The Mob Programming Guidebook

Llewellyn Falco & Maaret Pyhäjärvi

Mob Programming Guidebook

Maaret Pyhäjärvi and Llewellyn Falco

This book is for sale at http://leanpub.com/mobprogrammingguidebook

This version was published on 2017-08-19

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Maaret Pyhäjärvi and Llewellyn Falco by spreading the word about this book on Twitter!

The suggested hashtag for this book is #MobProgrammingGuidebook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#MobProgrammingGuidebook

Contents

To the person reading this book in progress,

We wanted to say thank you. Please enjoy and give us feedback on both what you liked and didn't to help us further grow the book.

Creating a book takes a lot persistence and support from early adopters like yourself helps us to keep going.

Thank you.

Maaret Pyhäjärvi & Llewellyn Falco

maaret@iki.fi isidore@setgame.com

PART 1

First Time Mobbing with a New Group

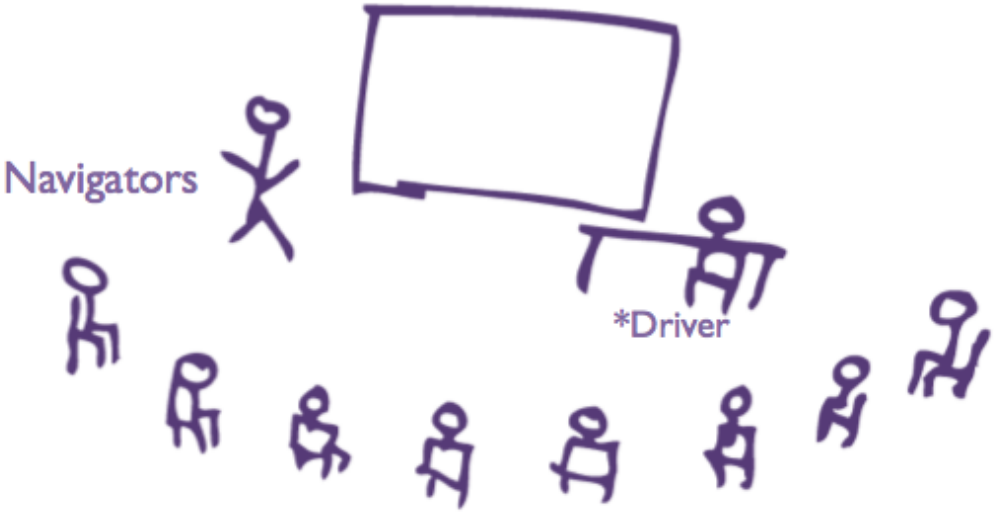
In this section, we will show techniques and methods when dealing with a new group of people, who are not familiar with mob programming. In these situations, you will see more structure to allow people to grow into working as a mob.

First time mobbing commonly occurs with teams at a company, temporary groups at a conference or meet-up, or at classes and workshops.

What is Mob Programming?

> *"All the brilliant people working on the same thing, at the same time, in the same space, on the same computer." – Woody Zuill (the discoverer of Mob Programming)*

Mob Programming is a style of programming in which the entire team sits together and works on a single task at a time. Teams that have worked this way have found that many of the problems that plague normal development have just melted away, possibly because communication and learning goes way up. Teams also find that the quality of their code increases. They find their capacity to create increases. However, the best part of all is that teams end up happier and more cohesive.



Mob Programming

In mob programming, there is a driver, who is the person at the keyboard. Many people thinks this means having five people watch one person work. This is not mob programming. In fact, mobbing has the rule that the driver is not allowed to think while at the keyboard. This means that you can't have people just watching the person type. Mobbing is about the

entire team collaborating together. Having a single computer and driver allows everybody's thoughts and insights to be captured for the task at hand.

Why would you have 5-8 people working on one thing?

A lot of people think that it must inefficient and wasteful to have your entire team working on one problem. Couldn't you get more out them if you divide and conquer? The answer is: No. It's better to think of this in a completely different fashion.

Instead of thinking how can I get the most out of my team, with mob programming we ask **how can I get the best out of my team?** This process was discovered by Woody Zuill and his team at Hunter Industries through the process of constantly paying attention to what they were doing, and doing **more** of the things that were working.

Maybe you've had the experience where the team came together and swarmed over a particularly tough problem. By working together as a team, the swarm might have been able to resolve that problem. We've talked to many people who have had this experience. The big difference is that after having success working in this manner, most teams say "Problem solved, let's go back to normal". Mobbing says "We were very high performing, how can we do more of that?".

Getting the best out of your team

All of us have good and bad skills areas and moments. When you work by yourself, both your best and your worst makes it into the code. When you have a team of people working together but separately, their best and their worst makes it into the code.

In the end it's only what makes it into the code that actually matters.

Quality - Separate Programming

When you work as a mob, everybody still has their highs and lows. However, this is not what makes it into the code. What makes it into the code is only the highest points. This can be particularly empowering for team members whose programming skill might not be the best. We have found that on many teams, some of the best ideas come from members that trouble turning those insights into production code. Left by themselves, those insights die. In the mob, they flourish.

Quality - Mob Programming

The benefits of high communication

Very often people make mistakes. Maybe we misunderstood a detail of a requirement. Maybe your understanding of small is different than my understanding of small. Or maybe there is just an aspect that was left out and forgotten. If I make this mistake on Monday morning but I don't find out about it until Thursday afternoon, I am going to compound that mistake all week. Worse, because I've spent so much effort, I will become defensive of my work and try to protect my mistake.

When we work in a mob, many of these mistakes are detected the moment they occur. Mainly because somebody on the team has that knowledge. Sometimes simply because someone on the team is willing to ask a question.

There are tremendous benefits to this just-in-time knowledge. And unfortunately, the cost of the delays is mainly hidden and does not show up on anybody's accounting sheet. If the hours programmers waste because of gaps in knowledge or understanding were actually accounted for, there would be much less resistance to mob programming.

In other words, everyone thinks that five people working by themselves means that everyone is working well independently. This is rarely the case.

Imagine a jazz band. Are you going to get better music if you put everyone together and have

them play a song or if you send them off to record separate pieces and then try to integrate them later?

How to use this book

This is a recipe book for how to create a highly functional and collaborative mob. There are two main recipes in this book.

- Facilitating a new mob
- Growing a long-term mob

Too often, we find people altering the recipes before they know how the overall dynamics work or what purpose each ingredient provides. Here is our suggestion.

The recipes are for how to work with your groups. How closely you adhere to these recipes should be connected in your experience as a mob facilitator.

Beginner facilitator

If you are a beginning mob facilitator, follow each step very specifically. Some of them will seem weird, counterintuitive or just wrong. Don't worry about that yet. Many facilitators need 20-40 iterations in this mode before they are ready to advance. You may need less. However, we suggest you do a minimum of five times before advancing. This will give you a foundation to build on.

Intermediate facilitator

Once you have facilitated a mob many times, you will start to get a sense of what is needed and what each part contributes. You will start to move in a direction uniquely your own. At this stage, focus on the purpose of each step and start to play with the variations suggested in this book and the variations that come up from your own team. Go slowly. Try to vary only one or two things per session.

Advanced facilitator

Disregard the steps. Just pay attention to the things that are working for you and focus on doing more of those things. Pay attention to subtleties of your team and the lessons of the retrospectives. At this point, the book has served its purpose and is only here to remind you of some of the foundational principles. Now is the time to create your own way of working together.

Not Just Programming

A lot of times because of the term Mob Programming includes the word programming, people wrongly assume that is about the programming. It is actually about the mobbing. You can have mob programming, mob testing, mob designing, mob writing, mob anything. A lot of times we just refer to this under the generic term mobbing. In this book, however, we will often use the term mob programming to be distinctive to the method. Please do not limit this process to programming and feel free to substitute which ever word you desire for programming.

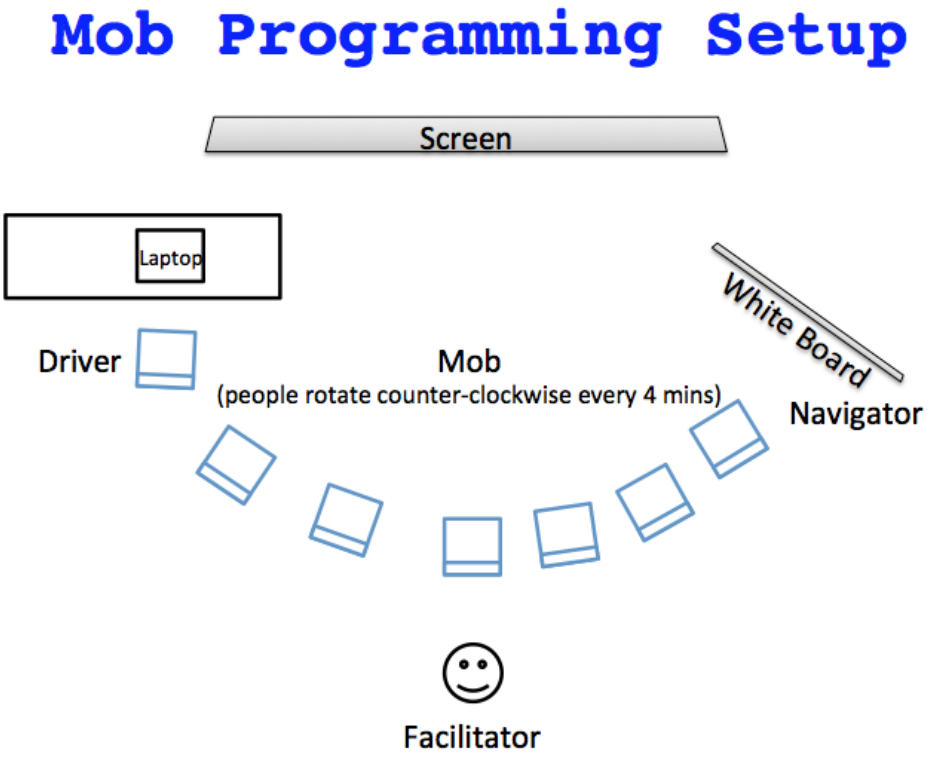
TL;DR: It's really **Mob _____ (insert activity) _____**

Setting Up the Space

For your first time doing mob programming, you do not need to worry too much about the space. We're going to lay out just the most critical things that are helpful.

Note: If you are doing this with more than ten people, please check out the chapter *mobbing with an audience*.

Basic Setup



Mob Programming Setup

The Screen

The screen, projector or TV should be visible and clear to everybody in the mob. The chairs should be facing forward towards the screen as much as possible. People will need to stand up and move around frequently, so there should be enough room to do so comfortably. It is not helpful to have backpacks and laptops during this. Usually we put bags in the corner of the room. However, if people are uncomfortable with this, it's not a big deal.

It is also important to have a whiteboard where the navigator can express ideas.

The Facilitator

As you are reading this book, you will most likely be the facilitator. The important part to know is that mob programming works much better in the beginning when there is a facilitator. As a facilitator, your job is to ensure that all the steps of mob programming are being carried out appropriately. Most of the time, you will not be doing the programming yourself. The exception being if you need to pause the mob to introduce a new idea by temporarily stepping into the navigator role. You do not need to be a good programmer or even a programmer at all to be a good facilitator. This is not a team lead position. When everything is going well, you will be doing nothing at all.

The Work

The first question is always: "What are we going to work on?" While there are many answers to this question, whatever you decide to work on, it should be simple. You are going to learn a lot working together as a mob. Don't try to do that while adding the extra complexity of a super-hard task. After you've learned to work together, it's a good time to tackle the hard tasks.

There are three common items to work on as your first task.

1. **Simple work task**

   If you have a simple task to do, this can be a perfect place to start. Just do it as a mob.
2. **Refactoring large methods**

   Many teams have code that is hard to read and understand. A refactoring for readability exercise makes a great first experience in mobbing. Simply choose a

method that everyone agrees is troublesome and you are going to work with soon anyways. When you do this exercise, we suggest starting with the simple extraction of paragraphs of code and giving them better names. The only two refactorings we suggest are extract method and rename. We also suggest that you commit frequently, usually after each paragraph.

3. **Programming Katas**

   Katas are simple exercises that are used to practice programming. The more common ones include FizzBuzz, Roman Numerals, and Tic-Tac-Toe. For more Katas, check the reference in the back. However, any problem usually makes a good Kata. These are often done in test-first style of programming.

The Computer

While you can get away with just about anything for your first mob, here are some tips to make things easier.

1. **Keyboard and mouse**

   Having an external keyboard and mouse just makes everything simpler for the driver. An external keyboard and mouse can also allow you to close the laptop, which is even better because then everybody is looking at the same the screen. This allows the driver to cue off of people pointing to the screen.

2. **Screen**

   While you usually have little control over the screen or projector, when you have a choice, try to get a bigger screen and a high-resolution projector. Remember that the code often has many more details than a regular slide presentation.

3. **Simple editor with line numbers**

   Line numbers make it easy to talk about where your focus is at. Editors that allow for simple typing and scrolling make it easier for everybody to track what is going on. In particular, editors like vi and Emacs add to the cognitive load if not configured in the way each individual member of the team normally uses them. Finally, make sure the font size is big enough so that everyone can easily read it.

Seating and roles

The Driver

The driver is the typist. There should be "no thinking" going on by the driver. This means that for anything to happen, there will need to be talking involved. It is important that the driver trusts the navigator and does their best to listen and do what is asked.

The Navigator

The navigator is the main person programming. While they will take insight and help from the mob, they are the person who has to make the final decision on what to do. They should be talking in the highest level of abstraction possible. However, in the beginning, this is often at the level of keystrokes and simple programming structures.

We will go into this more in the chapter on **Strong-Style Pairing**.

The Mob

The mob is checking the navigator and contributing insights when appropriate. Remember that you will be rotating fast and soon a new person will be navigating. This forces the other people in the mob to pay attention.

The Facilitator

The facilitator sits in the back and does not rotate with the rest of the mob. If it is necessary for them to step in, they can pause the mob and assume whatever role is needed, except that of the driver.

The Rotation

In the beginning, you will be using a 4-minute timer. This is usually your phone. It should have an audible (but pleasant) sound at the end of each turn. At the end of each turn, everybody stands up and rotates to the next seat. The navigator should become the driver. The driver should join the mob.

Note: Experienced mobs will use a special software instead of phones as their timer. We recommend not using this the first time people are mobbing.

Congratulations!
You are now ready to start your first mob programming session!

Working in Your First Mob

Mob Programming

Is the mob working?

Heuristic: If everything is going right, rotation will not disrupt the flow of the mob.

Preparing the navigator

Part of having a good mobbing experience is being able to have a clear direction that is shared throughout the team. A few little tricks can be done to enable this to happen as it is not a habit most people have acquired yet.

Ask the navigator what they are going to do. Here's some examples of when a navigator has a clear direction versus when they are lost.

Lost:

Q: *"What are you going to do?"*
A: *"I don't know."*
A: *"Something with the test."*
A: *"Get it to work."*
A: *"The next scenario."*
A: (stares blankly)

Has direction:

Q: *"What are you going to do?"*
A: *"The code does not compile. We need to fix that."*
A: *"We need to write the test for handling positive numbers."*
A: *"I'd like to give that paragraph a better name."*
A: *"We haven't checked in the code for a while, let's do that."*

Sometimes you can get to a direction through a series of questions. For example:

Q: *"What are you going to do next?"*
A: *"I don't know."*
Q: *"Does the code work?"*
A: *"No."*
Q: *"Why?"*
A: *"It doesn't compile."*
Q: *"Why not?"*
A: *"The class does not exist."*
Q: *"What are you going to do next?"*
A: *"We need to create the class."*

Examples

If you are working on a task or doing test-first development, it is very useful to have an example written on a whiteboard. The examples should be very simple, and only show one path at a time. This means you can **not** use conditional words like "or", "if", "depending", or "maybe". If there is an example, it might take many turns before its finished, and having it on the board will give guidance for the whole team on what they need to do as each new navigator takes over and continues on the task.

Examples should make you feel like the person telling the example actually did the thing yesterday. Imagine a teenager talking to their parent.
Q: *"What did you do last night?"*
A: *"I went to a friends house and played Scrabble."*
Q: *"How did the game go?"*
A: *"Well, if I had played words on a triple letter score, then they would score three times."*
Q: *"Oh, who won?"*
A: *"The person with the highest score."*

Does this sound like the teenager actually played Scrabble last night? This is what bad examples feel like. You want a good **concrete** example.
Q: *"How did the game go?"*
A: *"There were three players: me, Jennifer and Samantha. Samantha went first playing the word 'again' for 7 points. ... Jennifer ended up winning with 297 points."*

Now you have an example that you can turn into code.

These examples should be drawn on a whiteboard and they will help everybody in the team be on the same page. Being on the same page means that everyone is working towards the same goal.

Consume first

It is extremely beneficial to use the "consume first" style of programming as opposed to the more common "build-up" style of programming. This allows the group to have a clear idea on where we are going and what is needed. Let's look at two examples to show how this works.

Build-up example

```
public int x;
public int y;
```

Q: *What do you need to do next?*
A: *It's almost impossible to know what to do next right now!*

The navigator had a plan, but unless you are in their head, you can't continue it on. And it is hard to check that what they were doing is correct.

Consume first example

```
Point p = new Point(10, 20);
```

Q: *What do you need to next?*
A: *I notice that Point does not compile yet. Let's create that.*

```
public Point(int x, int y)
{
}
```

Q: *What do you need to next?*
A: *I notice that we are not holding on to the x and y.*

```
this.x = x;
this.y = y;
```

Q: *What do you need to next?*
A: *I notice that it does not compile. There is no this.x.*

```
public int x;
```

Q: *What do you need to next?*
A: *There's no this.y.*

```
    public int y;
```

Because the overall picture was created and used, we were able to fill in the spaces even if the navigator changes. Also, if in the very beginning it turns out that we should have used a double (20.5 vs. 20) we would have caught it right away and changed it to a double instead of an int. These kind of mistakes and assumptions happen all the time in programming. Consume first style allows the group to catch them and come to a shared understanding.

Yes, and...

When working as a mob, it is important to follow the "**Yes**, **and**..." rule of improvisational theater. The idea here is to continue with what you have. Do not to delete and undo what the previous navigators did before you. You can refactor but do not rewrite. This allows progress to be made continually and prevents people from being shut down in the group.

If you follow this rule, then each step in the rotation moves the mob further ahead than they were before.

Intentional code

Related to the **consume first** style of programming and writing on the whiteboard, it can be very helpful to write a code comment in English to state what you intend to do. Note that you can refactor this comment as well.

```
//take the individual points of 6,7 and 0,0 and combine them to form a line
```

While this says what we want to do in English, it's not as clear as it could be. We can refactor at this point.

```
//create a line from 6,7 to 0,0
```

Now that the English is cleaner, there's a much better chance that the resulting code will be cleaner.

Writing the English allows you to quickly write 3-4 lines at once, which will then serve as bookmarks for what still needs to be done. Once the code is written, you can delete the English. It is merely a stepping stone to write from intention.

When you are translating the intention to code, you should usually end up with one line of English per one line of code. So if you write five comments, you should end up with five lines of code in that method. This will influence encapsulation either by methods or by classes in the resulting code.

Native language

If your native language isn't English, start by stating the intention in your native language. Translate into English and then translate that English into code. Even though most groups code in English, regardless of their native language, the intention and subtleties that occur in one's native language are powerful and valuable. Skipping the step of forming your intention in your native language will give you worse results.

Code in English

A lot of times programmers are so used to thinking in code, that you have to remind them to think in English. Here is a common example.

We are trying to write the method that calculates the perimeter of the triangle.
Q: *"How do you calculate the perimeter of the triangle?"*
A: *"First we create a variable to hold the perimeter.*
Next we iterate through the different sides and for each side we add the length to the existing perimeter.
Finally we return the perimeter."

Notice two things. First, there is an awful lot of programming words in this English (variable, iterate, return). Second, the programmer is giving the comments for a for-loop implementation that is already in their head. This is not how a regular person would tell someone to calculate a perimeter.

To correct this, you might say something like
Q:*"How would you tell a child?"*
A:*"You sum up the lengths of the sides."*

Because this intention is more intentional and less prescriptive, there is a lot more freedom to come up with different code translations.

Small steps

There are two measures to pay attention to when it comes to small steps.

1. How long has it been since you've seen feedback?
2. How long has it been since you checked in?

These two measures serve separate purposes.

The first measure of feedback lets you know if the group is taking too large steps when completing their tasks. Feedback can come in many forms. But the two big ones are from the compiler and from executing the program.

Feedback helps to bring everyone in the team to the same mental space. Do not worry about the feedback always being positive. Many times seeing how something fails helps us to know what success looks like. Likewise, a different type of failure is also good feedback. It lets us know that we are making progress. Finally, even seeing failure that confirms that we are where we think we are is helpful, especially for the times when we aren't.

The second measure of feedback is how long since you committed code to your repository. This has to do with how large of a task you are taking at a time as opposed to making progress within a task. It is possible to take very many small steps, while still taking too large of a task overall. Not being able to check in frequently has other negative side effects as well. If you cannot check in frequently, it can be hard to quit when you need to (lunch, end of workday, etc.). Large changes between commits can also make it hard to work with other teams because of merge conflicts. If you are checking in frequently, you will not experience merge conflicts - however, other teams might. One saying that emphasizes this is:

Be the bird, not the statue.

The Rules for Working with Each Other

It is useful to post the following rule on the wall as well as get a general working agreement from the mob.

We will treat everyone with kindness, consideration and respect.

One interesting aspect of treating each other with kindness, consideration and respect is that it tends to make everyone in the mob treat each other better and like each other more.

Ironically, cognitive science teaches us that the people we treat kindly we end up liking, rather than the other way around.

Kindness is rather self-explanatory, but consideration and respect are worth going into.

Consideration

Consideration is really about listening. This is something we don't get a lot of practice with because of the focus on individual contributions. The place it is going to show up the most is at the driver seat. Often the driver will start by not listening to the navigator.

The facilitator can usually fix this by simply saying "No thinking at the keyboard" or "There's too much thinking going on at the keyboard".

Another way that this will manifest itself is that good ideas will be spoken by members of the mob and utterly ignored. Usually these ideas are softly spoken by more introverted members. As a facilitator, it is your job to call attention to those ideas and make sure that everyone gets heard.

Over time the mob will learn the habits of listening to everyone and people will find their spots to contribute. As we gain experience being heard we will also change how we speak.

Another aspect to consideration is to remember to allow other people to shine. You don't always need to be showing how much you know. And if someone has a differing way of doing it, try their way first and then consider if your idea would make it better. It's ok to try a solution in multiple ways.

One final violation of consideration is when members who are not in either the driver or designated navigator spot, tune out, usually opening their laptops or phones. This can be a tricky one to handle. I prefer to handle it with environmental settings. Chairs without tables to put a laptop on, and frequent rotations will do more to keep everybody engaged than disciplining them when they are not paying attention.

Respect

> ***We always assume that the person who wrote the code before us did the best they could with the knowledge and circumstances they were in at the time they wrote it.***

Nothing is more corrosive than disrespect in a mob. A group can bond over making fun of some code that was written by someone else but you will pay a high price for that

bonding when everybody is nervous that some day it might be them that everybody is ridiculing. We want to create a space that is safe. Safe to experiment. Safe to learn. Safe to show your vulnerabilities and weaknesses. Safe to look at and improve code without judging and criticizing the author. Remember that mob programming exposes a lot about everybody involved and we need to be safe and supported.

Closing a Mobbing Session

Learning

Learning is fundamentally made up of two parts. The first part is experiences which are the fuel for learning. You need to have experiences to learn from. You've just had an experience of mobbing. This is a great opportunity to learn. Your tank is full of fuel.

However, in and of itself, fuel is not enough to get you anywhere. You need to burn it. The way you harness the potential energy of your experiences is by retrospecting over them, thinking through them, examining them and imagining new experiments for future experiences.

When we finish a mobbing session, we will run a retrospective to help us learn all we can from the session. There are many ways of running a retrospective. We are going to show you the most common way that we use.

Observation Retrospective Framework

A basic idea for an observations retrospective is silently collect observations from each participants on sticky notes. The facilitator will then read out each of the observations while placing them on the whiteboard. The facilitator tries to group similar things near each other so that groupings and patterns can emerge.

Part 1: The Explanation

You want a safe and fertile place in which to gather observations from your group. If the group has never done an observation retrospective before, you will want to take a moment to explain why you are doing this so that people will feel more comfortable and you will get more and better observations.

There are four parts to this explanation to do:

1. Selective attention
2. Expectation biases

3. Single perspective
4. Using sticky notes

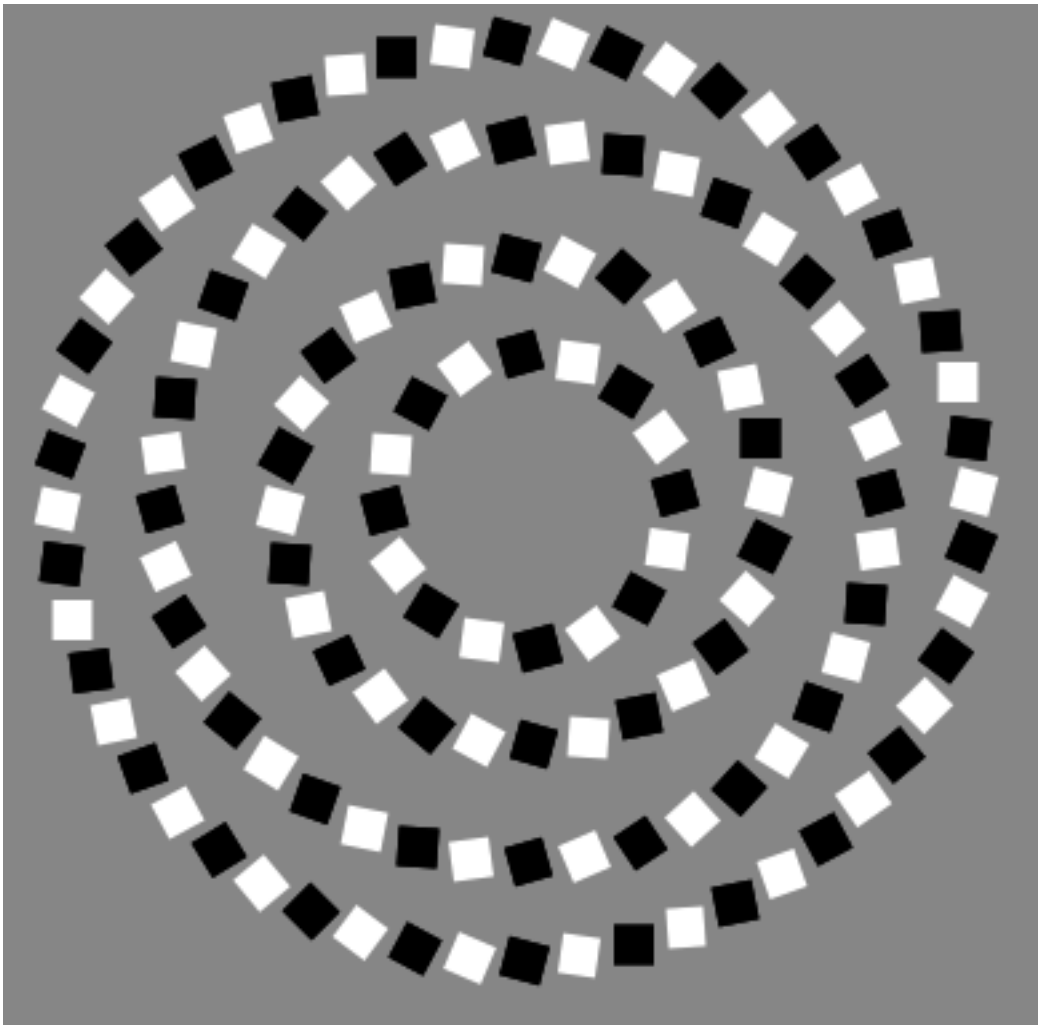
Selective attention

(We usually show this video: youtube search: Selective attention)
Take a look at these two pictures.

You might notice a few things that are different. And you might think the things you notice are obvious. But if you have a group write down the things, each one will see things the other one doesn't. No one knows which ones are obvious to them and not obvious to someone else. This is why we ask all your observations, no matter how obvious they are. Everyone misses a lot of things right in front of their eyes.

Expectation biases

(We usually show this video: youtube search: Faa Baa)
Look at this picture.



As you look at the picture, you might notice it moving or that it's spiraling. It is doing neither. However, while many people think of our eyes as cameras and our ears as microphones, what we experience as sight and sound is really put together in the visual and auditory cortexes of our brains. And this is a lot more akin to photoshop.

Our expectations and biases mean that we see and hear things that might not have ever happened. One reason we collect observations is to get the things that happened in front of our eyes that we missed. Another reason is to dispel illusions that we think happened but never did.

Single perspective

(We usually show this video: youtube search: assumptions)
Here's another picture.

What you are seeing is a trick called forced perspective. When you only have one viewpoint, it is easy to be mislead. Everybody is vulnerable to this because everybody experiences things from a single perspective. But we have multiple perspectives in the room when we mob. By taking collective observations, we reduce the chance of being fooled by a single perspective and get a better understanding of the full reality.

Using sticky notes

On our list of cognitive biases is rationalization. There are many simple things that I have learned long enough ago that I forgot I ever learned them. When faced with knowledge and no memory of acquiring that knowledge, it is rational to assume that I always knew it and that is just another aspect of common sense.

How do you write sticky notes is one such simple task that often falls prey for this rationalization. Here's what you need to know.

While each of these rules is simple, notice that when taken together the green side is much easier to read than the orange side. And because humans engage in fight or flight responses, most people, when faced with the challenge of reading the orange side, instead choose to skim over or zone out. Following these four simple rules helps to ensure that the mobs observations get read. A good example of overwhelming information is:

> Lazy person fact #6543671283

You were too lazy to read that number

For sticky notes, it is also important to have good markers to write with. We suggest Sharpies.

Part 2: Collecting observations

When you ask for observations, if you just say *"give me any observations"*, then people get overwhelmed by the vastness of possibilities. Instead, you want to highlight the different areas that observations can come from. Think of this as a spotlight pointing out the different corners so that people will give you a greater quantity and variety of observations. For example, try saying something like:

"Any observations are valid. You can have observations about syntax of the language, the code we used, the IDE and tooling, how we worked as a group, emotions you felt, worries or concerns about the feature, things you liked or disliked. Or things that surprised you about today."

After you ask for observations, you want to give enough time for everyone to write them down. I usually try to count to twenty in my head after I have heard the last person stop writing, just to make sure we got everything.

Collect all the sticky notes and bring them up to the whiteboard. It's time to start reading observations.

Part 3: Reading observations

Mainly you are just going to read the observations out loud and place them on the whiteboard to allow for groupings to emerge.

Here are a couple of key things to remember.

- Use your whole space on the whiteboard or wall.
- If you do not have enough space for the amount of stickies, don't worry. Just read them out loud and don't worry about the groupings.
- If there is any confusion, this is the time to ask for clarification.
- If you feel there is something to add or compliment or even dispel, you can do that while reading the sticky.

- If a sticky opens a larger discussion, place it to the side and return to those stickies after going through all the rest.
- After everything is done, it can be useful to circle and label 3-4 groupings that have emerged.

Final thoughts

The entire observation period should take 15 minutes on average. It is important to plan for that time as it is easy to run over and skip it if you don't.

If you are mobbing for a full day, we recommend two observations retrospectives, one before lunch and one at the end of the day.

When retrospectives are frequent and short, the outcomes from them are small. But those small outcomes are usually acted upon and problems can be nipped in the bud. When they are long and infrequent, you get big outcomes that are largely ignored.

In your first retrospectives, don't expect a lot of observations. Like everything else, it takes time to build the habit of noticing what is going on. Part of the reason to do regular retrospectives, is to build the metacognition abilities of your team. Once those are built, you might notice people making observations while the mobbing is happening and even writing them down. At this point you might be tempted to abandon the retrospective all together, as it is no longer needed. Don't. If you do, you have to rebuild that skill up after a couple of weeks when the habit fades away.

Pictures used from Google image search labeled for reuse. The last picture of sticky notes is from Michael Sahota.

Mobbing Cheat Sheet

Mobbing

1. Arrange people in a circle with the driver at the keyboard and the navigator as far away as possible
2. Rotate every four minutes, navigator becomes the driver
3. Make sure driver is not thinking at the keyboard
4. Small steps, frequent feedback
5. Small steps, frequent check-ins
6. Monitor for kindness, consideration and respect
7. End while still happy (2 hours or less)

Retrospective

1. Retrospective
2. Explain observations and sticky notes
3. Collect observations
4. Read observations and place similar ones near each other

Strong-Style Pairing

An important aspect of mobbing is manner in which the driver and navigator(s) work together and split roles. This style of collaboration comes from Strong-style pair programming as discovered by Llewellyn Falco.

The basic idea is:

> ***For an idea to go from your head to the computer, it must go through someone else's hands.***

While programmers have years of practice taking an idea from their head to the computer through their fingers, most programmers have little to no practice articulating their thoughts so that another person can understand them.

Here are some tips and exercises to help develop the skill.

Basic navigation flow

When you are navigating, there is a 3-step process you tend to follow.

1. State intent
2. Location
3. Keystrokes

The navigator is always trying to speak at the highest level of abstraction that can be understood. Even if the high level intention is not enough, it is important to say it so that the driver can map the actions to higher level concepts.

If the driver does not seem to react when you give the intention, or if they move to the wrong location, your next step is to get them to the right place. Line numbers can be very helpful for this, as well as the use of screen terminology such as:

- **Menu bar**
  The top-level text menus usually starting with File, Edit etc.

- **Tool bar**
  The top-level icon menus.
- **Task bar**
  On Windows machines, the task bar is usually at the bottom of the screen and holds which programs are open.
- **Tabs**
  Editors and browsers use tabs to allow many documents to be open at once.
- **Tab well**
  Editors will sometimes have section areas on the right, left, bottom or top to hold things like file navigators. These areas are called tab wells.
- **Gutter**
  The sideline next to the text, usually where the line numbers are.

Once you've gotten the driver to the right location, if they are still not moving, you will need to give keystrokes. Remember to also use keystrokes for keyboard shortcuts. Keyboard shortcuts can be a little tricky because often they are muscle memory for you and you don't know them yourself. One trick is to pretend there is a keyboard in front of you, and do the keyboard combination on the pretend-keyboard. Also remember that when giving keystrokes, you also want to convey the intent. For example, "Let's extract the method. That is ctrl+alt+m".

Cellphone exercise

One exercise that can be helpful in giving people a sense of navigating and driving is to have a driver use your cellphone to do a basic task of yours on your cellphone using only your words. You do not need to give your own passwords. Unlock your phone and hand it to the driver. Then state your intention. For example, "I'm going to show you how to play my favorite game."

Remember, you are going to do lots of little cycles of the three step basic navigation flow. For example,

"Open up the game"
-no response-
"Move to the next screen" -no response-
"Swipe left"
-moves to the next screen-

"Click on the game icon"
-no response-
"It's in the upper right corner" -opens game-

After you have practiced the driver-navigator, switch roles and phones and give the other person a chance to try the roles reversed. Usually this exercise takes around 5 minutes per person.

Mobbing with an audience

Mob makes a great way to do trainings, and hands-on workshops at conferences. While you can scale regular mob programming up to about 20 people, there's some points where it is more convenient to do mobbing with an audience.

Scaling

Mobbing can scale to about 20 people. But there are lots of aspects to pay attention to.

Number of rotation cycles

When you are mobbing, you want everyone to get several times times on a computer, a minimum of 2-3. If you take the number of people you have times the number of minutes at the keyboard, you get the total amount of time per cycle. Here's some examples.

# people	time	cycle time
6	4 mins	24 mins
10	4 mins	40 mins
10	2 mins	20 mins
15	4 mins	60 mins
15	2 mins	30 mins
20	4 mins	80 mins
20	2 mins	40 mins
20	1 mins	20 mins

The next question is how much time do you have? If you are doing an all day training, 1 1/2 hours cycle time is fine. Everyone will still get 4-5 chances at the keyboard. If you are doing a 1 1/2 hour conference session, you need a smaller mob.

Event Based Rotation

Training and conferences also mean that you may be doing work that is a little more understood by the facilitator. This offers the opportunity to rotate on something other than

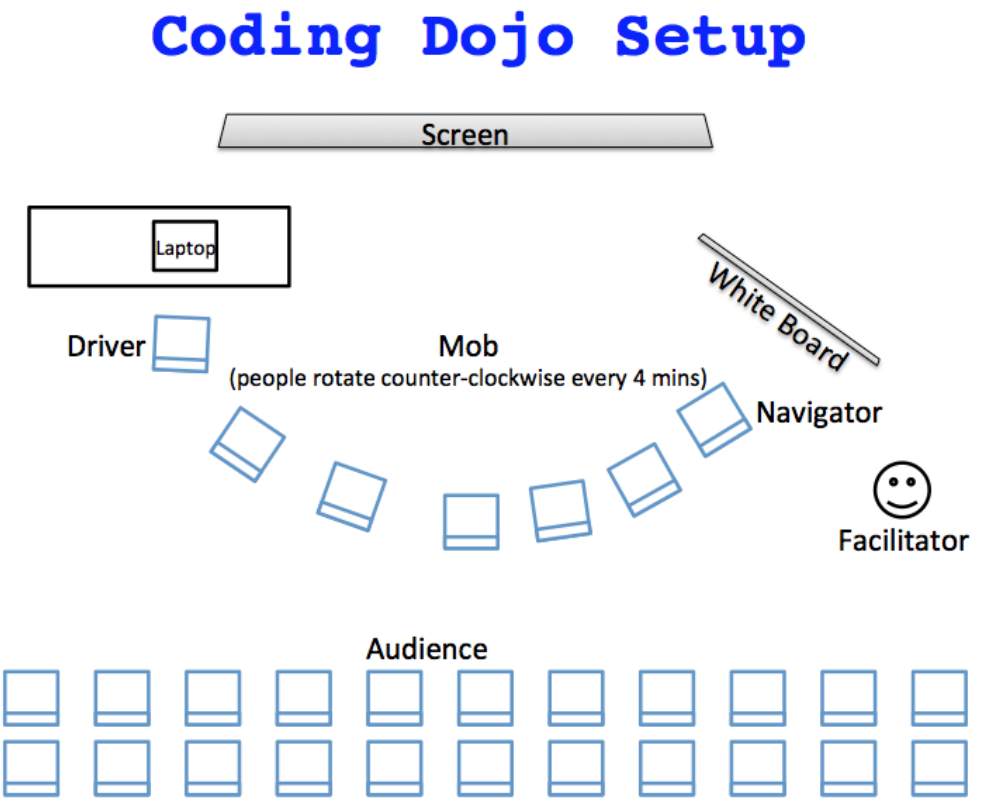
time. When event-based rotation keeps you within an average of four minute rotation time, I prefer it as a method of rotation. It is satisfying to rotate on accomplishment.

Here are some events that I will use for rotation:

1. Unit test is written
2. Unit test is passing
3. Small task is completed
4. Single line of code is written
5. Story (example) is written on a board

> When teaching kids in Africa, we had 30 students on a single computer. We rotated on single line of code written. This allowed everyone to get a couple chances at the computer, even in a short two hour session.

Audience



Mob Programming with Audience

Sometimes scaling the mob doesn't work or give the experience you want for the size of the group in the time you have. In these cases, we recommend having a mob with an audience.

To the main part, this works the same part as a regular mob, with a couple extra rules.

1. People can join or leave the mob at will.
2. When joining, sit right behind the driver.
3. The audience can also volunteer insights and information to the navigator.
4. Audience participates in retrospectives.

If the audience is volunteering too much information, so that the navigator does not get a chance to get their focus and flow, quiet them down to give the navigator a chance. Sometimes loud audience members are a sign of people who need to be invited to join the mob.

On the other hand, if the navigator is stuck or frightened, you may need to encourage the audience to help them out.

Diversity in the mob

When you asked for volunteering from the audience to become the mob, you will end up with a subsection of your audience. The makeup of the mob will effect how and what they will do. The people who feel more comfortable volunteering, will volunteer first. Since you're only selecting a small group, you might have to encourage minorities to join early on, or your mob will not reflect your audience. Instead the mob will end up being made up entirely of the most common majority in the room.

There are different types of diversity. Sometimes you may need to encourage programmers without experience in the particular language, non-programmers, different roles, different experience levels, different ethnicities, or different genders. The more diversity you get in the mob, the more the audience can connect, feel safe to join and the better your session will go.

When People Get Stuck

There are times in the mob when the person who is navigating will not know what to do. Here are some strategies to deal with that situation.

Situation 1 - navigator is confused

If you're with a new group, and the navigator is confused and maybe a little bit scared, the best thing to do is to ask the rest of the mob to help them out. This allows progress to continue, and makes people feel comfortable being exposed during that short formative period. It also allows things to go back to the designated navigator as soon as things become unconfused.

Situation 2 - only one person knows what to do

Sometimes you'll have only one person who has any idea how to do the task at hand. In these situations, keep the rotation but allow that person to continue navigating, effectively removing the designated navigator. Also, when the single expert would rotate to the keyboard, rotate twice so that they are skipped. We want to avoid thinking at the keyboard.

This situation should not persist for long. Maybe an hour or two. But then the knowledge should be starting to pick up with other people in the circle. At every chance to you that someone else might be able to do even a little piece of the puzzle, you should facilitate pausing the expert to allow other people to start filling the role of navigator.

Situation 3 - purposefully transferring knowledge

When you have a team that is mobbing, it is good to practice improving the underdeveloped navigation skills in your team. A good way to do this is to do a navigate the navigator - session.

In this session, the more experienced navigators will purposely pull back from navigating. They will guide the navigator through the use of questions and high level concepts.

For example, instead of laying out the steps of unit testing, they might instead ask “Have you thought about writing a test?”. They might even get more direct and say “Please write a test”. But they are simply getting the navigator to navigate the driver. They are never navigating the driver themselves.

Situation 4 - opting out

Sometimes it is a good idea to allow people to opt out of a mob. Sometimes it is a bad idea to allow people to opt out. Here are our overall suggestions.

Don’t start by allowing people to opt out. New things are always a bit scary. By allowing people to opt out before you even start, you make the situation more scary. When everybody is just doing things, it’s what everyone does. If only some are doing it, now you have to be brave and volunteer.

When you start, put everyone in the circle, get the timer quick and don’t offer the option of backing out. After they’ve taken their turn a couple of times, then you might want to offer the option of opting out.

There are two ways of opting out. One is they are not in the room. This method is useful only if the person opting out is having a very negative impact on the rest of the mob. While opting out might allow the rest of the mob to function well, it rarely allows people not in the room to improve. Sometimes, however, the social aspect of the rest of the mob talking and laughing together, will draw people who opted out back into the mob.

The other way of opting out is being in the room but not be part of the mob. Think of this as being part of the audience, observing the mob. The important thing here is to make it welcoming to them when they are ready to join.

Also realize that sometimes people are just full. They need a small break some of what they have just experienced. Again, make it easy for them to rejoin as they are ready.

Remote Mobbing

Remote mobbing is much harder than regular mobbing. We don't suggest it as the way to start. However, sometimes it is the only choice you have.

It will require screen sharing tools (see Reference 3).

There are three different models of remote mobbing. We will outline all of them.

The Remote Employee

In the remote employee model, most of the team is colocated with one member being remote. Sometimes this member is always remote, and sometimes this member is part-time remote. In this situation, the best starting point is to learn to mob colocated. And later extend it remote with the remote employee always skipping the keyboard. This is quite possible once everyone has learned the navigation skills. We also suggest you use the video setup mentioned below.

The Remote Company

In this model, everyone on a team is located in a separate place. There is no company office and there is no chance to learn mobbing colocated. In this model, you will be using screen sharing software that allows for the change of the control of the keyboard. It is also useful to include the use of a mob timer (See reference 2). An odd advantage of this model, is that while everyone is at a disadvantage, they are all at the same disadvantage. Remember to keep your rotation short. Be extra careful that the driver is not thinking. Also, remember to take time for retrospectives and video chat.

The Remote Team

In this model, half of the team is located somewhere and the other half is located in a different spot. The key here is to set up the mob timer so that at any given time, the designated navigator and driver are **not** located in the same spot. The reason for this is to force communication between the two groups. It is too easy for a group talking amongst

themselves not to be heard by the other group. And unfortunately, we have too much practice not paying attention on a conference call. In this model, we again suggest use of a mob timer (See reference 2) and setting up an external video system mentioned below.

Video

The question of video is a tricky one. In general there is not much gained from having video on, especially when it is just your face looking at the screen. The teams that have reported success with video, have a separate computer that is sharing the video and taking the picture. The advantage of this is most of the time you are not looking at the video. When somebody turns their head, you get to pick up many non-verbal signals that the person wants to talk to you.

PART 2

Developing into a Full Mob

All the chapters up until this point have been about getting started. Although we have referred to that as mob programming, it is actually closer to the randori technique with strong-style pairing. The difference between randori and mobbing seems trivial from the outside. It is closer to the difference between little league and professional sports. While much of the trappings are the same, the level of skill and ability is greatly increased. One of the biggest differences is the level of trust and understanding between the participants.

In this section, we are going to start to talk about how to transition a team from their very first mob to fully functional mob. This takes some time. This can take several weeks. Remember to go slow. Introducing one new thing at a time.

Building habits

Now that you´ve gotten started with your first mob, the most important thing is to keep going. As the saying goes: “The best exercise is the one you do”. As a coach or facilitator, you are trying to build habits. Habits take time and repetition to form. As such, there are many good intentions, that get in the way of forming habits. Heres what we´ve learned to avoid and nurture.

End happy

No matter how good a workout may be, if you are exhausted, you don´t want to show up the next day. One of your primary goals as a facilitator should be to end while you´re still happy so that everyone wants to show up tomorrow. Try to limit sessions to two hours in the beginning for this reason. Also be very careful about running overtime, especially when overtime means staying late after work or missing lunch. You don´t want anyone saying “oh great, we´re mobbing again. I´m going to be late picking up my kids”.

There are two big things that prevent people from ending on time. The first is forgetting to leave enough time for the retrospectives. The second is working in too big of chunks, so you feel uncomfortable leaving the task in such a stage of incompleteness.

Also, look for natural endpoints that might come a bit early. It could be better to end 20 minutes early at a good place.

Learning to mob

Another common distraction on the way to learning how to be a functioning mob, is to place too much importance on being productive. While we highly encourage you to do actual production work, don´t worry about being too productive in the beginning. The first few sessions people need to learn to work together. It can be easy to forget that people have had 10-20 years of practice working by themselves and only a few hours of practice working together as a mob. Give yourself permission to slow down while learning a new skill.

Too often people feel stressed and overburdened. Because of this, they don´t allow themselves enough time to learn new skills, and improve. Because they are not improving, they continue

to feel stressed and overburdened with no time to improve. It is an unfortunately common trap.

Timing

A lot of time people will try out new things like mobbing saying “let´s try it every other week for a few hours”. It is very hard to build any habits doing something only twice a month. If you are going to spend 8 hours mobbing, it is better to spend 1 hour over 8 days than 8 hours on one day. Also remember that rest is a very important factor. You want to have a little bit of learning, a little bit of rest.

Monitoring your mob

Now that your team is doing a little each day, the next step is to start paying close attention to how the members of the mob are interacting with each other.

Engagement

Engagement is the basis for shared understanding in your team.

The main difference between a high-functioning mob and a beginning group is engagement. You want to make sure that everybody is engaged. This is particularly problematic for the people who are neither the driver or the navigator. The first step is to model the behavior. If you are distracted on your laptop or your phone, you´re sending a clear message that they should be too.

Engagement signals

1. Quiet

The mob should never be quiet. Quiet not only means that the other participants aren´t engaged, but it means that the navigator isn´t navigating and probably that the driver is thinking. If your mob is quiet, that is a distinctive warning sign that something is wrong.

1. Head orientation

Even though you´re sitting at the back of the room, it is easy to tell if people are engaged simply from their head orientation. Are they looking at the screen? Good. Are they looking somewhere else? Not good.

1. Laptops and smartphones

Although later you will want them to have laptops, it is not a good idea in the beginning. Often laptops will encourage people who are not driving or navigating be completely

disengaged, checking email or doing other activities. Sometimes this will go away naturally by keeping the rotation time tight and making them move around a table. Although we will also implement a no laptop rule when necessary.

This is a very tricky things. When members of the mob check out when they are not at keyboard, it is akin to saying: “this is not important enough to require my attention”. This quickly becomes: “YOU are not important enough to require my attention.”

Full team engagement

When you're learning to mob, you need to have a designated navigator so that you can learn the skill of how to talk. If you don't do this, you will naturally isolate the more quiet people in your team. Then they will not learn to express themselves and not be able to keep up. Once you learn how communicate your ideas on tasks and code to another person you want to move from a single navigator to an entire team contributing at the same time.

Here are some of the changes to introduce as you move towards full team engagement.

No designated navigator

This means that you don't need someone standing up while they're doing navigation. The only position that is actually relevant is the driver. This is a very short period of transition and after a little bit of that I will introduce…

The mob timer

The mob timer is a program that not only tracks the amount of time, but also who is supposed to be at the keyboard. Once you have a mob timer, you no longer need to rotate seats. And when you don't rotate seats, the concept of the designated navigator becomes even more blurry. This frees everybody to focus on the code instead of their turn.

For a list of Mob Timers checkout the **Mob Timers** Reference in the back of the book.

Encouraging participation

Many times multiple people will have differing ideas on how to do something. When you have a designated navigator, this isn't so much a problem because the navigator gets to decide. But once you've moved to a full mob, you are going to have to facilitate. The first rule is to encourage action. As they say in agile, **Don't discuss something for 20 minutes that you can do in 10**. As quickly as possible, try to get to doing something.

However, you also want to encourage everyone in the mob to contribute. Quieter and more introverted people will easily let go of their ideas and not fight for them. The best way to encourage all ideas to stay present is to do multiple solutions. If Sally and Tim have a way to solve a problem, try one first then the other. Afterwards decide which one was best. You get very different conversations with hindsight than you do beforehand. Sometimes what results is that one of them won't work. Sometimes a third or fourth way might present itself. This is ok, do all four.

In the beginning, it is important to try each one so that everybody knows they can be heard. After you've done that a few times, people will start to trust that they can be heard and then it is ok to confirm after the first one that you still want to do the second. It is surprising how often you will get the response "Who cares, it works". It is important that you establish trust and safety so that this comes from a place about the code and not defeat. But there are other things to do to guard against team members feeling defeated.

Preference the weird and the quiet

A good way to encourage participation and learning is to preference people who speak less. If one of your team members rarely speaks and they speak, try their way first. Also, if one of the ways sounds very weird, preference that. This gives people courage to speak up and sets the habit of stepping outside your common routines.

Avoiding decision paralysis

There are two ways that people tend to get stuck. One is when into the vastness of a large open space. Think of this as you're staring at a blank page. There's so many different things. Where do you even start? It is very easy to get caught up in the abstract concepts and if you find your team stalled, what you need to do is get them to focus on a specific task.

Almost always an example is the right thing to do right now. Even if it is the wrong example, its failure can help you figure out what success looks like. Choose a path and go.

Conversely, you can also get tied up in the details. What is the perfect shade of red? Is it #ff0000 or is it #ee0000? Have you considered #dd0000? Or even #ed0000? People can spend an absurdly long amount of time talking about these types of details. This is normally referred to as bike-shedding. The term comes from a saying "If you give people plans for a nuclear power plant, they will look at it and think it is ok. But if you give them plans for a bike shed, they will argue for days about the color of the paint." Although this paralysis is the exacts opposite of looking into the vastness, the solution is still the same. Pick one and go.

Normally, asking the group a roman vote is an easy way to force a decision without the decision coming from the facilitator. This allows the facilitator to stay a facilitator instead becoming a decision-maker.

Roman vote

Roman vote is a quick and easy way to get input from the entire team. Simply have everybody stick their fists out with their thumb up. If they agree, point their thumb up. If they disagree, put their thumb down. If they don't have strong feelings, they can put their thumb sideways. Either way, everybody votes, at the same time, all at once. Majority wins.

Walking away

As a facilitator, one of your jobs is to make the team self-functioning. This means they can function without you there. If you are going to help them to function without you there, you can't do that while always being there. What you want to do is start adding little periods of time while you're not there so that they are used to you not always being there. Walk away to get something to drink. Have a conversation on your phone. Take an extra long bathroom break. When you come back, you notice the team slips a little bit. This is good. Get them back on course, then give them another chance to continue without you. Over time, you want to grow the periods when they are alone. Think of this as turning your team from kids to adults. You can't expect self-organized decision making if you always do decisions for them. You need to give them safe spaces to try, fail and succeed on stuff on their own.

Sometimes it is really hard not to step in when you are a facilitator. You know the right answer. And it kills you to stay silent. Walking away at these moments is a great solution.

Tip. Carry a refillable water bottle. This gives you an excuse to go refill it and will also remind you to go to the bathroom.

Holding the space

Kids will play with their friends differently when they are home alone compared to when their parents are there. This is true even through in both situations they will not talk to their parents, but play by themselves. This concept is referred to in the facilitator-world as holding

the space. If you do you set up the right habits and patterns, you can enforce them by your mere presence in the room.

This is not that surprising. But it can be hard to value holding the space. To realize that you are doing nothing but being present is something that you need to prioritize and give time to.

What Established Mobbing Looks Like

On the first day Joy came into the office, the team was already working. She sat down watching them as they said hello. She was surprised when they just continued on their way, but even more surprised when a few minutes later timer went off and they added her name on the rotation list. She was nervous at first, a lot of what they were saying was unfamiliar. And she did not want to appear stupid in front of the group. But everyone else was taking their turn, so when it was her turn, she went up to the keyboard making a small joke to remind that she was new.

It was not so hard. People seemed to know where she needed help, and gave lots of details and hints to help her through the new areas. Still she was relieved when the timer went off and she went back to the group. The ideas seemed to be coming from the team as opposed to a single leader. Some of the team members would become more quiet, but then speak up from time to time, warning of a mistake or forgotten requirement.

She started to notice that different people of the team had different strengths they brought to the mob. Eric was the most senior member and been at the company for a long time. Many times people would ask for information about obscure code or piece of history from him. Jessica seemed to always be one step ahead on her laptop, often pulling up sample code from Stack Overflow right about when team was about to run into problems. Timothy did the same thing but with real people, often going to a product manager, analyst or another section of a company and bringing over the person we needed to talk to. Steve seemed to be the most into the geeky language features of the code, often suggesting optimizations or technologies we should be using right now.

Soon Joy started to feel comfortable asking questions. Not just comfortable but like she understood enough to ask valuable questions. Sometimes instead of direct answer, the mob would just say "Oh, we had not thought of that". It was odd being able to contribute without knowing the answer herself. One time when they were stuck understanding why the prediction engine was working the way it was, she asked "Why do you expect it to give this result?" "Because of the pricing history", Timothy responded. "Not the pricing history", said Eric. "It's the browsing history." "Oh, right", said Jessica. "We need to cross-reference that data or it will never work." Joy saw this a lot. Different insights into the code coming from different people. Even in the code they had just written. She could not identify whose idea it really was. This also came into play when the code broke. In her last job, she remembers people getting teased mercilessly when it was discovered they had written a bug. Here, when

something went wrong, there was no sense of defense or attack.

Instead, they would often do small retrospectives. Asking for ideas on how to prevent it from happening next time. Brainstorming on sticky notes and categorizing them on the wall. There was a lot of writings on the wall. People would sketch out scenarios on the whiteboard. And she mentioned she did not understand the mechanics of S-expressions, a couple of other people admitted they did not either and they added a post-it note on the learning hour ideas. She'd have to make sure to get in earlier tomorrow so she did not miss the learning hour.

By the end of the day, she was already feeling part of the team. She noticed that everyone had their chair set up to comfortably look at the wall all day. She was planning on bringing in her cushion for tomorrow. But more than the environment, she noticed that the ideas she was having were being listened to. It seemed like people were giving her a little extra space, because surely the other members of the mob could contribute the things she was doing right now. The fact that they were willing to let her, the new person, guide a little bit, really meant a lot.

The day came to an end at 5 pm, Joy was relieved to see that they were not working late into the night. It made it easier for her to get home on time.

REFERENCES

Online as well

Here are some handy references to use while mobbing. They are online as well in case you need them and left your guidebook at home.

http://www.mobprogrammingguidebook.com/references

Deliberate Practice

Development katas are simple exercises that we can do for the sake of practice.

FizzBuzz

1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,11,Fizz,13,14,FizzBuzz
Print the numbers from 1-100 replacing numbers divisible by 3 with Fizz, divisible by 5 with Buzz and both with FizzBuzz.

Roman Numerals

1. Given a number, convert it to a roman numeral.
2. Given a roman numeral, convert it to a number.
3. Program roman numeral addition, without allowing to convert to numbers in between.

Games

Most games make for good katas. Here are a few of our favorites

1. Tic-Tac-Toe
2. Battleship
3. Minesweeper
4. Bowling (score card)
5. Blackjack
6. Checkers/Chess
7. Yahtzee
8. Scrabble

Checkouts

Create a receipt and checkout program. Interesting pricing rules include

1. Multiple items
2. Discounts if 3 are purchased
3. Discounts if 3 or more are purchased
4. Weighable items (3.5 pounds of candy)
5. Rates per ranges (0-1 lbs $5/lbs, 1-10 lbs $4 per lbs, 10+ 3$ per pound)
6. Taxable items
7. Buy one get one free promotions
8. Combo Promotions ($5 off hotdogs & buns)

Math

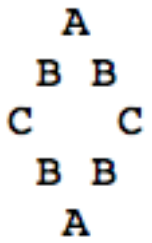
Math problems are also fun.

1. Triangles (perimeters, area, angles)
2. Shortest route
3. Integer Range (does 1-100 contain 65? overlap 50-200? contain 5-10?)
4. Prime Factors
5. Change Machine (which coin combinations to give for 55 cents)
6. Game of Life
7. Leap Year

Words / Text

1. Anagrams
2. Secret Decoder Ring
3. Word Wrap
4. LCD numbers
5. Natural Sort (1.txt,2.txt,10.txt,20.txt)

6. Diamonds



Exploratory Testing

Test Target

You can use any software, website application or electronic device to do an Exploratory Testing kata. Some of the qualities that make a good test target are:

1. Understandable Domain (either common or team specific)
2. Target rich - a fair amount of problems to find
3. Small area to test
4. GUI, API or Physical Interface

Charter

Choose a single charter for your Kata. Possible charters include:

1. List Features while testing (Mindmap)
2. Find Bugs & Questions
3. Toolings (Bug magnet, Chrome Developer Tools, Burb Suite, etc…)
4. Create tests scripts (Selenium, Cucumber, etc…)
5. Create bug reports
6. Thorough notetaking
7. Test Planning (list possible areas and charters)
8. Performance Testing
9. Security Testing

10. Enhancing Testability & Supportability
11. Different Environment Testing
12. Usability Testing (personas, Nielsen heuristics)
13. Analysing logs
14. Heuristics (SFDPOT, HICCUPS, etc…)

Non-development

Sometimes you want to practice mobbing in a non-technical setting.

Writing

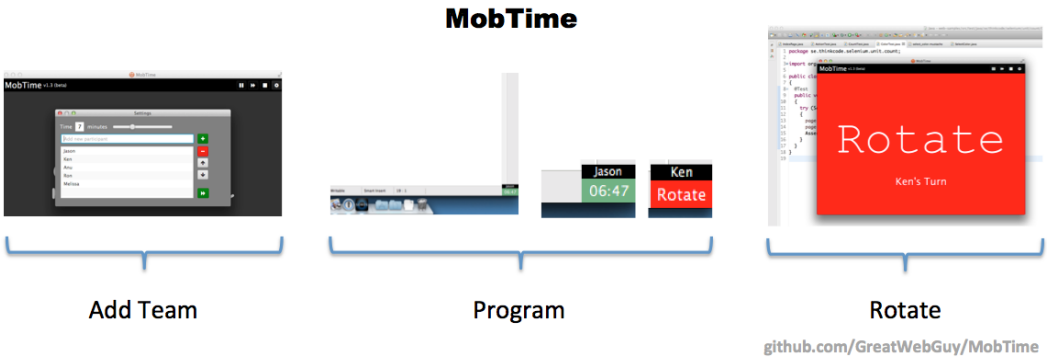
Write a blog or an essay. Remember to outline (at minimum the paragraph you're working on) to keep everyone in sync.

Other

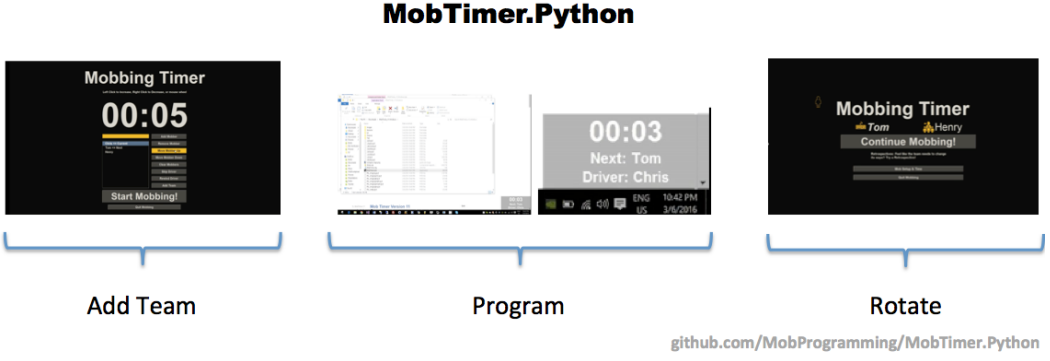
Any suggestion can make a good kata. Our rule is whoever suggested the kata becomes the product owner and is responsible for defining the user stories and prioritizing them.

Mob timers

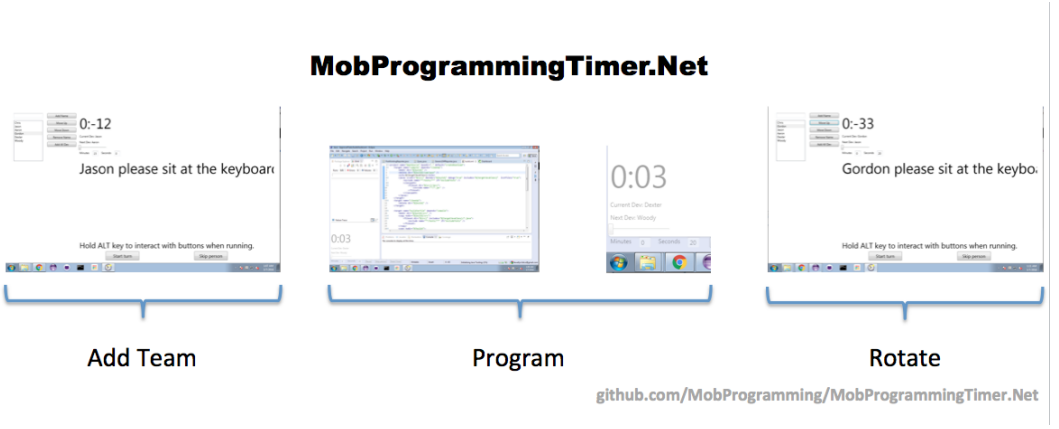
Here are a few mob timers *(In order of our preferences)*:



https://github.com/GreatWebGuy/MobTime/releases
Windows & Mac
Easy to see countdown; Easy to pause.



https://github.com/MobProgramming/MobTimer.Python/releases
Windows & Mac
Full-screen forces rotation; Easy to see countdown



https://github.com/MobProgramming/MobProgrammingTimer.Net/releases
Windows
Hard UI; Full-screen forces rotation

http://oss.jahed.io/agility/timer.html
Web-based, no install needed
Can't see while programming

Screen sharing tools

Here are a screen sharing tools:

Features * Remote control - not all tools allow others to control the desktop, these do * No install - these tools allow viewing without installing

Voice and video

Skype

This is a popular video conferencing app. It will share screen but without remote control. On a plus side, it's voice and video is usually good. It is often used with a different app that is doing screen sharing.

Google hangouts

This requires a plugin installation on a browser, but not an app of its own. Almost everybody has an account. But, it does not do remote control but will run from your phone if you want to use that as your secondary video device.

Screen and control

Join.me

This has remote control and the viewers and can do it without install through a web page.

Virtual environment with Shadow

Nowadays, it is easy to set up a virtual machine on Amazon and have everyone remote into it using Windows Shadow (for administrators). This has many advantages around speed and control. Because everybody is going into Amazon, personal bandwidth is minimized and

no user has a speed advantage because it's their machine. Also, Shadow is extremely quick, meaning IntelliSense can feel responsive. This also has the advantage that the dev machine always available regardless who is sick. Just remember to turn the machine off when you're not using it.

VNC

VNC is a screen sharing app from the days before screen sharing was an internet phenomenon. You have to set up your own server and open ports or create tunnels to connect if you are going through the Internet. However, it does give you complete control and will work in systems that don't have internet access. It does not handle voice and video.

Full package

Screenhero

This screensharing has had closed circle of users, meaning only those who are already in can give you access. It is to be integrated with Slack. It has nice optimizations around screen sharing and remote control.

Appear.in

Video-conferencing software. Works well with anonymous guests. No install.

WebEx

Video-conferencing software. If your company is already using this, you already know about it and will probably use it. If not, don't worry about it.

Zoom.us

Video-conferencing software. Works well with anonymous guests. No install.