Extracted from:

Code with the Wisdom of the Crowd

Get Better Together with Mob Programming

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Code with the Wisdom of the Crowd

Get Better Together with Mob Programming

Mark Pearl

edited by

Tammy Coron

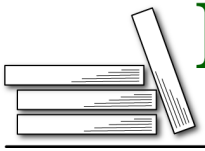
Code with the Wisdom of the Crowd

Get Better Together with Mob Programming

Mark Pearl

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Development Editor: Tammy Coron
Copy Editor: Jasmine Kwityn
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-615-0
Book version: P1.0—July 2018

CHAPTER 6

Preparing Your Team for Regular Mobbing

Different teams use Mob Programming at different intensities. Some teams mob all day, every day; some do it for a scheduled part of the day; and some do it as-needed (ad-hoc) for key work where they want everyone involved. If you plan to have your team mob more than just ad-hoc, you'll need to get them prepared.

Like healthy eating, for mobbing to happen regularly within your team, you need a lifestyle that supports it.

In the last few chapters, the focus was on the individual people within the mob and the mob's workspace. In this chapter, you'll examine the team as a whole and learn how to prepare them for regular mobbing.

Linking Mobbing Needs to Team Needs

Your team is a system with needs. If you were to put the needs into categories, they'd fall into two main groups: the needs of the individuals in your team, and the needs of the sponsors paying for your team—and it's important to understand how Mob Programming fulfills the needs of each. The payback if you successfully link Mob Programming to your team's underlying needs is it will stand the test of time; if you don't, it will become a fad and fall away.

Before trying to map the needs for each group, your team should experiment with mobbing a few times so everyone can have a meaningful conversation (rather than it being all theoretical). You should also have this conversation before starting to mob regularly, and before you forget what problems you had before mobbing.

Some examples of individual needs that Mob Programming fulfills include:

- Having a more complete understanding of the problems you're solving.
- Spending less time investigating defects.

- Having fewer meetings to synchronize team members' work.
- Having fewer interruptions while working.
- Writing higher quality code.
- Being able to take leave whenever you need it.
- Feeling more connected with others in the team.

When connecting mobbing to fulfilling your needs, make them specific to your team instead of generic-sounding. For instance, instead of pointing out a generic need:

- "Having a more complete understanding of the problems we're solving."

Aim for something that relates to your team:

- "John and Darren have great domain knowledge, and we'd like to share this knowledge with the rest of the team. Mob Programming gives us a complete understanding of the problems were solving."

Your team also has sponsors paying for your team who have needs that must be met. Identifying what needs Mob Programming fulfills for your sponsors is just as important as linking Mob Programming to individual team member needs. Some examples of sponsors' needs include:

- Fewer defects impacting users
- Fewer key-person dependencies
- Up skilling team members

Does this list look strangely familiar? These are the same points we covered in *Getting Support from Management*, on page ?. Having your team be aware of how Mob Programming helps fulfill the needs of your sponsors as well as their own individual needs is a powerful way of cementing Mob Programming as a practice in your team.

Adjusting Team Processes for Regular Mobbing

The more your team mobs, the more likely it is that your team processes will need some adjusting, or at the very least, consideration for adjustment.

Mob Programming and Scrum, Kanban, Waterfall, etc.

A common concern for new mobbers is how mobbing impacts their current methodology: "My team does methodology *X* (put Scrum, Kanban, Waterfall or any other development methodology in its place), can I do Mob Programming and still follow this methodology?"

The short answer is "Yes." Mob Programming doesn't discriminate on a specific development methodology. When I asked the Mob Programming community what methodology they use, the response was varied: Scrum, Kanban, Scrumban, FAST Agile, XP, Directed Discovery, Lean, you name it. Chances are, you can keep your current development methodology in place and mob.

If there's one thing that might prevent your team from mobbing while following a specific development methodology, it's if your development methodology explicitly requires tasks or pieces of work to be assigned to specific individuals upfront, with the rule that nobody else is allowed to work with that person while they're working on that task. However, I'm not aware of a development methodology that does this (although I wouldn't be surprised if there was one out there).

While development methodologies vary from team to team, there are two team activities common enough across most teams that they're worth examining further in the context of Mob Programming: daily stand-ups and code reviews.

Your Daily Stand-up

If your team is following an agile process, you probably have a daily stand-up where the team gets together to talk through what they did the previous day, what they plan on working on today, and how they plan on doing it. The more your team mobs together, the more in sync they'll be; this often means fewer things to talk about at stand-up meetings. In fact, you can even consider reducing stand-ups to just two questions:

- What is the most important thing we need to get done today?
- How are we going to get it done?

Getting your team to align on these two things encourages appropriate mobbing. But keep an eye out for generic answers. If at the end of the stand-up you're not sure who's going to be involved in what for the day, speak up—you're probably not alone. The primary purpose of this kind of stand-up is to figure out how you're going to collaborate with others on the work. Once it's clear what the most important thing is for the day, getting people to work together to get it done becomes a natural next step.

Dropping Daily Stand-Ups

Some teams that adopt Mob Programming stop doing daily stand-ups altogether because they no longer find that it adds value. While this may sound sacrilegious to some, think of it this way: if the purpose of the stand-up is to sync up on how different work is progressing, and the team already knows what's going on because of the mob, are stand-ups still needed?

Before you drop daily stand-ups, be aware that it's not for everyone; usually you should only consider it if your team mobs all day, every day, and everyone in the team is also in the mob; for everyone else, a daily stand-up still makes sense and adds value to your team.

Code Review Process

If your team has an existing formal code review process, what happens to your review process on code that's been created by the mob?

Mobs handle this in various ways. The two most popular options are (a) making no changes at all, or (b) using shared mob credentials.

No Changes

The first approach is to simply not change anything, and to treat mob code the same as any other code. However, for some mobs, this may seem like an unnecessary waste—why formally review mob code if the review has been happening by the mob while the code is being written?

Shared Mob Credentials

Another option is to keep code review and use a shared mob credential for checking in mobbed code. This allows you to easily maintain your code review processes for solo code while dropping it for mobbed code.

If you go this route, and you have a security admin that maintains your version control credentials, expect some resistance when requesting a generic shared version control account. One of the pillars of security is traceability (the ability to trace something back to its original creator). At first glance, having a shared version control account seems to violate this idea. However, in this case, you still have traceability; the code is created as a group, not as an individual, and it's owned and maintained as a group, or in other words, the team has collective code ownership. Collective code ownership was advocated by early extreme programmers. The code base is owned by the entire team and anyone may make changes anywhere. Martin Fowler[1] explores the concept on his bliki.[2]

The challenge for many security admins is that most version control tools are designed with a "single commit" in mind, and to deviate from this means you're doing something wrong. If you can get them to understand this is a restriction on the tool rather than doing things the wrong way, you'll have

1. https://twitter.com/martinfowler
2. https://martinfowler.com/bliki/CodeOwnership.html

better luck getting it approved. Often, simply explaining the concept of collective code ownership to your security admin resolves this concern.

Attributing a Single Commit to Multiple Developers in Git

As far back as 2011, on StackOverflow there was a discussion of how to attribute a single commit to multiple developers, and there have been various techniques and tricks suggested to make this possible.[a]

If you use GitHub you'll be pleased to know that in January 2018 they announced a feature that allows people to commit together with co-authors.[b]

a. https://stackoverflow.com/questions/7442112/how-to-attribute-a-single-commit-to-multiple-developers
b. https://blog.github.com/2018-01-29-commit-together-with-co-authors/

Team Retrospectives

Hopefully, regardless of whether your team follows an agile methodology or not, you have a regular recurring meeting where you can reflect as a team on how things have been done in the past and how to become more effective going forward—in the agile world this is often called the team retrospective.

As part of the mobbing recipe in *Learning from the Experience*, on page ? it was suggested that you have a mob retrospective after each mobbing session. In a mob retrospective, you reflect on how the session went and what can be improved going forward. But don't confuse the mobbing retrospective with the team retrospective—they are different things. The mobbing retrospective is focused exclusively on the activity of mobbing, while the team retrospective has a much wider focus on all things team-related.

Over time, as your team gains experience in mobbing, you may feel that the mobbing retrospective is unnecessary overhead—if you start feeling this way, drop the mobbing retrospective; it's a useful activity in the early days but can become overkill with time.

Keep the Team Retrospective

Regardless of whether or not you're having mob retrospectives, continue to have your team retrospectives as normal—I've found them the lifeblood of a healthy team. If your team is currently not having regular team retrospectives, it may be time to start doing them—they are extremely useful! If you are unsure how to get started, I've found *Agile Retrospectives [DS06]* is a great resource for those less familiar with the practice.

Who Should Be in the Mob

In *Deciding Who's in Your First Mob*, on page ?, you tackled who should be in your first couple of mobs, and you kept the number of participants small, ideally having 3 to 4 people at most. When preparing your team for regular mobbing, it's worth exploring this further.

First, there is no definitive answer on how may people should be in a mob; current trends are that mobs tend to be between 3–5 people, although some mobs are significantly larger. It varies depending on the problem you're solving and who you need in order to solve it.

Instead of giving your team a hard rule on who should be in a mob, and who shouldn't, it's better to teach them the principle for being in the mob and allow them to self govern. In this case, the principle is, "If you're in the mob and you find yourself contributing, you're meant to be there. If not, either start contributing or find another way to add value."

In other words, regardless of your role—be it developer, tester, business analyst, product manager, or anyone else—if you're finding value from being in the mob, you are entitled to be there. If not, take a walk.

The Law of Two Feet

The principle for being in a mob is an adaptation of the "law of two feet." The law of two feet originated from open spaces.[a] In its original form, it goes like this: "If at any time during our time together you find yourself in any situation where you are neither learning nor contributing, use your two feet, go someplace else."

a. https://en.wikipedia.org/wiki/Open_Space_Technology

Being Around to Mob

For most people, having some sort of flexibility on what hours you can work is important. Some people like to come in early and leave early, while others prefer to come in late and leave late. This can prove a challenge for teams that want to mob.

There are several different approaches teams can use to overcome this problem, from the most popular (establishing collaboration hours) to alternatives like having a revolving mob or even remote mobbing. Take a moment to go through the specifics of how collaboration hours work.

Aashiq's Experiments with Remote Mobbing

What's it like to do remote mobbing? While I only have limited experience with it, Aashiq, a seasoned remote mobber, shared his experience with making it work:

> "My journey began when I joined a team who believed in experiments and finding better ways to add value. This team worked in the complex domain of banking and had been mobbing for well over a year when I joined.
>
> We began experimenting with remote mobbing because a key member of our mob was unable to come into the office for an extended period of time due to health reasons. We wanted to still work with him but didn't want to give up mobbing. At the time we had no idea if remote mobbing would work—we had concerns because most of us were visual thinkers and used the whiteboard a lot.
>
> On our first attempt at remote mobbing we tried to share our mob screen over Skype, but we soon discovered a security network limitation on the mob machine meant we could not get the audio and video working. Undeterred, we ended up making it work by video calling him on an iPad—think of the iPad as our virtual team member; he could see the mob via the iPad's camera and we could see him on the iPad screen. Funny enough, we worked this way for several months and it actually felt productive, though never as effective as the real-time coding experience when all physically together.
>
> With time, we improved the experience by experimenting with tools like Screenhero, various online whiteboards, and some visual representation tools. We also got the security limitations on the network lifted, which made things a lot easier. At one point, we managed to successfully mob with multiple people working remotely.
>
> Putting all the technicalities of the technology aside, I think we were able to mob well because of the relationship and understanding the team formed prior to remote mobbing. The level of communication we had already established—and our ability to express intent and create a shared understanding—allowed us to be effective.
>
> We also found having agreed mobbing hours for the day helped us make it happen.
>
> Recently, I've had the opportunity to experiment with remote mobbing with a new team; even though the network and tools this time were much better, the experience wasn't as smooth. Based on this, I believe that as a team you need to get the fundamentals of mobbing covered well and have a good collaborative relationship before trying remote mobbing."

If you're going to experiment with using remote mobbing, I recommend reading *Remote Pairing [Kut13]* for some ideas on tools and configurations to experiment with.

Collaboration hours consist of establishing a set or core hours where everyone in the team is around to work together. It still provides people with some flexibility but also blocks out a set time for mobbing, making scheduling easier. During collaboration time, it's appropriate to mob; outside of collaboration time, it's acceptable to do other things. Having agreed hours for collaboration also removes the pressure from people to mob all the time.

Deciding on the appropriate collaboration time for your team is context-specific and will change depending on your team's needs. While collaboration

hours is one way to ensure everyone is around to mob, it certainly isn't the only solution—there are alternatives, like having a revolving mob where some people start early in the morning (at normal starting hours), and some start late and work late. In this scenario, they'll overlap each other to preserve context as they progress throughout the day.

Gathering the Mob

In *Finding a Place to Start Mobbing*, on page ?, it was recommended that you start mobbing in a meeting room. When you're mobbing in a meeting room, getting everyone into the mob to start is easy—you're all there already, so it just happens.

When you move mobbing back to your normal work area, it's not so simple. Often, despite your team agreeing about the massive value added from mobbing and their desire to do it more often, when it comes to actually getting together to mob, they struggle to get started.

One common behavior that many teams suffer from is everyone waiting for one another to get the mob started. I've seen it happen many times: your team agrees that they need to mob on something, but first they have some odds and ends to finish individually. Each time someone finishes their work, they check to see if the mob has started; each time, it appears that everyone else is still busy on their own. To avoid being idle, the person who's just finished their work, starts a new piece of solo work.

Unfortunately, this pattern repeats itself throughout the day: everyone finishes and begins new solo work, at different times, with no actual mobbing ever happening. One approach that helps counter this "continual waiting for someone else to get the mob started" is to embrace mob rustling.

Mob Rustling

Mob rustling means being intentional about starting a mob. Often, all that a mob rustler needs to do is let others know they're starting to mob on something and then move to the mob machine and begin. Usually that's it, once that person is by the mob machine and starting the work, within minutes, others join and the mob is gathered.

Most times, this simple approach is sufficient to get things started; sometimes, however, it doesn't work, and ends up with someone working at the mob machine on their own. If you're rustling, and this happens to you, don't stress—you're doing the right thing; it could just be a lot of unplanned items in-flight that other people are sorting out. The number one rule of mob rustling

is never force someone to join the mob—encourage, invite, and inspire, but never force.

Establishing a Mob Starting Ritual

Another way to help mobbing get started each day is to form a daily starting ritual. Humans are creatures of habit; forming a ritual that naturally lends your team to start mobbing in an unforced way can make all the difference.

For example, a team does a stand-up, which is the first activity inside the agreed core hours. This creates alignment, and sets focus for the day. After the stand-up, most of the team goes together to the coffee station. Think of starting the mob as just an extension of the coffee run, making it feel natural and unforced.

Handling Mob Fatigue

The first time I mobbed, it was a weird experience, we worked on a problem for a few hours, it felt like time had passed by quickly, although I also found myself exhausted by the end of the session. In fact, I ended up going to bed early that night because I was so tired. The feedback from others is that this is common for most people that mob. It's an intense way of programming, and without adjusting your workflow to handle this intensity, it's easy to suffer from mob fatigue or extreme tiredness.

First off, it's important that your team is aware of mob fatigue, and that you have a plan for how to handle it. Also, it's a problem even if only one person is experiencing fatigue, because that one person can impact the entire mob.

Now for the good news: mob fatigue is preventable. Doing something as simple as taking regular breaks when mobbing, and not pushing it too hard, will help most people stay energized and fresh. Here are a few things you can do to reduce its impact and speed up recovery.

Regular Breaks

When mobbing, encourage individuals to take regular breaks. Rely on the approach described in *The Pomodoro Technique*, on page ?; every 25–30 minutes, take a quick break from the mob. Whether you do this as a group, or quietly leave and rejoin the mob individually, is up to you. Just be aware of some of the common mistakes people make when leaving the mob (see *Disrupting the Mob When Leaving and Joining It*, on page ?).

In addition to the small regular breaks, you want slightly longer breaks throughout the day. If you mob for more than an hour and a half, call for a

break where the whole mob takes 15 minutes together to get coffee or stretch their legs. Regular breaks increase energy and productivity.

Speeding Up Recovery

While taking regular breaks makes a huge difference in prevention, there will still be times when people get exhausted. Here are a few tips on how to speed up the recovery process when this happens.

Let Others Know How You Are Feeling

Communicate early. Let your team know at the beginning of the day that you're feeling tired. If they don't know how you're feeling, they can misinterpret your fatigue as you being displeased with their behavior.

Work On Something Else

Sometimes you just need to work on something else. If you're experiencing fatigue because what you're working on is extremely draining, there's a good chance others in the mob are feeling the same way. If this is the case, it may be worth changing what you're working on for a period of time to give the mob an opportunity to recharge.

If the rest of the mob is feeling fine and it's just you, find out if there's something that needs to be done that you can do on your own. If you do this, pick something small (at most a day's worth of work) and make sure the rest of the team is comfortable with you doing it before commencing.

What's Next?

In this chapter, you explored some of the adjustments you should consider making to your team's lifestyle to allow for regular mobbing. You learned that it's important to identify the needs of people in your team, as well as the needs of your team's sponsors, and that linking those needs back to mobbing helps to avoid mobbing becoming a passing fad. Finally, you reviewed the important issues regarding flow breakers in mobbing, getting the mob started, scheduling, and avoiding mob fatigue.

Now that your team has a lifestyle that supports mobbing, it's time to learn more about the flow-centric way of creating software and how you can get your mob on board.