One Year of Remote Mob Programming

JONATHAN TURNER, Emmersion

Mob programming is an extremely effective way to develop software, not only in-person, but also remotely. Over the past year, my team and I have been exploring ways to collaborate while working remotely. While there are some requisites to make remote mob programming effective, I have found it to be a surprisingly productive way to work. In this report I share the most important lessons learned about the challenges we faced with remote mob programming.

1. INTRODUCTION

Mob programming is a software development practice that has seen increased adoption since its popularization by Woody Zuill (1). It can be considered an extension of pair programming (2), but instead of having only two developers working together on the same piece of code, an entire team of developers works on the same code at the same time on the same computer.

I have been a professional software developer for almost 20 years. During my career, finding ways to develop software more collaboratively has become increasingly important to me. I was familiar with the idea of pair programming for some time, but only got to practice it occasionally in an actual working environment. In 2013, I started working at Pluralsight, which was my first experience using pair programming essentially all day every day. Later during my time at Pluralsight I was able to start experimenting with mob programming, including being on one team for several years that did the vast majority of our development work using mob programming.

In early 2020, I decided to transition from Pluralsight to Emmersion. Emmersion also practiced mob programming. At the time, the majority of the single team was co-located, with a small percentage of the team members being remote. My expectation in making this transition was to practice mob programming full time, but primarily in-person. As we all know, 2020 and the COVID-19 pandemic introduced many challenges and changed many expectations. My situation with mob programming was no different. Instead of practicing mob programming in person as I had expected, I spent the following year practicing exclusively remote mob programming.

Initially I was concerned that using mob programming in an exclusively remote setting would be significantly less effective than the in-person mob programming I had experienced in the past. Fortunately, this turned out not to be the case at all. I found it to be a highly effective way to develop high quality software, delivering value frequently to end users.

While remote mob programming shares many of the same benefits and challenges as in-person mob programming, I have found that it has some distinct challenges as well that need to be overcome. This paper will present the major challenges we faced and the options my team and I explored to overcome these challenges.

2. CHALLENGES

Outside observers sometimes perceive mob programming as an inefficient practice, since multiple team members are working on a single piece of code when they could be working on multiple pieces of code at once. While I can understand this concern, I have found the opposite to be true. I have found mob programming to be an effective way to write high quality, maintainable code quickly and deliver it to customers frequently.

However, this does not mean that simply placing several people in front of a computer and saying, “Go!” is guaranteed to result in a productive or even pleasant mob programming experience. Mob programming, like all technical coding practices, requires practice. It also has certain social and technical requirements in order for mob members to be able to fully contribute and write high quality code quickly. This is true of in-person mob programming as well as remote mob programming. The remainder of this section details some of the primary

Jonathan Turner; email: jonathan.m.turner@gmail.com

requirements for effective remote mob programming and the solutions to those requirements that we have implemented at Emmersion.

2.1 Environment

The physical environment in which a mob member is remotely mob programming can greatly affect the quality of the mobbing experience. This is true not only of remote mob programming, but also of in-person mob programming as well. A shared space at home with a lot of ambient noise and distractions, much like an open office space with little sound dampening material, can make the mobbing experience frustrating. A relatively quiet area on the other hand can allow mob members to communicate more freely. With remote mob programming, this type of environment can help a mob member achieve much of the same rich communication that exists with in-person mobbing.

Before I joined Emmersion the team mobbed primarily in-person, with some members remote. This hybrid approach to mob programming introduces some challenges that are different from either completely in-person mobbing or completely remote mobbing. But the focus of this paper is on completely remote mob programming, since I joined right as the team transitioned from this hybrid approach to completely remote mobbing. Fortunately for us, most of the team members had an environment in their homes that was conducive to remote mob programming. Most team members had an isolated room they could use separate from those they lived with.

A few team members had less than ideal environments at home. They were able to use their home environments sometimes, and go into the office other times. Their ability to do this of course varied with the local governmental COVID-19 restrictions. For some team members this did help make up for a less than ideal environment at home. Working in the office was also fairly comfortable for most team members as that was how they were working before having to work remotely. Since most employees were not in the office during this time, the experience for the rest of the team was mostly unaffected by ambient noise around the mob member in the office.

While having a quiet, distraction free environment is necessary for remote mob programming to be effective, it is not necessary to have an environment that is 100% devoid of distractions. For us, most team members usually have at least a couple of distractions per day, such as a spouse walking by, a delivery at the door, kids coming in, or a cat needing attention. This is actually one of the strengths of mob programming: one or two mob members can briefly disengage from the mob to deal with a minor distraction and then quickly rejoin the mob.

2.2 Communication

If you can't communicate, you can't collaborate. If you can't collaborate, you can't mob. For in-person mob programming, this communication takes place via speaking, viewing the other mob members' facial expressions and body language, and through materials (often, but not always code) on a large, shared screen. With remote mob programming these same types of communication happened, though somewhat differently and sometimes augmented by other communication streams.

Remote mob members need to have a way to hear each other, and ideally see each other. They also need a way to show each other materials such as code. At Emmersion we currently use Zoom (3) for all three of these functions. We have a persistent Zoom meeting for each team that we consider to be the team's "mobbing room". Whenever mob members are not engaged in meetings or other activities that require them to work individually, they are in the team's mob meeting. Mob member cameras are usually on, and mob member microphones are usually enabled. When it is a given individual's turn to fill the driver role, they will share their local screen for the duration of their mob rotation.

We use several other tools for communication besides Zoom. We use Slack (4) to transfer files, links to websites, and some other types of persistent communication. Zoom has a chat feature, but it is ephemeral, so we use it infrequently. Kanbanize (5) is the online kanban board that we use. With in-person mob programming, the mob members will often use a whiteboard to share ideas and discuss solutions. When transitioning to remote mob programming we initially felt the loss of this type of communication. We experimented with a few different digital whiteboards. We currently use Miro (6) as a digital whiteboard, as well as a sticky note tool.

There are some technical considerations to keep in mind when remote mob programming. Sending audio and video streams and screen sharing puts extra demand on computational resources (CPU usage, RAM, etc.), in addition to whatever development resource needs there are. Depending on the resources of the computer and the development resources needed (such as running multiple Docker containers, compiling large code

bases, or running heavyweight IDEs), this can adversely impact communication or the ability to develop or both. Additionally, the cameras, microphones, and speakers on some laptops are not particularly high quality. Most of our mob members have separate dedicated audio-video equipment, either that they already had, that they purchased, or that Emmersion purchased for them, depending on need and urgency. Finally, bandwidth can be an issue if it is too constrained (such as by sharing it with others in the home who are also using bandwidth intensive applications) or has too high a latency.

2.3 Contribute to the Code

If mob members are unable to contribute to the code they will be unable to fulfill all the mob roles (driver especially) and their ability to contribute to the overall mob programming process will be limited. With in-person mob programming, all mob members work on a shared machine. Switching from one mob member as driver to another is as simple as sliding the keyboard from one person to another. With remote mob programming, the requirement to easily switch between drivers still exists, but the mechanism is different.

The approach that we at Emmersion have used the most is to have each person work on their own computer when they are the driver, and to use a tool to transfer code between mob members' computers. The tool we have used to do this is a command line tool called *mob* (7). It easily allows mob members to start a new piece of work, start or end their turn as the driver for an existing piece of work, or indicate that a piece of work is finished. It does this by creating a "mobbing" branch in a Git repository. This temporary branch is then used to transfer work in process code between mob member computers. These "mobbing" branches are shorter lived than typical feature branches, typically lasting anywhere between a couple of hours and a couple of days. We try to size our units of work so that they can be completed and deployed to production within this time frame. For smaller pieces of work we will often commit those changes directly to master/trunk/main.

When it is a given mob member's turn to drive and the mob is starting on a new piece of work, the current driver will share their screen (using Zoom), issue a *mob start* command using the mob tool, follow the navigator's instructions until the end of their mob rotation period, then issue a *mob next* command to commit their changes to the "mobbing" branch, and push the changes in that branch to the central Git repository. We currently use Github for this and have used Azure DevOps Git repositories in the past. The tool should be compatible with any Git hosting platform.

When a piece of work is complete and ready to be merged to master/trunk/main (and thus be released to our staging environment and potentially our production environment), the current driver will issue a *mob done* command, which squashes all mobbing branch commits into a single commit and allows the current driver to review the changes, provide a single concise commit message, and push those changes.

Some mobbing teams (notably those at Hunter Industries, which is where Woody Zuill worked when he initially popularized mob programming) use a single shared mobbing machine approach, rather than the approach of moving code between individuals' machines that we primarily use at Emmersion. For the single shared mobbing machine approach to work, there must be a single machine somewhere (either a physical machine or virtual machine that is accessible by all mob members) that has the necessary developer and mobbing tools set up on it. The mob members are then able to remote into that machine to write code and develop new features. Hunter Industries uses AnyDesk (8) to do this, which allows multiple people to interact with the mobbing machine at the same time, rather than a remote desktop session, which typically only allows one person at a time to remote in. At Emmersion, we have experimented with AnyDesk and a shared mobbing machine approach, but do not use it as our primary mobbing practice right now. We have also occasionally used the shared control feature on Zoom for similar purposes, though we currently prefer the experience with AnyDesk. We will continue to experiment with these tools and this approach to remote mob programming.

One challenge with using the approach of moving code between individuals' machines is having to share configuration and state between mob members. We use Docker, combined with various setup scripts, to make sure that it is easy for mob members to get a known good state. But sometimes a specific configuration is needed for a given feature or task. We typically don't create setup scripts for these types of situations since they are generally short lived. In these situations, we either have one mob member drive for longer, or pay the cost of doing that configuration multiple times. This is an area of potential improvement for us.

An interesting side effect of the approach we use at Emmersion is that we tend to make more use of the researcher role. The two primary roles in mob programming are driver and navigator, but there are many other possible roles as well. Willem Larsen (9) has documented several of these "auxiliary" roles. One such role is researcher. When a mob member is in this role, they disengage somewhat from the active process of translating ideas into code and instead search other resources to find answers to help the rest of the mob. At

Emmersion, since each mob member is using their own computer, quickly transitioning to this researcher role is often easier than if we were in person using only a single shared mobbing computer.

2.4 Remote Mob Timer

Having mob members rotate frequently between the main driver and navigator roles helps ensure that everyone is engaged, helps maximize learning across the team, and helps all team members contribute to the solution. With in-person mob programming this rotation frequency is typically quite short, 10 minutes or less. Some type of physical timer or one of various mob timer applications is used to keep track of the time.

With remote mob programming, these frequent rotations are also beneficial for the same reasons as with in-person mob programming. But physical timers can make it difficult for everyone to see and interact with the timer. Mob timer applications are useful if the mob members are using a shared machine, but less so if each mob member is using their own machine, like we have done at Emmersion. In this type of situation a shared, online mob timer has been a great option for us.

When I first joined Emmersion my team was using a 20 minute rotation timer. This was after reducing the rotation from much longer periods, such as 30-40 minutes. These longer rotations were mostly due to a lack of familiarity with the tools we were using. The team I am currently on typically uses a 10 minute rotation cycle. We occasionally use a shorter rotation cycle.

We experimented with several online mob timers, including one we built ourselves. The one our team currently uses is *mobtime* (10). This timer is open source and can be self-hosted, though so far we have successfully used the version hosted by the author. It uses websockets, so everyone using the timer has the same time and all the same settings almost instantaneously. Mob members can easily be added or removed from the timer list, and can be easily skipped if they are away or busy when their turn as driver or navigator comes up. This timer defaults to having designated navigator and driver roles. Mob members can change the names of these roles or add additional roles if desired. There is also a "goals" feature that our team hasn't used, but that some could find useful.

One drawback to using an online mob timer instead of one installed locally is a lack of integration with the operating system. The biggest example of this is a feature that many installable mob timers have which blocks the entire screen when it is time to switch roles. This may sound disruptive, but is a great way to force the mob members to not ignore the timer. With online mob timers, having the timer go full screen automatically when the rotation time elapses is not possible. It is possible to have browser notifications that can optionally integrate with the operating system notification system. This helps, but is still easier to ignore than a full screen application.

Because of this drawback of online mob timers, we do sometimes find ourselves forgetting to start the timer when a new driver and navigator take over. This sometimes causes one set of mob members to have a longer rotation than they otherwise would. As mentioned above, it is also easier to miss when the rotation timer has elapsed, which causes one set of mob members to have a longer rotation. This is an area of potential future experimentation and improvement for us.

3. WHAT WE LEARNED

Over the course of the last year, we have learned that many of the skills needed for in-person mob programming translate to remote mob programming. Many of the things needed for successful in-person mob programming are also needed for remote mob programming, such as having adequate equipment and having a mob timer. Some requirements for successful in-person mob programming translate mostly unaltered to remote mob programming, while other requirements need to be adapted, such as ways to be able to contribute to the code. We have also learned that there are some requirements for remote mob programming that are mostly unique, such as ensuring that mob members have the correct tools to communicate.

We have learned that remote mob programming is effective. Given the right tools and conditions, remote mob programming is as effective as in-person mob programming. We have learned that remote mob programming is a productive way to craft high quality code and deliver valuable features to our customers frequently. Remote mob programming pairs well with technical and other practices, such as test driven development, continuous integration, push button deployments, limiting work in process, and eliminating waste.

4. ACKNOWLEDGEMENTS

I would like to thank Woody Zuill for, if not inventing mob programming, definitely popularizing it. I would like to thank those who have been on my team during this last year. Thanks Ryan, Kevin, Charles, Eric, Danny, Neil, Judson, Dani, Zac, and Nate! Thanks to Dave, Mike, Allan, Jacob and others in management positions at Emmersion for understanding and promoting the importance of mob programming and other technical practices. And thanks to the conference organizers and volunteers for their huge amounts of effort that go into organizing events and schedules, selecting presenters, and all the other numerous things required to organize a quality conference. And finally, thanks to Ademar, my shepherd, for his great input on and help with this paper.

REFERENCES

1. Zuill, Woody. Mob Programming - A Whole Team Approach. *Agile Alliance.* 2014. https://www.agilealliance.org/resources/experience-reports/mob-programming-agile2014/.
2. Fowler, Martin. PairProgramming. *MartinFowler.com.* https://martinfowler.com/bliki/PairProgramming.html.
3. *Zoom.* https://zoom.us/.
4. *Slack.* https://slack.com/.
5. *Kanbanize.* https://kanbanize.com/.
6. *Miro.* https://miro.com/.
7. Harrer, Simon. *Remote Mob Programming.* https://github.com/remotemobprogramming/mob.
8. *AnyDesk.* https://anydesk.com/en.
9. Larsen, Willem. *Mob Programming: The Role Playing Game.* https://github.com/willemlarsen/mobprogrammingrpg.
10. Barry, Alex. *MobTime.* https://mobti.me/.