Software Development Practices Patterns: from Pair to Mob Programming

Herez Moise Kattan[1]

[1]Department of Computer Science - University of Sao Paulo (IME-USP), Sao Paulo, Sao Paulo, Brazil

herez@ime.usp.br

Abstract. *Software development is a social activity. A positive aspect of techniques of software development like Pair Programming and Mob Programming, as collaborative approaches empathize those social aspects. Some benefits are, e.g., an increase in the technical level of the team, related with the dissemination of knowledge, collective responsibility, a cutback of the number of system defects, an increase of satisfaction, of motivation, on the collaboration, in the communication, and trust among the team members. This paper reviews the literature of Pair Programming and Mob Programming cataloging the variations of pairing and mobbing for developers with different expertise levels on distinct tasks with diverse complexity levels grouped by context.*

1. Introduction

Products developed by pair programming have generally a higher quality compared with ones developed by solo programming. However, a literature review of pair programming revealed some drawbacks, such as a possible loss of productivity when using the technique, especially for experienced programmers and simple tasks [Kattan et al. 2018b] [Kattan et al. 2018c]. It is observed that when using pair programming, there might be an increase in the effort in the project [Nosek 1998].

Loss of productivity is a common criticism of pair programming since it devotes two professionals to perform the same job simultaneously on a single computer. There is a constant debate in academic and professional circles if the technical advantages justify this possible loss of productivity [Allen Parrish and Hale 2004] [Kattan 2015] [Kattan and Goldman 2017] [Kattan 2018] [Kattan et al. 2018b] [Kattan et al. 2018c].

There are related techniques, such as mob programming, where the whole team participates in around a computer [Kattan et al. 2018a]. Also, there is code review, another well know and applied alternative, which might increase productivity compared to pair programming [Rajendran Swamidurai and Kannan 2014]. So, this is the motivation of to do a more broad study in the context of these related techniques that allow collaboration. It is reviewed the literature about Pair Programming and related techniques. Through this review, It is identified Software Development Practices Patterns related to these collaborative software development techniques.

Following is the literature review including the protocol, the descriptions of the Software Development Practices Patterns and it is finalized describing the limitations and conclusion.

2. Literature Review

2.1. Protocol

It is reviewed the literature on Pair Programming and Mob Programming looking for reliability, avoiding trends in papers selection or interpretation bias made when summarizing them [Kattan 2016] [Kattan 2017].

The sources of academic studies for this paper are IEEE Xplore (ieeexplore.ieee.org), ACM Digital Library (dl.acm.org) and SpringerLink (springerlink.com). As Mob Programming is still an emerging software development technique, early adopters publish experiences reports, for this reason, the inclusion criteria only to Mob Programming accept gray literature (not controlled by commercial publishers), although sometimes peer-reviewed, are only experiences reports, possibly without a rigorous research method. The search string used was 'Pair Programming' or 'Mob Programming'. The complete reading of all studies was performed. To try to avoid bias, the author performed an entire reading of the literature and of the interpretations four times.

Based on this literature review It is proposing a catalog of software development practices patterns, it is based on preliminary finding and heuristics to help software development teams in the choice of a more appropriate approach for their job context.

2.2. Pair Programming

Although it has been known as a technique associated with the eXtreme Programming method, the creation of pair programming precedes the XP method. There are many old tech-savvy reports, such as [Brooks 1975] 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.

[Coplien and Harrison 2004] defend the advantages of making pairing compatible designs work together. In this way, they can produce more than the sum of the two individually.

2.2.1. Definition

A definition of pair programming is: two programmers working collaboratively at the same activity using a single computer. While one person is programming, the partner is observing with attention looking for defects or suggesting improvements [Begel and Nagappan 2008]. There are similar approaches [Al-Jarrah and Pontelli 2016], where two monitors and mouses are used, but there is no conceptual difference.

2.2.2. Dissemination of knowledge, quality, and reduction of defects

To help to spread the knowledge among the team pair rotation was also introduced. In this technique, a balance among the pairs of each individual is reached. As an important side effect I also have the sharing of skills and competencies among team members [Williams and Kessler 2000] [Cockburn and Williams 2001] [Nosek 1998].

[Constantine 1995] has published about the reduction on the number of defects when programming in pairs. Nosek [Nosek 1998] 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.

[di Bella et al. 2013] have published a case study, conducted to evaluate the effect of pair programming on the quality and efficiency of defect corrections. A software development project was studied at a large Italian company for a period of fourteen months. Compared with existing case studies of pair programming, the longer period makes the study more realistic. In an exploratory analysis, the effectiveness of the pair programming was investigated in the context of defect corrections and implementations of the user stories. The analysis showed that the introduction of new defects tends to decrease when pair programming is used. The results are consistent for both contexts.

2.2.3. Informal review, inspection process and communication

The use of the pair programming technique acts as an informal review process because each line of code is read by at least two people. Code inspections help a lot in finding a high percentage of software errors. However, they are time-consuming to organize and often represent delays in the development process [Williams et al. 2000].

For [Cockburn and Williams 2000] [Cockburn and Williams 2001] pair programming is a less formal review process and probably does not encounter as many errors as code inspections, but it is much cheaper than a formal inspection process.

[Williams et al. 2000] 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, so there will probably be fewer errors and less rework. In addition, the number of errors avoided by informal inspection is such that less time is spent repairing discovered bugs during the testing process.

[Du et al. 2015] performed a basic programming experiment using the C programming language. The results are analyzed through interviews after the experiment and the conclusion is that pair programming is effective in improving communication in a basic exercise using the C language.

2.3. Simultaneous Style Pair Programming

A possibility to increase the productivity of pair programming is the use of parallelism and multidisciplinary teams of Concurrent Engineering of [Pithon 2004]. Another alternative is the incorporation of a process of pair code review, and a stage for projecting pairing simultaneous development. In the pairing project, whether or not adopting pair/mob programming is defined collaboratively and if it is adopted, define in which software requirements each programming technique will be used [Kattan 2019b].

As [Coplien and Harrison 2004] argue, the design is adopted compatible with the pairing of working together. In this way, they can produce more than the sum of the two individually. The selection of pairs depends on the project, the task to be performed, the availability of the team members, the need to disseminate knowledge, the pair rotation

and the experience of each one. When incorporating Concurrent Engineering practices into pair programming, one should note the importance of communication, even if the activity is performed in different physical dependencies. It is also important to stress that the more the activity is developed simultaneously, the greater the productivity gain.

Communication and collaboration are the pillars to join the work performed simultaneously on different computers successfully, without compromising with mistakes during the joining of the work, the productivity gain obtained by the parallelism during its execution according to [Pithon 2004].

Developers work in pairs to accomplish their tasks to promotes collective, collaborative work, uniting the team, improving communication and the quality of the code. The work is developed simultaneously and if there is more than one team pair, the iterations are designed to be simultaneous.

The Programming and Review Simultaneous in Pairs (PRSP) of [Kattan 2015] [Kattan et al. 2018b] [Kattan et al. 2018c] is also known as Simultaneous Style Pair Programming or as pair development. PRSP has the following definition:

"A programming activity wherein planning is at the beginning including the pair selection, the pairing of tasks is collaboratively designed and based on this two programmers work collaboratively in the same activity. Only in the beginning of one activity sitting side by side to exchange experiences (this way there are more algorithms and solutions) or communicate in the beginning, if they are working in a distributed way (different locations). Still, in this initial phase, they decide how to divide the task, and do not need to sit together all the time on a single computer, or communicate at all times if they are working in a distributed way, only when necessary and useful. Whenever possible, the work should be performed simultaneously on separate computers. Unlike traditional pair programming, in PrsP each programmer revises the work of the other one simultaneously using two computers if an error is found then the task returns to the programmer fix it. In the end, they unite the work of the pair. During the rest, it is suggested to speak or to think about the best way for the accomplishment of the work, mindset zero defect, adoption of a process of stress reduction and for resolution of conflicts".

2.4. Mob Programming

The idea of mob programming originated from technical meetings where a team member presented a code he knew. The group could work together, exchange information on design, architecture and programming techniques, encourage change in the code and provide feedback [Hohman and Slocum].

The main difference, comparing to pair programming, is that the whole team works together as part of the pairing. In addition to software coding, Mob Programming teams works 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 [Kattan et al. 2018a] [Kattan 2019a].

Mob Programming can be seen as an evolution of the learning environment of a Dojo Setting to a production environment [Bravo and Goldman 2010]. The whole team stays around a single workstation, but with only one member having the possession of the keyboard and mouse, to actually modify the code. The coder is changed every 5 minutes,

thus, with a group of 4-6 people, a member must wait 15-25 minutes to use the keyboard again [Zuill 2014].

Among the difficulties in adopting the mob programming, Hohman and Slocum [Hohman and Slocum] report that it is difficult to keep in focus, especially for those who are not with keyboard possession. Wilson [Wilson 2015] shows the concern of managers about the difficulty of having the whole team working together in a single requirement, while is more productive and effective to have peers working on different tasks. On the other hand, Zuill [Zuill and Meadows 2016] highlights the benefits for communication, collaboration and team alignment.

Concerning weaknesses, Hohman and Slocum [Hohman and Slocum] report that it is difficult keep focus, especially for those who are not in the possession of the keyboard. Wilson [Wilson 2015] cites two problems, the effect of dominant personalities within the Group and appearance of Groupthink.

Among the strengths Zuill [Zuill 2014][Zuill and Meadows 2016] emphasizes that it strengthens communication, team alignment, collaboration and self-organized. Wilson [Wilson 2015] note that the dissemination of knowledge was stimulated and the technique worked well for critical codes and complex.

There are two points of convergence in the conclusion of Hohman and Slocum [Hohman and Slocum] and Wilson [Wilson 2015], when using two projectors or monitors, the technique works best and it is advantageous to switch the use of group programming with other techniques, if the particular problem stem from the team not having a shared understanding of the project, using a collaborative method that involve the entire team is a great approach [Zuill 2014] [Wilson 2015] [Hohman and Slocum] [Lilienthal 2017].

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 [Balijepally et al. 2017].

3. Software Development Practices Patterns grouped by context

Based on the literature review it is proposed the following Software Development Practices Patterns.

3.1. Pair Coaching

When a company grows, it is normal to hire new collaborators or when one collaborator leaves the team, frequently is necessary to hire a new one. Another usual situation is when a software developer goes to work in a different department of the company or goes to work in an unknown part of the system. The inclusion of a new member in the team that does not know the system creates the problem of the first contact with the code.

How to reduce the time for a team member to get familiar with the application code and software architecture of a part of the system that he or she did not work yet?

By the use of pair coaching that is one more senior member of the team in training the new member. It is not necessary to be someone more experienced, but with familiarity

with the code. In this case, the choice of who is more appropriate to work in pair with the new member depends on availability and familiarity with the code of the team members. It is important to give developers freedom of choice on how they conduct the work. When she or he talk that is ready, the team gets conscience that the new member already has the autonomy to new challenges.

Typically, the more senior member starts being the pilot to show how the system really is. When the new member feels confident is appropriate do the rotation to she or he starts being the pilot with the keyboard possession.

Training the novice of the team is one positive consequence, his or her integration with the whole team is another, but it is necessary to have available one senior professional to help the newcomer. Training one novice is an investment of work effort because the senior could work in another more complicated task, but this transfer of knowledge occurs when the pair is working on something productive.

3.2. Metrics and Practices to share knowledge

In a professional environment, usually, directors and managers say that, if there are only one professional with knowledge of something, so there are not any, because unexpected problems occur and sometimes obstructs the only one professional to collaborate, impeding the team makes delivery the software with quality. The problem typically is how to know the highest probability of changes in specific system parts that only one professional has knowledge.

There are metrics available nowadays as open-source software tools and patented that are useful in this situation. Mob Programming and Coding Dojo are a possibility to detect points where only one Professional has knowledge.

Metrics to the complexity of code, crusades with who is updating the source code repository is a possibility to guide the pair selection in a more efficient way for transfer knowledge. Another possibility is collaboratively in a Mob Programming detect who exclusively knows a specific part of the system and immediately share this knowledge.

Collaboratively, using metrics or Mob Programming to the detecting of lacks in team knowledge about only one professional with knowledge in a specific system part, carefully are chosen the pairs.

A positive consequence is to avoid bad intend of a professional in the creation of dependency of him or her through getting exclusively knowledge of a system part. Typically, the professionals who looking for creating dependency, when is created an adequate work environment, they admit transferring the knowledge with benefits of enjoying more the weekends and holidays.

3.3. Cognitive Learning

When the pair is composed by one Senior member working with one Junior. The Senior usually is not learning very much, only teaching. The Junior learn much more but sometimes only watch. Considering learning outcomes how a factor of productivity, only the Junior is learning.

The senior developer verbalizes his teaching and the novice being supported by the senior when he is developing a task are teaching methods used in cognitive learning.

The suggestion is the Junior with the possession of the keyboard (driver/pilot) and the Senior not only being a simple navigator/co-pilot but teaching and verbalizing. Thus, the Junior is programming with the possession of keyboard and listening to the teachings of Senior.

While seniors make an additional effort to transfer knowledge to the newbie by raising awareness about knowledge transfer practices, it generates a learning opportunity for the senior. A negative consequence could be a decrease in productivity in the short term. The learning could be motivational and bring satisfaction to the programmers, increasing the productivity of the team as a whole. This heuristic too could be useful to reduce the rate at which developers leave the team, yielding the next heuristic.

3.4. Reducing the rate at which developers leave the team

[Williams and Kessler 2003] *apud* [Weinberg 1971] 'If a programmer is indispensable, get rid of him as quickly as possible'. Although this contextualization works very well to talk about avoiding build a house of cards, we are talking about an important productivity factor: the problem of if your team has a 'key' person and this person leave the team. We hope you could say 'Houston, We've Had a Problem and we're looking at it through this heuristics'. The problem is to retain talented people.

The incentive to the Programmers to adopted the technique of **Pair Programming** or **Mob Programming** ideally needs have the support of the president. Sharing and recognizing competencies must be appreciated by the organization culture. Fighting against bad programming practices too. Creating an environment-friendly to the conversation between programmers.

Change the culture of the organization to **collaboration** could stimulate the programmers to use it and get the motivation to stay working to your company.

A very positive consequence is the retaining of the top talent. In the long term increase in productivity and in the short term could increase the cost and maybe decrease the productivity are negative consequences.

3.5. Sharing operational and domain problem knowledge

When you get in a taxi, you need to inform the driver where you wanna go if you intend to go to the location that you need. Agile is about adaptation on changes, not about developer without domain knowledge, the customer always present remember? Developers usually know more about programming and technical tasks than business domain problem knowledge. Thus, developers need learning about operational and business domain problem.

The solution suggest is have a planning phase to collaboratively detect the points at which are need sharing operational and domain problem knowledge. After, work using the Mob Programming.

An example is if in the planning phase is detected some tasks could be useful to share operational and domain problem knowledge, the decision made by the team is using Mob Programming. But, to the other's tasks could use other Programming Technique.

A positive consequence is in the long term this learning of domain problem and operational knowledge help to improve the productivity of the organization. A negative

aspect could be the cost to have rooms and projectors adequate to Mob Programming.

3.6. Programming of complex and critical code

Metaphorically, in determinant situations, the cops call reinforcements. Sometimes, developers need help to review code or to develop a complex part of the software system.

The solution is to work using the Pair Programming, Simultaneous Style Pair Programming or Mob Programming.

An example is in the planning phase was to detect the complex and critical parts, facilitating the choice of Programming Technique. Another example is the team chose work using Pair Programming or Simultaneous Style Pair Programming but unexpectedly happens a critical problem at a complex part of the code of the software, immediately the team decides use Mob Programming and after come back to work in pairs.

The positive consequence is taking all the advantage of working in pairs and mob programming together. The negative could be the cost to have rooms and projectors adequate to Mob Programming.

4. Limitations and Conclusion

The limitation founded in the literature reviewed of Simultaneous Style Pair Programming is the reduced number of case studies, only eight involving fifty-five developers and in that empirical part, it was not possible to make all possible combinations regarding the level of developer experience with the complexity of the task. The limitation founded in the publications related to Mob programming is the lack of an empirical validation using research methods because the major part of the literature is composed of experience reports.

The main conclusion is Mob Programming could be an advantage technique for complex and critical codes and works combine with Pair Programming and Simultaneous Style Pair Programming. The secondary contributions are software development practices patterns describing a context which could be advantage use each approach.

Referências

Al-Jarrah, A. and Pontelli, E. (2016). *On the Effectiveness of a Collaborative Virtual Pair-Programming Environment*, pages 583–595. Springer International Publishing.

Allen Parrish, Randy Smith, D. H. and Hale, J. (2004). A field study of developer pairs: productivity impacts and implications. *IEEE Software*, 21(5):76 – 79.

Balijepally, V., Chaudhry, S., and Nerur, S. P. (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*.

Begel, A. and Nagappan, N. (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, pages 120 – 128, New York, NY, USA. ACM.

Bravo, M. and Goldman, A. (2010). *Reinforcing the Learning of Agile Practices Using Coding Dojos*, pages 379–380. Springer Berlin Heidelberg, Berlin, Heidelberg.

Brooks, Jr., F. P. (1975). The mythical man-month. *SIGPLAN Not.*, 10(6):193–.

Cockburn, A. and Williams, L. (2000). The costs and benefits of pair programming. In *In eXtreme Programming and Flexible Processes in Software Engineering XP2000*, pages 223–247. Addison-Wesley.

Cockburn, A. and Williams, L. (2001). Extreme programming examined. chapter The Costs and Benefits of Pair Programming, pages 223–243. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Constantine, L. L. (1995). *Constantine on Peopleware.* Yourdon Press Computing Series. Prentice Hall Ptr, Englewood Cliffs, N.J., 1 edition edition. Paperback: 219 pages.

Coplien, J. O. and Harrison, N. B. (2004). *Organizational Patterns of Agile Software Development*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

di Bella, E., Fronza, I., Phaphoom, N., Sillitti, A., Succi, G., and Vlasenko, J. (2013). Pair programming and software defects–a large, industrial case study. *IEEE Transactions on Software Engineering*, 39(7):930–953.

Du, W., Ozeki, M., Nomiya, H., Murata, K., and Araki, M. (2015). Pair programming for enhancing communication in the fundamental c language exercise. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 3, pages 664–665.

Hohman, M. M. and Slocum, A. C. Mob Programming and the Transition to XP. Technical report.

Kattan, H. M. (2015). Programming and review simultaneous in pairs: a pair programming extension. Master's thesis, Institute for Technological Research of the Sao Paulo State, Brazil.

Kattan, H. M. (2016). Illuminated arrow: a research method to software engineering based on action research, systematic review and grounded theory. In *13th International Conference on Information Systems and Technology Management.* Paper submission in 1 Dec 2015. Presented at Session4A. Pages: 1971-1978. Sao Paulo, SP, Brazil.

Kattan, H. M. (2017). 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, in May 22-26*.

Kattan, H. M. (2018). Theory of altruism on software development practices patterns. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, pages 44:1–44:4, New York, NY, USA. ACM.

Kattan, H. M. (2019a). Mob programming and simultaneous style pair programming in the development of a battle royale game: an action research. In *Agile Methods*, Cham. Springer International Publishing.

Kattan, H. M. (2019b). Pair programming: a step beyond. In *Agile Methods*, Cham. Springer International Publishing.

Kattan, H. M. and Goldman, A. (2017). Software development practices patterns. In Baumeister, H., Lichter, H., and Riebisch, M., editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 298–303. Springer International Publishing.

Kattan, H. M., Oliveira, F., Goldman, A., and Yoder, J. W. (2018a). Mob programming: The state of the art and three case studies of open source software. In Santos, V. A. d., Pinto, G. H. L., and Serra Seca Neto, A. G., editors, *Agile Methods*, pages 146–160, Cham. Springer International Publishing.

Kattan, H. M., Soares, F., Goldman, A., Deboni, E., and Guerra, E. (2018b). Swarm or pair? strengths and weaknesses of pair programming and mob programming). Poster - DOI: 10.13140/RG.2.2.18105.06249, XP'18, Porto, Portugal. May 21-25.

Kattan, H. M., Soares, F., Goldman, A., Deboni, E., and Guerra, E. (2018c). Swarm or pair?: Strengths and weaknesses of pair programming and mob programming. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, pages 43:1–43:4, New York, NY, USA. ACM.

Lilienthal, C. (2017). *From Pair Programming to Mob Programming to Mob Architecting*, pages 3–12. Springer International Publishing, Cham.

Nosek, J. T. (1998). The case for collaborative programming. *Commun. ACM*, 41(3):105–108.

Pithon, A. J. C. (2004). *Projeto organizacional para a engenharia concorrente no ambito das empresas virtuais*. Doctoral thesis, Escola de Engenharia da Universidade do Minho Departamento de Produção e Sistemas, Portugal.

Rajendran Swamidurai, B. D. and Kannan, U. (2014). Investigating the impact of peer code review and pair programming on test-driven development. In *IEEE SOUTHEASTCON 2014*, SOUTHEASTCON '14, Conference Location: Lexington, KY, USA. IEEE.

Weinberg, G. (1971). *The Psychology of Computer Programming*. Wiley - Van Nostrand, New York, NY.

Williams, L. and Kessler, R. (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, pages 59–65, Washington, DC, USA. IEEE Computer Society.

Williams, L. and Kessler, R. (2003). *Pair programming illuminated*. Pearson Education, Boston, MA.

Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Softw.*, 17(4):19–25.

Wilson, A. (2015). *Mob Programming - What Works, What Doesn't*, pages 319–325. Springer International Publishing, Cham.

Zuill, W. (2014). Mob programming: A whole team approach. In *Experience report*, Agile '14.

Zuill, W. and Meadows, K. (2016). *Mob Programming. A Whole Team Approach*. LeanPub, this book is 95% complete - last updated on 2016-10-29 edition.