Does the Improved Agile Team Cohesion from Pair Programming Affect Software Quality?

COMP150 - Agile Essay Draft Assignment

Samantha Wills

May 5, 2016

Introduction

Agile practises are often used in game development studio to allow for group discussion and improve retrospective evaluation of current projects to improve project quality. For many companies, the cost of quality assurance and testers can prove costly during and post production. If pair programming can be used to improve software quality then it could reduce the costs of fixing bugs and player approval.

In the games industry, the quality of the software can be effected and interpreted in different ways. For many, it is the functional cohesion of the software which constitutes as high quality while others include code comprehension and commenting. During the process of programming, there can be a variety of factors influencing the programmers ability to code and can often change the speed and quality. Therefore, in many agile practices, pair programming is used to reduce common mistakes and focus code direction. This essay aims to investigate the ways pair programming can be used and has proven to improve standards of work. It also aims to investigate the effect of team strength and relationship of the programming pair in their attempt to improve software quality.

Pair programming is where one person takes on the role of the "driver", who is responsible for implementing code with control of the keyboard[1], and the other is the "navigator" whose role is to maintain code focus [2].

Main Body

The definition of software quality differs depending on the individual and their role within the game industry. A programmer would perceive code commenting, naming conventions and data structure. For this essay, the measurement of software quality is a combination of code length, the number of errors found or cyclomatic complexity number (CCN)[3] . There is also the measure of the qualitative description of game software written.

People who pair program notice an increase in coding confidence and feel more compelled to refactor code, write more tests and use more code integration [4]. This has been attributed to the effect of your partner watching you and so you are less likely to put these things off [5].

This suggests that groups who pair programme are more likely to use their partner as the motivation to improve their coding and testing practice. This also suggests that they could feel more confident knowing that any mistakes made from refactoring is more likely to be recognised by their navigator. For a game studio, this means that groups are more integrated into their project and could explain all aspect of their project to any new members or teams.

Another industry advantage, and the focus for most employers, is that those who pair programme have also reported higher quality code. A survey from the University of Memphis reported a 100% agreement, from its software engineering class, that using pair programming improved the code quality in comparison to individual work [4].

In addition, many student or junior programmers reported that pair programming improved their coding ability [6].

This was supported by a comparative study, from University of California Santa Cruz, that looked at the same programming task completed by pairs and individuals. To objectively measure the quality, they measured the length of the programs and the cyclomatic complexity number (CCN)[3] of programs. The study resulted in 23% mean increase in pair programming quality for one assignment, and then 17% for another [7].

For many projects, the code review process was reported to have worked well as their "partner was familiar with the design and code"[8]. For agile practices, the ability evaluate and manage the project is crucial. This suggests that team cohesion benefits from pair programming with more responsive code reviews. This also suggests that from better code comprehension provides an improved ability to give feedback to the agile process and development team.

For game studios, the appropriateness and opinions of the development team are a crucial factors in the effectiveness of this approach. While different companies have used a varying amount of pair programming in their production team, it was reported that 69% in a study of 31 developers wanted more pair programming than the average of 19% time allocated [9]. This questions what the limitations are for pair programming, and what prevents management from using this agile practice.

The personality of the individual and approach to conflict are also factors in the effectiveness of agile practices. It has been found that programmers with an "integrating" style of conflict handling are concerned with both their work and their partners work. This style results in good problem solving skills, collaboration, cooperation and solution oriented [10].

Conclusion

For game studios, the size of the development team could impact whether pair programming is feasible for producing a product, and the amount of pair programming allocated to production. For smaller teams, where one programmer is assigned to a certain task, there isn't an option to put programmers in a pair. Similarly, the tools required to pair programme over distance are restrictive to by each other's time schedule and forms of communication.

However, it has been shown to be effective in both education and industry settings, to improve both programming confidence and some measurable form of software quality.

It is the individual development teams to make the choice to use pair programming, and how much to do, as the personality of the programmers can also affect the effectiveness of agile practices. For some, this may be that the team would benefit from agile practices with only some pair programming during reviews or testing. However for most, especially teams with difficulty in team cohesion, the benefit of agile practices in combination with pair programming could improve the team moral and the product overall.

References

[1] A. Höfer, "Video analysis of pair programming," in *Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral*, ser. APOS '08. New York, NY, USA: ACM, 2008, pp. 37–41. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/1370143.1370151

[2] H. Hulkko and P. Abrahamsson, "A multiple case study on the impact of pair programming on product quality," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 495–504. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/1062455.1062545

[3] M. Xenos, D. Stavrinoudis, K. Zikouli, and D. Christodoulakis, "Object-oriented metrics-a survey," in *Proceedings of the FESMA*, 2000, pp. 1–10.

[4] L. B. Sherrell and J. J. Robertson, "Pair programming and agile software development: Experiences in a college setting," *J. Comput. Sci. Coll.*, vol. 22, no. 2, pp. 145–153, Dec. 2006. [Online]. Available: http://dl.acm.org.ezproxy.falmouth.ac.uk/citation.cfm?id=1181901.1181927

[5] K. Beck, *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, 2000.

[6] A. D. Radermacher and G. S. Walia, "Investigating the effective implementation of pair programming: An empirical investigation," in *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, pp. 655–660. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/1953163.1953346

[7] B. Hanks, C. McDowell, D. Draper, and M. Krnjajic, “Program quality with pair programming in cs1,” in *ACM SIGCSE Bulletin*, vol. 36, no. 3. ACM, 2004, pp. 176–180.

[8] J. Vanhanen and H. Korpi, “Experiences of using pair programming in an agile project,” in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on.* IEEE, 2007, pp. 274b–274b.

[9] L. Plonka and J. van der Linden, “Why developers don’t pair more often,” in *Proceedings of the 5th International Workshop on Co-operative and Human Aspects of Software Engineering*, ser. CHASE ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 123–125. [Online]. Available: http://dl.acm.org.ezproxy.falmouth.ac.uk/citation.cfm?id=2663638.2663664

[10] M. A. Domino, R. W. Collins, A. R. Hevner, and C. F. Cohen, “Conflict in collaborative software development,” in *Proceedings of the 2003 SIGMIS Conference on Computer Personnel Research: Freedom in Philadelphia–leveraging Differences and Diversity in the IT Workforce*, ser. SIGMIS CPR ’03. New York, NY, USA: ACM, 2003, pp. 44–51. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/761849.761856