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

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 1

Why Mob Programming

Leading a software team can be difficult, and there are many things you need to consider:

- Are you solving the correct problem?
- Do you have the right mix of technical skills within the team?
- Is your team producing things at a fast enough rate?
- Is the quality at a high enough level?
- Is everyone able to work together effectively?
- Will the team be all right if someone leaves?

As a team leader, it rests on your shoulders to find solutions to these challenges—that's where Mob Programming can help. With Mob Programming, you and your team can produce quality software, at a steady rate, with a low dependency on specific individuals.

This chapter provides a high-level introduction to Mob Programming and will help get you prepared to start mobbing.

What Is Mob Programming?

Mob Programming is when three or more people work at a single computer to solve a problem together. It is about leveraging distributed knowledge[1] when programming.

Distributed knowledge is all of the knowledge that a group of people possess and might apply in solving a problem—Mob Programming brings those people together and puts them in front of a single computer.

Mob Programming was first spoken about in the Extreme Programming (XP) community in the early 2000s. At the time, getting even two people to work

1. https://en.wikipedia.org/wiki/Distributed_knowledge

together at a single computer was seen as "extreme," so largely the XP community settled on Pair Programming as the de facto way of writing code and Mob Programming faded into obscurity.

In 2015, Mob Programming came into the limelight again, largely due to the efforts of Woody Zuill[2] who began speaking about it at conferences and user groups. Woody and his team at Hunter Industries[3] had begun experimenting with Mob Programming as a way of group learning.

After some success as a once a week activity, they decided to try it as their default way of developing software and were pleasantly surprised. As Woody puts it, it allowed them to get all the brilliant people working on the same thing, at the same time, in the same space, and on the same computer.

Since then, the benefits that Woody's team have seen from mobbing at Hunter Industries have been repeated by teams all over the world.[4]

Joe asks:

Why Is It Called Mob Programming?

Moses Hohman[a] first coined the term "Mob Programming" in 2002 as part of the title to an article he wrote in *Extreme Programming Perspectives [MSWW02]*. He had been inspired to use the word Mob because of a talk given at OOPSLA 2000 by Richard Gabriel titled "Mob Software—The Erotic Life of Code."[b]

Woody Zuill later became aware of Moses's article and started using the term "Mob Programming" as a bit of humor when doing coding dojos and other group programming activities at user groups and conferences. Woody would explain to those attending these events that the process they used was like Pair Programming, but with more people—like a Mob. He'd then explain that anyone could contribute provided they keep the basic rules for it—so it wasn't like an "Unruly Mob."

a. https://twitter.com/moseshohman
b. https://www.dreamsongs.com/MobSoftware.html

So how does it work? Is Mob Programming just a free for all? No, absolutely not. While there isn't a standard rule book to define exactly how to mob, it does require a bit of structure to be effective—somewhat in the way Pair Programming does.

2. https://twitter.com/WoodyZuill
3. https://twitter.com/Hunter_Ind
4. https://markpearlcoza.github.io/Talk_PeopleMobbing

For instance, one of the structures of classic Pair Programming is the driver/navigator concept. The driver is the person behind the keyboard and the navigator is the person next to the driver. Each has a specific task: the driver solves the immediate problem by writing code, while the navigator looks at the bigger picture of where the code is headed and points out potential issues to the driver. In the end, they work together as a complementary unit.

Mob Programming has a similar structure: there's a driver, yes, but instead of having one navigator, there are multiple navigators. Also, how you interact in a mob is different than how you interact in a pair, simply because there are more people with whom you need to interact.

If it all sounds a little confusing don't worry—it will make sense as you work through the basics of mobbing; for now, the important part is that mobbing has some structure to it.

Dropping the Driver/Navigator Metaphor

Personally, I don't like the driver/navigator metaphor; the thought of having multiple navigators shouting directions to me while I'm driving gives me an awful feeling. So, from here on out, the person behind the keyboard is referred to as the "typist" and everyone working with the typist is called the "rest of the mob."

When people start mobbing, or have been asked to participate in mobbing, they often have many questions. Being able to answer those questions effectively puts you in a much better position for getting everyone's support for mobbing.

Mobbing vs. Pairing

Because Mob Programming evolved from Pair Programming, you'll likely have questions—or get asked questions—about the differences between the two. In fact, there's a good chance it'll be one, if not all three of the questions we'll explore in the following sections.

I've never paired before, should I do that first?

Many teams successfully skip Pair Programming and go straight to Mob Programming. While there are benefits to being comfortable with both practices, there is no prerequisite that states you must first master, or even participate in Pair Programming. In fact, there are some advantages in going straight to mobbing.

For example, one benefit is that it's easier to resolve stalemates with mobs compared to pairs. When pairing, there are only two of you, which means when you have a difference of opinion, it can quickly turn in to you versus your partner. With mobs, you have a group; when you hit a difference of opinion it often feels less personal. Think about it, when two or more people suggest they prefer a certain approach, the mob listens, action is taken, and the stalemate that can sometimes plague pairs is avoided.

I already pair program, why switch to mobbing?

As Joe Kutner[5] points out in his book on *Remote Pairing [Kut13]*, several academic studies have shown that pairs solve problems faster than individuals doing the same work.

While there have not been any studies directly comparing Mob Programming with Pair Programming, a 2006 study by the American Psychological Association[6] found that groups of three, four, and five were better than pairs (and individuals) at solving complex problems. Mobs have an edge on pairs when tackling complexity because there are more people working toward a common goal and offering ideas.

In addition, mobs generally have more continuity than pairs. When you're pairing with someone and they get interrupted with a phone call, or a meeting, or anything that requires them to step out, pairing stops. This has always been a frustration for me and over the years I've tried different methods to reduce this, and the Pomodoro Technique (discussed in the following sidebar) is the one I've had the most success with.

The Pomodoro Technique

The Pomodoro Technique is a productivity technique you can use on your own or as a pair. Your day is broken up into pomodoros. Each pomodoro is a 25-minute period during which you focus on one thing.

When pairing using the Pomodoro Technique, during each 25-minute pomodoro you remove all interruptions—including the phone, emails, and instant messaging. Once the 25 minutes runs out, you take a 5-minute break; you can stretch your legs, check your email, go to the bathroom, etc. At the end of the 5 minutes, you start a new pomodoro, resetting the timer back to 25 minutes, and so the cycle repeats itself.

While this is the general format, it's also an over simplification. For more specifics on the Pomodoro Technique I recommend reading *Pomodoro Technique Illustrated [Nöt09]*.

5. https://twitter.com/codefinger
6. http://www.apa.org/news/press/releases/2006/04/group.aspx

While using the Pomodoro Technique with Pair Programming can help manage many of the little disruptions, it doesn't address the challenge of the bigger ones—like what to do when someone in the pair isn't available due to a meeting, illness, or something else.

With Mob Programming these sorts of interruptions don't have nearly as big of an impact, provided you are careful on how you step-in and out of the mob. You'll learn more about that in *Disrupting the Mob When Leaving and Joining It*, on page ?.

I hate pairing, will I hate mobbing too?

Not everyone likes Pair Programming. There are people who try pairing and hate it. Unfortunately, because of the many similarities between pairing and mobbing, these same people might be worried they'll feel the same way about Mob Programming. So much so that they question whether or not they should even give it a try.

If you fall into this category, try to set aside your feelings about pairing and give mobbing a try. Mobbing is different from pairing. And while there's a chance you may not enjoy it, there's also a very real chance that you'll love it—regardless of how you feel about pairing.

A while back, I was invited to spend a day with a team of first-time mobbers. At the start of the day, a team member pulled me aside and told me he hated Pair Programming. While he was willing to give Mob Programming a try, he couldn't see how it was any different from pairing.

As the day progressed, and he got past the mechanics of mobbing, his team began to make some real progress on the problem at hand. At the end of the day, I asked him what he thought about mobbing. His response: he loved it and thought it felt totally different from pairing. Since then, he's been mobbing regularly, even though he still hates Pair Programming.

With mobbing, the personal interactions are different, the intensity is different, and the way you set things up is different, to name just a few things. So, yes, it is quite possible to hate pairing and love mobbing.

Getting Support from Management

Another challenge you may face is getting buy-in from your manager or other team leads.

In *Driving Technical Change [Rya10]*, Terrence Ryan tackles the topic of talking with your boss about professional development techniques. Terrence suggests that the single biggest reason that management offers resistance to technical practices is because they don't understand them.

The trouble is, technical people often make the mistake of pushing an idea because it's a solution to technical problems. Instead—and especially when you're talking to managers and team leads—the focus should be on solving management problems, not technical problems.

If you want to have a conversation with your manager about why you should start Mob Programming, you first need to think about the rest of the organization and how mobbing can solve the issues important to management. You also need to be prepared to answer a few questions.

Why have three or more people do the work of one?

This is the most frequently asked question, quickly followed by, "Surely, this isn't a cost-effective way of working?".

While it's true that Mob Programming is not the cheapest way to make software, it is cost effective. In fact, there are many benefits of having a group of people work together on one thing. One of the big benefits with mobbing is flow efficiency.

Resource Efficiency vs. Flow Efficiency

At its core, Mob Programming is flow centric—you optimize things to get features finished quicker instead of getting them done cheaper. To put it another way, you're optimizing for flow efficiency, not resource efficiency.

As Johanna Rothman explains in her book *Create Your Successful Agile Project [Rot17]*, when you work in a resource-efficient way, you get the most skilled person for the specific tasks; this creates experts. Johanna's book explores the theory behind this and why you should avoid it, but here's a quick example to give you a better idea.

Say you have two developers, Mary and Jason, who both get paid the same. Mary is better at front-end work and can do it in half the time as Jason. Knowing this, whenever you need front-end work done, you make sure it's done by Mary because she's quicker at it (Jason's good at other things). That's a resource-efficient way of working.

The problem with working in a resource-efficient way is that, in time, Mary becomes the front-end specialist and Jason doesn't. It may not sound like a problem. But what happens when you suddenly have a lot of front-end work and Mary gets overloaded? You're left with delays and a bottleneck.

Compare this with optimizing for flow efficiency. Remember, flow efficiency is about getting features to market quicker. With flow efficiency, the team works on a new feature together. If anyone needs to be away from work for a day or a week (even two), the team can continue to do the work. Yes, the team might be a little slower, but the feature will still progress toward release.

Optimizing for flow efficiency makes sense when the return on getting something to market sooner outweighs the cost of getting it developed. As it turns out, a lot of software falls into this category, making Mob Programming a cost-effective solution.

Fewer Key-Person Dependencies

Flow efficiency is not the only benefit you get from having a mob work on something versus an individual working on the same thing; you also get fewer key-person dependencies.

When people work on their own, they become the expert on what they're working on. With time, the organization becomes dependent on them. But what happens when they leave? The organization experiences a lot of pain.

When you work as a mob, you build redundancy into the group. Having more than one person know how to do things means the impact on the organization when someone is no longer around is significantly reduced.

Upskilling

In addition to reducing key-person dependencies, Mob Programming helps upskill the less experienced people in your team at an accelerated rate.

Also, when you work on your own, you tend to do things in a certain way. In contrast, when you work as a mob, the team can share ideas with one another. As the sharing continues, the knowledge of the team grows, making everyone more effective and efficient.

Fewer Defects Impacting Our Clients

Last, but not least, having a group working together on one thing leads to increased quality. Multiple people reviewing the code as it's being written results in fewer defects getting to production. Detecting and avoiding defects early on saves money because you have fewer support calls, bug fixes, emergency patches, and so on. Not to mention, fewer defects in production also leads to happier clients.

Better Decisions are Made by Groups

While there haven't been many studies done directly on Mob Programming, several have focused on groups and the quality of decisions they make compared to pairs and individuals. The conclusion: groups are better at making decisions and solving complex problems.

In 1989 a group of researchers performed a study to determine who was better at making decision: individuals or groups. They reviewed 222 teams involved in 25 organizational behavior courses, all 222 teams outperformed their average member with 215 groups outperforming their best member.[a]

In 2006, a set of researchers tried to determine which group size was best at solving complex problems.[b] In their study, 760 people participated, ranging in group sizes of two, three, four, or five people. The researchers concluded that groups of three or more outperformed the best individuals, while groups of two only did as well as the strongest of the two individuals. Their conclusions: having a group of three or more people working together to deliver a piece of work introduces the necessary dynamics for optimal problem solving.

a. http://psycnet.apa.org/record/1990-04483-001
b. http://www.apa.org/news/press/releases/2006/04/group.aspx

How do we measure success?

This is an important question and something most managers want to know, so set the expectations up front: you are not going to be able to measure the success of Mob Programming by measuring the short term "velocity" of the team.

While Mob Programming will increase flow efficiency, you'll need some time for the team to jell before this can happen.[7] This makes measuring success by the numbers difficult. In fact, the short-term numbers will likely show a dip. However, they'll recover and then increase in the long term.

Unfortunately, the problems that Mob Programming solves are complex and therefore difficult to measure; they are also largely dependent on the challenges your team faces. However, here are a few measurements worth considering:

Less time spent on merge conflicts: With Mob Programming, you'll spend less time dealing with merge conflicts, which are sometimes difficult and time consuming. Mobbing can save a lot of time and a lot of headaches.

Fewer defects getting into production: If your team has a high defect rate and frequently releases bugs into production, Mob Programming can help decrease

7. https://www.youtube.com/watch?v=cKH1zyJf5h8

that rate. Because most teams already record bugs that are released into production, this makes for a convenient way to measure the value a team gets from Mob Programming. With time, you should see fewer defects make their way into production. In addition to the overall defect rate decreasing, you should also see a decrease in the impact these defects have on other things.

Less experienced team members learning more: If your team has a big difference in experience and knowledge between the team members, one way to measure the success of mobbing is to ask the less experienced members of the team if they feel like they're learning and growing at an accelerated rate.

Does the team feel this is an improvement?: Although hard to define with a number, the best measurement of success is after a number of mobbing sessions, get everybody together to share what benefits they've observed.

Keep a Wins Log Book

It can be really useful to keep a log book of the wins you have while mobbing, especially when you first start out.

In it, store daily events with specific dates of successes. The sort of things you want to look out for are knowledge share opportunities, quality issues discovered, and so on. In true mob fashion, get everyone in the mob to help maintain it.

The wins log book is an excellent document to refer back to when talking to management.

What do you need from your manager right now?

When first approaching your manager and other team leaders, make it clear that all you want to do, initially, is experiment with Mob Programming. The goal is to determine if the team recognizes the same benefits others have reported.

The experiment should consist of several mobbing sessions, spread over a few weeks. It should be opt in, meaning you'll encourage others to participate but they can choose whether or not they actually want to be involved. At the end of the experiment, share the team's findings, including what worked and what didn't.

Up front, all you need is support for trying something new. That doesn't mean you won't need help down the line. If your team sees value with mobbing, you'll need to have a chat about how to make it more sustainable, but for now, all you need is support.

Joe asks:

Why Foster a Culture of Experimentation?

Having a culture of experimentation in your team goes beyond just experimenting with Mob Programming.

Experimentation is the foundation to working in highly collaborative software development teams. And let's face it, human interactions and software development are complex, and the same interaction doesn't necessarily lead to the same result.

For instance, take human interaction: if I greet you today, I may get a friendly hello in return; whereas tomorrow, you may be having a bad day, and it can be a completely different response. Yes, people are complex.

A useful model that describes why experimentation is important in complex environments is the Cynefin model.[a] It proposes that in complex environments you should use a probe-sense-respond approach, which at its essence, is an experimentation culture.

While I'm only briefly touching this topic, it's worth understanding better. I recommend watching Aurelien Beraud's talk on the topic, which covers it in more depth.[b]

a. https://en.wikipedia.org/wiki/Cynefin_framework
b. https://www.youtube.com/watch?v=aLCabiwShvc

The Disillusioned Mobber

"We tried Mobbing but it didn't go too well." Every once in a while you come across someone who's had a terrible past experience with mobbing.

Usually, when you dig a little deeper, you discover that they heard about Mob Programming at a conference or from a friend, attempted it for a few hours—often with with total strangers or a team that didn't quite understand what they were doing—and ultimately became disillusioned about mobbing because of that negative experience. It's heartbreaking when I hear of people in this position. I've personally seen the many benefits of mobbing, and it's a shame to think what they're missing.

What do you do if someone like this is in your team? The first thing is don't force them to participate. Forcing participation is a recipe for failure (my mother forced me to eat my peas when I was young, and on principle, to this day I still hate them).

Instead, listen to these individuals; hear their concerns. Make it clear that participation in the mob is optional and that while you would love for them to join, it's up to them if they want to participate or not.

If they do decide to be in the mob, it’s important that they judge it on how it’s working now, not on their previous experiences. Often the difference between the two is an open mind, a few tweaks, and a bit more time.

And if it’s you that’s had the bad experience, first off—well done on being open minded enough to read this book. I hope this time around your experience is different.

The success of Mob Programming is determined by so many things: who’s in it, how long you have worked together, what experiences you bring to the table, how you and your mob communicate, and the physical setup of where you mob—and all of these factors have a major impact on the experience. The purpose of this book is to help you get the best outcome possible.

What’s Next?

You’ve learned what Mob Programming is and why it works. You discovered that Mob Programming is flow centric and you explored the benefits of working this way. You also went through some of the typical questions people might ask when they first hear about Mob Programming.

But you’ve only just started your mobbing journey. In the next chapter, you’re going to see how to have your first Mob Programming session and what you need to do to get started.