Swarm or Pair? strengths and weaknesses of Pair Programming and Mob Programming

Herez Moise Kattan
IME-USP
Sao Paulo, SP
herez@ime.usp.br

Flavio Soares
IME-USP
Sao Paulo, SP
fcs@ime.usp.br

Alfredo Goldman
IME-USP
Sao Paulo, SP
gold@ime.usp.br

Eduardo Deboni
IPT
Sao Paulo, SP
jose@eduardodeboni.com

Eduardo Guerra
INPE
Sao Jose dos Campos, SP
eduardo.guerra@inpe.br

ABSTRACT

Create a robust software with long live, cheap to maintain is related to the quality of software product. This paper is a review of the literature looking for strengths and weaknesses of two popular practices for increasing the quality of the software development: Pair Programming and Mob Programming.

CCS CONCEPTS

• **Software and its engineering** → **Software creation and management**; **Agile software development**; **Programming teams**; • **Applied computing** → *Collaborative learning*;

KEYWORDS

Pair Programming, Mob Programming, Programming and Review Simultaneous in Pairs, Collaborative Programming, Collaborative Problem Solving, Collaboration

ACM Reference format:
Herez Moise Kattan, Flavio Soares, Alfredo Goldman, Eduardo Deboni, and Eduardo Guerra. 2018. Swarm or Pair? strengths and weaknesses of Pair Programming and Mob Programming. In *Proceedings of XP '18 Companion, Porto, Portugal, May 21–25, 2018,* 4 pages.
https://doi.org/10.1145/3234152.3234169

1 INTRODUCTION

The evolution of humankind developed social characteristics in the humans. Pair programming is a practice for software development teams, and it is also a process of informal review. It improves the code perceived quality and the technical level of the team. Pair programming boosts a collective feeling, both on ownership and responsibility for the system. Stimulating egoless programming idea [40] and Information Flow and Knowledge Sharing [16], the software is owned by the team as a whole and individuals are not responsible for problems with the code; the entire team has a responsibility to address these problems.

Mob Programming is a software development approach where the whole team works on the same thing, at the same time, in the same space, and at the same computer. Mob Programming, as Zuill [47] describes, is similar to pair programming [6], where two people work on the same computer and collaborate on the same code at the same time.

The main difference, compared to pair programming is that the whole team works together as part of the pairing. In addition to software coding, Mob Programming teams work together on almost all tasks that a typical software development team tackles, such as defining stories, designing, testing, deploying software, and collaborating with the customer [47] [26] [27].

We reviewed the most relevant literature about Pair Programming and Mob Programming. Our goal is to provide an answer to the following research question:

- **What are the most important strengths and weaknesses of Pair Programming and Mob Programming?**

2 RESEARCH METHOD

The sources of academic studies for this paper are IEEE Xplore (ieeexplore.ieee.org), ACM Digital Library (dl.acm.org), SpringerLink (springerlink.com), Elsevier, Agile Alliance and Google. As Mob Programming is still an emerging software development technique, early adopters publish experiences reports, for this reason, we put one inclusion criteria only to Mob Programming accepting gray literature (not controlled by commercial publishers), although sometimes peer-reviewed, only experience reports, possibly without a rigorous research method.

The search strings used: Pair Programming and Mob Programming. The papers were evaluated based on the inclusion criteria: peer-reviewed and grey literature [37] only to Mob Programming. The exclusion criteria were: incomplete studies, drafts, slides, abstracts and repeated studies. Due to the page limit, we report here only a summary of our literature review.

After reading all studies, a paper was included if scoped all the inclusion criteria and excluded if hadn't at least one of the exclusion criteria. To avoid bias, the first author performed the complete reading four times, including the interpretations made of all papers.

XP '18 Companion, May 21–25, 2018, Porto, Portugal

ACM ISBN 978-1-4503-6422-5/18/05…$15.00
https://doi.org/10.1145/3234152.3234169

The others co-authors also reviewed the interpretations to assure its correction [24] [25].

3 LITERATURE REVIEW

3.1 Pair Programming

3.1.1 Origins and Introduction. A pattern called Development in Pairs published in 1995, on the authors' views, some people occasionally only feel able to solve a problem with help. There are problems seem to be greater than individuals [15].

There are many old tech-savvy reports, such as Brooks's [10] report. It describes a situation when programming in college between 1953 and 1956, he and his pair wrote 1500 lines of code, which they executed without any error on the first attempt.

Pair programming helps to decrease the number of software defects [15]. It helps to improve software quality, but it also increases the effort of software development. On an experiment, the pair spent more work hours compared to two developers working alone [13].

Coplien and Harrison [14] defend the advantages of making pairing compatible designs work together. Again, in this way, they can produce more than the sum of the two individually. Constantine[13] has published about the reduction of the number of defects when programming in pairs. Nosek [33] found that pair programming improves code and algorithm quality, but also noted the increased development effort, as it observed that the pair spent more working hours than if a single programmer worked alone.

A formal definition: two programmers working collaboratively at the same activity, sitting side by side in front of the same computer. While one person is writing the code, then another is paying attention to the work, looking for defects and suggesting improvements [7]. There are similar approaches, using two monitors and mouses, but without any conceptual difference [2].

A famous phrase by Brooks [10], adding human resources to a late software project makes it later. Adding a new member to the team that does not know the system creates the problem of the first contact with the source code. The pair programming contributes to solving this problem because the new team member can pair with someone more experienced [41] [42].

For Padberg et al. [34] [35], the code produced using the Pair Programming is more defect-free. They conclude, when market pressure is strong, to add developers and to create more pairs, can accelerate the project and can increase its value for the business, but it also increases the cost.

3.1.2 Informal review and Communication. Pair programming is a less formal review process and probably does not encounter as many errors as code inspections, however, is cheaper [11] [12].

Cockburn et al. [11] [12] and Williams et al. [43] observed that productivity when using pair programming, seems to be comparable to that of two people working independently. The reason is that programming in pairs will discuss the system before developing it.

3.1.3 Confidence and Satisfaction. Pair programming stimulates programmers' motivation and satisfaction [32]. It improves code quality, increases programmers' confidence and satisfaction (noted a 24% increase for women and 15% for men) and reduces student dropout in computer courses, they collected data from 554 students who attempted the course at the University of California-Santa Cruz [30].

3.1.4 Reducing the rate at which developer exits the team; Programmer Experience and Task Complexity. Arisholm et al. [3] and Parrish et al. [36] carried out their work with experienced programmers. They found that there was a significant loss of productivity compared to two programmers working alone. There were some benefits to quality, but they did not fully compensate for the loss of productivity of pair programming, in spite of the knowledge sharing during pairing is very important as it reduces the overall risks of a project if any team member leaves, so this justifies its use.

Hulkko and Abrahamsson [21] have observed in four case studies the effectiveness of pair programming in disseminating knowledge. Another of work carried out by Vanhanen, Lassenius, and Mantyla [38] noted that it might be more useful in complex tasks.

Dyba et al. [17] observed an improvement in the quality, a reduction in the number of defects, and an increase of the effort in the project compared to individual programming. They report a direct relationship between the benefits of pair programming and programmer experience with task complexity. But, only one study analyzed, though large and consistent, [3] considered the complexity of the task and the programmer experience.

3.1.5 Mechanisms to improve the performance. Wray [46] [1] published four mechanisms to improve the performance of pair programming: chat, notice more details, fight against poor practices, and share and recognize competencies.

The developers chatting when are doing pair programming is a good indication of they are doing right and will have a good performance using the technique. A part of the effectiveness of the pair programming is presumably due to the interaction between programmers. When programmers chat about the problem in the moment of doubt, they find the solution faster and consequently productivity increases.

More details are perceived when doing pair programming. Thus, the pilot/driver frequently must be warned because probably not noticed the same issue, the reason is the human limitation of perception capacity that depends on the purpose of the search. There is a blind spot in the human attention on the unexpected, usually, have difficulty about notice something that is not paying attention. Fatigue decreases productivity, and changing the person from the pair typing on the keyboard during work can increase productivity. Pair programming reinforces the importance of keeping in mind that a beginner programmer increases her/his productivity when is pairing with someone more experienced.

The fighting against poor programming practices helps to increase the performance. Pair pressure helps to improve the programming quality. Pair Programming reinforces the importance following the programming standards and to program in the best possible way so that the code does not become bad and disorganized, this avoids rework and increases productivity.

Sharing and recognizing competencies cooperates with the dissemination of knowledge. Discover who are the experts on specific subjects and share this knowledge with the entire team, could help to increase the productivity. Wray [46] suggests the reworking of Dyba et al. [17] toward it to consider its four mechanisms with new

methods and criteria to effectively measure the performance of the team using Pair Programming.

A new approach called Programming and review simultaneous in Pairs [23], a kind of Simultaneous Style Pair Programming, benefit some productivity factors in software development compared to the traditional style Pair Programming.

3.1.6 Distributed Pair Programming. Pair programming works remotely on the view of Bandukda et al. [5] they observed that distributed pair programming, where pairs are in different physical dependencies communicating each other by voice and text messaging software, could be used to increase development productivity and the quality of the software product.

A paper by Williams and Stout [44] described some modifications in the agile practices adopted by a company, with teams in North America and Europe, to reduce problems in the development of software. They concluded that it is possible to distribute and scale agile teams, but keeping them small and together still is the best alternative.

3.1.7 Camaraderie and Personality. According to Plaue et al. who did an experimental study on pair programming to individual programming. The result observes pair programming increase the camaraderie. It also worked how a protective factor for preserving and reducing the evasion of women and novice programmers in computer science courses [28].

According to Hannay et al. [19], personality could be a valid indicator of team performance in the long-term. However, there was no significant evidence of how affects the performance of the team benefiting some related aspect consistently, especially when including indicators of skill, task complexity, and country cultural differences.

In the short-term, it pays off for industry and researchers, to focus on other performance preceptors, including experience and task complexity. Attention to the factors that may be relevant, such as team learning, team motivation, and programming skills. The reason is the personality traits are fixed in a person and could be a focus on malleable factors, such as learning, motivation, and commitment to perform improvements to the software produced.

3.2 Mob Programming

The idea of mob programming originated from technical meetings where a team member presented a code he knew. The group work together, exchange information on design, software architecture, learn programming techniques with others programmers, write code everybody together and provide feedback.

The main difference, comparing to pair programming, is that Mob Programming teams work together on almost all tasks that a typical software development team tackles, such as defining stories, designing, testing, deploying software, and collaborating with the customer.

The whole team works around a single workstation, only one member has the possession of the keyboard to modify the source code [47]. Mob Programming can be observed as an evolution of the learning environment in a Coding Dojo setting to a production environment [9].

Among the difficulties in adopting the mob programming, Hohman and Slocum [20] report that it is difficult to keep in focus, especially for those who are not with keyboard possession. According to Wilson [45], there is the concern of managers about the difficulty of having the whole team working together in a single requirement, in the view of them seems to be more productive to have pairs working on different tasks. On the other hand, Zuill [48] highlights the benefits of communication, collaboration, and team alignment.

Concerning weaknesses, Hohman and Slocum [20] report the difficulty to keep focus for the co-pilots/navigators without the possession of the keyboard. Wilson [45] cites two problems, the effect of dominant personalities within the Group and the appearance of Groupthink.

Irving Janis[22] studied the effect of extreme stress on group cohesiveness and how people make decisions under external threats. Groupthink is a quick and easy way to refer to the mode of thinking that persons engage in when concurrence-seeking becomes so dominant in a cohesive group that it tends to override realistic appraisal of alternative courses of action.

Among the strengths, Zuill [47][48] emphasizes that it strengthens communication, team alignment, collaboration and self-organized. Wilson [45] observe an increase in the dissemination of knowledge and the technique worked better for critical and complex software products.

There are two points of convergence in the conclusion of Hohman and Slocum [20] and Wilson [45] when using two projectors/monitors Mob Programming works better and is advantageous to switch the use with other techniques. If the main problem derived from the team not having a Shared Understanding of the project, Mob Programming involves an entire team is a great approach [47] [45] [8] [20] [29].

The initial evidence for the effectiveness of mob programming by its early adopters is quite encouraging, it still requires validation through rigorously designed empirical research, so that more organizations could adopt it fully understanding the conditions that accentuate its benefits and minimize potential risks [4] [31].

Fun is fundamental in life since it fosters interaction and learning [39]. In three case studies of open source software, all developers reported that had fun with Mob Programming and the main conclusion was an improvement of the learning experience [26].

Mob programming is surprisingly inclusive. Done right, it can provide a safe and nurturing environment for the fledgling programming skills of novice programmers or new team members. Not only this, remote mobbing offers significantly more support and flex for family life or other interests. Most unexpectedly of all, the ability to uniquely tailor a physical environment that is right for you to work in would be almost impossible to achieve in a traditional work environment. Couple this with a level of optionality about level and duration of social that is also difficult to achieve in an office and could be a blueprint for maximizing inclusivity for the neurodivergent team members [18].

4 CONCLUSION

Distributed Pair Programming and Mob Programming are topics with a lot of open questions nowadays. Open source projects and home-office work are a kind of remote work, thus Distributed Pair

Programming is a relevant problem to be investigated and Mob Programming requires validation through designed empirical research. Pair and Mob Programming seem to increase the learning and satisfaction of the developers compared to solo programming.

5 ACKNOWLEDGMENTS

Authors would like to thank the CAPES and the IME-USP.

REFERENCES

[1] 2010. Responses to "How Pair Programming Really Works". *IEEE Software* 27, 2 (March 2010), 8–9. https://doi.org/10.1109/MS.2010.51

[2] Ahmad Al-Jarrah and Enrico Pontelli. 2016. *On the Effectiveness of a Collaborative Virtual Pair-Programming Environment.* Springer International Publishing, Cham, 583–595. https://doi.org/10.1007/978-3-319-39483-1_53

[3] Erik Arisholm, Hans Gallis, Tore Dyba, and Dag I.K. Sjoberg. 2007. Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Trans. Soft. Eng.* 33, 2 (Feb. 2007), 65–86. https://doi.org/10.1109/TSE.2007.17

[4] Venugopal Balijepally, Sumera Chaudhry, and Sridhar P. Nerur. 2017. Mob Programming - A Promising Innovation in the Agile Toolkit. In *23rd Americas Conference on Information Systems, AMCIS 2017, Boston, MA, USA, August 10-12, 2017.* http://aisel.aisnet.org/amcis2017/SystemsAnalysis/Presentations/10

[5] M. Bandukda and Z. Nasir. 2010. Efficacy of distributed pair programming. In *2010 International Conference on Information and Emerging Technologies.* 1–6. https://doi.org/10.1109/ICIET.2010.5625667

[6] Kent Beck and Cynthia Andres. 2004. *Extreme Programming Explained: Embrace Change (2Nd Edition).* Addison-Wesley Professional.

[7] Andrew Begel and Nachi Nagappan. 2008. Pair Programming: What's in It for Me?. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08).* ACM, New York, NY, USA, 120 – 128. https://doi.org/10.1145/1414004.1414026

[8] Karel Boekhout. 2016. *Mob Programming: Find Fun Faster.* Springer International Publishing, Cham, 185–192. https://doi.org/10.1007/978-3-319-33515-5_15

[9] Mariana Bravo and Alfredo Goldman. 2010. *Reinforcing the Learning of Agile Practices Using Coding Dojos.* Springer Berlin Heidelberg, Berlin, Heidelberg, 379–380. https://doi.org/10.1007/978-3-642-13054-0_41

[10] Fred P. Brooks, Jr. 1975. The Mythical Man-Month. *SIGPLAN Not.* 10, 6 (April 1975), 193–. https://doi.org/10.1145/390016.808439

[11] Alistair Cockburn and Laurie Williams. 2000. The Costs and Benefits of Pair Programming. In *In eXtreme Programming and Flexible Processes in Software Engineering XP2000.* Addison-Wesley, 223–247.

[12] Alistair Cockburn and Laurie Williams. 2001. Extreme Programming Examined. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, Chapter The Costs and Benefits of Pair Programming, 223–243. http://dl.acm.org/citation.cfm?id=377517.377531

[13] Larry L. Constantine. 1995. *Constantine on Peopleware.* (1 edition ed.). Prentice Hall Ptr, Englewood Cliffs, N.J. Paperback: 219 pages.

[14] James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[15] James O. Coplien and Douglas C. Schmidt. 1995. *Pattern languages of program design* (1 edition ed.). Addison-Wesley Professional, New York, NY, 183–237.

[16] F.S.C. da Silva, J. Agusti-Cullell, M.E. James, and F. van Harmelen. 2008. *Information Flow and Knowledge Sharing.* Elsevier Science. https://books.google.com.br/books?id=QsdPMk7hdNQC

[17] T. Dyba, E. Arisholm, D. I. K. Sjoberg, J. E. Hannay, and F. Shull. 2007. Are Two Heads Better than One? On the Effectiveness of Pair Programming. *IEEE Software* 24, 6 (Nov 2007), 12–15. https://doi.org/10.1109/MS.2007.158

[18] Sal Freudenberg and Matt Wynne. 2018. The Surprisingly Inclusive Benefits of Mob Programming.. In *Experience report (XP '18).*

[19] Jo E. Hannay, Erik Arisholm, Harald Engvik, and Dag I. K. Sjoberg. 2010. Effects of Personality on Pair Programming. *IEEE Trans. Softw. Eng.* 36, 1 (Jan. 2010), 61–80. https://doi.org/10.1109/TSE.2009.41

[20] Moses M. Hohman and Andrew C. Slocum. [n. d.]. *Mob Programming and the Transition to XP.* Technical Report.

[21] Hanna Hulkko and Pekka Abrahamsson. 2005. A Multiple Case Study on the Impact of Pair Programming on Product Quality. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05).* ACM, New York, NY, USA, 495–504. https://doi.org/10.1145/1062455.1062545

[22] Irving L. Janis. 1971. Groupthink. *Psychology Today Magazine* 5, 6 (Nov 1971).

[23] Herez Moise Kattan. 2015. Programming and review simultaneous in Pairs: a pair programming extension. *Master Thesis. In: Institute for Technological Research of the Sao Paulo State.*, 88. https://doi.org/10.13140/RG.2.2.15831.68004

[24] Herez Moise Kattan. 2016. Illuminated Arrow: a research method to software engineering based on action research, a systematic review and grounded theory. *13th International Conf. on Inf. Systems and Technology Management, p. 1971-1978.*

[25] Herez Moise Kattan. 2017. Those who fail to learn from history are doomed to repeat it. *Agile Processes in Software Engineering and Extreme Programming: poster presented in the 18th International Conference on Agile Software Development, XP 2017. held in Cologne, Germany, in May 22-26.* https://doi.org/10.13140/RG.2.2.20864.02563

[26] Herez Moise Kattan, Frederico Oliveira, Alfredo Goldman, and Joseph William Yoder. 2018. Mob Programming: The State of the Art and Three Case Studies of Open Source Software. In *Agile Methods*, Viviane Almeida dos Santos, Gustavo Henrique Lima Pinto, and Adolfo Gustavo Serra Seca Neto (Eds.). Springer International Publishing, Cham, 146–160. https://doi.org/10.1007/978-3-319-73673-0_12

[27] Herez Moise Kattan, Flavio Soares, Alfredo Goldman, Eduardo Deboni, and Eduardo Guerra. 2018. Swarm or Pair? strengths and weaknesses of Pair Programming and Mob Programming. In *Poster at 19th International Conference on Agile Software Development: XP '18, May 21–25, 2018, Porto, Portugal.* https://doi.org/10.13140/RG.2.2.18105.06249

[28] Zhen Li, C. Plaue, and E. Kraemer. 2013. A spirit of camaraderie: The impact of pair programming on retention. In *2013 26th International Conference on Software Engineering Education and Training (CSEE T).* 209–218. https://doi.org/10.1109/CSEET.2013.6595252

[29] Carola Lilienthal. 2017. *From Pair Programming to Mob Programming to Mob Architecting.* Springer International Publishing, Cham, 3–12. https://doi.org/10.1007/978-3-319-49421-0_1

[30] Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald. 2006. Pair Programming Improves Student Retention, Confidence, and Program Quality. *Commun. ACM* 49, 8 (Aug. 2006), 90–95. https://doi.org/10.1145/1145287.1145293

[31] Herez Moise Kattan and Alfredo Goldman. 2017. Software Development Practices Patterns. In *Agile Processes in Software Engineering and Extreme Programming*, Hubert Baumeister, Horst Lichter, and Matthias Riebisch (Eds.). Springer International Publishing, Cham, 298–303. https://doi.org/10.1007/978-3-319-57633-6_23

[32] Matthias M. Muller and Frank Padberg. 2004. An Empirical Study About the Feelgood Factor in Pair Programming. In *Proceedings of the Software Metrics, 10th International Symposium (METRICS '04).* IEEE Computer Society, Washington, DC, USA, 151–158. https://doi.org/10.1109/METRICS.2004.8

[33] John T. Nosek. 1998. The Case for Collaborative Programming. *Commun. ACM* 41, 3 (March 1998), 105–108. https://doi.org/10.1145/272287.272333

[34] F. Padberg and M. M. Muller. 2003. Analyzing the cost and benefit of pair programming. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717).* 166–177. https://doi.org/10.1109/METRIC.2003.1232465

[35] Frank Padberg and Matthias M. Müller. 2003. Analyzing the Cost and Benefit of Pair Programming. In *Proceedings of the 9th International Symposium on Software Metrics (METRICS '03).* IEEE Computer Society, Washington, DC, USA, 166–. http://dl.acm.org/citation.cfm?id=942804.943771

[36] Allen Parrish, Randy Smith, David Hale, and Joanne Hale. 2004. A Field Study of Developer Pairs: Productivity Impacts and Implications. *IEEE Softw.* 21, 5 (Sept. 2004), 76–79. https://doi.org/10.1109/MS.2004.1331306

[37] Joachim Schöpfel. 2010. Towards a Prague Definition of Grey Literature. In *Twelfth International Conference on Grey Literature: Transparency in Grey Literature. Grey Tech Approaches to High Tech Issues. Prague, 6-7 December 2010.* Czech Republic, 11–26. https://archivesic.ccsd.cnrs.fr/sic_00581570

[38] Jari Vanhanen, Casper Lassenius, and Mika V. Mantyla. 2007. Issues and Tactics when Adopting Pair Programming: A Longitudinal Case Study. In *Proceedings of the International Conference on Software Engineering Advances (ICSEA '07).* IEEE CS, Washington, DC, USA, 70–. https://doi.org/10.1109/ICSEA.2007.48

[39] Luiz Carlos Vieira and Flavio Soares Correa da Silva. 2017. Assessment of fun in interactive systems: A survey. *Cognitive Systems Research* 41 (2017), 130 – 143. https://doi.org/10.1016/j.cogsys.2016.09.007

[40] Gerald Weinberg. 1971. *The Psychology of Computer Programming.* Wiley - Van Nostrand, New York, NY.

[41] Laurie Williams and Robert Kessler. 2000. The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education. In *Proceedings of the 13th Conference on Software Engineering Education & Training (CSEET '00).* 59–65.

[42] Laurie Williams and Robert Kessler. 2003. *Pair programming illuminated.* Pearson Education, Boston, MA.

[43] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. 2000. Strengthening the Case for Pair Programming. *IEEE Softw.* 17, 4 (July 2000), 19–25. https://doi.org/10.1109/52.854064

[44] W. Williams and M. Stout. 2008. Colossal, Scattered, and Chaotic (Planning with a Large Distributed Team). In *Agile 2008 Conference.* 356–361. https://doi.org/10.1109/Agile.2008.25

[45] Alexander Wilson. 2015. *Mob Programming - What Works, What Doesn't.* Springer International Publishing, Cham, 319–325. https://doi.org/10.1007/978-3-319-18612-2_33

[46] Stuart Wray. 2010. How Pair Programming Really Works. *IEEE Softw.* 27, 1 (Jan. 2010), 50–55. https://doi.org/10.1109/MS.2009.199

[47] Woody Zuill. 2014. Mob Programming: A Whole Team Approach. *(Agile '14).*

[48] Woody Zuill and Kevin Meadows. 2016. *Mob Programming. A Whole Team Approach.* LeanPub.