality of games to be implemented.

One of the advantages of test-driven is that it makes sure that each functionality has its associated test and that everything that we want the software to do is documented as a test. Therefore, test-driven provides a way to document the game application since programmers are reluctant to do so during coding.

On the other hand, in order to achieve the robustness of the game application, automated tests are needed. Test-driven performs such task. It is important that tests are automated, which means that they could be run automatically without user interactions. That makes it easier to always run all the tests, which prevent game programmers from unintentionally break any already existing functionality when adding a new functionality to the game application.

Although testing user interface of game applications is a hard job, it is still possible to use test-driven with user interface implementation [11] [14].

2.3. Refactoring

Refactoring means consistently cleaning and improving the code which makes code easier to maintain

and extend, but without changing its observable behavior. It is a technique used to improve the design of the existing code. On the other hand, if the code is well structured, it is easy and efficient for programmers to add new functionality to it.

Refactoring might introduce new bugs, however, unit testing, which is part of test-driven approach, helps ensure that refactored code does not break existing functionality and introduce bugs.

Applying refactoring during the game development process can enhance the game quality. Since game applications take much time to compile and run, refactoring becomes quite necessary since high quality of code take less time to run. Time spent on the refactoring may shorten the entire game development time as well since refactoring makes debugging easier and makes adding new functionality faster.

3. Case Study Design

The case study design is based on the following hypothesis: "XP practices, such as pair programming, test-driven and refactoring are not applicable to game development" which is falsified by the case study. The rest of this section presents case study design in detail.

3.1. Participants

Eight undergraduate students from the Department of Computer Science at Wayne State University, and four undergraduate students from the Department of Computer Science at Algoma University College, who were taking software engineering courses, are classified as novice programmers. They voluntarily joined the case study as a part of their course projects. They had small-to-intermediate programming experience with C++, and Java, but they had never used pair programming or test-driven and refactoring practices. Paired students were from Wayne State University and learned C++ in introduction courses, and those individuals were from Algoma University College and took Java in introduction courses. All 12 students are advanced students in the classes. We randomly chose students to form the pairs. The pairs worked with test-driven and refactoring practices. The remaining four individuals worked alone using traditional waterfall-like approach.

3.2. Material

The task to be solved in the case study is to implement an application which records the scores for bowling games. All the participants worked on the same task. Our novice programmers were not originally familiar with the bowling domain.

3.3. Recording Method

In order to trace the development process and the time spent, we used free software "Microsoft Producer" [15], to capture computer screens and to record voice at the same time when programmers conduct their work. The recorded media files were mainly used for cognitive research, and partial results have been published in [24].

3.4. Procedures

Our case study was carried out in 2005 and 2006. Two pairs did their work in February, 2005 and two pairs completed their task in June, 2005. The four individuals finished their work in February, 2006. All pairs were asked to use Eclipse, an open source Java compiler, and JUnit. Two individuals chose Eclipse and another two used TextPad with JDK.

We performed short training sessions prior to the case study for pairs because programmers were new to pair programming, test-driven development and refactoring. During the training session, the paired programmers were provided with reading materials on pair programming, test-driven, and refactoring techniques, and they were asked to implement a simple program using the Eclipse environment and JUnit in order to understand the procedure and to be familiar with the tools.

The four individuals were advised to use traditional waterfall-like development process. All of pairs and individuals were asked to write a high quality program in an efficient way.

One of the authors acted as the mentor, who monitored the programming process for all the pairs and the individuals. Programmers were provided with a list of bowling scoring rules. After the case study, programmer pairs were asked to conduct a survey.

4. Results and Discussion

In this section, we summarize the case study results and the survey results.

4.1. Results of the Case Study

Table 1 summarizes the characteristics of the developed programs by each pair and individual and the actual time used. Table 2 contains the summery results for all the four pairs and four individuals. Please note that the times listed in Table 1 and Table 2 are the duration that the pairs and individuals spent on the task, therefore the "total time cost" for a pair is twice as much as the time indicated. Therefore, it is slightly bigger than the total time cost spent by individuals who worked on the same task. Please also note that first individual lost the development direction twice and gave up all the previous

code and that is why he took much longer time. On the other hand, programmer pairs worked on the tasks for a significantly shorter duration than individual programmers. The average time spent by the pairs is 216 minutes, while as that for individuals is 418 minutes (as Table 2). This indicates the pairing may reduce the time of game development, which is consistent with the result for pair programming in other software development reported [6, 23].

Table1. Main characteristics of the programs developed by each pair and individual

Items	Pair 1	Pair 2	Pair 3	Pair 4	Ind. 1	Ind. 2	Ind. 3	Ind. 4
LOC	360	249	289	269	195	215	150	162
Number of class members	24	28	26	27	18	40	1	2
Number of classes	4	4	3	2	2	4	1	1
Number of classed created in first half of code	3	2	2	2	2	4	1	1
Number of refactoring	1	2	3	1	0	1	0	0
Numbers of test cases passed	12	12	11	12	11	9	9	10
Time used (minutes)	154	241	260	210	568	268	380	459
LOC written per hour	140	62	67	77	21	48	24	22

The average number of lines of code in the programs developed by pairs is 291 and that of the programs written by individuals is 180 (see Table 2). Although the programs written by the pairs have more lines of code than those by individuals, LOC (line of code) written per hour by pairs are much higher than those by individuals, which indicates the high efficiency of pair programming. Please notice that two individuals put everything in one class and that is why their programs have less number of lines of code than other two individuals. That also indicates individuals lack the advantage of pairing where partner could provides some suggestions for the program design.

Table 2. Summery of data from the programs developed by pairs and individuals

Items	Pairs		Individuals	
	Av.	SD	Av.	SD
LOC	291.7	48.3	180.5	29.8
Number of class members	26.3	1.7	15.2	18.2
Number of classes	3.3	0.96	2	1.4
Number of classed created in first half of code	2.3	0.5	2	1.4
Number of refactoring	1.8	0.96	0.3	0.5
Numbers of test cases passed	11.8	0.5	9.8	1.0
Time used (minutes)	216.3	46.3	418.8	126.7
LOC written per hour	86.5	36.2	28.8	12.9

A comparison between the final programs constructed by pairs and those by individual shows that the former has a better design. Pairs created much more (26 in average, and almost twice) class members (methods) than individuals (15 in average) in general (see Table 2), which indicates modules in pairs' programs have higher cohesion. It coincides with the finding of Beck [3] who stated that codes written with test-driven technique tend to be more cohesive and less coupled than codes that are written with traditional approach.

Table 3. Result of quantitative questions

Questions	Lowest (0.0)	Highest (5.0)	Answer(av)
How effective do you think pair programming was for the project?	Not effective	Very effective	4.5
Did you and your partner contribute equally to the project?	Very unequal	equal	3.5
What is your rating of your performance?	Did very little	Did most of work	4.0
What is your rating of your partner's performance?	Did very little	Did most of work	4.0
Do you think you learned more or less than you would have if you had worked on your own?	Much less	Much more	3.5
How do you think the time that you personally spent on this project compares to the time it would have taken you to do it on your own?	Pairs much slower	Pairs much faster	4.0
Would you like to use pair programming during your future graduate course project?	Not at all	Very like	4.5
How do you like the test-driven technique?	A little	Very much	4.0
How do you like the refactoring?	A little	Very much	3.0

The pairs' class members are also much more elegant and readable. Please note second individual created a lot of unnecessary data members and that is why the program contains over 40 data members. However, in general, the individuals created one or two classes at the beginning and kept them until the end, while pairs created classes as needed at beginning and they added one more class in the middle of the process. The program implemented by our third individual is also hard to read with less meaningful variable names and the programs written by pairs have more meaningful variable names, which proves that with the help of each other, programmer pairs are able to write higher quality code, which supports the same finding by [23]. It is interesting

to point out that all the paired programmers have less experience in using Java than those individuals as mentioned in previous section.

Since our paired programmers were new to the refactoring technique, they applied it several times at the beginning of the process by only renaming the variables, but they did not extract methods or classes [9]. Three individuals did not clear up the code at all. Using refactoring certainly contributed to higher quality of code.

In order to test the quality of the programs written by pairs and individuals, we created 12 black-box test cases and run them on all the eight programs. The test-cases can verify how well the requirement specifications were met by the program and how robust the program is. The programs written by the pairs pass almost all the 12 test cases since they were written with test-driven techniques. The programs by individuals only passed 9 to 11 test cases.

Since test-driven approach provides a way to do documentation, programs written by pairs have been documented once they are completed and therefore they are easy to understand.

In general, pair programming, test-driven and refactoring techniques all contribute to the higher quality of code written by four pairs.

4.2. Survey Results

After completing the case study with pairs, we conducted a survey on pair programming, test-driven and refactoring with the survey questions listed in Table 3 and Table 4.

In general, participants thought that pair programming is an efficient way to develop software. All were happy about the performance by themselves and their partners in general; however, they thought the contributions from the two people in the pair were slightly different, which is consistent with the two roles in the pair: one is the driver who controls the keyboard and the other one is observer performing code review and helping driver to make decisions [2]. Previous work has indicated that a role rotation is needed for further promoting learning between the two programmers in the pair [21].

Programmers indicated that pair programming can actually saves time in projects, which corresponds to the shorter project durations as shown in Table 1 and Table 2. The four pairs showed that they liked the pair programming technique more than them programming alone and it seems that they are willing to apply it in their future projects.

The pairs seem to prefer pair programming more than test-driven and refactoring techniques since they rated test-driven and refactoring with lower scores. Pair programming is easier to learn and easier to use than the other two practices. Test-driven technique is new to all the programmers and is hard to apply. Refactoring is less favorable since programmers often forgot to use it.

Table 4. Answers of qualitative questions

Questions	Answers (the number of answer)
Do you feel like you have learned anything just by reading your partner's code?	• Yes (7) • Others: I did not learn too much (1)
What was the biggest problem you have had to overcome as a paired programmer?	• No problem (4) • Others: The pair should be in good relation; should know how to choice different options; should know how to explain their idea
What do you think is the biggest concern in pair programming?	• No problem (4) • Others: agreement, hard to cooperate; time conflict; comparable
What are the advantages of pair programming?	• Improve the quality; more options; more confidence; find the bugs quickly; learn from each other
What do you think is the biggest advantage of test-driven?	• Features can be tested one by one; gives us a correct direction; confidence with previous testing; increases the correctness
What do you think is the biggest problem with test-driven approach?	• No problem (2) • Others: add amount of code for testing; not familiar; forget to use it; hard to find next function
What do you think is the biggest advantage of refactoring?	• Code become more and more readable; make code shorter; makes code easy maintainable
What do you think is the biggest problem with refactoring?	• No problem (1) • Others: time consuming; could change the concepts; not familiar

Based on the case study and the survey results, it seems that pair programming, test-driven and refactoring can be applied in game development in some aspects since paired programmers with test-driven and refactoring produced higher quality results within a shorter period of time, compared with those working individually with traditional waterfall-like approach.

Programmers often have resistance to do documentation when they implement the program. Documentation increases the understandability of the programs. With test-driven development, the documentation is done at the same time, which not only can reduce the resistance, but also can save programmers' time.

4.3. Limitations

While our case study can be replicated, some of its features may cause some limitations in the result:

- The problem solved by the programmers is a part of a game application and is relatively simple. Most game applications have heavy interface. The results may be different when solving more complex problems.

- Our sample size was relatively small with only 4 pairs and 4 individuals.
- The subjects of the case study are classified as novice programmers in term of XP knowledge. The result might be different if we conduct the case study on experienced professional programmers.

5. Conclusions and Future Work

XP practices such as pair programming, test-driven and refactoring increasingly attract attention. They have been reported to have benefits in software development. However, few researches and no experiment have been done on XP in game development.

We performed a case study with 12 advanced undergraduate students, who worked as either pairs or individuals. Paired programmers were also using test-driven and refactoring techniques, while the four individuals were applying waterfall-like approach. All the subjects worked on implementing an application which scores the Bowling game.

The case study showed that programmer pairs spent much shorter period of time in completing the task than individuals. Programmer pairs completed the tasks with higher quality than individuals since their programs passed more test cases than those by individuals. The modules in the programs written by pairs have higher cohesion and more meaningful variable names. Paired programmers also wrote much more lines of code per hour than individuals, showing its efficiency. According to the survey, all students are satisfied with the performance by themselves and their partners and they are willing to use pair programming, test-driven and refactoring techniques in the future. Based on the case study and survey results, it seems that some XP practices may be useful in game development.

For the future work, we would like to replicate this case study on different sizes of gaming problems in order to further validate our observation.

6. References

[1] "Sammy Corporation:http://www.sammy.co.jp/ english/index.html ", 2006.

[2] Beck, K., *Extreme Programming Explained*, Maassachusettes, Addison-Wesley, 2000.

[3] Beck, K., *Test-Driving development: by Example*, Addison Wesley, 2003.

[4] Canfora, G., Cimitile, A., and Visaggio, C., "Working in pairs as a means for design knowledge building: an empirical study", in Proceedings of International Workshop on Program Comprehension, Bari, Italy, 24-26 June 2004, pp. 62-69.

[5] Coad, P., LeFebrve, E., and Luca, J., "Feature-driven development", in *Java Modeling in Color with UML*, Prentice Hall, 1999.

[6] Cockburn, A. and Williams, L., "The costs and benefits of pair programming", in Proceedings of Extreme Programming and Flexible Processes in Software Engineering, Cagliari, Italy, June 21-23, 2000, pp. 223-243.

[7] Coyoto Development, "http://www.coyotedev.freeserve.co.uk/", 2006.

[8] Falstein, N. and Fox, D., "Collaborating in Game Design, http://www.gamasutra.com/features/19970601/collab_in_game_design.htm", 1997.

[9] Fowler, M., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

[10] George, B. and Williams, L., "An initial investigation of test driven development in industry", in Proceedings of the 2003 ACM Symposium on Applied Computing Melbourne, Florida, 2003, pp. 1135 - 1139

[11] Hamill, P., *Unit Test Frameworks*, O'Reilly, 2004.

[12] Llopis, N., "By the Books: Solid Software Engineering for Games, http://convexhull.com/sweng/GDC2003.html", 2003.

[13] Llopis, N., "Games from within: GDC 2004: Software Engineering Roundtable Summary, http://www.gamesfromwithin.com/articles/0403/000015.html", 2004.

[14] Llopis, N., "Stepping Through the Looking Glass: Test-Driven Game Development, http://www.gamesfromwithin. com/articles/0502/000073.html", 2005.

[15] Microsoft, "Microsoft Producer for Microsoft Office Powerpoint 2003", http://www.microsoft.com/windows/ windowsmedia/technologies/producer.aspx,

[16] Muller, M. M. and Hagner, O., "Experiment about test-first programming", *IEE Proceedings Software*, vol. 149, no. 5, October 2002, pp. 131-136.

[17] Nawrocki, J. and Wokciechowski, A., "Experimental evaluation of pair programming", in Proceedings of European Software Control and Metrics (Escom), London, England, April 2001, pp. 269-276.

[18] Nosek, J. T., "The case for collaborative programming", *Communications of the ACM*, vol. 41, no. 3, 1998, pp. 105-108.

[19] Pressman, R., *Software Engineering A Practitioner's Approach*, McGraw Hill, 2001.

[20] Schwaber, K. and Beedle, M., *Agile Software Development with SCRUM*, Prentice Hall, 2001.

[21] Srikanth, H., Williams, J. G., Wiebe, E., Miller, C., and Balik, S., "On pair rotation in the computer science course", in Proceedings of the 17th Conference on Software Engineering Education and Training, Norfolk, Virginia March 1-3 2004, pp. 144-149.

[22] Williams, J. G. and UpChirch, R. L., "In support of student pair-programming", in Proceedings of the 32th SIGCSE Technical Symposium on Computer Science Education, Charlotte, NC, February 21-25, 2001, pp. 327 - 331.

[23] Williams, L., Kessler, R., Cuningham, W., and Jeffries, R., "Strengthening the case for pair-programming", *IEEE Software* July/August 2000, pp. 19-25.

[24] Xu, S. and Rajlich, V., "Dialog-based protocol: an empirical research method for cognitive activity in software engineering", in Proceedings of the 4th ACM/IEEE International Symposium on Empirical Software Engineering, Noosa Heads, Queensland, November 17-18, 2005.

0-7695-2613-6/06 $20.00 © 2006 IEEE