Mob Programming: The State of the Art and Three Case Studies of Open Source Software

Herez Moise Kattan[1(✉)], Frederico Oliveira[2], Alfredo Goldman[1], and Joseph William Yoder[3]

[1] Department of Computer Science, Institute of Mathematics and Statistics of the University of Sao Paulo (IME-USP), Sao Paulo, Sao Paulo, Brazil
{herez,gold}@ime.usp.br
[2] Samsung SIDI Institute, Campinas, Sao Paulo, Brazil
f.oliveira@samsung.com
[3] The Refactory, Inc., Urbana, IL, USA
joe@joeyoder.com

Abstract. Mob programming is a whole team technique that includes programmers and others such as product owners or testers working together in the same space and time, discussing solutions and writing code in a fast succession on a shared screen and keyboard. This paper includes a literature review and case studies of Mob Programming in software development of three open source software in an academic setting. Aspects and practices involved in the Mob Programming are analyzed. The identification of common practices can serve as standards in the Mob Programming sessions. We carried out experiments with teams practicing this technique. The bond formed among the members were the strengths of the three teams experience. The noise from work in an open room irritated two members, but two members of the same team did not get bothered and was not a problem for the remaining ten other participants. The approval of Mob Programming was unanimous in each retrospective. Providing the infrastructure to use more computers could be useful for parallel searches when a task on the Mob Programming computer takes too long, or when the team needs learn new technologies. We conclude that improved the team learning.

Keywords: Collaboration in software development · Agile practice
Programming teams · Programming technique
Software development approach · Mob programming

1 Introduction

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 [1]. Mob Programming, as Zuill [1] describes, is similar to pair programming [3], where two persons work on the same computer and collaborate on the same code at the same time.

V. A. Santos et al. (Eds.): WBMA 2017, CCIS 802, pp. 146–160, 2018.
https://doi.org/10.1007/978-3-319-73673-0_12

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 [1].

Since the popularization of Mob Programming by Zuill [1] there has been Mob sessions experiences described in some papers [1,5,11–13,15,17,18]. Mob Programming techniques also resembles the Randori [4] style of programming popular at Coding Dojos commonly used during sessions to learn new technologies and techniques [5].

This report describes common practices that could serve as standards in the Mob Programming sessions. We reviewed the most relevant Mob Programming literature and carried out experiments with teams practicing the technique. Our motivation is to find a way to develop quality software in the most productive way possible. The goal of this paper is to deepen the knowledge about Mob Programming by the conduction of case studies in an academic setting with three different teams and discover what we could learn about Mob Programming in practice.

2 Research Method and Organization of the Work

The authors have experience practicing Mob Programming [16]. The scope of the research involved a review of the literature and application examples to learn in the practice [6–8]. The literature review aimed to identify, analyze, interpret and report the relevant studies available to answer the research question "What are the practices used in the industry on Mob Programming?".

The sources of the search for this paper are IEEE Xplore (ieeexplore.ieee.org), ACM Digital Library (dl.acm.org), SpringerLink (springerlink.com), Elsevier, others websites like Agile Alliance, Google, and Portal for Periodicals of the CAPES (periodicos.capes.gov.br). The search string used was 'Mob Programming' that found thirteen studies, that were evaluated based on the inclusion and exclusion criteria described in Table 1, with emphasis on EC3, to exclude abstracts and slides. The eight accepted papers passed in all inclusion criteria and the five rejected papers did fail in either one of the exclusion criteria.

We conducted Case Studies of Mob Programming with three teams in an academic setting, trying to validate the literature findings and also to provide

Table 1. Inclusion and exclusion criteria.

Inclusion criteria	Exclusion criteria
IC1. Studies about Mob Programming, including grey literature [9]	EC1. Repeated studies
IC2. Studies applied in industry	EC2. Incomplete studies and drafts
IC3. Qualitative or quantitative research	EC3. Slides and abstracts

new insights. We looked for common aspects and practices involved. The Participant Selection process was: who had enrollment at LabXP and voluntarily did accept to collaborate with this research. Fourteen people participated, thirteen men, and only one woman. The fourteen people were members of the three teams and answered all the questions, twelve individual questions, and forty-one team questions (two questionnaires). The response rate for the survey was 100%.

The survey design split into two parts, a first about one questionnaire with individual questions, and a second with questions thought to the team, looking for examining their experience practicing Mob Programming about relevant aspects described in the literature.

The first questionnaire got fifty-six individual answers of the all fourteen participants. The team questions of the second got one hundred twenty-three answers of the three teams. Below are more details about the two questionnaires.

Individual Questions of the First Questionnaire:

- How many years have you been programming? (educational and professional experience in any language)
- How do you evaluate your current level of knowledge in the programming language used currently?
- Do you like Mob Programming? Did you Have practical experience with Mob Programming before this course?
- How would you compare Mob Programming with other practices?

The Second Questionnaire is About the Experience with Mob Programming at LAB XP Was Answered by Each Team and Had Questions Related to:

- The setup of the room and also a description of the Mob setting;
- Driver rotation;
- Retrospectives and mini-retrospectives;
- Automation of the job;
- Avoiding idle time;
- Learning as a team;
- Groupthink;
- Collective intelligence;
- Learning and mentoring;
- Continuous Improvement.

All projects are open source. The consent term, photos, two questionnaires and all answers of the fourteen members of the three teams are available online at the IME-USP CCSL Wiki [28].

The paper has six sections. The literature review is following, in the next section, and looking for aspects related to Mob Programming. In the fourth section, are the details of the case studies. After that, in the fifth section, is the analysis of the results. In the last section of this paper, we conclude and address the limitations. The acknowledgments are in the end before the references.

3 State of the Art

We review the literature of Mob Programming looking for patterns of relevant aspects and practices involved most frequently used that guided the case studies. The next subsection is the background. Subsequently, Mob Programming aspects are analyzed.

3.1 Background

Tens and hundreds of interactions between people occur every day in their work. The people express ideas, discuss problems, explore possible solutions, and share thoughts all day long. To make it possible and to keep this high level of communication happening throughout the day, the team must adopt the principle of treating each other with kindness, consideration, and respect [1].

Retrospectives are used in Mob Programming sessions [1]. In these ceremonies, the team frequently evaluates what is working for them, what problems they might be having, and how they can improve. These are usually quick and focused on one item.

A possibility to Mob Programming is adopting the Driver/Navigators pattern adapted from Llewellyn Falco's "strong" pair programming style [10]. The basic rule is for an idea to go from your head into the computer it MUST go through someone else's hands.

There are two roles: the Driver, and the Navigator. The Driver has the possession of the keyboard. The Navigators discuss the idea being coded and guide the Driver in creating the code [1].

The team adopting Mob Programming has the possibility of a timed rotation. Each team member plays the role of Driver with the possession of the keyboard for a short time. The timer remembers the current driver when can pass the possession of the keyboard to the next driver when their turn ends up [1].

Another important aspect is the configuration of the room. The room must be physically comfortable while the team members work relatively close to each other, using shared monitors, keyboards, computer setup, and programming tools. There is only one computer in use for programming [1].

A set up of the room possible is two projectors or monitors. The goal is to keep the screens at about the same size, general position, resolution, and brightness to make them comfortable to work with all day long. There are also two keyboards and two mouses (a simple one and another ergonomic one).

Everyone has the choice to suit him/herself in the better way. Each team member has its own chair which is moved around on the different roles (Driver or Navigator). Thus, the people not need constantly readjust the chair settings making each one stay as comfortable as possible [2].

Finally, the room has a rolling magnetic whiteboard to keep track the work in an informative workspace. Figure 1 illustrates a setup that worked very well and had been seen it at some companies [2]. A possibility is to have one, two, or more monitors. The readjustment of the monitor height to be a height suitable for everyone.

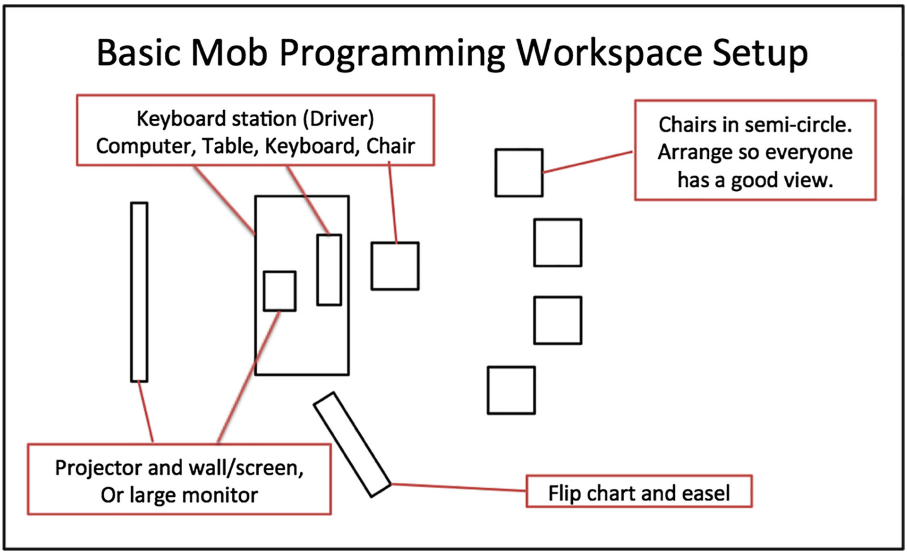

Fig. 1. A setup that has been used it at some companies [2].

3.2 Aspects and Practices Involved in Mob Programming

Here, we present the results of the literature review about aspects and practices involved in Mob Programming. Table 2 is a correlation of the selected papers for this study and all others papers. We only consider a common aspect when is cited in at least two papers.

Table 2. Practices versus papers.

Aspect or practice	[1]	[5]	[11]	[12]	[13]	[15]	[17]	[18]
Room setup	X	X	X	X	X	X		
Driver rotation	X	X	X	X	X		X	X
Retrospectives	X	X	X	X				
Automation			X	X				
Structured breaks				X	X			

Setup of the Room. The setup of the room is one of the pillars of Mob Programming [1] and the configurations are described in details by all papers [1,5,11–13,15].

According to Boekhout [12], the team found the screen too small, so it was hard to get everyone in a position near enough to it, due to all the desks. Distractions from outside were an issue that causes many interruptions. They have

Fig. 2. The setup of the room using one projector [12].

a room designed for training that was a little more isolated, with fewer interruptions, a projector and a big monitor for presentations. The team quickly discovered that the resolution of the projector was low and harmful. A high-resolution projector is a suitable solution. Then they had to use the big monitor. The team also decided to use proper office chairs, and have everybody move around keeping their chair to save time and avoid continuously fiddling with the chair configuration. Figure 2 illustrates a setup of the room using one projector.

Wilson [5] also describes an issue about the lower resolution of television or projector. The solution was to supplement paired workstation with an additional 50-in monitor that was a mirror of the workstation screen. It has been placed at right angles to the actual monitor, with the intention of the team watching the big screen and the driver can see both the screen and the rest of the team. Figure 3 illustrates this possible configuration.

Driver Rotation. In the opinion of Wilson [5], the ideal time of rotation for a Mob of 4–6 people seems to be 5 min. But, for a Mob of 3 people, ten minutes was more appropriate.

In the report of Kerney [11], a team consisting of five people rotates who has the possession of the keyboard at each fifteen minutes. But, not always means that the team must rotate when the timer is up, the team decides in each case and sometimes not use fifteen-minute rotations. Change the time of rotation in special circumstances, such as for visitors or learning sessions.

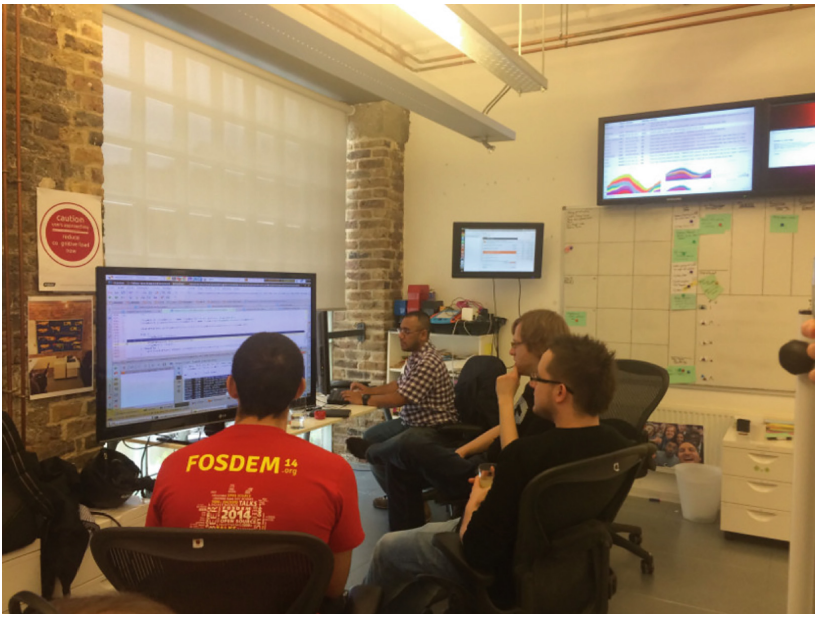

Fig. 3. Running a four people mob. The driver is in the background [5].

In the report of Boekhout [12], the team copying Woody's video [19] started with a rotation of ten minutes. Unfortunately, it seemed to them that every change of driver and navigator became an interruption and it took the team time to get back to focus on the problem at hand. There was clearly no sign of flow. So, they followed a tip that Boekhout [12] got from Llewellyn at Agile 2015 [33] and proved to be very important: lower the rotation cycle from ten to four minutes. By rotating so quickly, the switch has to go smoothly, so that you really need to make sure the workspace is good, you have a good timer and most important, that everybody is fully involved all the time. After a while, another team slowed to five minutes and declared that worked better for them too.

Retrospectives. In the view of Zuill [1], the team always looks for 'action items', and limit themselves to only one or two that they can use to 'tune and adjust' the process. They have found that having more than one or two 'action items' is almost always counter-productive. The team also set aside from half an hour to an hour to reflect on the last week or two. In these sessions, they gather information on sticky notes, do affinity groupings, dot-voting, and have conversations about the things they have observed and new things they would like to try.

There are also Just-In-Time Ad-hoc retrospectives. When anyone on the team notices something they feel we should reflect on, the team simply go ahead and

Fig. 4. An early task and retro board - notice the focus on the hourly retro [12].

do it while the experience is fresh. These are usually quick and focused on one item [14].

Kerney [11] described that the team does the retrospectives immediately when they found a problem. For example, the team realized that the projectors they had were constantly going dim, and it made the text hard to read. The immediately team tried several adjustments to fix it until they found a solution.

According to Boekhout [12], a core practice for the teams was the hourly retrospectives. An hourly retro needs to be short and to go to-the-point. The team started with a simple positive/negative items system and made sure this was visualized on their daily scrum board. Figure 4 illustrates this board.

The Fig. 4 shows the basis of the board is the horizontal axis for the hourly blocks. According to Boekhout, every hour the corresponding column is used. The top part for positive, the lower part of the improvements. The left of the board is a basic scrum board (To Do/In Progress/Done), turned on its side, where the team keeps track of the tasks about the user story of the day [12].

Automation. Kerney [11] mentioned that the team strives to make things as fluid as possible so that they do not have to break their flow. Some benefits as such as the team have an environment that is easy to maintain, and people cannot focus all the time on something that is tedious.

Structured Breaks. According to Boekhout [12] and Griffith [13], full involvement all the time can be exhausting. So the team made sure that every hour

there was a 5–10 min break after the retrospective where people weren't allowed to be behind the screen. Even then a full day can feel like a marathon [12].

4 Case Studies of Open Source Software

The programmers are attendees of LABXP, a regular course offered by the University of Sao Paulo, to graduate and undergraduate Computer Science students. First, the teams watched a lecture about Mob Programming with Alfredo Goldman and Joseph Yoder. They showed one video to the attendees about the time-lapse of a full day of work of a team of Mob Programmers [19].

The course requires a minimum of at least eight hours per week of dedication, and there is a lunch once a week, to allow the students to share experiences. The conduction of these examples of Mob Programming application in an academic setting with three different teams began in 08/08/2016 and finished on 12/12/2016. All the projects are open source. A questionnaire was applied to deepen the knowledge about the experience of the fourteen team members.

4.1 GeoXPerience: gitlab.com/geoxperience

A georeferencing platform for Casa dos Meninos, a social project [27]. It is a web georeferencing system that allows the user to create customized maps [26].

The website through a CSV file upload latitude and longitude data on a map to show the coordinates of the Basic Health Units of the City of Sao Paulo, as well as any other point on the map of interest of the user.

4.2 The Game of Life: github.com/Automata-Life

Automata.Life is a multiplayer web-based game based on JohnConway's Game of Life where several players need to fight for the survival of their single-celled species in an arena like Agar.io.

You can 'program' your cells to better try to dominate your space, or yield under the forces of Darwinism [24,25].

Figure 5 is a photo of the team of the game of life working in the programming, note that they are using a laser pointer, making easier to point an issue in the source code to be fixed. They had two projectors available and computers to parallel searches.

4.3 Mezuro: mezuro.org

Mezuro is a free/libre web platform for collaborative source code evaluation. Able to evaluate source code with the most popular SCMs (like Git and SVN), just by providing its URL.

For now, it can evaluate C, C++, Java, Ruby, Python, and PHP source codes, but the Mezuro team are looking forward to supporting more languages in the future. Mezuro is continuously under development [20–23].

Fig. 5. The team of the game of life is working in the programming, note that they are using a laser pointer, making easier to point an issue in the source code to be fixed.

4.4 Questionnaire: ccsl.ime.usp.br/wiki/SwarmQuestionnaire

Consent term, photos, two questionnaires and all answers of the fourteen members of the three teams, are available online at the CCSL Wiki of the IME-USP [28].

5 Result Analysis

Table 3 correlates the answers of the questionnaires with the aspects that were considered relevant by previous works and explanations emerged using the axial code technique of Grounded Theory.

During all process, we (the researchers) takes photos and wrote memos. These memos contain the key points observed to help us in their categorization. After, we reviewed again the literature looking for answers that would contribute to formulating a theory based on the results produced by the questionnaires and metrics. Figure 6 shows the programming experience of all the fourteen members (thirteen men and one woman).

Mob Programming reflects very well the Weinberg [32] idea of ego-less programming, where the software is owned by the team as a whole, instead of the individuals being responsible for problems with the code.

Table 3. The answers that justify the aspects and new insights.

Aspects	The answers that justify the aspect
Room setup	Sometimes, a team member was able to help a member from another group due to the shared room. The noise from work in an open room irritated two members, but two members of the same team did not get bothered and was not a problem for the remaining ten other participants. The team of Automata.Life worked with two projectors and more computers available to use if needed. The team of GeoXperience worked with one projector and two notebooks available to use. The team of Mezuro worked with one big screen television and more computers and notebooks available to use
Kindness and respect	The bond formed among the members were the strengths of the three teams experience
The Driver/Navigators Pattern	Automata.Live team used the laser pointer during some work sessions to the navigators communicate with the driver. GeoXPerience team said that the laser pointer is a good idea. Mezuro team members usually stand up to show something with the hand on television to the entire team
Automation	The team of Automata.Live automated the timer, build, test, and deploy processes. They did it because those tools helped them perform daily tasks. The Mezuro team tried to reduce wasted time by making use of our text editor's automated functionalities (Vim of Linux). Furthermore, they frequently used shell commands to speed up some processes. The GeoXPerience team automated the tests and CI and reported that was useful, and would have been more productive if they had automated more stuff
Retrospectives	They did daily mini-retrospectives, a mid-project retrospective with Professor Alfredo, and technical retrospectives with Joe Yoder, coaching all the three teams
Structured Breaks	They like to have spontaneous breaks, where the team members usually left to and came back from the breaks individually. Thus, was valued individual freedom of choice of breaks and the team continues to make progress even when individuals left for having a break. So, occurred few structured team breaks
Driver rotation	All teams developed their own timer tool, which received an integer as argument and warned them when the time was up. They enjoyed because it helped us organized the rotation, typically used 15–20 min. Automata.Live and Mezuro teams were able to use the Strong Style at some points. Automata.Live had difficulty enforcing it all the time because found it difficult to force the rotation when the current driver had an idea to solve the problem, but they were able to do that sometimes. Mezuro team used more the Strong Style with a timer to Driver rotation
Others Computers	Providing the infrastructure to use more computers could be useful for parallel searches when a task on the Mob Programming computer takes too long, or when the team needs learn new technologies
Using Mob Programming is not advantageous in doubt moments	When no one of the team knows a specific technology, it is better to do individuals searches to learn and after regroup again. Mezuro: *"There are some tasks that the team does not seem fit for Mob Programming all the times. The solving of some DevOps tasks were improved using Mob Programming, but others not so much. Especially the ones which the team did not know what to do for sure and the approach was trial and error"*
Sharing knowledge	The capacity to distribute the knowledge through a streamlined method and working with friends, we observed that increased the learning
Developers approved and enjoyed	The fourteen members of the three teams approved and enjoyed. Mezuro: *"Yes. The approval of Mob Programming was unanimous at every retrospective"*

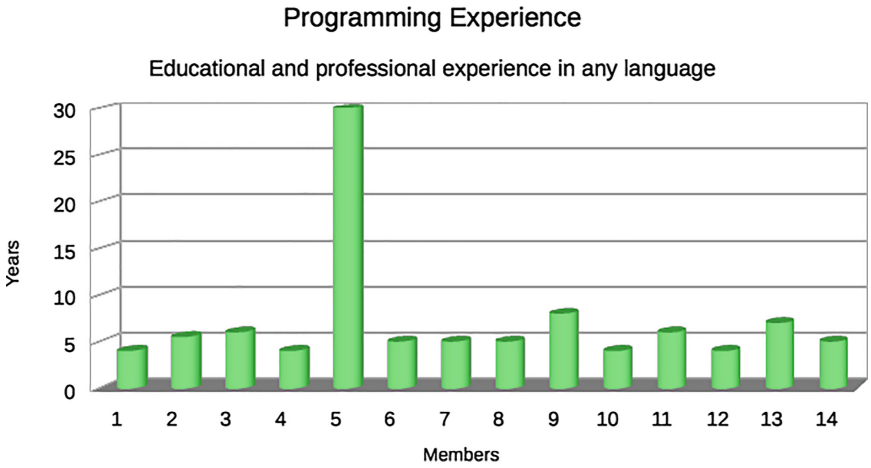

Fig. 6. Programming experience of team members.

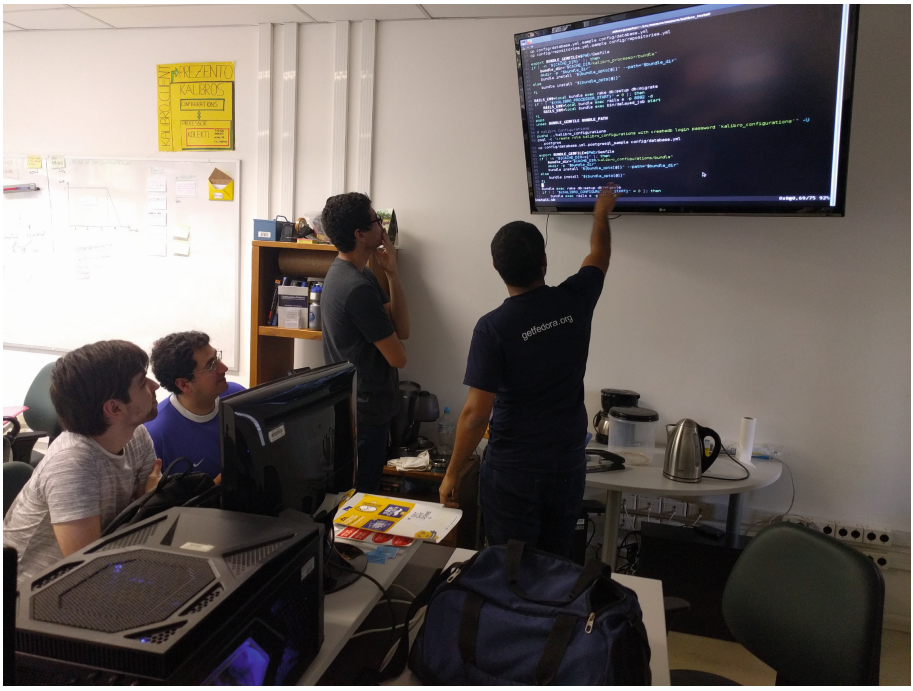

Fig. 7. One member of the team is pointing with the hands an issue in the source code.

We agree that different types of collaboration are suited to different kinds of problems. If the particular problem stem from the team not having a shared understanding of the project, Mob Programming that involve the entire team

is a great approach [1,5,11–13,15,16,29]. Western Electric made experiments attempted to determine the relationship between light levels and worker efficiency. The data compiled by the Illumination Experiments indicated only a minor correlation between light levels and worker productivity [30]. However, one interesting observation is sometimes when increasing the light level, the productivity grows up and when decrease the light level the productivity grow up too. Thus, the light experiment showed that **increased interest of the workers** increased productivity. Mob Programming increase the satisfaction, bond among team members, and learning of the software developer. These are factors of productivity that could justify the use of Mob Programming.

Mob Programming in an open work-space environment increases the Integration Among Teams in the LABXP course, that is one relevant factor to Inter-team Knowledge Sharing and open work-space environment provide great value, but software developers have to deal with the problem of noise [31].

Overall, there was a very positive experience to all teams. Figure 7 is showing the team of the Mezuro working in the programming. In the photo, one member of the team is pointing with the hands, an issue in the source code. A laser-pointer seems to be a good idea.

6 Conclusion and Limitations

We confirm the literature reviewed about the advantages of the use of Mob Programming over other techniques to sharing knowledge, learning, and satisfaction of the programmer. We also confirm that sometimes it is better to alternate with other techniques like pair programming. Additionally, we noted that collaboration among teams is improved and the bond formed among the team members.

Our answers to questionnaire did not confirm the literature that most programmers have practically non-existent moments of frustration in a Mob Programming session. We observed that in moments when nobody of the team knows a tool/framework was exhaustive to the whole group to approach these situations. When no one of the team knows a specific technology, it is better to do individuals searches to learn and after regroup again.

Limitations. The case studies performed are a qualitative research strategy and not permit generalizations, but all our data (source code hosted at Gitlab or Github, metrics, and questionnaires) are open source, thus could be audited. Our questionnaire is very extensive because we tried to confirm all details observed, these data are auditable observations available publicly in the Wiki.

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

References

1. Zuill, W.: Mob Programming: A Whole Team Approach. Experience report, Agile (2014). https://www.agilealliance.org/resources/experience-reports/mob-programming-whole-team-approach-woody-zuill/
2. Zuill, W., Meadows, K.: Mob Programming - A Whole Team Approach. This book is 95% complete (2016). http://leanpub.com/mobprogramming. Last Updated on 29 Oct 2016
3. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley, Boston (2004). 75 p
4. Rooksby, J., Hunt, J., Wang, X.: The theory and practice of randori coding dojos. In: Cantone, G., Marchesi, M. (eds.) XP 2014. LNBIP, vol. 179, pp. 251–259. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06862-6_18
5. Wilson, A.: Mob programming - what works, what doesn't. In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) XP 2015. LNBIP, vol. 212, pp. 319–325. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18612-2_33
6. Kattan, H.M.: Illuminated arrow: a research method to software engineering based on action research, systematic review and grounded theory. In: CONTECSI 2016, 13th International Conference on Information Systems and Technology Management, Paper submission: 1 Dec 2015 - Presented at Session4A - AUD Systems Auditing and IT Governance 02 Jun 2016-15H30, pp. 1971–1978 (2016). https://doi.org/10.5748/9788599693124-13CONTECSI/PS-3926
7. Kattan, H.M.: Those who fail to learn from history are doomed to repeat it. In: Agile Processes in Software Engineering and Extreme Programming: Poster Presented at the 18th International Conference on Agile Software Development, XP 2017, Held in Cologne, Germany, 22–26 May 2017 (2017). https://doi.org/10.13140/RG.2.2.20864.02563
8. Moise Kattan, H., Goldman, A.: Software development practices patterns. In: Baumeister, H., Lichter, H., Riebisch, M. (eds.) XP 2017. LNBIP, vol. 283, pp. 298–303. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57633-6_23
9. Schöpfel, J.: Towards a prague definition of grey literature. In: Proceedings of the Twelfth International Conference on Grey Literature: Transparency in Grey Literature. Grey Tech Approaches to High Tech Issues, Prague, December 6–7 (2010)
10. Falco, L.: Llewellyn's strong-style pairing (2014). http://llewellynfalco.blogspot.com.br/2014/06/llewellyns-strong-style-pairing.html
11. Kerney, J.: Mob Programming - My first team. Experience report, via Initiative of Agile Alliance (2016)
12. Boekhout, K.: Mob programming: find fun faster. In: Sharp, H., Hall, T. (eds.) XP 2016. LNBIP, vol. 251, pp. 185–192. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33515-5_15
13. Griffith, A.: Mob Programming for the Introverted. Experience report, Agile (2016)
14. Arsenovski, D.: Swarm: beyond pair, beyond Scrum. Experience report, Agile (2016)
15. Hohman, M., Slocum, A.: Mob Programming and the Transition to XP (2001)
16. Kattan, H.M.: Programming and review simultaneous in pairs: a pair programming extension. Master dissertation, Institute for Technological Research of the Sao Paulo State (2015). http://aleph.ipt.br/F or http://ipt.br, click on: Online Consultations, then click on: Library. https://doi.org/10.13140/RG.2.2.15831.68004

17. Lilienthal, C.: From pair programming to mob programming to mob architecting. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2017. LNBIP, vol. 269, pp. 3–12. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49421-0_1
18. Balijepally V., Chaudhry S., Nerur S.: Mob programming - a promising innovation in the agile toolkit. In: Twenty-third Americas Conference on Information Systems, Boston (2017)
19. Zuill, W.: A Day of Mob Programming (2012). https://www.youtube.com/watch?v=p_pvslS4gEI
20. Mezuro Wiki. http://ccsl.ime.usp.br/wiki/Mezuro
21. Mezuro. http://Mezuro.org
22. Mezuro GitLab. http://gitlab.com/mezuro
23. Mezuro GitHub. http://github.com/mezuro
24. The Game of Life Wiki. http://ccsl.ime.usp.br/wiki/Automata.Life
25. The Game of Life GitHub. http://github.com/Automata-Life
26. GeoXP: http://ccsl.ime.usp.br/wiki/Sistema_online_de_georreferenciamento
27. GeoXP GitLab. http://gitlab.com/geoxperience
28. Questionnaire. http://ccsl.ime.usp.br/wiki/SwarmQuestionnaire
29. GDS post. http://gds.blog.gov.uk/2016/09/01/using-mob-programming-to-solve-a-problem/
30. Western Electric Company Hawthorne Studies Collection, Baker Library, Harvard Business School. http://oasis.lib.harvard.edu/oasis/deliver/~bak00047
31. Santos, V., Goldman, A., Souza, C.: Fostering effective inter-team knowledge sharing in agile software development. Empir. Softw. Eng. **20**, 1006–1051 (2015)
32. Weinberg, G.: The Psychology of Computer Programming. Van Nostrand, New York (1971)
33. Falco, L.: Group Learning. Today's exercise: Unit Testing. Session at Agile (2015)