Mob Programming: Find Fun Faster

Karel Boekhout(✉)

Haarlem, The Netherlands
karel@boekhout.org

Abstract. The Mob Programming technique proves to be an effective learning instrument with a group of less experienced developers. It is also used to explore topics outside of just software development.

This paper describes how, with a set of weekly Mob Programming sessions, the teams as a whole and all its individuals have grown much faster than they could have done otherwise. They improved their coding skills, mastery of tools, involvement in Scrum ceremonies, estimation skills, process modeling (!) and learned to be much more self-sufficient.

This didn't happen without plenty of experimentation, and some dead ends. I will describe the different approaches we tried, how we ended up with a surprisingly strict process for our mobbing sessions, and how acceptance was easier with a team that had fewer ingrained habits of work.

Keywords: Learning · Pairing · Mob Programming · Discovering unknown territories

1 Introduction

It started with two teams that needed to improve their skills in many different areas but with very little support available to get them there. No seniors, no training, and a single, non-technical, coach trying to help them.

Having been impressed by the Mob Programming [3] session at XP2015 in Helsinki, I started an experiment to see whether that technique could help us accelerate the learning process. We tried Mob Programming for a period of two months, with one full day of mobbing a week. It proved to be a great experience for both the teams as well as for this coach, albeit one with a steep learning curve.

In this experience report, I show the effects mobbing had on different aspects of our work. How the adoption of the practice was different between two differently structured teams. What we did to make it work, and how mobbing was particularly effective in supporting learning and discovery in both technical skills as team maturity.

2 Situation

This experiment happened in a small department of a company (consisting of 18 people, 14 developers in 3 teams) based in Rijswijk, The Netherlands. The main period of the experiment was in the summer of 2015, fresh after the inspiration from the XP2015 conference in Helsinki.

H. Sharp and T. Hall (Eds.): XP 2016, LNBIP 251, pp. 185–192, 2016.
DOI: 10.1007/978-3-319-33515-5_15

The experiment involved two teams within the company: a junior team (let's call them team Red), and a less junior team of developers (team Yellow). The junior team, the Reds, consisted entirely of young programmers with no previous work experience as developers. There was no senior developer available to guide them, and we needed some way to accelerate their learning process. The Yellow team was also very junior but had some programming experience, some from working at another company.

This isn't a group who by themselves who would scout the outer limits of IT innovation, so I didn't expect them to be eager to try Mob Programming out. The reception was not even lukewarm; I would have to earn my pay to get this accepted.

3 Introduction of Mob Programming to Teams

I started out by showing Woody Zuill's video 'A Day of Mob Programming' [4] to the whole group. I explained that even I, a non-developer, had had a lot of fun participating in a session of Llewellyn Falco and that I would like to try it with the teams.

I told them I thought it would be fun, and mentioned similarities I saw with some of the online games that I knew the junior team members liked playing in their own time. These games are high paced shoot-'m-ups, in which they acted as a team with a lot of online communication, and I hoped the similarities would spark an interest.

A second argument was, via feedback in the retrospectives the teams had raised, that there was lot of difference in the skill level and use of tools between the team members, regardless of seniority and time with the company. Everyone agreed that it would be in the interest of the teams, the individuals and the company to spread those skills more evenly. I emphasized how Mob Programming could help us achieve this.

Still, the first reactions were lukewarm. Even while stressing that the goal was learning and not delivery, people were complaining about the apparent lack of efficiency. A senior colleague, who was not part of these teams but influential in the company, openly said he would dislike working on a daily basis as was shown in the video. I stressed that if people liked this way of working we could do it more often, but that it was not my goal to make this the new default way of working in the office. In the end, I decided to just try.

4 Experiments

As part of the series of Mob Programming sessions, we continuously adapted our way of working. When you're doing hourly retrospectives, the rate of change can be very high. In the following Sect. 1 describe some of the larger, and more important, changes that happened, and the process we ended up with.

4.1 Room Setup

We went through a few iterations before we arrived at a room setup that worked for us. Starting by just using the big screen we use for giving demo's in the main team room,

we quickly found that the screen was too small, and it was hard to get everyone a position near enough to it, due to all the desks. Distractions were also an issue, with many interruptions from outside the team, and simply from people's workstations (see Fig. 1).

Fig. 1. First try: mobbing in the team room

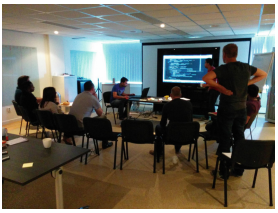

Fig. 2. Second attempt: training room in conference-inspired setting

Fig. 3. The final setup: comfort and proper lighting

At the company we have a room designed for training. In this room we were somewhat more isolated from interruptions. The room has a projector and a big monitor for presentations. We quickly discovered that the resolution of the projector was too low and not clear enough. A good high resolution projector is the most optimal solution. We had to use the big monitor.

The setup in the training room initially resembled the setup used in the XP2015 conference setting that I participated in. We rearranged the tables to create one central table, directly in front of the monitor, with a keyboard, mouse and one laptop. We arranged some chairs in a semicircle around that driver position, and simply switched places (see Fig. 2).

We still found issues with this setup. The simple chairs, though good enough for a short mobbing session at a conference, wouldn't do for sitting on all day. So we 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 (see Fig. 3).

Lighting also turned out to be important; if there's too much, it makes it hard to read the screen. But when it's too dark the bright screen strains the eyes too much. By switching to a side-wall for the screen we could alleviate that particular issue.

4.2 Cycle Time

Copying Woody's video, we started with a rotation of 10 min. Unfortunately, it seemed that every change of driver and navigator became an interruption and it took the team time to get back in focus on the problem at hand. There was clearly no sign of flow.

A tip I got from Llewellyn Franco at Agile 2015 [1] proved to be very important: lower the rotation cycle from 10 to 4 min. 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, Team Red slowed to 5 min, and declared this the sweet spot for them.

4.3 Structured Breaks

Full involvement all the time can be exhausting. It's worth noting that more mature teams might experience Mob Programming less stressing than Pair Programming, due to breaks in the rotation. However, in this situation the primary goal was not delivering software, but focus was on learning though training.

So I 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. The biggest advantage of a full day, was that you can do a full Sprint and finish work in one day, which people found fulfilling.

4.4 A Sprint a Day Keeps the Coach Away

I emphasized that the Mob Programming days were an experiment with the focus on learning, rather than delivering. Apparently I was a little bit too effective with the emphasis on learning and creating an environment of not delivering.

The sessions were not always happening with the full attention of everyone in the team. The results were incomplete and would bleed over into additional work outside of the mob, and inside of the containing, normal sprint. Reflecting on this, we decided to put a little more focus on the mobbing day, and have a clear goal of taking a small story and having it deployed to production. I called this the 'sprint in a day', and it did put the whole process in a pressure cooker.

The structure of that single-day-sprint was as follows:

- the team picks a user story during the planning session, taking into consideration that it had to be possible to finish the story in one day,
- the day starts with a tasking session, where the team does a breakdown of all the tasks needed to deliver the user story,
- then hourly cycles of development, which each ended in a retro and break,
- at the end of the day, the user story would be deployed to production
- the retro for the last hourly cycle is extended, looking back at the last hour and the whole day,
- the day is closed by making the retro report together as a mob,
- and we clean up the room before we leave.

Giving ourselves this goal of completing the work had, perhaps predictably, an immediate effect. The first time the whole team stayed for an additional two and a half hours to deploy. Thus, they decided in the retro that perhaps more automation in the deployment process would not be a bad idea. It also raised awareness in the team about the advantages of small stories.

4.5 Hourly Retrospectives

Already mentioned above, a core practice for our teams were the hourly retrospectives. With our focus on team learning, the most important outcome of the mob was in learning how to improve.

An hourly retro needs to be short and to-the-point. We started with a simple positive/negative items system, and made sure this was visualized on our daily scrum board. Here's an example of an early Task & Retro board (see Fig. 4):

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

The basis of the board is the horizontal axis for the hourly blocks. Every hour the corresponding column is used. The top part for positive feedback (e.g. "We chose a good user story to work on"), the lower part for the improvements (e.g. "tests fail").

The left of the board is a basic scrum (ToDo/In Progress/Done) board, turned on its side, where we kept track of the tasks for the day's story.

As we refined the retro, we changed the board from having distinct sections for positive and negative points to one where we have a gradual scale from top to bottom, inspired by the happiness metric [2].

In the example below (Fig. 5), note the trend towards negativity as the day progresses, undoubtedly influenced by the lack of progress on the tasks shown in the task board on the left.

In the next example (Fig. 6), a board from team Red, we do see a clear upward trend in day. The team has further extended the board by adding a task burn-up chart.

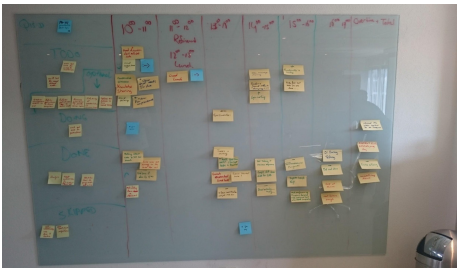

Fig. 5. An incremental improvement to the retro board: a gradual scale of positive to negative

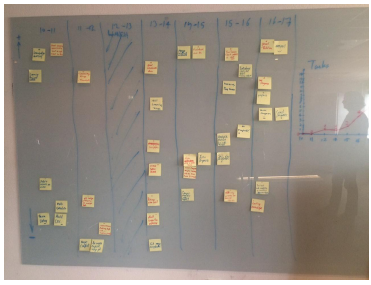

Fig. 6. A more upward trend of the day, in both retro-points and task burn-up

Retrospectives are also put in Confluence. Easier to read, for reference, and as this is done as a Mob at the end of the day, it is another moment where the retro points are digested by the team. Additionally, actions for the next session are added as tasks.

4.6 A Special Mob: Process Flow

In one of our mobbing sessions it turned out that the story required the creation of a process flow. As no-one in the team had any experience in this area, this was a session where the coach (who used to be a process manager in a previous life) took on the role of navigator for the start of the session. It was interesting to see that, even though whiteboard drawings and Visio diagrams were the output instead of Java code, the same effects occurred as with other techniques: after a while our process manager could see that the basic skills had landed, and he could step back and let others take on the navigator role.

5 Acceptance

Team Red, our junior team, embraced Mob Programming the most. They've indicated that they don't want to work this way all week, because they had to do an individual study as well, and time spent in the mob meant less time to prepare for their OCA and OCP exams. But, even after the initial 8 sessions of our summer experiment, the team continues to have regular mobbing days.

On the other hand, Team Yellow, the less junior team, disliked the experiment, didn't like the working in a group, and kept saying that they thought it was inefficient, no matter how much I made clear that this wasn't a consideration. So they stopped after only 3 attempts at mobbing.

A few months after they had stopped, team Yellow needed to work in a new technical domain, with some pressure on learning this domain quickly due to a new project that they had landed. Team Yellow then decided to split in two groups, both addressing a particular area. Although they didn't do the strict rotation, the interaction (with driver/navigator roles, frequent updates in group and between the two groups) was clearly reminiscent of the setup they had experienced a few months earlier while mobbing.

In an unexpected late update, only a few weeks before finalizing this paper team Yellow decided independently to Mob Program for a day to tackle a difficult user story. My initial conclusion that they had rejected the technique was premature: they did find value in mobbing and added this new tool to their toolbox.

6 Team Growth

Learning software development is the primary goal for our group of junior developers. They were already delivering demos weekly, proving they understood the studied chapters in their books. Working in a mob with their peers accelerated everything, from exchanging coding practices to learning to have an opinion and to share or even defend it.

For instance, one day Team Red discovered that the training room was not available, so they decided to move to the boardroom. With unfavorable light conditions and table arrangement, they themselves decided the room as unfit and changed to a better room. All without any intervention of the coach, something that would be unthinkable a few weeks earlier.

Team Red started to identify user stories that were suitable for Mob Programming and those which were not, in light of the very short cycle and strict rhythm. The characteristics of these stories proved to be mainly how clear the goal was and whether enough contextual information was available. Basically, they were finding shortcomings previously undetected in the Backlog. This dramatically improved the interactions with the Product Owner and within the team during the regular refinement sessions.

Having to finish a story at the end of the day, the Yellow team noticed *together* that deployment to production took far too much time. Since they had the rule in place that everyone stayed until the day was closed with deploying the user story, it felt even longer. This drastically changed their attitude to deployment automation.

A set of scripts had been disabled in Jenkins, because it gave too many errors. To speed up the deployment process the team had to re-enable the scripts and fix the problems. The process went down from 40 min to 10.

With both teams lacking experience in Pair Programming, I made sure that the rules for Mob and Pair Programming were almost the same. So basically we do Mob Programming, but if the group consists of only two people we call it Pair Programming. Many people would perhaps dislike this approach, but after the experiment the result is that people more often choose to do Pair Programming than before. I am positive that with additional exposure they will reach a point where a more relaxed approach to pairing will also work for them.

Overall, the level of discipline/cadence/structure went up for both teams, while at the same time the evaluations during the retrospectives were more positive and productive.

The Red team is now very mature and self-sufficient in their day to day processes. Though they still need support on technical issues, they only need help from a scrum master when they run into conflicts or other situations that have a need for more life and work experience.

7 Conclusions

I can state that because of these weekly Mob Programming days, the team as a whole and all its individuals have grown much faster than they could have done otherwise. Not only did they improve their coding skills, they improved in many other aspects, such as their requirements process, deployment procedures, appreciation for focus, and perhaps most important of all, their much higher degree of self-sufficiency. Learning would have been quicker and more directed with a senior as part of the team, but they progressed greatly, even on their own.

As a coach, I had my own learning experience. The difference in reaction between the teams indicates that a different approach might be more effective with more senior people. Perhaps that is not surprising. We all get more set in our ways the longer we are

used to our particular habits. The experience for me as a coach has resulted in lessons learned I'll take into my next Mob Programming experiments. And those will certainly happen!

Acknowledgements. First I like to thank the teams at Qualogy Solutions for being my test subjects. Woody Zuill and Llewellyn Franco deserve credit for inspiring me, in their sessions and through their ideas, to conduct this experiment. Many thanks go to Wouter Lagerweij, for encouraging me to submit this report and helping me clean up the mess afterwards. Final thanks go to Joseph Yoder, for shepherding me through the process. His knowledge of the process and interest and enthusiasm for the subject helped me greatly.

References

1. Falco, L.: Group Learning. Today's exercise: Unit Testing (session at Agile 2015)
2. Kniberg, H.: What is Crisp (on The Happiness Metric) http://blog.crisp.se/2010/05/08/henrikkniberg/what-is-crisp
3. Zuill, W.: Mob Programming: A Whole Team Approach, experience report at Agile (2014). https://agile2014.sched.org/event/1exPSc0/mob-programming-a-whole-team-approach-woody-zuill
4. Zuill, W.: A Day of Mob Programming. https://www.youtube.com/watch?v=p_pvslS4gEI