Pair programming

Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the *driver*, writes code while the other, the *observer* or *navigator*,[1] reviews each line of code as it is typed in. The two programmers switch roles frequently.

While reviewing, the observer also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address. This is intended to free the driver to focus all of their attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.

Two co-workers pair programming, 2007

Contents

Economics

Pair programming increases the man-hours required to deliver code compared to programmers working individually. Experiments yielded diverse results, suggesting increases of between 15% and 100%.[2] However, the resulting code has about 15% fewer defects.[3] Along with code development time, other factors like field support costs and quality assurance also figure in to the return on investment. Pair programming might theoretically offset these expenses by reducing defects in the programs.[3]

In addition to preventing mistakes as they are made, other intangible benefits may exist. For example, the courtesy of rejecting phone calls or other distractions while working together, taking fewer breaks at agreed-upon intervals, or shared breaks to return phone calls (but returning to work quickly since someone is waiting). One member of the team might have more focus and help drive or awaken the other if they lose focus, and that role might periodically change. One member might have knowledge of a topic or technique which the other does not, which might eliminate delays to find or test a solution, or allow for a better solution, thus effectively expanding the skillset, knowledge, and experience of a programmer as compared to working alone. Each of these intangible benefits, and many more, may be hard to accurately measure, but can contribute to more efficient working hours.

Design quality

A system with two programmers possesses greater potential for the generation of more diverse solutions to problems for three reasons:

1. the programmers bring different prior experiences to the task;
2. they may access information relevant to the task in different ways;
3. they stand in different relationships to the problem by virtue of their functional roles.

In an attempt to share goals and plans, the programmers must overtly negotiate a shared course of action when a conflict arises between them. In doing so, they consider a larger number of ways of solving the problem than a single programmer alone might do. This significantly improves the design quality of the program as it reduces the chances of selecting a poor method.[4]

Satisfaction

In an online survey of pair programmers from 2000, 96% of them stated that they enjoyed their work more than when they programmed alone and 95% said that they were more confident in their solutions when they pair programmed.[5]

Learning

Knowledge is constantly shared between pair programmers, whether in the industry or in a classroom, many sources suggest that students show higher confidence when programming in pairs,[5] and many learn whether it be from tips on programming language rules to overall design skill.[6] In "promiscuous pairing", each programmer communicates and works with all the other programmers on the team rather than pairing only with one partner, which causes knowledge of the system to spread throughout the whole team.[3] Pair programming allows programmers to examine their partner's code and provide feedback which is necessary to increase their own ability to develop monitoring mechanisms for their own learning activities.[6]

Team-building and communication

Pair programming allows team members to share problems and solutions quickly making them less likely to have hidden agendas from each other. This helps pair programmers to learn to communicate more easily. “This raises the communication bandwidth and frequency within the project, increasing overall information flow within the team.”[3]

Studies

There are both empirical studies and meta-analyses of pair programming. The empirical studies tend to examine the level of productivity and the quality of the code, while meta-analyses may focus on biases introduced by the process of testing and publishing.

A meta-analysis found pairs typically consider more design alternatives than programmers working alone, arrive at simpler more maintainable designs, and catch design defects earlier. However, it raised concerns that its findings may have been influenced by "signs of publication bias among published studies on pair programming". It concluded that "pair programming is not uniformly beneficial or effective".[7]

Although pair programmers may complete a task faster than a solo programmer, the total number of man-hours increases.[2] A manager would have to balance faster completion of the work and reduced testing and debugging time against the higher cost of coding. The relative weight of these factors can vary by project and task.

The benefit of pairing is greatest on tasks that the programmers do not fully understand before they begin: that is, challenging tasks that call for creativity and sophistication, and for novices as compared to experts.[2] Pair programming could be helpful for attaining high quality and correctness on complex programming tasks, but it would also increase the development effort (cost) significantly.[7]

On simple tasks, which the pair already fully understands, pairing results in a net drop in productivity.[2][8] It may reduce the code development time but also risks reducing the quality of the program.[7] Productivity can also drop when novice–novice pairing is used without sufficient availability of a mentor to coach them.[9]

Indicators of non-performance

There are indicators that a pair is not performing well:

- *Disengagement* may present as one of the members physically withdraws away from the keyboard, accesses email, or even falls asleep.
- The *"Watch the Master"* phenomenon can arise if one member is more experienced than the other. In this situation, the junior member may take the observer role, deferring to the senior member of the pair for the majority of coding activity. This can easily lead to disengagement.

Pairing variations

Expert–expert
: Expert–expert pairing may seem to be the obvious choice for the highest productivity and can produce great results, but it often yields little insight into new ways to solve problems, as both parties are unlikely to question established practices.[2]

Expert–novice
: Expert–novice pairing creates many opportunities for the expert to mentor the novice. This pairing can also introduce new ideas, as the novice is more likely to question established practices. The expert, now required to explain established practices, is also more likely to question them. However, in this pairing, an intimidated novice may passively "watch the master" and hesitate to participate meaningfully. Also, some experts may not have the patience needed to allow constructive novice participation.[10]

Novice–novice
: Novice–novice pairing can produce results significantly better than two novices working independently, although this practice is generally discouraged.[3]

Remote pair programming

Remote pair programming, also known as **virtual pair programming** or **distributed pair programming**, is pair programming in which the two programmers are in different locations,[11] working via a collaborative real-time editor, shared desktop, or a remote pair programming IDE plugin. Remote pairing introduces difficulties not present in face-to-face pairing, such as extra delays for coordination, depending more on "heavyweight" task-tracking tools instead of "lightweight" ones like index cards, and loss of verbal communication resulting in confusion and conflicts over such things as who "has the keyboard".[12]

Tool support could be provided by:

- Whole-screen sharing software[13][14][15]
- Terminal multiplexers
- Specialized distributed editing tools
- Audio chat programs or VoIP software could be helpful when the screen sharing software does not provide two-way audio capability. Use of headsets keep the programmers' hands free
- Cloud development environments
- Collaborative pair programming services

See also

- Mob programming
- Extreme programming
- Chavrusa
- Joint attention

References

1. Williams, Laurie (February 19–20, 2001). *Integrating pair programming into a software development process*. 14th Conference on Software Engineering Education and Training. Charlotte. pp. 27–36. doi:10.1109/CSEE.2001.913816 (https://doi.org/10.1109%2FCSEE.2001.913816). ISBN 0-7695-1059-0. "One of the programmers, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects, and also thinks strategically about the direction of the work."
2. Lui, Kim Man (September 2006). "Pair programming productivity: Novice–novice vs. expert–expert" (http://www.cs.utexas.edu/users/mckinley/305j/pair-hcs-2006.pdf) (PDF). *International Journal of Human–Computer Studies*. **64** (9): 915–925. CiteSeerX 10.1.1.364.2159 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.364.2159). doi:10.1016/j.ijhcs.2006.04.010 (https://doi.org/10.1016%2Fj.ijhcs.2006.04.010). Retrieved 2012-11-18.
3. Cockburn, Alistair; Williams, Laurie (2000). "The Costs and Benefits of Pair Programming" (http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF) (PDF). *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000)*.
4. Flor, Nick V.; Hutchins, Edwin L. (1991). "Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance" (https://books.google.com/books?id=KT_bpSSJBgcC&pg=PA36). In Koenemann-Belliveau, Jürgen; Moher, Thomas G.; Robertson, Scott P. (eds.). *Empirical Studies of Programmers: Fourth Workshop*. Ablex. pp. 36–64. ISBN 978-0-89391-856-9.
5. Williams, Laurie; Kessler, Robert R.; Cunningham, Ward; Jeffries, Ron (2000). "Strengthening the case for pair programming" (http://sunnyday.mit.edu/16.355/williams.pdf) (PDF). *IEEE Software*. **17** (4): 19–25. CiteSeerX 10.1.1.33.5248 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.5248). doi:10.1109/52.854064 (https://doi.org/10.1109%2F52.854064).
6. Williams, Laurie; Upchurch, Richard L. (2001). "In support of student pair-programming". *ACM SIGCSE Bulletin*. **33** (1): 327–31. doi:10.1145/366413.364614 (https://doi.org/10.1145%2F366413.364614).
7. Hannay, Jo E.; Tore Dybå; Erik Arisholm; Dag I.K. Sjøberg (July 2009). "The Effectiveness of Pair Programming: A Meta-Analysis". *Information and Software Technology*. **51** (7): 1110–1122. doi:10.1016/j.infsof.2009.02.001 (https://doi.org/10.1016%2Fj.infsof.2009.02.001).
8. Arisholm, Erik; Hans Gallis; Tore Dybå; Dag I.K. Sjøberg (February 2007). "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise" (https://web.archive.org/web/20101029033020/http://simula.no/research/se/publications/Arisholm.2006.2/simula_pdf_file). *IEEE Transactions on Software Engineering*. **33** (2): 65–86. doi:10.1109/TSE.2007.17 (https://doi.org/10.1109%2FTSE.2007.17). Archived from the original (http://simula.no/research/se/publications/Arisholm.2006.2/simula_pdf_file) on 2010-10-29. Retrieved 2008-07-21.
9. Stephens, Matt; Doug Rosenberg. "Will Pair Programming Really Improve Your Project?" (http://www.methodsandtools.com/archive/archive.php?id=10). Retrieved 28 May 2011.
10. Williams, L. & Kessler, R. (2003). *Pair Programming Illuminated* (https://books.google.com/books?id=LRQhdlrKNE8C). Boston: Addison-Wesley Professional. ISBN 9780201745764.
11. Flor, Nick V. (2006). "Globally distributed software development and pair programming". *Communications of the ACM*. **49** (10): 57–8. doi:10.1145/1164394.1164421 (https://doi.org/10.1145%2F1164394.1164421).
12. Schümmer, Till; Stephan Lukosch (September 2009). "Understanding Tools and Practices for Distributed Pair Programming" (http://www.jucs.org/jucs_15_16/understanding_tools_and_practices/jucs_15_16_3101_3125_schuemmer.pdf) (PDF). *Journal of Universal Computer Science*. **15** (16): 3101–3125. Retrieved 2010-04-30.
13. Agile Ajax: Pair Programming with VNC (http://blogs.pathf.com/agileajax/2007/09/pair-programmin.html) Archived (https://web.archive.org/web/20080402003711/http://blogs.pathf.com/agileajax/2007/09/pair-programmin.html) 2008-04-02 at the Wayback Machine
14. Pair Programming – The Ultimate Setup and the other options we tried. – Jonathan Cogley's Blog (http://weblogs.asp.net/jcogley/archive/2004/10/13/242117.aspx)
15. Ola Lindberg › Computer ergonomics and pair programming (http://olalindberg.com/blog/2009/07/06/computer-ergonomics-and-pair-programming/)

External links

- wikiHow: How to Pair Program (http://www.wikihow.com/Pair-Program) How-to guide; contains common wisdom on how to make pair programming work.
- Tuple:Pair Programming Guide (https://tuple.app/pair-programming-guide) Pair programming guide that covers paring styles, antipatterns, and more. Includes example paring videos.
- c2:PairProgramming
- c2:PairProgrammingPattern
- c2:PairRotationFrequency