The Pragmatic Programmers

Pragmatic exPress

Code with the Wisdom of the Crowd

Get Better Together with Mob Programming

Mark Pearl

edited by

Tammy Coron

Early praise for *Code with the Wisdom of the Crowd*

Code with the Wisdom of the Crowd is a shining gem to have emerged from the agile community's relentless search for more meaningful and productive ways of working. Mark has done a sterling job of collating this body of knowledge in a way that will help you navigate past rookie to proficient. Stand on the shoulders of pioneers and enjoy the fruits of this fulfilling and highly effective way of working.

➤ **Ron Quartel**
Independent Consultant, Cron Technologies

Answers all the questions I had about mobbing—why, who, how, etc.—and opened my eyes with great insights about mobbing that I hadn't thought of. Essential.

➤ **Jeff Langr**
Developer, Author, Consultant, Coach, Trainer, Langr Software Solutions

Software development is a "team sport." This book offers valuable insights and practical steps on how technical leaders and software teams can apply mob programming techniques effectively. I highly recommend it to anyone who values collaborative software development.

➤ **Darren Sim**
Lead Technologist, IAG New Zealand

Full of practical tips and real-life examples that will accelerate your adoption of the practice. A must read!

➤ **Erwann Jooris**
Agile Lead, Stuff

We've left this page blank to make the page numbers the same in the electronic and paper books.

We tried just leaving it out, but then people wrote us to ask about the missing pages.

Anyway, Eddy the Gerbil wanted to say “hello.”

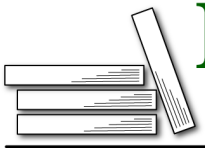
Code with the Wisdom of the Crowd

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

Contents

Acknowledgments

Code with the Wisdom of the Crowd could not have happened without the support of my wife, Genevere, and my three children, Daniel, Juliet, and Jamie. I'm so grateful for your patience and understanding, especially when I had to spend time away from the family to be working on the book.

Many thanks to Tammy Coron for all your help throughout the writing process; your feedback and guidance was invaluable.

Thank you to my technical reviewers for your suggestions and insights—to Allan Stewart, Daniel Irvine, Eric Jutrzenka, Gareth Stephenson, and Janco Wolmarans—each of you contributed something and helped shape the book.

A special thank you to Allan, Eric, and David for taking the time to share their craft with a total stranger. Who would have thought a single day would have such a major impact on how I make software?

Finally, thank you to my original mob: Janco, Theo, Steven, Oz, Gwen, Coenie, Mandla, Lindo, Sandile, and Hendrik. We had a great run together, mobbing with you was so much fun—this one is for you.

Preface

Welcome to Mob Programming—an exciting, emerging form of collaborative programming. The outcome of effective Mob Programming is great software produced by teams; it has fewer defects, requires fewer key person dependencies, and is more robust, better thought out, and just flows. People all over the world[1] are realizing the benefits of mobbing, and now you can too.

About This Book

Having helped several teams start mobbing, I've found there are two big hurdles most teams encounter: the early days, when you're just getting started and still figuring things out, and then a few months later, after the shininess of mobbing has worn off. This book is here to help you get past both of these obstacles to a point where your mob is sustainably humming.

Code with the Wisdom of the Crowd is written for a range of audiences, including:

- Team leads who want to introduce mobbing to their team.
- Software developers who haven't tried Mob Programming before or who have previously tried and had a negative experience due to poor execution.
- Coaches who would like to introduce Mob Programming to their development teams.

Because this book takes you on a journey of mobbing, I recommend you read it start to finish and then come back to the sections that apply to where your mob is at that point in their journey.

With that, let's get started—it's exciting to be with you on your mobbing journey.

1. https://markpearlcoza.github.io/Talk_PeopleMobbing

Online Resources

This book has a dedicated page on the Pragmatic Bookshelf website.[2] If you find a problem with the text, please report it using the errata submission form. There are no additional online resources.

2. https://pragprog.com/book/mpmob

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 84.

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.

CHAPTER 2

Starting to Mob

This chapter covers some common things you need to do to get started with your first mobbing session. It also includes a recipe to follow when you're just starting out. This recipe is there to help you effectively complete your first mobbing session—however, after you have several sessions under your belt, you'll develop an intuitive understanding of how mobbing works and what adjustments are needed. As you make these adjustments, update the recipe. But for now, follow it as-is, and by the end of this chapter, you'll be hands-on with your first mobbing session and in a good place to keep the momentum moving forward.

Preparing for Your First Mob

Before starting your first mobbing session, there is some up-front preparation that must be done. This includes:

- Framing mobbing as an experiment
- Deciding who's in your first mob
- Finding a place to mob
- Finding a problem to solve
- Configuring a machine for mobbing
- Organizing a mob timer

Framing Mobbing as an Experiment

When you first start Mob Programming, it's important to frame it as an experiment. Framing it as an experiment is helpful in setting expectations, and it goes a long way to establishing a psychological safety net for those participating.

When speaking to others, let them know that you heard about the benefits of Mob Programming and you want to try it out to see if it's useful. Then, find

out their level of interest in trying it with you. Make it clear that it's temporary and will take the form of several mobbing sessions. After each session, and as a group, you'll evaluate what worked and what needs to be adjusted. After several sessions, you will once again as a group determine if there were enough benefits to proceed. The number of sessions is largely determined by the group's willingness to participate, but you should aim for no less than five times.

Deciding Who's in Your First Mob

While you're experimenting with Mob Programming, keep the mob small. A good starting size is about three to four people in a single mob. If more people want to get involved, have multiple mobs instead of one large mob. With larger mobs, more storming can occur because of the increase in the number of people, and this can have a negative impact on the experiment.

Tuckman's Model and Storming

The Tuckman model of group development was first proposed in 1965 by Bruce Tuckman. He explained that there are four phases that a group goes through when working together: forming, storming, norming, and performing.

We all work slightly differently and sometimes these differences result in disagreements or cause others to become frustrated. When people begin discovering these differences, resolving disagreements, and figuring out how to work with each other, it's called storming.

With Mob Programming, storming usually starts early on as people discover the differences in working styles a lot sooner when compared to teams that work more independently. Storming continues until these differences are resolved.

While storming can be hard to go through, it's not necessarily a bad thing—once things are worked through successfully, the result is a greater intimacy and understanding of those around you. You can reduce the amount of storming by getting a mob to understand each other better up front (see *Running a Group Session on Personality Types*, on page 46). Storming happens whenever a mob forms or a new member joins a mob—and it's something to be on the look out for.

Often when you start mobbing, people from different teams get curious and want to get involved. However, it's important to keep your first few mobbing sessions limited to people from the same team. People from the same team already have a lot of alignment on practices because they've worked on the same code base, used the same or similar tools, and have an understanding of how to work together—all these things can reduce the amount of storming the mob might go through.

Also, be specific about who to include in the mob. Knowing exactly who's participating in your first few mobbing sessions can help to avoid some of the minor setup issues, like having the right editor installed or the correct shortcuts configured. Both of these things can have a major impact on the overall mobbing experience.

For the first few mobbing sessions, only include people who are able to code so that everyone has an opportunity at the keyboard. This doesn't mean that others like dedicated non-coding Business Analysts or Quality Analysts can't be involved; it just means that for now, you're going to keep the group small and focused. Later, when mobbing becomes more regular in your team, you'll be able to better decide who will be part of the mob (more about that in *Who Should Be in the Mob*, on page 72).

Now that you have a better idea of who should be in the mob, it's time to find a suitable place to do it.

Finding a Place to Start Mobbing

Having a suitable area to mob is important. When searching for a place, look for something that can comfortably fit the entire mob and allows everyone in the mob to see the code at the same time. Unfortunately, many team areas are not designed for this sort of thing, but most workspaces have meeting rooms.

Meeting rooms work well for beginner mobs because they're usually insulated away from the rest of the office giving you a private place for a group to experiment. They also tend to have two key tools necessary for mobbing: a large screen to view the code and a whiteboard.

While the large screen does not require much explanation, often overlooked but almost as important is the whiteboard. The whiteboard is a critical tool because it provides a place where the mob can put reminders, explain ideas, and brainstorm together. If you don't have a whiteboard in the room, get a portable whiteboard with non-permanent markers.

The key to selecting your mobbing space is to select one that helps people feel comfortable and connected. Avoid places that are too cramped or too spread out. Having a mob crammed in front of a single cubicle does not lend itself to a great mobbing experience. Likewise, mobbing in a large room that forces everyone to be spread out results in a poor disconnected experience.

Keep distance in mind when choosing a meeting room. Prefer places closer to your normal work area. I once worked with a team who chose a meeting room that was a fair distance away from their normal team area. While it was easy to get people to attend the first session, we struggled with attendance later because it required too much effort to get to the meeting room. As soon as we decided to use a meeting room next to our team area, mobbing started again. Usually the closer the meeting room is to your normal working space the more frequently you will mob.

Joe asks:

Can I Use Post-it Notes Instead of a Whiteboard?

One of the big benefits that a whiteboard has over Post-it notes (even the large format Post-it notes) is the space the whiteboard gives you to draw ideas on.

Post-it notes work great for recording written ideas and moving them around, but are just too small for drawing diagrams that the whole mob can see at a glance.

Certainly you can get the best of both worlds by using a mix of Post-it notes and the whiteboard but I wouldn't recommend using Post-its as a replacement for a whiteboard.

Selecting a Suitable Problem to Solve

For your first few mobbing sessions, find a problem that does not have a hard delivery date, and one in which everyone is on an equal footing. I recommend doing a code kata; they're designed to help programmers hone their skills and are generally something from which everyone can learn. There are a bunch of katas available at Pragmatic Dave's CodeKata site.[1] Choose one that appeals to the whole mob and that will take the full session to complete.

1. http://codekata.com/

Avoid katas that somebody is already an expert on; when there's an expert in the mob, things work differently (this is covered more in *Mobbing with an Expert*, on page 50). For now, work on a problem that is new to everyone in the mob.

Joe asks:

What's a Code Kata?

Kata, originally a Japanese word, is a series of detailed choreographed patterns of movements practiced either solo or in pairs to teach proven techniques and practice in martial arts. They can also be preformed in large groups, which is beautiful to watch when all members are synchronized.

A code kata is a programming problem suitable for practicing specific programming patterns found in software development. The concept was first popularized by Dave Thomas, co-author of the book *The Pragmatic Programmer*.

Configuring a Machine for Mobbing

While it may seem like a small thing, having a machine pre-configured for the mob before the session starts has a big impact on the overall mobbing experience.

Often, the first mobbing machine will be a laptop. Note that laptops present a couple challenges: (a) they are hard to pass around, and (b) they usually have custom keyboard layouts, which can be challenging for anyone not accustomed to that specific keyboard layout. If you're going to mob on a laptop, bring a standard external keyboard to the mobbing session. Standard external keyboards can be passed around and they have a layout that most people can adjust to using.

In addition to having external keyboards, the mob machine should have a code editor that everyone in the mob knows how to use. Also, make sure the editor's font sizes are adjusted (so that everyone in the mob can clearly read the code from a distance), the keyboard shortcut mappings are set to ones people are comfortable with, and the line numbering is enabled.

Choosing an Editor

For many, the discussion on which editor to use will be quick. If everyone in the mob already uses the same editor, this will be the editor you use in the mob.

For some, different people in your mob will have preferences for different editors. Bringing these people together in a mob raises the question: what editor do you use when you are mobbing?

In the past, I've tried having everyone's preferred editor installed on the mob machine, but the experience wasn't great: every time the typist would swap, we'd go through the pains and delays in switching editors too.

Instead, I recommend using a single editor. If your mob has a majority of people that use a particular editor, go with that one. If there's no majority, use the one everyone's most comfortable with. Be mindful of individuals that aren't using their editor of choice. Acknowledge they may be under some initial stress as they learn the new, shared editor, and you appreciate their sacrifice.

I've also seen a new mob try to use a completely foreign editor that no one has used in the past. My recommendation is to avoid doing this unless there is a real need to do so. Learning a new editor, while solving a coding problem as a mob, can cause unnecessary confusion and frustration—both of which you want to minimize. Bottom line: you're either solving a code kata together as a mob or you're learning a new editor—not both.

A Word on Vi and Vim

I'm a Vim fan. I've been using it for years, and it has significantly changed the speed at which I can write and morph code. If you're a Vim fan and others in the mob are not, don't start with Vim—the learning curve is too steep. Rather, choose an editor that everyone can use. With time, as your mob settles, you can look at showing them the light of Vim—but for now, for the sake of the mob, skip Vim.

Keyboard Shortcut Mappings

There are likely members in your mob who have spent time committing keyboard shortcuts to memory. Have a discussion with them and the rest of the mob about which shortcut mappings to use. Start off by identifying the shortcuts people are currently using. Don't assume that just because everyone in the mob uses the same editor they'll all have the same keyboard shortcut mappings.

If everyone in your team uses the same shortcuts, great. If, however, you have different people using different shortcuts, you have a few options.

Some editors might let you seamlessly switch between different keyboard shortcut mappings. However, many do not provide such an experience. Instead, they require you to navigate through a series of setting screens to find the mappings you want to use. Once you make your selection, you then need to apply it, exit the editor, and then reopen it. If your editor falls into the latter category, avoid switching shortcuts—it takes way too much time. Instead, use the editor defaults or standardize on a set of mob keyboard

shortcut mappings, and have those shortcuts printed and put in a place that whoever is typing can easily see.

Line Numbers

Be sure to enable line numbers in your editor. Being able to see the line numbers significantly speeds up the on-screen code navigation.

Line Numbers Off

```
public void CostOfFoo() {
    const costPerFoo = 10;
    const costPerBar = 15;
    return costPerFoo + costPerBar;
}

public void WriteFooBar() {
    var foo = "foo";
    var bar = "bar";
    var foobar = foo + bar;
    if (foobar.Length() > 10) {
        Console.WriteLine(foobar)
    } else {
        Console.WriteLine("Something else")
    }
}
```

Line Numbers On

```
32
33  public void CostOfFoo() {
34      const costPerFoo = 10;
35      const costPerBar = 15;
36      return costPerFoo + costPerBar;
37  }
38
39  public void WriteFooBar() {
40      var foo = "foo";
41      var bar = "bar";
42      var foobar = foo + bar;
43      if (foobar.Length() > 10) {
44          Console.WriteLine(foobar)
45      } else {
46          Console.WriteLine("Something else")
47      }
48  }
```

When navigating, use absolute line numbers rather than relative line numbers. Absolute line numbers are absolute, regardless of where the cursor is positioned; relative line numbers change as you move the cursor up or down. The latter can create confusion. For example, saying, “Move down five lines,” is more error prone than saying, “Move to line 45.”

Organizing a Mob Timer

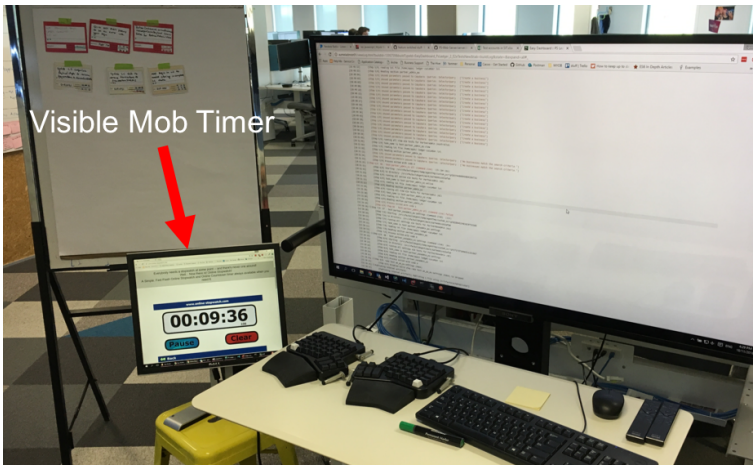
One final thing to organize before the mobbing session is a mob timer. You will use the timer to break a mobbing session up into smaller mobbing intervals; you’ll see how this works in more detail later on.

In the past, I’ve used various timers for mobbing, from cell phones to on-screen timers. I recommend getting a timer that can be clearly visible to the entire mob at all times during the mobbing session as shown in the figure on page 20. One benefit of making the timer visible is not being dependent on the audible alert. If the timer is visible, you have another way to notice when the time has elapsed.

There are various timers that have been specifically designed for mobbing, I recommend you try one of the following:

- Agility Mobbing & Retrospective Timer[2]
- Pluralsight’s Mob Timer[3]

2. https://agility.jahed.io/timer.html
3. https://github.com/pluralsight/mob-timer



A Recipe for Mobbing

The first mob session should be about two hours long; this gives enough time for the mob to get momentum without people getting too exhausted.

For your first mobbing session, let's break it down into a beginning, a middle, and an end.

At the beginning, you set the stage for the mob. This means explaining the two primary roles in mobbing and how each one works, selecting the problem the mob will solve, and the order in which the typists will rotate.

In the middle, there are a series of timed mobbing intervals where everyone works on the problem together as a mob, with a new typist taking over once each interval is finished.

At the end, the mob briefly reflects on how the session went and looks for things they can do to improve the next session.

Setting the Stage

When the mob gets together for the first time, start by introducing the concept of Mob Programming. To help guide this conversation, work through the next few sections in this chapter; I've included a set of complementing online slides[4] that you can use as a companion to these sections.

Set the stage by discussing what it means to work together on one thing as a mob.

4. https://markpearlcoza.github.io/MobProgrammingIntroduction

Basic Format for Your First Mobbing Session with Timings

Here's a rough outline of your first mobbing session, with timing:

The Beginning: Setting the stage (±30 minutes)

- Talk about working together on one thing (10 mins)
- Explain the roles in mobbing (8 mins)
- Overview of the problem the mob will solve (10 mins)
- Decide on the order of the typists (2 mins)

The Middle: Mobbing intervals (±1hr 30 mins)

- The first person on the list takes the keyboard (they become the typist)
- Start the timer on a 10-minute countdown
- Mob works together on the problem
- When timer reaches zero, the typist stops and the next person on the list becomes the new typist
- Reset the timer and repeat the previous three steps until 20 minutes before the end of session

The End: Learning from the experience (±20 minutes)

- Have a closing mob retrospective and discuss what went wrong, what went well, and how you can improve in the future

Working Together on One Thing as a Mob

The important points to cover are:

- Mob Programming is about a group of people working together at a single computer to solve a problem together.
- Effective mobbing requires each person in the mob to cultivate a flow mindset which means that as a group you work together to continually progress the problem toward completion.
- Cultivating the skills necessary for a flow mindset takes practice but is not impossible—teams all over the world have been able to do this.[5]

What does it look like when a group of people work together at a single computer to solve a problem together? To better understand this, as a group, watch this short time-lapse video of people mobbing[6] as shown in the figure on page 22.

5. https://markpearlcoza.github.io/Talk_PeopleMobbing
6. https://www.youtube.com/watch?v=Ev7uus12HRY

I've found watching others mob first gives you a better idea of what the interactions can look like than reading about them—turns out some things are just easier to understand by observing someone else do them.

After watching the video, it's time to go into the roles of mobbing. There are two main roles in mobbing: the typist and the rest of the mob.

The Typist

The typist is the person behind the keyboard. There's only *one typist* at a time in a mob. The typist *is not the programmer*; instead, think of the typist as a smart assistant, taking what the rest of the mob is asking them to do and implementing it.

Llewellyn Falco, an early advocate of Mob Programming, explained it best when he said, "For an idea to go from someone's head into the computer it must go through someone else's hands first." When you're the typist, you are predominantly the hands, not the head.

To be effective, the typist needs to:

- *Understand* what the rest of the mob is asking them to do
- *Ask* clarifying questions when they are unsure
- *Implement* what they are being asked to do in the code
- *Trust* the rest of the mob and be comfortable exploring approaches they would not normally explore
- *Learn* new things, like keyboard shortcuts and how their tooling works from others

When you're the typist, understanding what the rest of the mob is asking you to do can be a challenge. Sometimes you won't understand what's being asked. When that happens, speak up and ask clarifying questions until you know how to implement the code the mobs wants.

Trust is an important factor in being an effective typist. The best typists realize the rest of the mob may have a differing approach to the one they would personally take and are comfortable allowing that approach to be fully explored. Don't prematurely challenge things; rather, implement what's being asked and then review things.

Prematurely Challenging

As the typist, hold off prematurely challenging what you're being asked to do by the mob. If you haven't written the code yet, it's too soon to challenge it.

In my early days of mobbing I would challenge things before they had been expressed in code. When I did this, I found the mob would often have lengthy debates and little progress. After changing my approach to only challenging things after the code had been written, I noticed that work generally progressed faster.

Looking back, when I was challenging prematurely, one of two things was happening: either we were saying the same thing but with different words (and we didn't realize it because we couldn't see the code yet), or the approach the rest of the mob wanted me to take was appropriate, and I just didn't see it yet.

Only after an idea has been expressed in code should the typist request an explanation. Because typists change frequently, it's important they understand the code. If not, they won't be effective when moving back as part of the rest of the mob.

But what if you're the typist and you have an idea you think is worth implementing? You can either explain the idea to the rest of the mob and gain agreement for its implementation, or you can request someone else take your place as the typist and rejoin the rest of the mob—if you rejoin the rest of the mob, the mob interval resets and the person taking the keyboard becomes the new typist.

Typists Sneaking Code In

Occasionally I've seen typists sneak in code. One memorable instance happened with a team that was trying mobbing for the first time. Everyone was at the whiteboard discussing what approach they would like to take, except for the typist, who had snuck away to the keyboard and was already coding his solution.

If you're the typist, don't sneak code in—you need to let go and collaborate.

Finally, whenever you're the typist, it's a golden opportunity to learn different ways of using your tooling from others. Be receptive when others in the mob point out alternative ways of doing things or new keyboard shortcuts that will speed up implementation. You might be surprised how these little improvements will make you quicker and more effective.

The Rest of the Mob

As someone who's not the typist but still part of the rest of the mob, there are a few things expected of you. You should:

- *Help discover* what the next logical step is to solving the problem
- *Ask questions* until you understand
- *Contribute* to increasing the level of understanding in the mob
- *Focus* on the problem at hand
- *Listen* to others
- *Anticipate* needed information
- *Look* for areas to improve the system

As part of the rest of the mob, you're part of the problem-solving team. It's your job to help discover what the next logical step is to solving the problem at hand; this is done by working collaboratively with others and asking questions until you understand.

While there are always many angles and approaches to a problem, strive to get consensus on what the next logical step is. As part of the rest of the mob it's expected of everyone to actively contribute, talking at levels that everyone else can understand.

For example, if there is an opportunity to simplify the code by extracting a method, asking the typist to do so may be a sufficient. If they have a blank stare, be more specific, "Highlight line 114 to 127 and then press Ctrl+Alt+M to extract method."

When you're part of the rest of the mob, it's all about communication. You need to be able to clearly express your thoughts and make yourself understood.

Often, we think communicating effectively is focusing on what we should say; while that's important, it's only part of the skill. Real effective communication is about listening and hearing others. Next time you're struggling to convey an idea, check if you're listening. Chances are, you're not, and that's half the battle.

Listen to others. Be mindful of everyone in the mob, especially those that are not being heard.

Communicating Effectively

When you're part of the rest of the mob, communicating effectively is one of the most important skills to develop. It takes time and practice to get it right. I remember an experience I had with a new mobber, Kerrie.

Kerrie was a talented developer but lacked good communication skills. At one point, Kerrie wanted the mob to try something but couldn't articulate it well. You could see the frustration on Kerrie's face, laughing and saying, "This is hard, if only I could just show you."

But instead of swapping in to be the typist, Kerrie took a thoughtful moment and explained the concept again, this time listening carefully to our responses and clarify areas we didn't understand—and it was a success. Over time, and with practice, Kerrie got better at communicating ideas to the mob.

Sometimes the mob will get sidetracked and lose focus—look out for this. When it happens, bring it to the group's attention and then help get things back on track; park ideas that aren't related to the immediate problem at hand.

Anticipate needed information and proactively find it. You will be surprised how much it helps to have someone pre-screen documentation before its needed and highlight what's relevant. If you see an opportunity to do this, take it.

Finally, look for areas to improve the system. Things like repeated tasks and processes should be automated. If there are keyboard shortcuts and editor tricks that others do not know, help them learn. Identifying code smells and unnecessary complexity, and finding ways to simplify things, are part of the job.

Overview of the Problem You Will Solve

Now that the mob has an understanding of what's expected of the two main roles in mobbing, it's time to introduce them to the problem they will be solving. An appropriate problem should have already been selected in preparation for the session (see *Selecting a Suitable Problem to Solve*, on page 16).

As a mob, there needs to be alignment on the general plan of attack on the problem. Get everyone to brainstorm different approaches to solving the problem at the whiteboard but limit the conversation to, at most, 15 minutes. Once there is an agreed approach, write down the high-level steps on the whiteboard leaving detailed implementation specifics for the actual mobbing intervals.

Deciding the Order of the Typists

After going through the problem, the group needs to decide on the order of the typists. Everyone in the mob will have an opportunity to be the typist. To make sure of this, only repeat the typist role after everyone else in the mob has had the opportunity.

How you decide on the order of typists is up to you, but it's best to write the order down somewhere so that everyone can see and follow it. If someone really wants to start, put them at the top of the list. If nobody is eager to start, use a random order and write it down—it doesn't matter who's the first typist or what order you go in. As long as the order is clear and everyone has an equal opportunity, you're good.

Occasionally, you will find people in your mob who will be hesitant to be the typist but are happy to be in the rest of the mob. If watching makes someone feel safe, that's perfectly acceptable—safety in a mob is important. That said, you want to actively encourage everyone to participate as the typist—there's a difference between watching others solve a problem and getting your hands dirty. When in the mob, the act of working the keyboard helps you get immersed in the problem and engages your problem-solving abilities.

Sometimes people are hesitant to be the typist because they're not the expert in that area of the code or they are worried that they'll slow everyone down. First, being the typist is the best place to be when you're not the expert—by the very definition of the typist role, you just need to do what the rest of the mob is asking, while allowing the rest of the mob to be the experts. Second, there's a difference between knowing and doing; the goal is to get people to the point where they are able to do. Being the typist accelerates being able to "do it".

Mobbing Intervals

You've selected your first typist and you're ready to start the first mob interval. Wait, what's the difference between an interval and a session? Let me explain.

A mobbing session is a continuous period of time where a group of people are mobbing together (usually several hours). A mobbing interval is a short period of time (usually just a few minutes) with a single person as the typist. A mobbing session consists of a beginning, a series of mob intervals, followed

by a wrap-up at the end. The vast majority of your time mobbing is spent in mobbing intervals.[7] Let's take a look at how they're structured in a bit more detail.

Start the First Interval and Mob for Ten Minutes

You start the mob interval by starting a ten-minute countdown timer. Ten minutes works well because it gives everyone in the mob an opportunity to be the typist several times in the session.

Now comes the difficult part: deciding where to start. Different people will like to approach the problem from different angles; some prefer a top-down approach, while others may advocate a bottom-up approach. Because I'm a fan of test-driven development (TDD) I like to start with the question, "What's the first useful test we can write?" If you don't do TDD, asking, "What's the first useful step?" is a good alternative.

Often in the first interval, a new mob doesn't get much code written. Instead, there's still a lot of alignment happening. While this is fine, you want to push toward getting to code. Getting to code moves the conversation from theoretical to practical. If you have varying ideas of how to approach the problem, go with the simplest one first. If it works, it's usually a good enough solution; if it doesn't work, its flaws usually show up quickly when you try to code it. If you have multiple ideas that are equally simple, narrow it down to one—something you learn from Mob Programming is that there are many ways to solve a problem.

Changing Typists at the End of First Interval

As you progress through the first interval, keep an eye on the timer. As soon as the timer runs up, end the interval and change to the next typist. There's always the temptation for the typist to "finish" the thing they were on—resist this urge. You want the swap to a new typist to be quick. After the switch, reset the mob timer to ten minutes and start the countdown again. You may also want to take this moment to commit your work to version control.

7. For an illustration of the mechanics of mobbing intervals, watch this short animation (https://www.youtube.com/watch?v=DRo6wWuQY2Q).

Micro Commits in Your Version Control

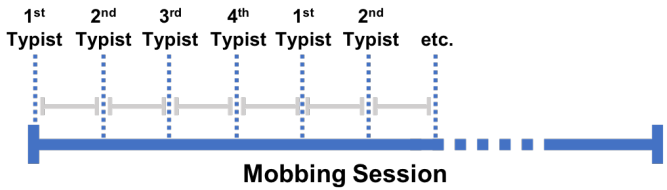
I'm a big fan of Git because it makes commits really fast. One approach I really like to use when mobbing is to make a temporary feature branch at the beginning of a session. As each interval in the session lapses, we make a "micro" commit which is a normal commit without any time spent on the commit messages. At the end of a mob session, or once we have finished an area of work, you can "squash" these micro commits into a single, larger commit with a meaningful message and merge this into the mainline.

Using this approach gives you a series of granular snapshots allowing you to revert code easily if you feel you've gone down a dead end.

What you realize after the first ten-minute interval is that ten minutes is a really short time to get code written, and yet it's still possible to make progress. Sticking to a short time interval helps keep everyone engaged.

Subsequent Intervals

After your first interval, you have a pattern for subsequent intervals during the mobbing session: work together on the problem as a group, and change typists at ten-minute intervals.



During these times, you'll have spurts when everyone is in agreement on what to do next and it's just a matter of coding it. You'll also have periods where you need to explain ideas at the whiteboard. If someone is the typist, and the mob spends their entire interval at the whiteboard, give that person the next typist's interval once you resume coding.

During each interval, be aware of your current role in the mob. If you're not currently the typist, you should not be touching the keyboard. People that mob for the first time really battle with this—it's not uncommon to hear comments like, "It is so frustrating, if I could just show you quick." With time, as everyone gets better at explaining things and listening to others, this becomes less of an issue.

Keep coming back to the high-level steps you outlined on the whiteboard at the beginning of the mobbing session. If the mob decides to change the general approach, update your high-level steps accordingly. Use a section of the

board to maintain a todo list. The todo list is where you can park ideas and tasks that will distract you from the problem at hand but that are still important to look at.

With three people in a mob working at ten minute intervals, everyone will have, at most, three opportunities to be the typist in the first session. Keep an eye out on the clock, as you might be surprised how quickly time flies during the mobbing intervals. And don't forget to finish the intervals 20 minutes before the end of the session so you have enough time for the wrap-up and retrospect.

A Word on Version Control

When Mob Programming, there's always the question of whose credentials to use for code check-in.

For your first few mobbing sessions, you aren't going to get caught up in this. For now, simply commit the code as one of the people in the mob and follow your normal review process. Later, when you make Mob Programming a regular practice in your team, you can review how to handle this (see *Shared Mob Credentials*, on page 70).

Learning from the Experience

For the last 20 minutes of the session, have a closing mob retrospective where everyone who participated in the mob can share insights and put forward suggestions on what to adjust for next time.

There are several ways to do this, my favorite is to use a technique borrowed from Edward de Bono's book *Six Thinking Hats [de 99]* that separates thinking into clear functions, keeping the discussion on track.

Thinking Hats

To do Thinking Hats effectively, you need blank Post-its and a Sharpie, and you can re-use the whiteboard you used for mobbing.

As the person facilitating, it's up to you to make the notes on what's being discussed, enforce time limits, and keep the conversation on track and moving. The retrospective is broken up into the following steps as shown in the table on page 30.

Step	Hat	Time
Explain How It Works	-	4 mins
Focus on Facts and Figures	White Hat	2 mins
Focus on Positive Thinking	Yellow Hat	3 mins
Focus on Critical Thinking	Black Hat	3 mins
Constructive Problem Solving	Green Hat	5 mins
Decide and Agree on Adjustment	-	3 mins

Step 1 - Explain How Thinking Hats Works

Draw four columns on the whiteboard. In each column, write the color hat, focus, and time limit.

Explain to the mob that as a group you're going to review the mobbing session with four distinct hats. Everyone will figuratively wear the same hat, at the same time, and discuss the mobbing session from that hat's perspective.

Each hat is worn for a limited time. When the time runs out, everyone will take off that hat and put on the next one. Repeat this process until all of the hats have been worn.

If during a discussion someone starts wearing the wrong hat, it's OK to interrupt them and bring the conversation back to focus on the current hat. You are the facilitator, after all.

In addition, and as the discussion is happening, write each idea on separate Post-its and stick them in the appropriate column on the board. A consensus isn't needed for something to go in a column. If one person feels a certain way about something, it's valid and can be put up, provided it meets the criteria for that hat.

Before moving on to step 2, make sure everyone understands how this works.

Step 2 - Focus on Facts and Figures (White Hat)

Start the timer and spend 2 minutes on facts and figures. Everyone in the mob shares facts about the session. Some examples of facts that people might give are:

- We changed typists every 10 minutes
- John was typist twice
- The editor font was font size 16

Be on the lookout for people wearing the wrong hat. A common mistake is for people to state opinions as fact.

- We spent spent two intervals on a *simple* part of the problem
- We solved the problem *quickly*

Both the above statements are opinions because *simple* and *quickly* are opinions, not facts. If someone states an opinion as a fact, get them to rephrase it to remove the opinion portion of their statement. A good indication that something is an opinion is if it can be taken as a positive or a negative statement. If we restated the previous two opinions as facts they would look something like this:

- We spent two intervals on the calculation part of the problem.
- We solved the problem in 2 hours 15 minutes.

Once the time has run out on focusing on facts and figures stop the discussion and move on to Step 3.

Step 3 - Focus on Positive Thinking (Yellow Hat)

Start the timer and spend 3 minutes for the group to come up with positive thinking around the mobbing session. Anything positive about the experience is appropriate.

Some questions you can ask to get the ball rolling include:

- What worked best?
- What was easier than expected?
- What did you like about the session?

As people suggest things, write them down and place them in the Positive Thinking column. Once the time has run out, stop the discussion and move on to Step 4.

Step 4 - Focus on Critical Thinking (Black Hat)

Start the timer and spend 3 minutes on critical thinking. In their opinion, what didn't feel right, what irritated them?

As people suggest things, write them on stickies and place them in the Critical Thinking column. Again, you don't need consensus for something to be valid. If one person feels critical about something, it should go in the Critical Thinking column.

Keep an eye out for people combining solutions with criticisms. For example, "I didn't like the ten-minute intervals, let's do five minutes going forward," is a combination of a criticism and a solution. Compare this to, "I didn't like the ten-minute interval," which is just the criticism. If people combine criticisms with solutions, get them to rephrase it with just the criticism.

Once the time has run out, stop the discussion and move on to the next step.

Step 5 - Focus on Constructive Problem Solving (Green Hat)

Start the timer and spend 5 minutes on constructive problem solving. Get the group to look at the items listed in the Critical Thinking column and think how they can improve those things. During this step, people are just suggesting ideas, and every suggestion is valid.

Once the time has run out, stop the discussion and move on to the final step.

Step 6 - Decide and Agree on an Adjustment

Start the timer for 3 minutes. You now need to decide and agree on one adjustment for the next mobbing session.

Ask the group which suggestions in the Constructive Problem Solving column they'd like the mob to try first. Often groups make the mistake of wanting to change too many things at once; encourage the mob to settle on just one thing.

If it's not clear what the one adjustment is, reiterate that the adjustment is just an experiment for the next mobbing session. If it doesn't work, you can always revert it.

If you get consensus on an adjustment that everyone is willing to try, write it on a Post-it and place it in the final column. Make sure the specifics of it are clearly understood.

If you can't get consensus to try something, don't change anything. Let the mob know that you'll repeat the exercise after the next mobbing session and that you'll need them to get consensus on one adjustment to try out.

Thank everyone for participating in the mobbing session and close the mob retrospective.

You've Completed Your First Mobbing Session

With the close of the mob retrospective, you've completed your first Mob Programming session—well done. You have now applied the basic recipe of mobbing.

Follow up the first mobbing session with additional mobbing sessions over the next few weeks, focusing on katas or problems where there's no expert and no heavy deadlines.

What's Next?

In this chapter, you learned how to prepare for and have your first mobbing session. You discovered that you can break a mobbing session up into intervals and what each person in the mob should be doing. You also went through the process of a mob retrospective and found out what to change in subsequent mobbing sessions.

With your basic recipe of mobbing in hand, it's time to dig a little deeper. In the next chapter, you'll focus on the people in the mob and why they're just as important as the programming.

CHAPTER 3

People and Mobbing

As you do more mobbing, you'll come to realize that mobbing isn't just about the code, it's also about the people. In 2017, through a Mob Programming Community survey,[1] I asked others about the challenges they've experienced with mobbing. Not too long into things, I noticed a pattern emerge in their responses:

- "Getting along. If you can solve the interpersonal issues, then software just flows."—Chris
- "The culture in the team and in the company must encourage feeling safe to express emotions and opinions."—Andrea
- "Some may be too stuck on their own ways to be open to other's ideas."—Jon
- "If a team already has personality issues, then implementing Mob Programming will be difficult."—Natasha
- "You have to accept opinions of others and accept you may not be right. Personality differences can cause collisions in mob."—Thomas

It should be no surprise that one of the biggest challenges with having a successful mob is creating an environment where people can get along and work effectively together. For a team to get good at mobbing, they have to get good at understanding people.

In this chapter, you'll focus on the people within the mob, and how you can help them (and you) be more effective at understanding each other and working together.

1. http://blog.markpearl.co.za/Mob-Programming-Industry-Survey

It Starts with Empathy

At its core, having empathy for the people in your mob helps you to be more effective with them. Although empathy means different things to different people, in Mob Programming, empathy is best applied using these three attributes (for more on this, see Theresa Wiseman's study):[2]

- See the world as others in your mob see it.
- Be non-judgmental.
- Emotionally connect with your fellow mobbers.

Take a moment to understand each of these attributes.

Seeing the World as Others in Your Mob See It

Seeing the world as others in your mob see it is often called cognitive empathy or perspective taking. It's a deliberate practice, and one that everyone can learn.

A common situation in mobbing where this plays an important role is when deciding the best approach to solving problems as a group. Because there are many ways to approach problems, being able to listen to alternative approaches from others, and understanding why they might be suggesting them, is an example of applying cognitive empathy.

Improving Your Cognitive Empathy Through Active Listening

One way to improve your cognitive empathy is to practice active listening. To be an effective active listener you should:

- Pay attention to who's speaking, ignore distractions, and put your own thoughts on hold.
- Show that you're listening, give visual queues like nodding your head.
- Provide feedback on what you've heard, paraphrase or repeat it back to make sure you've got the important points.
- Avoid interrupting, let them finish before you start speaking.

The short video "Improving Your Listening Skills with Active Listening"[a] is worth watching and digs into practicing active listening.

a. https://www.youtube.com/watch?v=t2z9mdX1j4A&feature=youtu.be

2. https://onlinelibrary.wiley.com/doi/abs/10.1046/j.1365-2648.1996.12213.x

Be Non-Judgmental

The next attribute that helps you practice empathy in mobbing is to be non-judgmental of others, and it's important to understand what, exactly, that means (as well as what it doesn't mean).

As Raj Raghunathan[3] explains in his article "Don't Be Judgmental, Be Discerning," being non-judgmental doesn't mean being blind to the differences between things—that's being discerning[4]. Raghunathan describes discernment as the ability to notice objective differences among things (activities, people, objects, etc.) along relevant dimensions.

For example, a discerning software developer with experience in writing unit tests can recognize differences in the quality of unit tests written by others, whereas a non-discerning software developer cannot. This is perfectly acceptable behavior—being discerning is valuable.

When you're judgmental, you go beyond discerning differences in other people's abilities to making inferences about their overall value as a person. To a judgmental person, someone who writes bad unit tests is inferior not just on that dimension, but inferior as a software developer and as a human being. Making the leap from bad unit tests means you're a bad software developer which in return means you're bad person is toxic and something you should never do.

Be Critical of the Code, Not the Person

A great piece of advice I was given from a fellow mobber was to be critical of the code, not the person.

Instead of saying, "I think you got this wrong," say "I think this section is wrong." Instead of saying, "This does not look professional," say, "We need to re-look at this part of the code."

It's a subtle change, but an important difference between being judgmental and being discerning.

Emotionally Connect with Your Fellow Mobbers

The last attribute related to empathy in mobbing is to emotionally connect with your fellow mobbers. In general an emotional connection happens when two or more people knowingly feel the same thing at the same time. This can

3. https://twitter.com/rajadon
4. https://www.psychologytoday.com/us/blog/sapient-nature/201105/dont-be-judgmental-be-discerning

happen at many levels—for example, emotional connectedness when mobbing can include:

- Experiencing satisfaction as a mob when you discover a particularly elegant solution to the problem on which you are working.
- Being frustrated as a group when you can't figure something out.
- Feeling concern for a fellow mobber when you know they're having a particularly hard day.

Feeling connected to others is important. Brené Brown,[5] a well-known researcher and author on empathy says feeling connected to others is "the energy that exists between people when they feel seen, heard, and valued; when they can give and receive without judgment." Being able to be seen, heard, and valued in a mob is powerful. When someone feels connected to others their best self emerges.

While practicing empathy will not remove all people challenges in mobbing, it often resolves many of them. Interestingly, just understanding why someone is doing what they're doing often makes it possible to effectively work with them.

Now that you understand the concept of empathy in mobbing better, let's spend some time on one of the other important people aspects of mobbing: adopting a collaborative mindset.

Adopting a Collaborative Mindset

A collaborative mindset starts with putting the focus on "we the group" not "me the person." While it may sound like a subtle difference, it's essential to have a collaborative mindset for someone to be effective in a mob.

When you have a collaborative mindset, you're willing to learn from others, and to work through conflicting ideas while keeping the group's purpose or larger goal in mind. When differing ideas to your own are presented, you seek to understand what others are thinking because you believe everyone has a piece of the truth and brings value to the mob. You look for win-wins instead of win-losses. In short: the success of the group is more important than a personal victory.

One of the best explanations on understanding a collaborative mindset can be found in *The Facilitator's Guide to Participatory Decision-Making [KLTF96]*

5. https://brenebrown.com/

Three Ways to Intentionally Build Connections

The challenge with emotionally connecting with others is that you just click with some people while others require a bit of work. For those connections that don't just happen, you need to invest time and effort to help foster them. Here are three suggestions to help you get started:

- Greet them each morning, ask how their evening/weekend was, listen to their response and follow up with additional questions where appropriate. This is a great way to become better connected, the trick is to keep things genuine. When asking, you need to listen to what the other person has to say. Only after you have demonstrated that you are genuinely interested in them will they begin to share more personal experiences. Be careful not to interrogate them, let them decide what level of detail they want to share. Once someone has shared something with you, share something about yourself back so it becomes a two-way interaction.
- Eat lunch together. You will be surprised how effective this is in becoming more connected. Many of the teams that I've mobbed with had a tradition of eating lunch together at least once or twice a week. Over lunch we would share war stories on past jobs, share personal experiences related to whatever was the topic of the day or just talk about something we did at home. Ultimately we would find common ground that made us see each other as people with lives outside of code that we could relate to. If you are eating in a group, be intentional about sitting next to different people during lunch. Often we tend to sit next to the people we are already connected with.
- Show your appreciation whenever they offer to help. Genuinely compliment them when they do something well. Everyone wants to feel that their work is appreciated. Appreciation goes a long way toward feeling more connected to someone.

These are just three suggestions to help you get started, there are ton more available online—a simple Google search will yield a range of ideas.

where they compare the differences between a collaborative mindset and a competitive mindset. In many ways, a collaborative mindset is the polar opposite to a competitive mindset. For instance, with a competitive mindset a personal victory often outweighs a group win.

Fostering a collaborative mindset is harder for some compared to others. Many things impact your mindset, from your up bringing to your personality traits. In the next section we'll look at the Big Five model, a framework that will help you better understand personality traits and people in general. Understanding people better makes it easier to collaborate with them.

Avoiding Groupthink

When speaking with others around Mob Programming some have raised concerns that mobbing can lead to groupthink.

Groupthink is a term first used in 1972 by social psychologist Irving Janis. It refers to a psychological phenomenon where people who are opposed to the decisions of the larger group remain quiet, setting aside their own personal beliefs and adopting the opinion of the rest of the group.

Why it happens varies, from people wanting to "keep the peace" to others fearing rejection from the larger group because of their differing opinions.

Regardless of the cause, groupthink is extremely dangerous, and in many ways, it is the antithesis of mobbing! It gives a false sense of security because you appear to be going through the motions of mobbing but are not realizing any of the major benefits of diversity in thought.

Some techniques to avoid groupthink include:

- Promoting an environment where people are non-judgmental.
- Allowing quieter personalities in the mob to state their opinions or preferences first.
- Assigning at least one individual in the mob to take the role of the "devil's advocate."
- Encouraging people in the mob to remain critical.
- Bringing outsiders into the mob on occasion.

The value in mobbing comes from having diversity in thought and opinion. Avoiding groupthink is critical to get the full benefits from mobbing.

The Big Five Personality Traits Model

In the same way that a leader becomes more effective by understanding their team, a mob becomes more effective by understanding each other.

You likely already make extensive use of models to help understand the system architectures and processes within your industry. In this section, you're going to use a model to better understand the people in the context of Mob Programming.

Before getting into this model it's worth mentioning that not all models are equal. Unfortunately, some popular "personality models" were based on pseudo-science[6] and were fabricated with no scientific foundation, giving the reputable ones a bad name.

6. https://www.psychologytoday.com/us/blog/give-and-take/201309/goodbye-mbti-the-fad-won-t-die

This, however, is not the case with the Big Five personality traits model. It has a solid scientific foundation that has proven reliable across many of the world's cultures; while it isn't perfect, it's extremely useful.

Joe asks:

Where Did the Big Five Model Come From?

The Big Five personality traits model has been around since the 1960s and is widely accepted by personality researchers as the dominant traits model.[a]

It emerged from several independent researchers, all studying the relationships between the large number of known personality traits. Each researcher used different methods to get to the five traits (so the traits had slightly different names and definitions) but essentially all of them independently ended up with the same model.

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

The Big Five personality traits capture the most important, basic individual differences found in personality. They are:

- Openness to experience
- Conscientiousness
- Agreeableness
- Emotional stability
- Extroversion

While all five traits are best to describe the individual differences in personality, the first four have a direct impact on the people who mob.

Interestingly, the extroversion trait does not seem to feature much when mobbing. Before exploring why, take a closer look at the four traits that do impact mobbing to understand them better.

Personality Traits Are Not Binary

The Big Five personality traits are not binary, they are scalar; everyone falls somewhere on a scale for each of them even though it's common for people to over simplify them and treat them as binary.

For instance, you may have been told you're an extrovert or an introvert when in reality you have a degree of extroversion or introversion which is different from others. This concept of scale applies to all of the traits.

Openness to Experience

The openness to experience trait has the most direct impact on mobbing. People who are strong in this area are creative, broadminded, willing to experiment and adapt easily to new situations—these are all attributes that make them great early adopters of Mob Programming. More importantly, these people are good at building on others' ideas, which is an essential element to effective mobbing.

Ironically, people often don't focus on developing the ability to build upon others' ideas during Programming 101; in fact, people tend to focus on the exact opposite—looking for the flaws in others' logic. Next time you're with your team discussing potential solutions to a problem, look out for how many times people in your team say, "No, but…" You might be surprised how frequently it happens.

For an industry that hasn't fully developed in this area, how can you get better at building up on other people's ideas? One way is to adopt a few principles from stage improvisation.

Stage improvisation—or improv—is a form of theatre where most or all of what is performed is unplanned; at its core it's about building upon others' ideas on the fly. The way improv works is one actor starts the scene by saying something, and everyone else tries to keep the scene going. While there are no absolute rules in improv, everyone on stage can succeed by applying the following three principles:

- Listen: always listen to what people around you are saying.
- Agreement: say, "Yes, and…". Then, add something to the conversation, but don't reject ideas.
- Team work: have a group mind, think of others.

Due to the close nature of collaboration when mobbing, it's key for everyone in a mob to get good at this. As an industry, adopting these three principles will help us get better at collaboration and building upon one another's ideas.

Conscientiousness

Conscientious people are task-oriented, thorough, hardworking, self-disciplined, organized, and self-motivated. It should be no surprise that conscientiousness is the most consistent predictor of individual performance; the more conscientious you are, the better you do at solo work.

Conscientiousness is also a strong predictor of how well a group will do at Mob Programming. Mobs that are made up of highly conscientious people

Breaking the "No, but..." Habit

Many of us have formed a "No, but..." habit without even realizing it.

Someone says, "I think we should do *X*". And you reply, ""No, but have you thought of Y?". Or, "No, but what about this...", and the list goes on.

Breaking the "No, but..." habit can be hard because you often don't realize you're doing it. If you enlist the help of a "No, but..." buddy, you can work to get past this habit.

Here's how it works: find someone who's around you often and ask them to listen for when you say "No, but...". When you do, the helper's job is to make you aware of it; it helps to agree on some sort of non-verbal signal. Whenever you see the signal, rephrase what you've just said with a, "Yes, and..." instead.

With a good buddy—and a few days practice—you'll soon break the "No, but..." habit.

will deliver results with one caveat: having a mix where some people are extremely conscientious and others are not can put undue stress on the mob.

While teams in general struggle when you have mixed levels of conscientiousness, Mob Programming can amplify things because code is produced as a group. Working this way can prove to be a challenge for some as they are directly impacted by those around them.

For instance, it's common to see extremely conscientious inexperienced mobbers get frustrated when the rest of the group doesn't share the same sense of urgency in getting the work done on time, or the same level of thoroughness in making sure it's complete. If left unaddressed, over time, highly conscientious members can feel like they're carrying the mob. They may even begin to resent their performance being measured and impacted by those around them.

Does this mean that if you have a team that has one or two highly conscientious people you should skip mobbing? Not at all, the mobbing is just bringing an underlying problem to the surface. Instead, be aware of this challenge and be proactive about it. Working with your more conscientious members to help them mentor your less conscientious members can turn out to be a major win in overall performance.

Agreeableness

Agreeableness is the third trait that has a major impact on mobbing. Agreeable people are friendly, tolerant, helpful, straightforward, and non-competitive. People who have an appropriate level of agreeableness have a positive impact on a mob because they create a positive environment that helps maintain harmony.

Mentoring Someone on Conscientiousness

When mentoring someone on improving their level of conscientiousness, pinpoint one specific thing to improve at a time instead of tackling multiple items at the same time.

For example, someone who is less conscientious may write sloppy commit messages, continually break the build, and always be late for meetings. Get them to choose one of those things to improve first, have them set regular reminders on it, and get them to measure their progress daily. Tackling one thing at a time will help keep the focus.

Let's say you decide to work with them on meeting arrival time; have them set a regular alert on their phone to "be on time." Then, get them to keep a day-to-day log of how many times they were late for meetings. Once you reduce the number of late arrivals, move on to the next thing.

While it's great to have a happy mob, agreeableness is only beneficial up to a certain point. If everyone is too agreeable, the mob loses its effectiveness at problem solving. There needs to be some friction when problem solving.

Speed Leas, an author on several books on conflict,incliding *Discover Your Conflict Management Style [Lea98]* and *Church Fights: Managing Conflict in the Local Church [Lea73]*, has described the behaviors you should see when experiencing problem-solving friction. Generally, information will be flowing freely; people will be collaborating and conversations will be clear, specific, and factual. This is what you should aim for when mobbing. When you're in this state, you become more effective as a group than on our own.

On one extreme, highly agreeable people avoid the conflict necessary to be effective problem solvers; on the other extreme, highly disagreeable people can create too much conflict for people to collaborate effectively. Getting the balance right is a challenge—if you have someone that falls on either extreme, you want to actively work with them to get them to adjust. This is best done away from the mob, in a one-on-one environment.

Emotional Stability

Emotional stability is a broad concept. People who are emotionally stable are self-confident and secure about their chosen goals and decisions; they don't tie their present emotions to previous negative experiences.

Emotional stability has a big impact on the long-term effectiveness of a mob, especially when there are continual differences in opinion on what the mob should be doing next or how to approach certain problems.

To understand this trait better, imagine you're mobbing and you suggest the group approach a problem in a certain way, someone else proposes an alternative and it's decided by the group to go with the alternative approach—even though you feel this is a mistake. As you might imagine, this can be frustrating.

If you struggle with emotional stability, your mind goes back into your past for all of the other times you felt frustrated in the mob. You begin to feel upset, which impacts your immediate effectiveness in the mob. You become reactive with an in-the-moment burst of emotion. To make it worse, you start to carry the feeling of frustration into future mobbing sessions making it more difficult for you to be effective in the group.

Compare this to how you might respond if you have a strong emotional stability trait. As you begin to feel frustrated, you stay present in your emotion. You don't revisit every previous time you experienced the same emotion, instead you focus on the here and now. This allows you to see the situation for what it is, without clouding your judgement on the negative emotions you experienced in the past, thus keeping you an effective contributor in the mob.

The Relationship Between Mindfulness and Emotional Stability

A study has shown a link between mindfulness and emotional stability,[a] drawing the conclusion that those who practice techniques of mindfulness tend to be more emotionally stable.

Mindfulness[b] is the ability to be fully present. There are various techniques you can use to increase your mindfulness, including meditation and breathing exercises.[c]

a. https://archive.unews.utah.edu/news_releases/better-living-through-mindfulness/
b. https://www.mindful.org/what-is-mindfulness/
c. https://www.pocketmindfulness.com/6-mindfulness-exercises-you-can-try-today/

What About Extroversion?

You may be surprised to discover that extroversion doesn't play a huge role in mobbing. To understand why, it's best to start with the differences between extroversion and introversion.

Jennifer Kahnweiler,[7] an expert on extroversion, gives a great way to identify whether you are extroverted or introverted:

7. https://twitter.com/jennkahnweiler

Extroverts tend to be:

- Energized by being around others.
- Usually talk to help their thinking.
- And often have a wide pool of friends and associates.

Introverts, on the other hand, are:

- Energized by being on their own.
- Need to think deeply before they talk.
- And usually like to have fewer friends, but get to know them better.

It's worth re-iterating that extroversion and introversion are not binary poles but instead a scale—everyone falls somewhere between the two. But which end of the scale is better suited to Mob Programming?

When I was first introduced to mobbing, I thought it would be best suited for people who were more extroverted; it's done in groups so it has a social element, and to be effective, you need to speak out loud a lot, snap extrovert.

After doing it for a while, I discovered it also appeals to people who would describe themselves as introverts, in particular you usually mob with the same group of people which means you get to know a small group of people really well.

Interestingly, a survey done by the Mob Programming community[8] supports this: 47% of respondents identified as introverts, 32% identified as extroverts, and 22% identified as neither extroverted or introverted. Based on this feedback, as well as my own personal experiences, it seems that Mob Programming is practiced by both introverts and extroverts, and more importantly, that it doesn't favor one group over the other.

While both extroverts and introverts mob, be aware of their different needs. Many introverts need time on their own to re-energize—there are various way of achieving this while still mobbing regularly. You'll learn more about that later in *Being Around to Mob*, on page 72 and *Handling Mob Fatigue*, on page 75.

Running a Group Session on Personality Types

One way to increase your mob's understanding of how to work and communicate better is by sharing your personality types with each other.

To run a group session on personality types, there has to be a high level of trust in your group and a clear understanding of what the motivation is—you're

8. http://blog.markpearl.co.za/Mob-Programming-Industry-Survey

Big Five Personality Tests

If you want to get an idea of how you rate against the Big Five personality traits there are many free online tests available. Truity[a] and 123test.com[b] are two of the more popular ones worth trying.

a. https://www.truity.com/test/big-five-personality-test
b. https://www.123test.com/big-five-personality-theory/

doing it to understand how to work and communicate better with each other, not to self discover.

First off, decide on an appropriate assessment for everyone to complete. You want one where the focus is on the strengths of each trait instead of just where someone is strong or weak in things.

There are currently two popular assessments you can use: Via Character Survey[9] and Gallup's CliftonStrengths.[10] At the time of this writing, the Via Character Survey has a free offering, while the CliftonStrengths survey requires payment; either survey is sufficient. If you decide to go with an alternative, make sure you review it and that it focuses on people's strengths before sending it to your team members.

Next, you want an external person to your group to facilitate the session; having someone external facilitate allows everyone in your group to participate. If you choose to go with the Via Character Survey or the CliftonStrengths survey, there are usually trained coaches you can bring in to facilitate the session.

Whoever you get to facilitate, make sure you get someone neutral. At all costs, avoid having the facilitator be a manager or someone seen as an authority figure in your organization; it's near impossible getting people to talk about areas of growth when they feel it can impact their career progression.

When booking the session, find a place conducive to meaningful conversations; you want people to relax and feel safe sharing their vulnerabilities. Consider an offsite location, as getting away from the office helps to create a neutral environment.

Before the actual day, everyone attending should have completed the assessment and reviewed their individual results; if you make use of a professional

9. http://www.viacharacter.org/www/
10. https://www.gallupstrengthscenter.com/

coach it's ideal for them to have met with each person before the group session to go over their individual outcomes—this often fulfills the need for individuals to self-discover.

On the actual day, when the facilitator starts the session, they need to clearly outline the objective: you're getting together to understand how to work and communicate better with each other and what unique strengths each person brings to the group.

As the facilitator reviews each person's strengths, they should get feedback from the individual. The facilitator should ask questions such as the following:

- Do you feel like this is a good representation of who you are?
- How does this assessment make you feel?
- Is it accurate? Get details about what's correct and what's not.

Once the individual has addressed the assessment, the discussion moves on to the rest of the group. Does the group feel like the assessment is helpful? What other strengths have they have observed with this individual?

A session like this can take up to half a day for a small group to complete. But the time invested is worth it. If done well, at the end of the session, everyone has a deeper understanding of who they are working with and what strengths they bring to the mob.

What's Next?

In this chapter, you learned about the people side of mobbing. You also examined the Big Five personality traits and how they relate to Mob Programming. Finally, you were introduced to two personality tests that you and your team can use. Using those (or similar) tests, you were encouraged to run a team building session to discuss the results, focusing on the strengths each member brings to the mob.

In the next chapter, you're going to find out how to adjust your mobbing according to different types of problems and different levels of expertise in the mob.

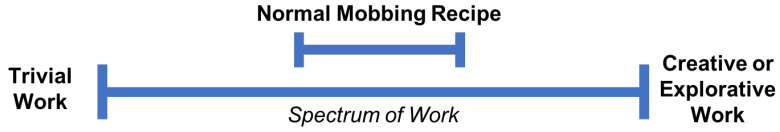
CHAPTER 4

Adjusting Your Mobbing

In *Selecting a Suitable Problem to Solve*, on page 16, you were asked to select a problem in which everyone was on equal footing; you then worked through a recipe for mobbing and (hopefully) applied that recipe to a few katas before moving on to regular work. As you start to mob on regular work, you're going to encounter a few challenges with recipe you've been using.

With the katas, you selected problems where everyone was on equal footing; with regular work, the people in the mob will have a different level of expertise or familiarity with a portion of the code base, mostly because the team is still using and relying on the knowledge they had when working on their own.

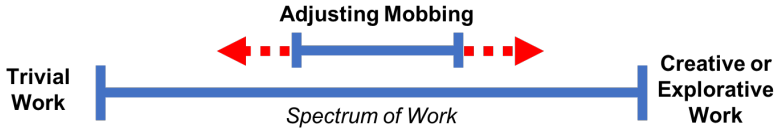
In addition to differences in expertise, sometimes you'll tackle certain types of work where the mobbing recipe isn't ideal. The following illustration shows the spectrum of work most development teams handle:



On the far right is the extreme creative and explorative type of work. On the far left is the trivial, mind-numbing tasks. In between are the usual day-to-day problems.

The mob recipe you learned so far works best on normal, day-to-day problems—the work in the middle—where the knowledge is spread evenly across the mob.

For work on the fringe—or work where someone knows substantially more than the rest of the mob—you need to adjust your approach to remain effective (many inexperienced mobs don't do this and have a poor experience) as shown in the figure on page 50.



In this chapter, you'll discover some ways to adjust your mobbing approach depending on the situation. You'll get a brief overview of the situation where an adjustment is needed, how it's usually handled by inexperienced mobbers, why that approach isn't the best, and what to do instead to help the mob remain effective.

Mobbing with an Expert

One of the most common situations you'll encounter—especially early on while the development team is adjusting from solo work to mob work—is mobbing with an expert. Mobbing with an expert happens when one person in the mob is held up as the expert and everyone else is viewed as a novice. It's not important if the expert's mastery is skill-based or perception—the result is generally the same.

Joe asks:

What's the Difference Between a Novice and an Expert?

In his book *Pragmatic Thinking and Learning [Hun08]*, Andy Hunt differentiates between novices and experts. A novice has little or no prior experience in a particular skill area, and they tend to have a preference or set recipe for how to do things and are less willing to change. Experts, on the other hand, are the primary sources of knowledge and continually look for new methods and better ways to do things. This is an oversimplification of Hunt's explanation, but it's enough to give you an idea between the two.

What Usually Happens

When there's an expert present in the mob, inexperienced mobbers may make the mistake of selecting the expert as the typist. For new mobbers, it seems like a great idea. In fact, with an expert as the typist, problems tend to get solved quickly while the rest of the mob watches and nods in agreement.

Why This Is a Bad Idea

When the expert is the typist, the perception of a healthy mob is only skin deep. Sure, it may look as if things are going well, and that the work is progressing and people are learning, but if you dig in a little deeper, you'll soon discover that the mob is broken.

Suppose you immediately undo everything the mob did together to get the problem resolved. Then, you ban the expert from the group and ask the rest of the mob to redo the work. More often than not, the mob will get stuck a few minutes later, even though they just saw how to do it.

One reason for the misconception of a working mob is because when the expert is behind the keyboard, things move at a pace that's comfortable for the typist but faster than the rest of the mob can follow. This results in knowledge gaps for the rest of the mob; as soon as someone misses a concept from that point onwards they risk misunderstanding why things are being done the way they are.

While not being able to keep up is often the main reason, there's another subtle reason why learning is only skin deep. When you're watching others solve a problem, you're not getting the same hands-on level of experience you would if you had been more involved. The act of working the keyboard helps people internalize learning, which the rest of the mob loses when the expert is the only one doing the typing.

What to Do Instead

Instead of making the expert the typist, do the exact opposite: adjust your workflow, and keep the expert away from the typist role—completely. Pull their focus from the keyboard and on to the rest of the mob. One approach is to make the expert a trainer to the mob. Have them look for gaps in knowledge and fill them in. They are there to help the mob learn, and this is one of the best ways to fulfill that duty.

Mobbing on Explorative Work

Every now and then you'll run into situations that have no clear next step for the mob. This is quite common when new technology is being adopted and nobody has used it before, or when you have taken over a system in a tech stack that's unfamiliar to everyone. This explorative work can often leave you scratching your head wondering what to do next.

Effective Training Moments

Effective training often means explaining a concept to the group; however, sometimes it's more effective to let people get stuck for a short period of time and figure things out on their own.

I remember once working in a mob where we were using a new technology and had already identified one person in the mob as the expert. He had taken on the trainer role and was walking us step by step through what we needed to do, explaining to us as as we went along. Unfortunately, you could tell we weren't really getting it.

After a few minutes, our trainer changed his approach. Instead of telling us what to do, he outlined at a high level what the next step was that we needed to accomplish, and a few resources we could look at to figure things out. He then explained that he was going to let us work a little on our own to get stuck and learn but that he would be nearby and keep tabs on our progress.

For the next hour, he remained with the mob but largely left us to figure things out as he worked on his laptop on something else. By the end of the hour, we had gotten the basics, at which point he put his work aside and normal mobbing resumed.

While this is not the only way to train people, it's certainly an effective approach for that specific situation. Knowing when to guide people and when to let them get stuck to have a learning experience is part of the art of effective training.

What Usually Happens

It's interesting to watch how inexperienced mobs handle this. First, there's usually an awkward moment of silence while everyone in the mob waits for someone else to suggest the next steps (everyone is secretly hoping that somebody else knows what to do). Then, as you realize you're all at the same level, there's a mad rush of suggestions from different people, with no clear idea on how to proceed.

Why This Is a Bad Idea

When there's no clear direction on what the mob should do next, it's easy for things to get out of hand. Individuals in the mob want to try different ideas first which can be confusing and frustrating for the typist, especially when the typist is receiving conflicting requests in quick succession.

What to Do Instead

First, acknowledge the situation for what it is: the mob has a lot of different divergent ideas at play—which is a good thing. In fact, it's one of the super-powers of mobbing, provided it's leveraged properly.

To handle the influx of divergent ideas, the mob needs to agree on which idea to try first. The mob can reach this agreement using one of these two techniques: timeboxed explorations or strong mobbing.

Timeboxed Explorations

A timeboxed exploration gives each person in the mob some time to individually explore their idea to gain a deeper understanding of the task at hand. This is often a good technique to use when there's an online manual that needs to be read or a series of tutorials that need to be done; these sorts of things are often done best on your own.

It works as follows: first, as a mob, clearly identify what you need to understand before you feel you can move forward effectively; this becomes the timebox objective.

Next, as a group, decide on how long the timebox should be; this is best done by everyone sharing how much time they think they need to figure out the objective, once all of the suggestions are put forward, select the smallest proposed period. If all of the proposed timeboxes are longer than 30 minutes, cap the timebox to 30 minutes.

Now, you start the timebox. Each person works on their own trying to figure out the objective. As soon as someone figures it out, they gather the mob and share their findings.

If the timebox lapses, and nobody has figured things out, you still get back together for a brief catch on what you have learned so far. Then, as a mob, you decide if you have enough context to continue as a group or if you need another timebox to continue working on your own, figuring things out.

Now for a quick pro tip: I've seen some mobs separate and go back to their own individual desks outside of the mobbing area when doing timeboxed explorations. I recommend not doing this. Instead, remain in the mobbing area but continue to work on your own.

When people go to their own desks, they open themselves up to distractions; someone comes by to ask for help with something else, or an email suddenly needs a reply. A timeboxed exploration isn't a break from the problem or an opportunity to introduce an unnecessary context switch. It's a time for focusing on the task at hand, without distraction.

Strong Mobbing

An alternative approach to timeboxed explorations is strong mobbing. Strong mobbing works well when the mob has diverged—when there are various

Gareth's Thoughts on Mobbing and Self-Learning

Gareth, who worked in a team that mobbed regularly for a number of years, shares his thoughts on mobbing and self-learning.

> "I've found mobbing doesn't lend itself to certain types of self-learning like learning the syntax of a new programming language, for example. I prefer to do that on my own. However, when working in familiar territory with a good understanding of the terrain, mobbing is extremely effective! Being able to relay your map of the territory to others within the mob and get insights on what they are seeing—and adjusting accordingly—is a powerful way of creating robust software. Personally, I've not found a better way!"

conflicting ideas proposed, and you need to systematically work through these ideas to converge on a solution.

It works as follows: first, as a mob, you nominate someone as the facilitator. The typist moves into command mode, only implementing what the facilitator asks of them, while ignoring everyone else. If the rest of the mob has an idea, they need to take it to the facilitator.

Next, the facilitator gets the mob to collate and clarifies the various suggestions by writing each idea on the whiteboard—the facilitator should do this regardless of how they personally feel about the idea; it's up to the mob, not the facilitator, to decide which ideas are worth pursuing.

Once all of the ideas are listed, the facilitator gets the mob to answer for each suggestion what assumptions have been made, what is known for certain, and what still needs to be confirmed.

The mob then decides on a plan of attack, which should include asking the following questions:

- Which idea is worth trying first?
- Which idea is easiest to validate?
- Which idea would be quick to eliminate?

Often at this point, some people in the mob will say they know something for certain, while others feel that same thing still needs to be confirmed. It's up to the facilitator to keep an eye out for when that happens, and to help the mob get alignment; that may mean the facilitator asks for further explanations or even better, gets the typist to write some code to confirm things.

When strong mobbing works well, the unknowns are clarified, people start converging on a solution, and the way forward becomes clear. Once that happens, strong mobbing can end and normal mobbing can resume.

Llewellyn's Strong-Style Pairing

Strong mobbing originated from Llewellyn's strong-style pairing. Llewellyn Falco,[a] an early advocate of Mob Programming, described the technique in a post written in 2014.[b] It's an interesting read, and a technique worth using when pairing as well.

a. https://twitter.com/llewellynfalco
b. http://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html

Mobbing on Trivial Work

Sometimes when you're mobbing, you'll come across trivial work that needs to get done. Trivial work isn't less important but it is simple and repetitive.

What Usually Happens

For new mobs, when they first come across trivial work, they often march along in lock step, mobbing as they would with any other type of work.

Why This Is a Bad Idea

There's little value in having the entire mob focus their energy on simple and repetitive work. Instead, you need a mindshift change.

What to Do Instead

Willem Larsen,[1] inventor of the Mob Programming Role Playing Game,[2] explained the mindshift change best in an email correspondence:

> "We treat the mob as almost the bridge of starship. When it's appropriate to all stare at the viewscreen and respond appropriately with each of our contributions (i.e., working on production code), we do so. But sometimes a smaller problem requires one of us to hunker down and create a short simple script. Or someone is doing "science officer" research, etc. But we're usually still all there, on "the bridge," still all have one eye on the mob as a whole, we still can discuss our movement forward or do a short retro on where we've come from and strategize. Due to the intensity of the colocation we're able to accomplish so much more, and form and re-form around the challenge."

1. https://twitter.com/cascadiawillem
2. https://github.com/willemlarsen/mobprogrammingrpg

Mob Programming Role Playing Game

Yes, a mobbing RPG is a thing, and it's worth playing. Willem Larsen came up with the idea and presented it at the annual Mob Programming Conference.

There are several roles within this game. There's the Driver (Typist), and there are several subroles that are part of the "Rest of the Mob," including Navigator (equivalent to the facilitator in strong mobbing), Mobber, Researcher, Sponsor, Rear Admiral, Automationist, and Nose.

I recommend going through each role in the game to get a deeper understanding of what you can do to be an effective contributor in a mob. Better yet—play the game next time you mob.

What's Next?

Now that you know how to adjust your mobbing to suit different situations, you're ready to tackle all sorts of work together as a mob. Next up, how to make mobbing sustainable by adjusting your workplace.

CHAPTER 5

Preparing the Workplace for Regular Mobbing

In *Finding a Place to Start Mobbing*, on page 15, you were encouraged to start mobbing in meeting rooms. While this works great initially, in time, you'll discover it's difficult to book the same meeting room all the time—as it turns out, other people need rooms too.

When someone else needs the room, your mob needs to move; the overhead of the mob moving regularly kills momentum, resulting in a poor experience. Instead, you want to move mobbing into your normal work area.

Mobbing in your normal work area has its own challenges, if you don't have a conducive place people won't do it regularly. In this chapter, you'll find out how to adjust your workplace to get everything you need for regular mobbing.

Guiding Principles for Your Workplace

The challenge with giving specific details on how to set up a mobbing environment is that every workplace is different—there are just so many variations on how your specific workplace is laid out that to cover all of them would be a book in itself (and a pretty boring book, at that). Instead, this chapter will outline a few guiding principles and then cover some of the most common scenarios; use these as a template and adjust to suit your specific needs.

The guiding principles around adjusting your workspace for mobbing are:

- Try before you buy.
- Movable is better than fixed.
- Nearer is better than farther.
- Space is important.

Here's a closer look at each:

Try before you buy. It's unusual to get the hardware setup correct the first time around. Before blowing your budget on that large, high-resolution TV that everyone wants, rent it for a few weeks. If things don't work out, you can try something else. Once you have a better idea of what works, then you buy it—renting everything for the first month keeps this option open, so try before you buy wherever possible.

Movable is better than fixed. When things are fixed to the ground or bolted to the wall, they're difficult to move, and that's something you want to avoid. For all you know, the size of your mob may change, and you'll need to move equipment around to maintain an ideal mobbing experience—so rather than keep equipment and other things firmly fixed in place, keep them movable.

Nearer is better than farther. The nearer you have your mobbing area to where your team works, the better. Even a few meters away from your regular working areas can mean the difference between a mob happening or people working on their own. In my experience, there seems to be a direct correlation between the distance between the mobbing area and where a team normally works and the amount of mobbing that actually happens.

Space is important. When you plan your mob area, make sure there's enough space for people to sit comfortably and not feel crowded. I've seen mobbing areas where people were on top of each other because there simply wasn't enough space available—this leads to a suboptimal experience. How much space you need depends on the mob area layout you choose; you'll learn more about that in *A Typical Mob Layout*, on page 58.

Don't Let Less Than Ideal Setups Stop You From Mobbing

Sometimes you won't be able to get the budget to set up an ideal mob station from day one—but don't let that stop you from mobbing. There's usually always a way to make a plan, you sometimes just need to think a little outside of the box.

The first mobbing station I set up cost the business nothing. I brought my 42" TV from home, we found a spare desk, made some space in our team area, and we were good to go. It wasn't ideal, but it worked—in time, as we showed the results from mobbing, we got the budget approved to get better equipment (and I was able to take my TV back home).

A Typical Mob Layout

Now that we've covered the overarching principles, let's go into a little more detail around the layout and the equipment you'll need to mob.

The following illustration shows a typical mob layout, with the screens in front, followed by a table; the typist is positioned in the middle of the table—center to the screens—and is surrounded by the rest of the mob, who are within easy reach of the whiteboard:

The distance from the table to the screens is determined by the ideal viewing distance. There are several screen-size-to-distance calculators[1] available on the web that you can use to figure out this distance.

When planning the mob area, try to have enough space for visitors to join the mob. Also, you may find it beneficial to have one or two additional chairs free for visitors to use.

Keep in mind, this is just one of many different variations of a mob area. The ideal layout for your mobbing area is reliant on the space you have available, the number of people in the mob, and the specific equipment you're using.

Equipment You'll Need

It should come as no surprise that when moving your mob into a more permanent work area you're still going to need a large screen and a whiteboard, just as described for mobbing in meeting rooms on page 15—the only difference is you can be a little more picky on the specs to make sure you get an ideal experience.

Screens

Select screens that have a viewing space that everyone in the mob can see comfortably at the same time. Comfortable means that everyone should have their own personal space respected and that they can read the code on the screens without straining their eyes.

1. https://www.rtings.com/tv/reviews/by-size/size-to-distance-relationship

Did you notice I wrote “screens” in the plural sense? With two screens, you can use one for the terminal or browser windows, and the other screen for the editor and code. With three screens you can do even more!

While you’re shopping for screens, larger is often better. Larger screens tend to have a larger viewing distance, which means you can have more people in the front row in the mob. When people are seated behind others, those in the back row often get blocked, resulting in a second-class experience.

While the size of your screens are important, the resolution of the screens is just as important; therefore, aim for high-resolution screens.

You may be wondering about projectors. If given a choice, prefer TVs and monitors over projectors; the challenge with projectors is that you need to project the image against something which reduces your options of where you can put the mob.

Joe asks:

What Size Screens Should I Get?

You may notice I’m avoiding giving you a specific size. Because screen technology is advancing at such a rapid rate, whatever specific hardware details I suggest would likely be out-dated almost instantly. With that disclaimer, 60" 4K screens worked well for our four-person mob.

Whiteboard

Whiteboards are one of the most important tools you need in your mobbing area. I cannot stress this enough. With whiteboards, you can outline approaches to the problem you’re working on and write down things you don’t want the mob to forget. Interestingly, you don’t need a huge whiteboard to do this—a simple, portable flip chart is often adequate.

When selecting a whiteboard, keep in mind that often you’ll be seated when using it. Given an option, prefer whiteboards that have a stable base and that you can adjust so you can comfortably use and read while seated.

Don’t make the mistake of thinking things will be fine because you have a whiteboard nearby. If the mob needs to move in order to have a discussion at the whiteboard, it’s not close enough.

Table

It may sound like a simple thing, but the table has a big impact on your mobbing experience. If it's the wrong shape or size, or set at an awkward height, you're going to have a poor mobbing experience. So, how do you choose the right table?

Going back to the guiding principles, remember movable is better than fixed. Having a table you can move is a big win because you never know when you may need to move your entire mob layout, including the table, to another room.

Something else to consider is adjustable height. If possible, invest in an electric adjustable height table. Being able to adjust the height based on the needs of the current typist—within seconds, thanks to a button—leads to a superior experience for everyone. As standing desks grow in popularity, more people are wanting the option of being able to stand while mobbing, so keep that in mind too.

Also, try to get a table that has built-in USB and electric outlets. Being able to plug devices ad hoc into power—right at the table—proves to be useful, especially when someone needs to work off their own laptop or device, independently, while still being in the mob area.

Finally, prefer rectangular tables over curved or half moon tables. A curved or half moon table angles the people at the table to face each other, and that's not what you want in mobbing; everyone should be facing the screen. With a curved table, some people will need to turn their head slightly to see the screen, which can cause neck strain over time.

Dedicated Mob Machine

In *Disrupting the Mob When Leaving and Joining It*, on page 84 you saw how to leave and rejoin the mob without disrupting flow. While this is great, what you'll soon discover is that if you need to pop out for a meeting, and the mob is using your laptop, problems may arise if you need to take your machine with you, disrupting the flow as the mob switches to someone else's machine. Having a dedicated mob machine solves this problem.

A dedicated mob machine is a development machine that is permanently set up in your mobbing area. Not only does a dedicated machine remove the disruption of swapping machines when someone leaves the mob, it also gives you a consistent development experience when mobbing.

The mob machine should be in a working state, ready for development—that means having all projects and tooling installed and up to date. Your mob machine should also be as powerful as any other developer machine; do not make it last year's hardware (time goes by slowly when a mob is waiting for something to finish running).

While having a dedicated mob machine makes it easier for people to leave and rejoin the mob, be careful that this does not become the only machine where development happens. Individual team members should still be able to do development on their own machines and work independently, if needed.

Mouse and Keyboards

In *Configuring a Machine for Mobbing*, on page 17 you read about the challenges of using the laptop keyboard for your first few mobbing sessions. Different laptops have different layouts which can cause chaos for those not familiar with them. Instead of using the laptop keyboard, use a standard external keyboard.

When moving mobbing into your normal work area, invest in a decent mouse and keyboard that everyone in the mob can use. Keep these permanently in the mobbing area and plugged into the mob machine.

If you decide to use a wired keyboard, make sure that the cable on the keyboard is sufficiently long—you should be able to move the keyboard to everyone in the mob without a fuss.

Many developers have a specific keyboard that they like to use. To cater for this it's worth getting an external multi-port USB hub plugged into your mob machine that is easily accessible to everyone in the mob. The USB hub proves handy should someone want to use their own custom keyboard.

One of the minor challenges of mobbing is managing the cables of the many peripherals that accumulate in the mobbing area. By the end of an aggressive mobbing session, it can look like a bird's nest of cables.

Where possible, keep peripherals wireless but don't compromise on the development experience. For instance, many wireless keyboards are inferior to their wired counterparts; yet, despite the potential of a wired keyboard becoming tangled within a nest of cables, they're usually the better option. Luckily, wireless mice don't seem to suffer the same fate.

Adjusting to Open-Plan Offices

If you're in an open-plan office, expect a few challenges around adjusting your workplace for regular mobbing.

First, you need to make a space large enough for your mobbing area; often, finding the space is the biggest challenge with mobbing in open plan offices. To figure out how much space you need, look at how much space you used when you were mobbing in a meeting room, and go with that.

Open-plan offices are filled with cubicles with fixed network cables, the antithesis to mobbing areas; luckily cubicles are modular, which makes them easy to remove, and WiFi is fast enough for us to no longer need fixed network cables.

Once you have the space cleared and cabling issues sorted out, the next challenge is managing the noise.

Managing the Noise

Mobs are louder than people pairing or working solo. People are interacting, they're talking about code, about different approaches—when a mob begins to really work, it gets louder. When this happens, especially in an open-plan office, people nearest to the mob either complain or they get louder to compensate for the noise. Ever been to a crowded restaurant?

To help deal with increasing noise levels, be proactive. Talk with those who are near the mob area to find out if they think noise will be an issue. Preemptively chatting to non-mobbers about potential noise problems—before they become problems—can be the difference between non-mobbers working with you or against you.

One way to reduce the impact of noise is getting portable sound barriers and placing them around the mobbing area. Not only does this reduce the noise, with enough of them, you can also create a temporary insulated space for mobbing.

No Noise Policy

If your open-plan office has a "no noise" policy, you'll need to get the policy changed or look at alternatives, like moving to a team room—mobs cannot work in silence.

Using a Team Room

The ideal setup for mobbing is a team room[2]—a dedicated space away from other teams where the mob has full control over their area[3] with the freedom to configure it how they like.

Many of the challenges with bringing mobbing into your normal workspace fall away when you have a team room. Often, the only challenge left is figuring out how you're going to move things around to fit the screen in the room.

Although team rooms are great, there's one benefit you lose by using them: casual contributions. In open-plan offices, it's not uncommon for someone to pass by, notice what the mob is working on, and then make a valuable contribution—something that's not possible when you're in a team room. While there are benefits from casual contributions, given the choice between a team room or adapting your open-plan office, take the team room.

A Quick Word on Costs

Bringing regular mobbing into your work area will have costs associated with it. Until now, all you needed from your manager was their support to try something new. But to really get the most out of the mob, you'll need help from your manager to implement some of the changes outlined in this chapter. Rest assured, however, the impact these changes can have on your mob is worth the investment and conversation.

There's a wonderful British saying, "Penny wise, pound foolish," which means to be extremely careful about small amounts of money, and not careful enough about larger amounts of money. Many organizations are penny wise but pound foolish when looking at the costs of bringing mobbing into their work area—the costs of adjusting your workplace for regular mobbing are minor compared to the cost of people not being able to work at their best.

When motivating budget, keep the focus on the what's important. If Mob Programming is making people in your development team more productive, then the savings in productivity far outweigh the costs of the additional equipment. It helps to frame budget conversations in that context.

2. http://bit.ly/Agile-Team-Room
3. https://martinfowler.com/bliki/TeamRoom.html

What's Next?

In this chapter, you saw some of the adjustments you can make to your workplace to make regular mobbing comfortable. You also learned about the guiding principles and the typical equipment needed, such as:

- Large screens
- Whiteboard
- Mob table
- Dedicated mob machine
- External multi port USB hub
- Wireless mouse
- Assortment of external keyboards
- Extra chairs for visitors
- Sound dividers (if in an open-plan office)

With a workplace that's conducive to regular mobbing, you might think people would be drawn to it. But that's not always the case. In the next chapter, you'll learn how to encourage and prepare your team to mob regularly.

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 5. 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 29 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 14, 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 15, 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 4; 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 84).

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.

CHAPTER 7

Focusing on Flow

At its core, Mob Programming is a flow-centric way to create software, meaning your mob is working together—continuously—to help move a piece of work from start to finish.

In *Why have three or more people do the work of one?*, on page 6, you saw the difference between resource efficiency and flow efficiency. To recap: a resource-efficient way of working creates specialists and bottlenecks while a flow-efficient way of working creates generalists and helps get features to market quicker.

In this chapter, you're going to take a closer look at the concept of flow. You're then going to look at mobbing in the context of flow. By the end of this chapter, you'll have a better understanding of flow and how to achieve it while mobbing.

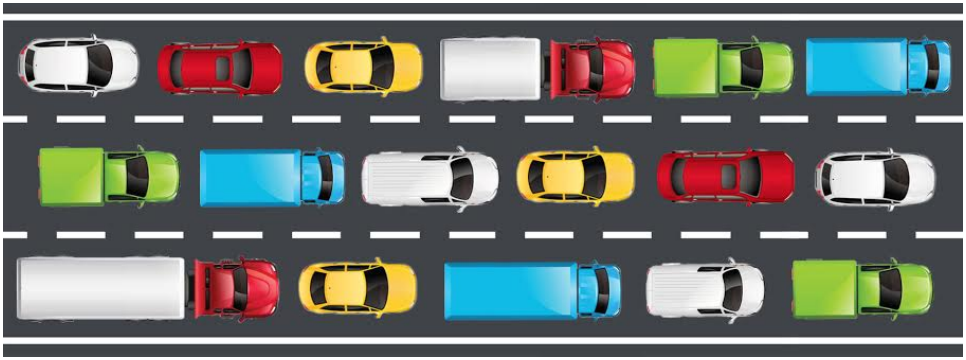
What Do You Mean by Flow?

The first time I truly began to visualize software as a flow system was when a friend of mine shared the freeway metaphor with me. Let's take a look.

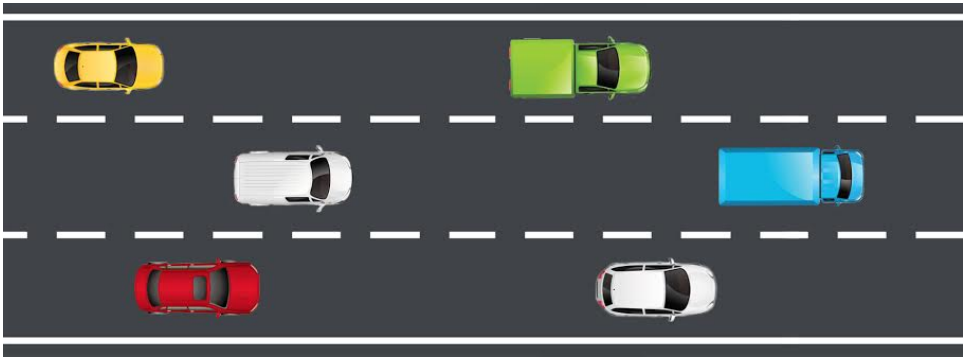
Freeway Metaphor

Imagine you have to travel on a freeway to get from one place to another. As you get on the freeway, you discover traffic is at a complete standstill; cars are bumper to bumper. Your trip is characterized by stop-and-go traffic, with vehicles in front of you moving slowly and unpredictably. It takes forever to get to your destination—traffic is not flowing as shown in the figure on page 78.

Compare this to another trip. As you get on the freeway there are fewer cars and you're able to accelerate, traveling at a much faster speed than is possible when there is standstill traffic. While there are still other vehicles on the road,



there is enough space to move and none of the stop and go you were experiencing in your first trip. Your journey is a pleasure, and you get to your destination on time—traffic is flowing.



The two trips are great examples of contrasting the difference between a system that flows and a system that doesn't. It's also a powerful metaphor because chances are you have experienced both types of trips before. At its essence, flow is about things moving steadily and continuously.

Resource Utilization and Flow Utilization in More Detail

Now that you have the image of flow in your mind, try to connect it to resource utilization and flow utilization.

In the freeway metaphor, think of the freeway as the resource. If you were optimizing for resource utilization, you would want every inch of the freeway to have a vehicle on it because that would mean you're using your resource to its full capacity.

Unfortunately, when you start filling up the freeway with vehicles, at some point, you discover traffic comes to a grinding halt. This is because the vehicles on the freeway don't all move at a constant speed—they're unpredictable and

one vehicle suddenly braking has an impact on all of the vehicles behind it. For a better understanding of flow and traffic, I recommend watching "The Simple Solution to Traffic".[1]

When you make resource utilization the primary focus, flow suffers; work becomes unpredictable as it's impacted by work in front of it.

Compare this to optimizing a freeway for flow utilization. If you were optimizing for flow, you'd want to limit the number of vehicles on the freeway so that if a vehicle suddenly brakes, there would be enough space between it and the vehicles behind it that the ones behind wouldn't be impacted.

Have you noticed that many cities have on-ramp signals restricting the number of vehicles allowed on the freeway—they're used to help maintain and control traffic flow.

When you make flow utilization the primary focus, resource utilization is reduced, which makes things move faster and become more predictable as they're not impacted by other things.

Keeping this image of the freeway in mind, think of the work you do as a software developer and how you get that work to its final destination, in production and being used.

1. https://www.youtube.com/watch?v=iHzzSao6ypE

The Biggest Impact on Flow in Software Development

When you start looking at things in terms of flow, you begin to notice flow breakers—things that prohibit the ability to continuously progress work toward completion. In my experience, the biggest impact on flow in software is not having the information needed to do the work. Unfortunately, no matter how much up-front analysis you do, there are usually unknown unknowns—in other words, the things you don't know you don't know, and will only discover when you do the actual work.

Has this ever happened to you? You start work on a feature thinking that all the analysis has been covered and that it's simply an implementation exercise. While doing the work, you discover something that needs clarification from someone else (an unknown unknown). The person with this information is busy, and so you have to wait for them to be available to answer your question. This results in a broken flow.

When they, eventually, get back to you with the information you need, you resume work only to discover another unknown unknown. More waiting for people to get back to you, more flow broken. And so the cycle continues, as you discover additional unknown unknowns you often have to wait for information from others, which in return, repeatedly breaks the flow of work.

Eliyahu Goldrat in his fictional book, *The Goal [Gol04]*, introduced a concept called the theory of constraints (TOC) which described some of the challenges of queues and bottlenecks in manufacturing. Donald Reinertsen[2] took this concept through to product development in his book, *The Principles of Product Development Flow: Second Generation Lean Product Development [Rei09]*. One of my favorite quotes from Reinertsen is this: "The enemy of flow is the invisible and unmeasured queues."

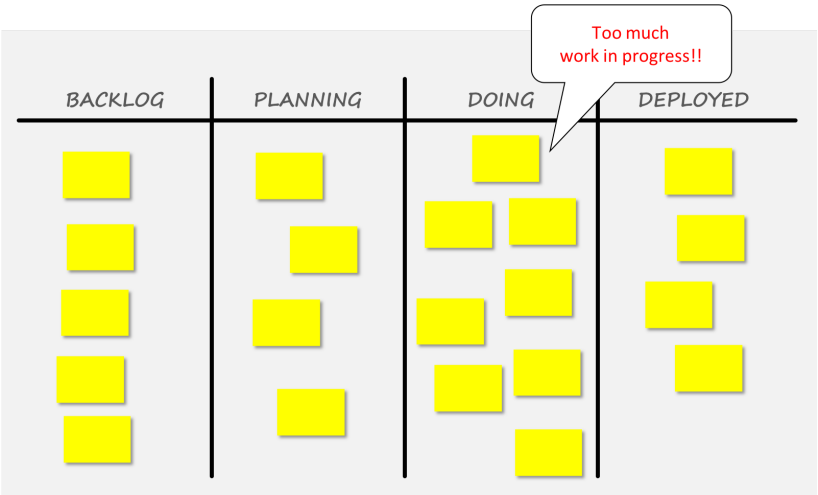
One of the invisible queues that he talks about is the request for information. Think of the information you're waiting on as an invisible queue. Theory of constraints would point out that most time is wasted by things sitting in queues; while this was originally suggested by Goldrat for manufacturing, it holds true for software.

One way to solve this problem is to bring more people to where the work is happening. In doing so, you reduce the likelihood of these queues occurring and slowing things down.

2. https://twitter.com/dreinertsen

Increase Flow By Reducing Work in Progress

Many teams struggle to get sufficient flow because they work on too many things at the same time. Sadly, for many teams, if they were to represent their work graphically it would look something like the following image:



The problem with this is that there's just too much going on in the Doing column. Encourage your team to reduce their work-in-progress or work on fewer things simultaneously. In spite of the many indications that reducing work-in-progress actually makes us deliver more, faster, as humans, we often feel that starting work is a sign of progress; the reality is it's an illusion. In many cases, starting work prematurely will actually slow you down because now you incur the costs of context switching.

The Benefits of Limiting Work-in-Progress

Two books that explore the benefits of limiting work-in-progress (WIP) in depth are *The Principles of Product Development Flow: Second Generation Lean Product Development [Rei09]* and *Kanban [And10]*. In Donald Reinertsen's book, he examines the power of WIP limits and their effect on cycle time in software development. In *Kanban*, David Anderson shares how to practically limit WIP as well as his successes in doing this at Microsoft.

When you focus on reducing work-in-progress, don't forget to take your stakeholders on the journey. If you don't, expect to face pressure from them when they see you suddenly not starting additional work—in the past, for them, starting work was an indication of progress, now that you are mobbing, they need to re-calibrate.

While the stakeholders are still re-calibrating, expect them to occasionally put you under pressure to start multiple pieces of work. At times like this, you need to stand firm, reassure them that you believe you're working in the most effective way possible (you really do need to believe this) and that your priority is to finish features faster, not start features sooner. Often, all it takes for them to be comfortable is reassurance from you that you're doing what you're doing for the right reasons.

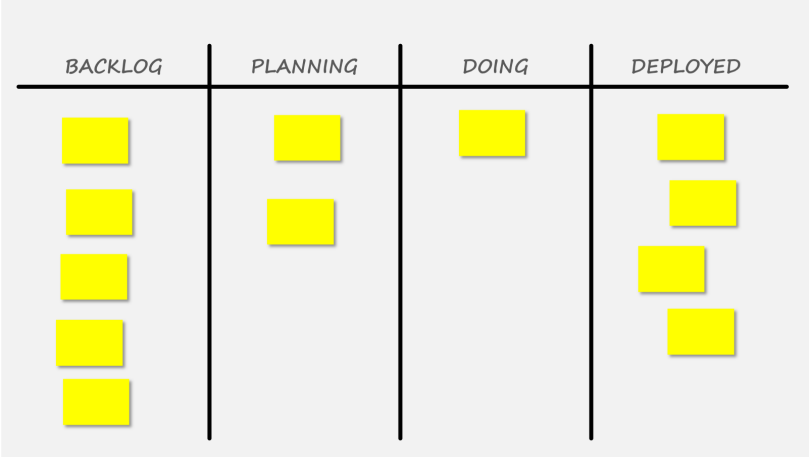
When your team starts to reduce work-in-progress and focus on flow, it helps to have a guide or set of principles they can operate by. When considering a set of principles, keep these in mind:

- Finishing "in-flight" work is better than starting new work.
- Working together is better than working alone.
- Sharing knowledge is better than being the expert.

Different development methodologies support these principles more than others; for mobbing to work, it's important that your overall development methodology supports flow.

While Mob Programming works well with a variety of processes (see *Mob Programming and Scrum, Kanban, Waterfall, etc.*, on page 68), it works particularly well with kanban. Over 50% of respondents on my industry survey of Mob Programming indicated that they were using a kanban methodology. Kanban is all about explicitly limiting work-in-progress with the idea that the fewer things in flight, the quicker they'll get done. If you're not familiar with kanban, it may be worth checking it out to see if it's a good match for your team's overall development needs.

It's quite possible, through mobbing, for a team to get to this:



Dealing with Other Flow Breakers

There are a few more common flow breakers that will impact your team when mobbing. Here are some strategies you can take to deal with them.

Ad hoc Disruptions to the Mob

External ad hoc disruptions to the mob while they're working negatively impact flow in a major way; this should not be a surprise, you experienced this problem before when everyone was working solo. Remember the time when you were in the middle of solving a complex problem, and someone popped by to ask you about something totally unrelated to what you were doing? Chances are, that made you lose your train of thought.

When you're working on your own, the impact of a disruption is high on you but low on the overall team (everyone else was doing their own thing); when mobbing, the impact of a disruption is still high on you but also high with everyone else in the mob.

You'll find when you start mobbing, many of the unnecessary ad hoc disruptions that were happening before don't happen anymore. It seems people are hesitant to disrupt a group of people working together, at least at first. With time, however, those outside your team get used to you mobbing, and the disruptions come back, making this an important problem to solve.

The good news is there are ways of solving this problem. Once such way is the Batman.

The Batman

The job of the batman is to protect the mob from ad hoc disruptions. Consider the batman as the buffer for the mob from the rest of the world. The batman is the first port of call if anyone outside of the mob has a question or needs help with something from the team.

For larger teams, the batman usually doesn't join the mob, but instead works independently. This makes it significantly easier for others to identify and approach the batman. Because larger teams typically have more ad hoc queries from others, the batman tends to stay busy.

For smaller teams, it may not be feasible for the batman to be someone from outside of the mob or you might not have enough people or enough work to justify another non-mob person. If you fall into this camp, the batman can be part of the mob but typically won't take on the typist role, because doing so would prevent them from being available to be approached at all times.

Joe asks:
Why Is It Called a Batman?

I know what you may be thinking, but no, it has nothing to do with the comic book character. Batman originated as a military term used to describe a soldier or airman assigned to help a commissioned officer with the mundane tasks. With the batman's help, the officer could focus on being a better officer.[a]

James Shore, in his book *The Art of on Agile Development [SW07]*, suggests taking a batman approach to remove disruptions in an iteration.

a. https://www.thefreedictionary.com/batman

This does make it more difficult for an outsider to identify and approach the batman—without disrupting the mob—but there are ways of pulling this off; you just need to be creative.

If you make use of a batman, change the batman at regular intervals to help your team avoid key-person dependencies, and to ensure that the knowledge of how to tackle common questions and requests from others continues to be spread across your team. Start by changing the batman weekly. Adjust it to find a cadence that works well for your team, keeping in mind that handovers are expensive and being the batman can be draining.

Discovering Who Batman Is

The trick with getting this to work is making it easy for outsiders to self-discover who the batman is. It doesn't help if someone has to go to the mob and disrupt everyone to ask who the batman is for the day; that defeats its purpose.

One team I was in used a large toilet plunger (we felt the toilet plunger was a good visual metaphor for the type of work the batman usually did) as a physical token to indicate who the batman was for the week. Whenever someone outside of the team needed help, they knew to find the person with the toilet plunger.

In addition to having physical tokens, it's useful to keep a generic email address that is monitored by the batman and moved as the batman changes—this allows people outside of the team to have a fixed email address to use regardless of who was the batman and when.

Disrupting the Mob When Leaving and Joining It

Often, mobs start to get their mojo, then it's all lost because someone in the mob needs to pop out for a few minutes, and in doing so disrupts everyone

else still in the mob. Here are a few strategies you can use to avoid breaking everyone's flow.

Generally, it's not a good idea to step out of a mob if you have the most context on the problem at hand, unless you're being intentional about it, as covered in *Effective Training Moments*, on page 52.

Once in a while, I've seen a mob at a critical point, and clearly reliant on a specific person for guidance, when there has been some distraction that has drawn that person outside of the mob; the result has always been a frustrated mob. If you're the only person that knows what's going on, stay in the mob until they're in a good place to continue without you.

If you're not at a critical point and need to pop out of the mob, indicate how long you think you'll be away. When you set expectations (for example, "I'm just going to pop out for ten minutes"), it makes it easier for the mob to plan around your absence. If you indicate you're going to be back by a certain time, do your best to make good on that promise; otherwise, you'll risk losing the mob's trust. If you're not sure how long you'll be away from a mob, make that clear as well.

When stepping back into the mob, do so with as little disruption as possible. Avoid asking questions like, "What did you guys do while I was gone?" or "What have I missed?" These sorts of questions disrupt the flow of the mob because people feel obliged to cover things they've just dealt with and often feel rude if they ignore your questions (like they should). Instead, quietly re-join the mob, and move in as a typist as quickly as possible. As the typist, you'll gain context as the rest of the mob directs the next steps.

Meetings

While not exclusively a problem when mobbing, meetings are a huge flow breaker; having a strategy around how to handle meetings in your team will help your mob maintain flow.

Paul Graham[3] has an enlightening post on the difference between maker schedules and manager schedules.[4] He explains that makers need larger blocks of time to create things compared to managers who are primarily paid to context switch from one meeting to another; mobs are makers. You should be looking for opportunities to provide your mob uninterrupted blocks of time that are meeting free. The sweet spot for mobs is ~3-hour blocks. This time is really important, and you need to protect it.

3. https://twitter.com/paulg
4. http://www.paulgraham.com/makersschedule.html

Unfortunately, there are some meetings that are unavoidable. In these instances, there are a few techniques you can use to reduce the impact to your team:

- Limit external meetings to a particular day of the week if possible.
- Encourage meetings to happen around natural context switches (just before lunch, just before everyone leaves, just before everyone starts mobbing).

Being with the Mob, but not in the Mob

It's possible to be with the mob, but not actually engaged in the mob because your mind is elsewhere, which is another flow breaker. I've seen mobs where a person was disrupting flow because they were continually zoning out, checking their phone for text messages, or reading their chat messages every few minutes.

While in the mob, turn off your notifications. It's difficult to stay engaged in mobbing if you're trying to maintain another conversation via chat, email, or text messages at the same time. Rather encourage regular breaks from the mob where you can check these things as recommended in *Regular Breaks*, on page 75.

What's Next?

Flow is a concept that's critically important to understand in any collaboration effort. In this chapter, you took a closer look at flow and flow blockers, and how they impact your mob.

With a better understanding of flow, it's time to cover some of the things you can expect to crop up over time when mobbing regularly.

CHAPTER 8

Mobbing Over Time

Well done, the hard work is complete. You helped introduce Mob Programming to your team, and they've embraced it and adopted a flow mindset. They've adjusted their physical workplace as well as their team processes to allow mobbing to happen, they now work together to get things done. Mobbing is not just something they have experimented with, it's something they do regularly and are getting value from.

While it would be great to say that your work is done, it's not—your team, and how they work, is constantly changing. Members of your team will leave and need to be replaced by others, stakeholder perceptions often change, teams fall back into old habits. In this chapter, you'll discover some of the most common changes that occur over time with mobbing and what you need to do to remain successful.

New People Joining Your Team

It's inevitable that at some point in time someone on your team is going leave and will be replaced by someone else. Remember from *Tuckman's Model and Storming*, on page 14, whenever someone new joins your mob, there will be some storming.

In addition to storming, your mob may encounter other challenges depending on the experience level of new members. But don't confuse experience with the number of years doing something. That's often not a good indicator of experience. Jeff Atwood[1] of StackOverflow calls it "The Years of Experience Myth."[2]

When you look at bringing a new person into a team, it can be difficult to get the balance of experience right in the team. Prior to mobbing, I always tried

1. https://twitter.com/codinghorror
2. https://blog.codinghorror.com/the-years-of-experience-myth/

to replace a departing team member with someone who was equally or more experienced. Since adopting Mob Programming, however, I've put less emphasis on technical skills and more emphasis on the underlying values brought to the table, and how well I think someone will work with others. I've found that mobbing tends to accelerate a person's technical skill development.

While mobbing might make you more confident to bring less technically experienced people into the team, keep a watchful eye on the impact this has to your mob. I've seen cases where the rate the mob was able to deliver work is reduced while getting the new mobber up to speed. In short: be careful not to fall into the trap of unintentionally creating a Daycare mob (I borrow this term from Daycare teams). Mobs are particularly susceptible to this because it's such a great place to learn.

Unfortunately, there's no set number on the combination of experienced vs. inexperienced developers you should have in your mob. Largely because the needs of every team is different. Instead of focusing on the number of years' experience, focus on what the new member can bring to the team.

Joe asks:

What Are Daycare Teams?

In Alistair Cockburn's book, *Surviving Object-Oriented Projects [Coc98]*, he speaks about Daycare teams. These are teams where things are not proceeding at a rate you would expect because the experienced members of the team are being diluted by teaching the less experienced ones.

Less Experienced Person Joining

If you decide to take on a less experienced person, be aware of how that person is going to feel in the mob. A mob is a vulnerable place to be; slowing the team down is often the number one fear less experienced people have when working in a mob for the first time, and because they're new, they're often hesitant to vocalize their concerns.

Here are four things you can do to help alleviate this fear:

- Bring the fear into the open.
- Focus on safety.
- Optimize their learning.
- Be clear on time pressures.

Bring Fear into the Open

The first step to addressing the fear of slowing the mob down is talking about it with the person. While this is a common fear for most inexperienced new people, it's not a fear all of them have. Meaning, if your new person says they are fine, don't dwell on it.

If it is something they're experiencing, and you want to put them at ease, let them know it's natural for them to feel this way and the mob is fine with moving a little slower provided they're learning.

Focus on Safety

Creating a safe environment is key for less experienced people to be successful in the mob. Do your best to create a place where new mobbers feel accepted, respected, and comfortable with getting things wrong without it permanently being held against them.

New, inexperienced people are often scared to ask questions for fear of appearing ignorant; this is an indication they're not feeling safe. Work with them to get them over this. Make it clear that asking questions is not a problem, and in fact it's encouraged. Often the questions you think show your ignorance turn out to be the most useful ones because they accelerate your learning and can also help uncover learning opportunities for others.

While they should not hold back on asking questions, some questions are best answered in the moment, while others are better answered after the mobbing session. Reassure them that if they ask a question that is better answered after the mobbing session the mob will let them know and park the question for later. Always come back to parked questions.

Be careful how you react to questions from others in the mob regardless of who is asking. Often, the time you'll forget to do this is when you're working in an area of the system where you really know what is going on. Listen to those around you and what they are asking without dismissing them; treat every question with care. New people will look at how people respond to questions in the mob as a indicator of how they will be responded to. Never shoot down a question.

Optimize Learning

Optimize their learning. You want to put your new less experienced people in the ideal position to accelerate their learning. One of the best ways to do this is for each person to have a mentor in the mob who can give them regular

feedback on where they need to focus their energies on improving and how best to do this.

Be intentional around setting up mentorships. In pairing someone with a mentor, make sure it is someone they respect and can learn from. Encourage the mentor/mentee to meet regularly in a one-on-one environment; meeting weekly often works well for new team members.

During mentoring sessions, the mentor should answer any questions the mentee has, follow up on previous goals set, share observations in areas the mentee can improve, share what has worked for the mentor in the past in the same areas, and help the mentee set new goals for improving until the next time they meet.

Be Clear on Time Pressures

Be clear on what the time pressures are for getting the current piece of work out and how fast the mob needs to move. If you have a new inexperienced person joining your mob, ideally work on non-time-critical work that will give the mob some slack to spend time explaining things. If you're working on time-sensitive work, it may make more sense for the new person to pair up with someone independently of the mob to get familiar with the code base before joining the mob. If you do this, be careful not to keep them away from the mob for too long or else they will begin to feel isolated.

More Experienced Person Joining

Whenever you bring an experienced developer into your team, they form part of your technical leadership and drive the culture of your team and should have an impact on the practices the team embraces—this is what you are paying a premium for: their experience.

The challenge with having a really experienced person join your team who has no prior experience with mobbing is often they have become accustomed to others being dependent on them, whether for technical leadership, mentorship, their technical and architectural abilities, or domain knowledge in a particular business area.

Because they're human, they need psychological safety as much as everyone else. Given they've already achieved a certain level of competency, they may feel pressure to perform in the mob in an effort to show their true worth. This may cause them to unintentionally dominate the mob.

It's important for them to have the right mindset going into mobbing. Help them understand their role in a mobbing session may be most productive

Karen's Evolving Feelings Around Mobbing

Karen, a young developer new in her career, shared with me her thoughts around joining an established mob after two weeks, and then again after six months. Here are her thoughts after two weeks:

> "In a mob, others work too fast for me so I'm not given time to come up with a solution myself. I understand the problem, but others can suggest a solution faster than I can so I'm not given an opportunity to think. As I'm not able to contribute much besides coding out the solution, I feel I'm not bringing any value to the team. I feel that this is also not optimal for my learning; I'm not given enough time to understand every step in the process.
>
> If something comes up that I'm not entirely sure of during the mobbing session, I'm more likely to ignore or forget about it. This makes my thoughts very fragmented—as a result, I don't come out with a solid understanding of the solution or the part of the system we were working on.
>
> I feel I need time set apart from mobbing to work on my own. It gives me time to think through a problem clearly without people handing the solution to me. This also gives me space to figure out holes in my understanding, exercise my thinking muscles, and gives me a chance to fail. I do find mobbing valuable, but to me it is not the primary way in which I learn."

Six months later, Karen moved teams and continued mobbing with them, and her experience with it evolved:

> "It definitely is a different experience being more experienced. I find it more engaging because I can contribute to discussions. I think part of the reason why I feel this way is because I now have more knowledge and experience in what we are working on.
>
> I'm actually finding my experiences in my old mob quite useful in our current work. I have picked up some skills from the people I worked with that I otherwise wouldn't have if I had worked on my own. Noting down things to discuss for after the mob was helpful.
>
> I still find value in working on some things on my own, however, I wouldn't do this too often. I find mobbing much more valuable than I did six months ago. I have learned a lot from mobbing with more experienced developers without me knowing it. I guess the results are only seen a bit later. Overall, mobbing has really helped my current team deliver, spread, and solidify our skills and knowledge. Thank you for being a huge advocate for it!"

when focused on flow—helping others in the mob get unstuck, looking out for behaviors that are counterproductive, making sure every voice and idea is considered equally. They should not feel like the success or failure of the work produced by the mob is solely their responsibility.

Interviewing Process

When interviewing someone who has no previous experience with mobbing it's hard to gauge whether they will enjoy and support it or hate it and discourage it—you just don't know enough about them and how they work.

If they're joining through an internal recruitment, the easiest thing to do is to get them to spend several days with the team mobbing on the teams work before they become a permanent member.

Within a relatively short period of time, they'll have a good idea of what it is like working with the rest of the team as well as what mobbing is like and whether it is something they will enjoy.

If they're joining through external recruitment unfortunately most companies have way too short a screening process—they do a quick code review, have a discussion on values, gauge whether they think the person is technically competent, and if so, make an offer. That's not enough time to know whether the person will actually enjoy mobbing or even if they're a good fit for the team.

Extend the Screening Process

Once you've done your initial screening and have a preferred candidate, pay them to spend a day or two with the team to do actual work before offering full-time employment. During this time, mob. By the end of it you will have a much better idea of who they are and what they will bring to your team and they will have a much better idea of how you work and whether they will actually enjoy mobbing. This substantially reduces the risk of any surprises.

While I believe this is the ideal, unfortunately it sometimes is not possible. If you only have one or two hours to make a hiring decision, I recommend not mobbing—it's just not enough time to get any momentum. Instead, leverage pairing.

Leverage Pairing When Interviewing

While there isn't a one-to-one mapping between pairing and mobbing, pairing will still give you a good indication of what the person will be like in a mob and how well they collaborate. Pair Programming has the added advantage of being more widely adopted in the industry and so you have a much higher likelihood the candidate is familiar with it and actually does it in their current job. Also, generally if someone pair programs well they will mob well too.

Now that you've spent some time over what to expect when someone new joins your mob, it's time to spend some time on the topic of slow mobbing.

Slow Mobbing

One day, someone in your team is going to say that mobbing is feeling slow—when this happens it is important to establish exactly what is making them feel this way. Is it just a general feeling or is it because someone in the mob is slowing them down?

General Feeling

If it's just a general feeling that things are slow, it's often a misconception; sometimes mobbing seems slower even though it is not. Once you take a closer look, you'll realize that mobbing has removed several things that used to slow things down even more, for instance: code reviews are rendered non-existent due to consensus decision making, stand-ups are quicker because there is less to talk about and fewer defects get to production, which means less re-work.

In fact, often the end-to-end development of software is faster as a mob compared to working solo or as a pair even though it may take a little more time during code creation part to get alignment.

Often, when someone is feeling that things are slow they are only seeing it in the context of the code creation part, they're not seeing the end-to-end picture. Usually all you need to do is help them take a step back and remember what the end-to-end process was like prior to mobbing and this feeling goes away.

Someone Slowing Things Down

The other common cause of someone in your mob feeling mobbing is slow is because someone else in the mob is slowing things down. Sometimes a mob only progresses as fast as the least experienced person in the mob. While the norm should be that your mob should share and learn while you work, there will be times when this slows the mob down too much and may not not be appropriate.

Intentionally Moving Faster

For instance, occasionally your team may have a tight delivery timeline for a piece of work. If one person in the mob is substantially behind the rest of the group in context or skills, slowing down the mob to work at that person's pace risks the mob not being able to meet delivery commitments, which in turn can create unnecessary anxiety and frustration.

At times like this it may be appropriate to intentionally move faster than is ideal for the person who is substantially behind everyone else. If you are going to do this make sure everyone is aware of the time pressure and that this is intentional and an exception.

Have the person who is behind everyone else keep a list of questions on things they didn't understand. Make sure you tackle these questions as soon as possible, do not let them fall by the wayside—they are important.

Losing Patience

For teams that have just started mobbing, it is not uncommon that if there was originally a wide skills gap in experience between the team, that after a month or so of mobbing some people in the mob may feel mobbing is too slow.

It is at this point I like to say that the shiny newness of mobbing has worn off; before this point everyone in your mob was patient with everyone else, happy to learn together; now, however, their patience has worn thin. They feel others are not improving at the rate they personally would like to see and this is slowing them down, which becomes frustrating.

The first thing you can do to manage this frustration is to recognize it. When people in your team worked on their own they could go as fast or as slow as they wanted; now that they are mobbing they are impacted by others.

After recognizing the frustration, find out what they think is causing their team member's slowness. Are things slow because there is a lack of domain knowledge, a lack of technical knowledge, or just something simple like being slow at the keyboard?

Next, establish what they have done to help improve the situation. Have they made the person aware of their frustration? What was their response? Have they offered to help the person?

Often the person slowing things down has not been clearly given this feedback and is not even aware that there is a problem. If that is the case, work with your frustrated team member to help give this feedback to their colleague in an appropriate way. Usually the most appropriate way to give this type of feedback is privately with just the two people involved. Creating a culture of direct feedback between team members around areas to improve leads to healthier mobs.

Creating a Direct Feedback Culture

Fostering a culture in your team where people can give clear feedback to each other about what they can do to improve leads to healthier more effective mobs, as people become aware of where they need to focus their energy on improving. This may sound simple, but it's more challenging than you would think.

Most software developers are good at talking about solving technical problems with each other but ask them to give and receive feedback directly to others about non-technical things and they really struggle.

Janco's Experience with Someone Slowing the Mob Down

Janco, an experienced mobber, shared with me his experience of someone slowing down the mob:

> "I have experienced one person slowing down progress in a mob because they were periodically in-and-out. Every time they joined the mob they'd have to be brought up to speed again. This was exacerbated by him being a less experienced developer, which meant he struggled to understand some of the decisions we had made while he was away from the mob. This frustrated other team members because his understanding seemed to always be behind the rest of the group.
>
> He spent a lot of time on the team, and in the mob, and never got to a level that added rather than subtracted from the team. This is the only time I have experienced this, and so maybe this is the exception rather than the rule. I want to emphasize that I value disagreement and debate in a mob, because that typically results in clearer understanding and alignment. It's just when one person's understanding is always behind the rest of the group I've found it can cause frustration."

Getting people in your mob to give and receive direct feedback to each other on how to improve leads to people being aware of the skills they need to develop to improve their pace and the behaviors they need to adjust to be more effective.

One-on-One Catch-Ups Between Team Members

One way to help create a direct feedback culture is to encourage team members to regularly have one-on-one catch-ups with each other to give and receive feedback. In trying this with my own team, I've found it's one thing for people to meet informally one on one, it's another thing for them to ask each other the right questions!

Personally, I have found that being direct is the best way to get clear feedback. Encourage your team members to ask each other questions along the following lines:

- How do I come across to you? Are there any adjustments I can make when interacting with you?
- Are there any gaps in my interactions that you feel I should focus on? Are there things I do that irritate you or you think I can do better?
- Where do you think I should focus improving so that you can have a great experience in the mob?

Don't forget when encouraging others to pull feedback, you should be doing the same; it is best to lead from the front.

You'll find usually the first few times you ask those you work with for feedback on where you can improve they will test the water by giving you *something* but not *the thing*. Usually you need to ask sincerely several times over an extended period of time before they begin to trust that you value their feedback and will not act negatively if they share the good stuff with you.

If you're the one receiving feedback, always be grateful for any feedback you are given from others to improve regardless of what the actual feedback is.

Creating a Direct Feedback Culture in More Depth

Giving and receiving feedback is a topic that regularly features in Mob Programming because of the closeness of the interactions in a mob.

I have briefly suggested a technique, one-on-ones amongst team members, to help foster a feedback culture but creating this type of culture extends beyond just Mob Programming.

If you feel your mob or individuals in your mob can improve in the way they're giving or receiving feedback, there are many resources available that explore the topic in greater depth. I would recommend you start by looking at the following books:

- *Radical Candor: How to Get What You Want By Saying What You Mean [Sco17]*
- *Fixing Feedback [Mur16]*
- *Nonviolent Communication: A Language of Life [Ros03]*

All three books give practical hands-on advice around feedback and communicating effectively and have proven invaluable in the past when I've had to help myself or people in my mob get better at giving and receiving feedback.

Promoting Mobbing to Others

In *Getting Support from Management*, on page 5, it was suggested that to get buy-in from your manager you talk about the management problems mobbing solves. While having buy-in from your manager is important to start, as you continue mobbing you need to look beyond your manager and think about others.

In one of the first mobs I participated in, we were careful to involve our manager but didn't think of promoting the benefits of mobbing to other stakeholders. As time passed we were seeing all the benefits of mobbing—domain knowledge was spreading, quality was increasing and it felt like we were delivering work at a steady rate. One day the product sponsor came to us: she had noticed that the number of things we had in progress had steadily

been decreasing and was concerned about this "mob programming thing" she heard we were doing and how it was impacting our performance. We had not thought to take her on the journey of understanding what problems we were solving by mobbing and why less work in progress was a good thing.

While it's important for others to trust that the people doing the work are professional enough to make the right decisions on how they work, trust is a two-way thing. The truly great software professionals actively build trust with their stakeholders; one way to do this is to share with your stakeholders why you are doing the things the way you are.

Instead of having a situation like the one we had, actively promote the benefits of Mob Programming to all your stakeholders. For an uninformed person observing mobbing, it often looks like a group of people sitting around talking with only one person actually working. To change this mindset, you need to communicate to them what benefits you are realizing. Be careful to frame these benefits in a way they can understand.

One of the most effective ways of opening up communication with your stakeholders is by giving them an in-person tour of your team area. Invite them to come during a time you know the team will be mobbing.

If you have a physical board, start the tour at the board explaining how work progresses; while doing this, talk to them about flow efficiency and why it is a good thing.

Talk with them about what software development is; there are many common misconceptions of what exactly it is. Software development is not simply churning out code to solve well-defined problems where the bottleneck is typing speed. Instead it is a complex problem-solving exercise, one in which the creative abilities of people are more important than speed of typing. In this light, mobbing makes a lot of sense. Listen to what they have to say.

Reverting to Old Habits

It's not uncommon for teams to slip back into old habits regardless of their effectiveness. A great word to describe this phenomenon is entropy, which is the gradual decline of something into disorder.

Entropy is a very human thing and mobbing is not immune to it—many teams who at one point in time were great at mobbing have gradually declined until one day they stop doing it even though they know it's a better way for them to make software.

How do you prevent entropy or a falling away? Like many things, early detection is key—the sooner you realize your team is slipping into old habits, the easier it is to get them back on track.

Two early indicators are:

- A sudden increase in the number of items actively being worked on.
- Nobody using the mobbing station for several days including you.

If you pick up on your team regressing early on, all that may be needed to do is to focus some energy around getting the mob started each day. This is covered in *Gathering the Mob*, on page 74.

If you pick up on it late, and your team is now in the habit of not mobbing. you're going to need to put some additional effort in. First, you will need to establish if mobbing is still meeting your team needs; if it is, you will need to start the recovery process—let's go through how to do this.

Establish if Mobbing Is Still Meeting Your Team Needs

As a team, establish if mobbing is still meeting your team needs. One way to do this is to re-examine what needs mobbing was fulfilling when you started (see *Linking Mobbing Needs to Team Needs*, on page 67).

There's a chance that your underlying team needs have changed since you started mobbing; if this is the case, establish if your current team needs overlap with the needs mobbing fulfills. If there's no overlap in needs then stop mobbing—you can always pick it up again when the underlying needs come back.

If there's still an overlap, your team has just gotten into a rut and you should work as a team at getting back into the habit of mobbing.

Getting Back into the Habit

A great way to get back into the habit of mobbing is to use the following four-step recovery approach:

1. Identify what contributed to you stopping.
2. Identify what you can change going forward to avoid these contributors.
3. Check in daily on your progress.
4. Remember slipping happens.

Identify What Contributed to You Stopping

It's tempting to say regression just happened, but there is always an underlying contributor and it's really important to identify what it is so that you don't repeat the same pattern going forward.

When doing so, be careful not to blame the rest of the organization for you not mobbing. The danger in blaming others is you risk framing your team as a victim of external circumstances ("We stopped mobbing because the Product Owner kept putting pressure on us to work on multiple things at the same time—it's the Product Owner's fault.", or "We stopped mobbing because we don't have a proper place to sit, management needs to up their game.").

While it may be true that there are external factors that impact your ability to mob, look at the situation as an opportunity to learn as a team what you can do to create a different outcome in the future—you always get to choose how you react to a situation. Identifying the specific contributor is important for you to adjust going forward.

Identify What You Can Change Going Forward to Avoid These Contributors

Once you've identified what the specific contributors were that stopped you mobbing, discuss what you can do differently going forward to avoid this. Write down any adjustments you will make as a team in the future and make these adjustments visible to the team so you see them every day.

Make sure the adjustments you agree on are specific and measurable. Vague statements like "Collaborate more" are less effective and harder to measure progress against than more specific statements like, "Get together at the mob station every morning at 10:00 a.m." Ask yourself: how do I measure this?

Check in Daily on Your Progress

Now that you have a plan going forward, as a team check in daily on your progress in getting back into mobbing. Checking in each day on how mobbing is going—say, during a daily stand-up—will help get things started. Find some way to put the spotlight on the behavior you want each day without making things unsafe.

Remember Slipping Happens

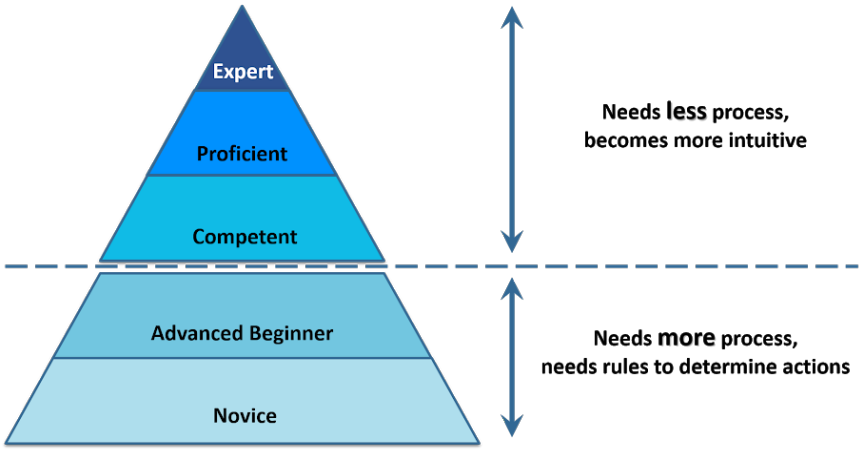
Remember slipping happens. Sometimes teams beat themselves up if they are not perfect ("We tried this before and it didn't stick, we are just not good enough for mobbing"). Realizing that sometimes you will fall back to old behaviors is part of being human.

Sometimes your team is going to have a few days when things are not ideal and they don't mob. When this happens it's important you realize it, inspect, adjust, and recommit. With time and continued effort, this will happen less and less until it is a thing of the past.

From Novice to Master

You've reached the last section before the end of this book. I would love to say everything about mobbing was covered, but the reality is it's an evolving practice and there are new things being discovered about it every day.

Before ending this book, I want to share an insight around your journey to mastering Mob Programming. In *What's the Difference Between a Novice,* on page 50, I briefly explained the difference between a novice and a expert. This explanation was based off the Dreyfus Model of Skills Acquisition, which is a model focused at an individual's journey toward mastering a skill.



The Dreyfus model puts people into five main categories in relation to their skills development; Novice, Advanced Beginner, Competent, Proficient, and Expert.

- *Novice*: has little or no previous experience in a particular skill area; preference for recipes or a clear step-by-step process
- *Advanced Beginner*: has some previous experience; can deviate slightly from the process but doesn't see the bigger picture
- *Competent*: has extensive experience in a skill area and can troubleshoot problems on their own; begins to seek out advice from experts and use it effectively

- *Proficient*: seeks out and wants to understand the bigger picture; can reflect on how they have done and adjust to perform better in the future
- *Experts*: deep understanding of the bigger picture; works from intuition

A Novice or Advanced Beginner needs rules and process to be able to be effective at performing a skill area. As they progress to being Competent, Proficient, and finally, with much effort, an Expert, the need for process is significantly diminished. In fact, for those who are at the Proficient and Expert levels of a skill, process often gets in the way. An Expert intuitively is able to look at the situation and adjust without having to go and read up on the step-by-step process to follow.

While the Dreyfus model focuses on individuals, there are some insights we can gain from it around mobbing and team process. When a group starts mobbing, it is a novice mob; it's important for it to follow a process. I've given you several processes for Novice and Advanced Beginner mobs to follow.

Over time, you're going to encounter scenarios and challenges I have not given you a formula to deal with; that's OK, you've got enough of a foundation in mobbing that you'll be able to adjust accordingly.

As you gain experience, your mob will progress past Advanced Beginner status toward Competent. At that point, the processes suggested in this book will begin to get in the way of you mobbing effectively. When it feels like the process is holding your mob back, move beyond the process.

No team is the same and no mob should be the same either. That means if you are doing Mob Programming right and have grasped its essence with time and experience how you mob will evolve.

Creating great software is all about learning and adapting, you should be learning and adapting with Mob Programming as well.

While mobbing is important, there's a deeper message behind it which is even more important—people collaborating and finding out different ways to create great software together.

As developers, we've moved from the era of individuals creating software to an era where teams create software systems. Mob Programming is one of the most effective ways I've seen teams do this.

It's an exciting time to be in this industry. Best of luck to you on your road ahead to mastering software development.

Happy Mobbing!

Bibliography

[And10] David J. Anderson. *Kanban*. Blue Hole Press, http://www.e-junkie.com/129573, 2010.

[Coc98] Alistair Cockburn. *Surviving Object-Oriented Projects*. Addison-Wesley Professional, Boston, MA, 1st Edition, 1998.

[de 99] Edward de Bono. *Six Thinking Hats*. Little, Brown and Company, New York, NY, 1st Edition, 1999.

[DS06] Esther Derby and Diana Larsen, Foreword by Ken Schwaber. *Agile Retrospectives*. The Pragmatic Bookshelf, Raleigh, NC, 2006.

[Gol04] Eliyahu Goldratt. *The Goal*. North River Press, Great Barrington, MA, Third edition, 2004.

[Hun08] Andy Hunt. *Pragmatic Thinking and Learning*. The Pragmatic Bookshelf, Raleigh, NC, 2008.

[KLTF96] Sam Kaner, Lenny Lind, Catherine Toldi, Sarah Fisk, and Duane Berger. *The Facilitator's Guide to Participatory Decision-Making*. New Society Publishers, Gabriola Island, BC, Canada, 1996.

[Kut13] Joe Kutner. *Remote Pairing*. The Pragmatic Bookshelf, Raleigh, NC, 2013.

[Lea73] Speed Leas. *Church Fights: Managing Conflict in the Local Church*. Westminster John Knox Pr (November 1, 1973), Louisville, Kentucky, 1973.

[Lea98] Speed Leas. *Discover Your Conflict Management Style*. Rowman Littlefield, Lanham, Maryland, United States, Revised Edition, 1998.

[MSWW02] Michele Marchesi, Giancarlo Succi, Don Wells, Laurie Williams, and James Wells. *Extreme Programming Perspectives*. Pearson Education, London, United Kingdom, 1st Edition, 2002.

[Mur16] Georgia Murch. *Fixing Feedback*. John Wiley & Sons, New York, NY, First, 2016.

[Nöt09] Staffan Nöteberg. *Pomodoro Technique Illustrated*. The Pragmatic Bookshelf, Raleigh, NC, 2009.

[Rei09] Donald G. Reinertsen. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing, Redondo Beach, CA, 2009.

[Ros03] Marshall Rosenberg. *Nonviolent Communication: A Language of Life*. Puddledancer Press, Encinitas, California, United States, Second, 2003.

[Rot17] Johanna Rothman. *Create Your Successful Agile Project*. The Pragmatic Bookshelf, Raleigh, NC, 2017.

[Rya10] Terrence Ryan. *Driving Technical Change*. The Pragmatic Bookshelf, Raleigh, NC, 2010.

[Sco17] Kim Scott. *Radical Candor: How to Get What You Want By Saying What You Mean*. Macmillan, New York, NY, First, 2017.

[SW07] James Shore and Shane Warden. *The Art of Agile Development: Pragmatic Guide to Agile Software Development*. O'Reilly & Associates, Inc., Sebastopol, CA, 1st Edition, 2007.

Thank you!

How did you enjoy this book? Please let us know. Take a moment and email us at support@pragprog.com with your feedback. Tell us your story and you could win free ebooks. Please use the subject line "Book Feedback."

Ready for your next great Pragmatic Bookshelf book? Come on over to https://pragprog.com and use the coupon code BUYANOTHER2018 to save 30% on your next ebook.

Void where prohibited, restricted, or otherwise unwelcome. Do not use ebooks near water. If rash persists, see a doctor. Doesn't apply to *The Pragmatic Programmer* ebook because it's older than the Pragmatic Bookshelf itself. Side effects may include increased knowledge and skill, increased marketability, and deep satisfaction. Increase dosage regularly.

And thank you for your continued support,

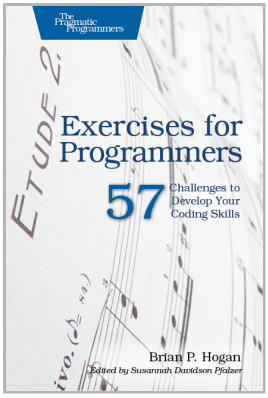
Andy Hunt, Publisher

Exercises and Teams

From exercises to make you a better programmer to techniques for creating better teams, we've got you covered.

Exercises for Programmers

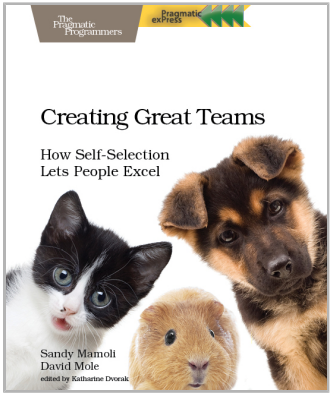
When you write software, you need to be at the top of your game. Great programmers practice to keep their skills sharp. Get sharp and stay sharp with more than fifty practice exercises rooted in real-world scenarios. If you're a new programmer, these challenges will help you learn what you need to break into the field, and if you're a seasoned pro, you can use these exercises to learn that hot new language for your next gig.



Brian P. Hogan
(118 pages) ISBN: 9781680501223. $24
https://pragprog.com/book/bhwb

Creating Great Teams

People are happiest and most productive if they can choose what they work on and who they work with. Self-selecting teams give people that choice. Build well-designed and efficient teams to get the most out of your organization, with step-by-step instructions on how to set up teams quickly and efficiently. You'll create a process that works for you, whether you need to form teams from scratch, improve the design of existing teams, or are on the verge of a big team re-shuffle.

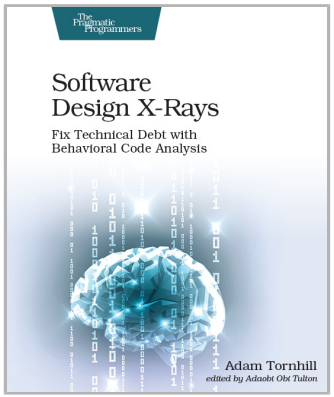


Sandy Mamoli and David Mole
(102 pages) ISBN: 9781680501285. $17
https://pragprog.com/book/mmteams

Pragmatic Programming

We'll show you how to be more pragmatic and effective, for new code and old.

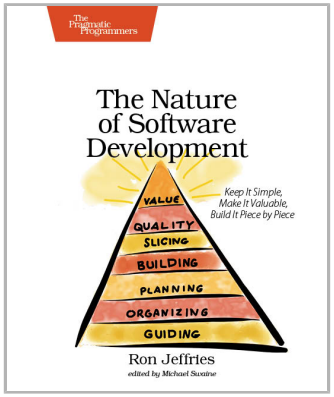
Software Design X-Rays



Are you working on a codebase where cost overruns, death marches, and heroic fights with legacy code monsters are the norm? Battle these adversaries with novel ways to identify and prioritize technical debt, based on behavioral data from how developers work with code. And that's just for starters. Because good code involves social design, as well as technical design, you can find surprising dependencies between people and code to resolve coordination bottlenecks among teams. Best of all, the techniques build on behavioral data that you already have: your version-control system. Join the fight for better code!

Adam Tornhill
(274 pages) ISBN: 9781680502725. $45.95
https://pragprog.com/book/atevol

The Nature of Software Development



You need to get value from your software project. You need it "free, now, and perfect." We can't get you there, but we can help you get to "cheaper, sooner, and better." This book leads you from the desire for value down to the specific activities that help good Agile projects deliver better software sooner, and at a lower cost. Using simple sketches and a few words, the author invites you to follow his path of learning and understanding from a half century of software development and from his engagement with Agile methods from their very beginning.

Ron Jeffries
(176 pages) ISBN: 9781941222379. $24
https://pragprog.com/book/rjnsd

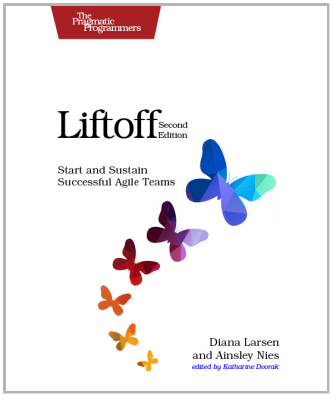
Start Great Teams, Keep Teams Great

See how to get great teams started, and keep them great by doing retrospectives the right way.

Liftoff, Second Edition

Ready, set, liftoff! Align your team to one purpose: successful delivery. Learn new insights and techniques for starting projects and teams the right way, with expanded concepts for planning, organizing, and conducting liftoff meetings. Real-life stories illustrate how others have effectively started (or restarted) their teams and projects. Master coaches Diana Larsen and Ainsley Nies have successfully "lifted off" numerous agile projects worldwide. Are you ready for success?

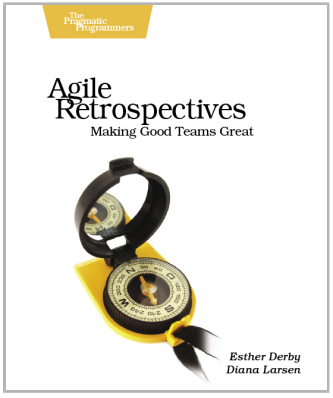
Diana Larsen and Ainsley Nies
(170 pages) ISBN: 9781680501636. $24
https://pragprog.com/book/liftoff



Agile Retrospectives

See how to mine the experience of your software development team continually throughout the life of the project. The tools and recipes in this book will help you uncover and solve hidden (and not-so-hidden) problems with your technology, your methodology, and those difficult "people issues" on your team.

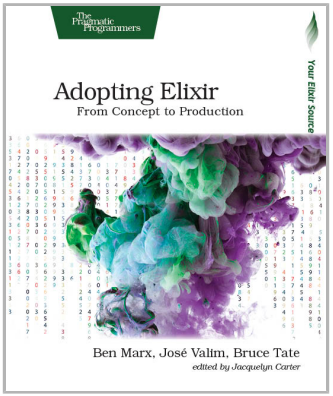
Esther Derby and Diana Larsen, Foreword by Ken Schwaber
(176 pages) ISBN: 9780977616640. $29.95
https://pragprog.com/book/dlret



Learn Why, Then Learn How

Get started on your Elixir journey today.

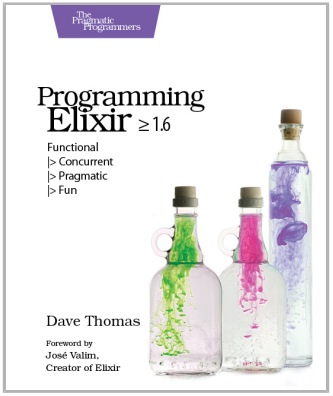
Adopting Elixir



Adoption is more than programming. Elixir is an exciting new language, but to successfully get your application from start to finish, you're going to need to know more than just the language. You need the case studies and strategies in this book. Learn the best practices for the whole life of your application, from design and team-building, to managing stakeholders, to deployment and monitoring. Go beyond the syntax and the tools to learn the techniques you need to develop your Elixir application from concept to production.

Ben Marx, José Valim, Bruce Tate
(242 pages) ISBN: 9781680502527. $42.95
https://pragprog.com/book/tvmelixir

Programming Elixir 1.6



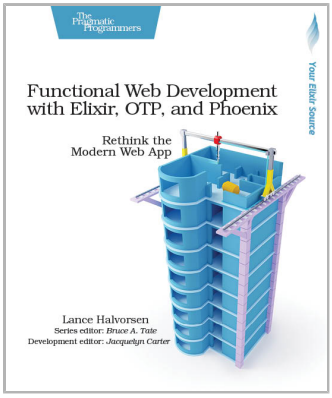
This book is *the* introduction to Elixir for experienced programmers, completely updated for Elixir 1.6 and beyond. Explore functional programming without the academic overtones (tell me about monads just one more time). Create concurrent applications, but get them right without all the locking and consistency headaches. Meet Elixir, a modern, functional, concurrent language built on the rock-solid Erlang VM. Elixir's pragmatic syntax and built-in support for metaprogramming will make you productive and keep you interested for the long haul. Maybe the time is right for the Next Big Thing. Maybe it's Elixir.

Dave Thomas
(410 pages) ISBN: 9781680502992. $47.95
https://pragprog.com/book/elixir16

A Better Web with Phoenix and Elm

Elixir and Phoenix on the server side with Elm on the front end gets you the best of both worlds in both worlds!

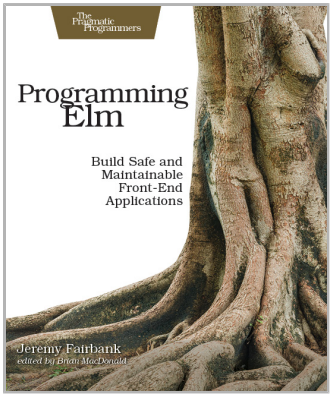
Functional Web Development with Elixir, OTP, and Phoenix



Elixir and Phoenix are generating tremendous excitement as an unbeatable platform for building modern web applications. For decades OTP has helped developers create incredibly robust, scalable applications with unparalleled uptime. Make the most of them as you build a stateful web app with Elixir, OTP, and Phoenix. Model domain entities without an ORM or a database. Manage server state and keep your code clean with OTP Behaviours. Layer on a Phoenix web interface without coupling it to the business logic. Open doors to powerful new techniques that will get you thinking about web development in fundamentally new ways.

Lance Halvorsen
(218 pages) ISBN: 9781680502435. $45.95
https://pragprog.com/book/lhelph

Programming Elm



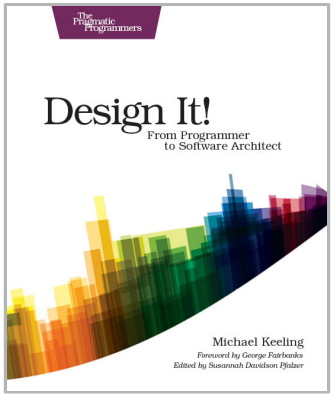
Elm brings the safety and stability of functional programing to front-end development, making it one of the most popular new languages. Elm's functional nature and static typing means that run-time errors are nearly impossible, and it compiles to JavaScript for easy web deployment. This book helps you take advantage of this new language in your web site development. Learn how the Elm Architecture will help you create fast applications. Discover how to integrate Elm with JavaScript so you can update legacy applications. See how Elm tooling makes deployment quicker and easier.

Jeremy Fairbank
(250 pages) ISBN: 9781680502855. $40.95
https://pragprog.com/book/jfelm

Better by Design

From architecture and design to deployment in the harsh realities of the real world, make your software better by design.

Design It!



Don't engineer by coincidence—design it like you mean it! Grounded by fundamentals and filled with practical design methods, this is the perfect introduction to software architecture for programmers who are ready to grow their design skills. Ask the right stakeholders the right questions, explore design options, share your design decisions, and facilitate collaborative workshops that are fast, effective, and fun. Become a better programmer, leader, and designer. Use your new skills to lead your team in implementing software with the right capabilities—and develop awesome software!

Michael Keeling
(358 pages) ISBN: 9781680502091. $41.95
https://pragprog.com/book/mkdsa

Release It! Second Edition

A single dramatic software failure can cost a company millions of dollars—but can be avoided with simple changes to design and architecture. This new edition of the best-selling industry standard shows you how to create systems that run longer, with fewer failures, and recover better when bad things happen. New coverage includes DevOps, microservices, and cloud-native architecture. Stability antipatterns have grown to include systemic problems in large-scale systems. This is a must-have pragmatic guide to engineering for production systems.

Michael Nygard
(376 pages) ISBN: 9781680502398. $47.95
https://pragprog.com/book/mnee2

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

This Book's Home Page
https://pragprog.com/book/mpmob
Source code from this book, errata, and other resources. Come give us feedback, too!

Keep Up to Date
https://pragprog.com
Join our announcement mailing list (low volume) or follow us on twitter @pragprog for new titles, sales, coupons, hot tips, and more.

New and Noteworthy
https://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: *https://pragprog.com/book/mpmob*

Contact Us

Online Orders:	*https://pragprog.com/catalog*
Customer Service:	*support@pragprog.com*
International Rights:	*translations@pragprog.com*
Academic Use:	*academic@pragprog.com*
Write for Us:	*http://write-for-us.pragprog.com*
Or Call:	+1 800-699-7764