MOB PROGRAMMING GUIDEBOOK

MAARET PYHÄJÄRVI

MOB PROGRAMMING GUIDEBOOK

This book is available for download at http://mobprogrammingguidebook.xyz

This version was published on 2019-04-07.

TWEET THIS BOOK

Please help Maaret Pyhäjärvi by by spreading the word about this book on Twitter.

The suggested tweet for this book is:

> I just got #MobProgrammingGuidebook by @maaretp.

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: #MobProgrammingGuidebook

THANK YOU

To the person reading this book in progress,

Thank you. For considering mob programming as something of interest. For considering my ideas and experiences as something that could help you on your path. For reading this book.

I self-publish so that I can make text available to you before it is all completed and finished. My work of learning and teaching Mob Programming is far from done. Writing a book incrementally enables me to create a place to collect bits of that knowledge with the goal of the guidebook: condensing information you will need to get started on a good foot with your Mob Programming journey.

I would love to hear from you. What questions you have, what lessons have been useful? Creating a book takes a lot of persistence, and for authors like myself, a sense of community. Every time I see an email saying someone purchased the book, I cannot thank you enough for the motivational boost.

Maaret Pyhäjärvi

maaret@iki.fi

CONTENTS

PREFACE

For Tampere Goes Agile 2014 conference, I invited Woody Zuill to deliver a keynote: Mob Programming. Whole Team Approach. I remember sitting in the audience, thinking that what he was describing was interesting but that it *would not work where I worked*.

With a few years and experiences in the software industry behind me, I've learned to recognize the triggers that often lead to new insights. Resisting without having tried something is one of the major triggers.

Back at office I started working to convince my team to try it with me. Them caring about my happiness was what allowed them to risk wasting a few hours. But it wasn't a waste, it was an enjoyable lesson of learning together about our codebase through refactoring.

We continued mob programming as a way of learning, having a session every two weeks. Mob programming became our gateway to pairing and improved collaboration in general. What I thought was a waste of time, turned out to be a time saver. What I thought was something I would never enjoy, turned out to be something that reminded me that I have been a programmer since age of 13.

After getting started, I have used mob programming as a way of consulting and teaching, having now perspectives to the difference people say they know and what they can do being put on the work. Mob programming - and mob testing - have become invaluable for growing the next generations of professionals as well as keeping the current generations continuously learning.

I still face resistance to mob programming from senior developers who get the job done alone. They learned their way, reading books and articles and writing and delivering code. They don't find their motivation for mob programming in learning, but enabling learning of others.

Enjoy mob programming guidebook. Try mobbing out and make it your own. This book is for you to get started but the journey with mob programming continues further. We are all still discovering ways to turn up the good.

CHAPTER 1

WHAT IS MOB PROGRAMMING?

“”All the brilliant people working on the same thing, at the same time, in the same place, and on the same computer.”

~ Woody Zuill

Mob programming is a software development approach where the whole group of people works together using a shared computer, with focus on real-time contribution from everyone to get the best out of them into the work they are doing.

Image: Mob Programming at Tech Excellence Meetup in Finland. Author of the book is second from left sitting down.

The work done can be targeted at writing code - all the activities around programming. The work done can be targeted at finding information - all the activities around testing. The work done can be targeted at creating written instructions - all the activities around documenting.

Since the activity with mob programming does not have to be programming, you will find we often use the term mobbing as a shorthand.

When the team works together using one computer, this guides the team to work on a single work item at a time. With increased communication and learning, quality of end result improves on the spot and many of the problems around distributing work to different people can be resolved on the spot. There's less back and forth and a shared drive to deliver consistently, and a group will improve whatever is slowing them down.

The Basic Dynamics

For the group to work together, we use basic roles and rules.

ROLES

We start off with roles for the Driver/Navigators pattern.

- **Driver** is the person at the keyboard. No decisions allowed.

- **Navigators** are the people off the keyboard. They are the ones doing the work by using their words to express intent the driver turns into action.

- **Designated Navigator** is one navigator channeling the group to the driver. This role is often necessary when first starting to practice mob programming and dissolves later.

RULES

Rules reflect basic expected behaviors.

- **Rotate on timer**. Whoever is on the keyboard moves away and makes space for the next one. Timer is in scale of minutes.

- **Yes, and.** Whatever we were doing when we rotated, we continue and improve. We don't erase previous work.

- **No decisions on the keyboard**. This is not a group watching one work. Whatever happens at the keyboard is initiated from navigators.

- **Highest level of abstraction**. Navigate on highest level of abstraction driver can consume. Tell your intent as a navigator. If more is needed, you see it from the lack of movement at the keyboard.

- **Bias to action**. Favor doing something and being ready to throw it away over discussing in length between options.

- **Kindness, consideration and respect**. We're working closely together and we are all valuable contributors. Make space for everyone to shine.

Why Would You Have 5-8 People On One Thing?

Many people look at mob programming and wonder on why would we use so many people to do the work one or pair could do. Their concern is that this way of working would be inefficient. However, that is not the experience of those who practice mob programming.

You need to think of software work in a completely different way. Typing the code is not programming. Creating a solution that stands the test of time in production is. Quality of what we produce matters - in the immediate solution, the supporting structures created to make it available for users, and the people's abilities who produce more software later.

Instead of thinking about how we can get the most out of our team, we ask how we can get the best out of our team. We've used swarming approaches to particularly tricky problems before, yet mob programming was discovered as a whole team way of working consistently by Woody Zuill and his team at Hunter Industries. They noticed how working on particularly tough problems benefited from everyone working together, and started paying attention to how they could do more of the things that were working for them.

Many teams bring a group of people together on tough problems. Most teams after coming together to solve a problem say "Problem solved, let's go back to normal". Mob programming asks "We had very high performance together, how can we do more of that?".

When working on something complex, a mob figures out a solution. When working on something simple, a mob innovates and automates simple

things in their workflow. Where an individual tolerates wasteful practices, a mob amplifies and addresses those.

Immediate Feedback

When we work alone, the best and the worst of us ends up in the code or the work artifact we are creating. When someone joins in later and reviews, they help correct some of the stuff, making priority calls to not address things that would have needed addressing.

Image. Working Alone.

When we work as a group, the best of each one of us end up in the code or the work artifact we are creating. We can correct mistakes when they are about to happen, without egos in play. Often our best is better in a group than our best alone as group generates ideas that would not emerge working alone.

Image. Working as a Mob.

Working in a mob can be particularly rewarding for team members whose programming skill is less evolved. They still have great ideas even if they have little ability to turn it into production code alone and we would like the best ideas to end up into the software we are building.

Learning or Contributing

For mob programming, we say the right size of a group is one where everyone is still either learning or contributing. Contributing raises the quality of the code or the work artifact we are creating. Learning raises the quality of the people contributing the code or the work artifacts. People, having learned in a mob, are better off working on similar activities solo.

Two people are a pair. Three and above are a mob. Moving from a pair to a group changes the interpersonal relationships, and we find that people who misbehave in a pair are on better behavior in a group. More people

maximize the chance of serendipitous learning from the other members in the group.

Mob programming is a software development method - all code is produced in mob. When the team works, they work together. Mob programming goes beyond a time-boxed practice session. It's what mob programming teams do, all day, every day.

Other teams use mob programming within a time-box. They work solo or in pairs except for mob programming sessions introduced to share knowledge amongst team members.

Consultants and trainers have found mob programming to be an effective way to teach groups of people habits and hands-on skills. Facilitating a mob allows the facilitator to impact the work as it is done instead of discussing an experience of doing something in the abstract.

Benefits of High Communication

When we are mob programming, we are all working on the same thing together. If we need help - whether we know to ask for it or not - it is available within the group immediately within the work we are doing.

When we work, we often make mistakes. Maybe we misunderstand a detail of what is required. Maybe we forgot to think about an aspect that would be relevant. If I make a mistake working solo and don't notice it myself, I will compound the mistake and build more on top of it until it gets corrected. And if I have used significant time building on top of that mistake, correcting it makes me defensive.

The just-in-time knowledge mob programming offers changes the dynamic. The mistakes people would correct get corrected in the moment, before the mistake reaches the code or the work artifact. Learning is turned up by the fact that you can learn things you know you don't know,

but mob programming guides you to learning things you did not yet know you could be learning.

Cost of rework we can track. Cost of delays of knowledge is mainly hidden. If the hours we waste because of gaps in knowledge or understanding were accounted for, there would be less resistance to mob programming.

CHAPTER 2

HOW TO USE THIS BOOK

This is a recipe book for mob programming. In advanced sense, there is no recipe for mob programming just like there is no recipe for agile. You need to create your own mix, appropriate for your team. Yet basic recipes on getting started and growing your mob will help you steps through with our experiences.

There are two main recipes in this book:

- Setting up and facilitating a new mob
- Growing your mob

These recipes have been instilled from both good and bad experiences mobbing with various groups. The first recipe addresses the issue of basic knowledge to avoid the most common pitfalls. The second recipe collects together ideas of what more you could do to enable a highly functional and collaborative mob.

Our advice for **beginner facilitators** is that you would follow steps we outline. Some may appear counterintuitive, or wrong. As soon as you build experiences as a mob facilitator, you can build your own style on top of the basic recipe. Share your beginning experiences and start connecting with the mob programming community.

Intermediate facilitators start to have a sense of the different parts and are able to move to their own, unique direction. Think of our recipes as foundation and variation. We may not recognize all the variation.

Discuss your variations in the mob programming community and help build the approach forward.

Advanced facilitators don’t follow our our recipes. They pay attention to what is working, and do more of those things. Their focus is on the subtle characteristics unique to their teams and lessons from the retrospectives. At this point the book has served its purpose, and may remind you of some foundational elements. You’re ready to create your own way of working together and share new variations to recipes with the mob programming community.

RECIPE 1:

A NEW MOB

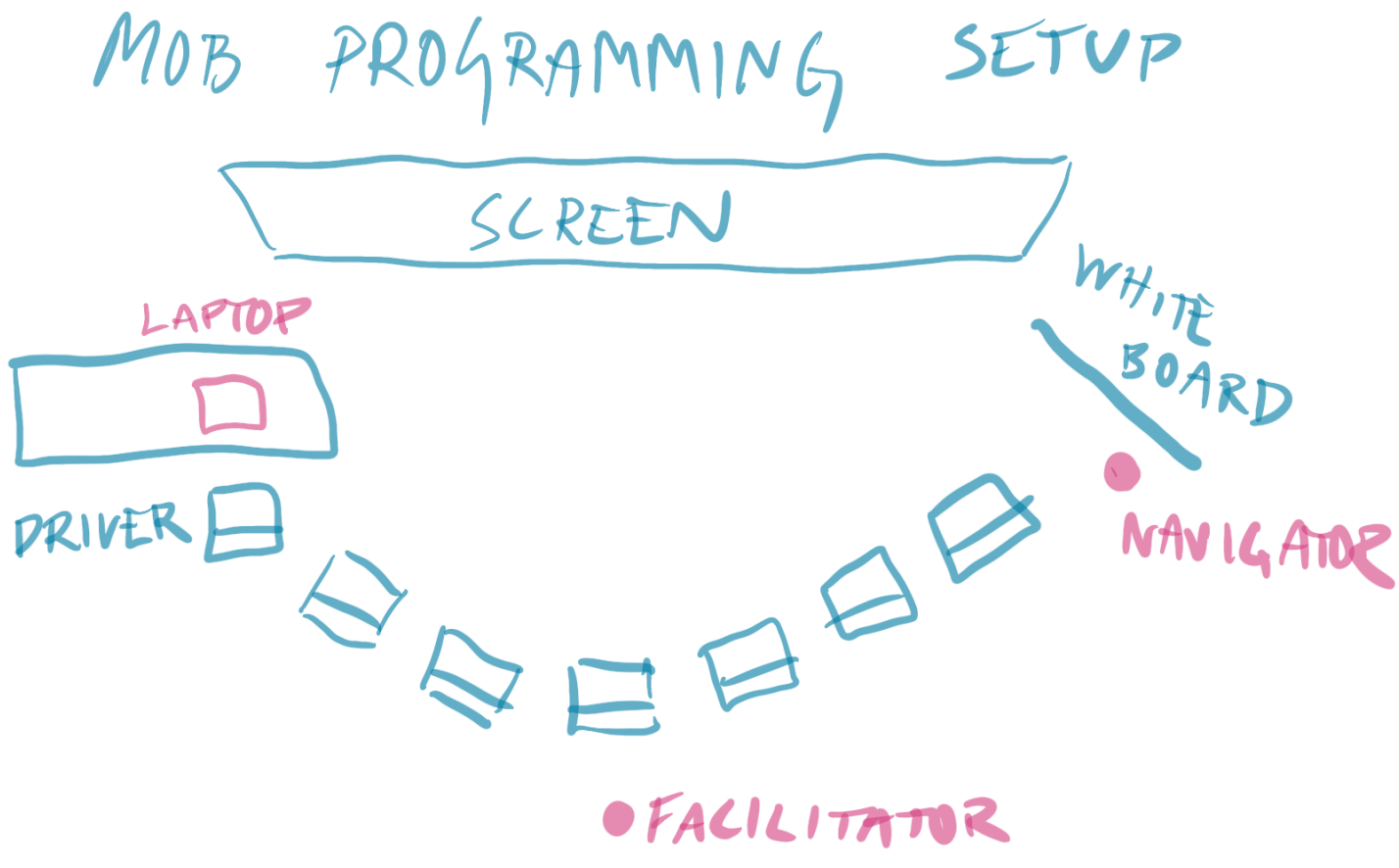
CHAPTER 3

SETTING UP THE SPACE

For mob programming, you need a space that allows for your group to work together on a shared screen and keyboard. For your first time mobbing, don't worry too much about the space. This chapter outlines the basic things you should pay attention to.

Note: If your mob will have more than ten people, check out chapter Mobbing with an audience.

Basic Setup



Your basic setup is a meeting room that allows for people to move for rotation and see the screen. You need something to work on, a computer to work with and basic roles and rules.

The Screen

The screen, projector or TV should be visible to everyone in the mob. It should be placed so that the person on the keyboard and separate screen also naturally sees the shared screen.

The chairs should be facing forward towards the screen as much as possible. Rooms with furniture you can move freely are better.

People will need to stand up and move around frequently, so there should be room to do that comfortably. Having your bags, notebooks and laptops with you isn't helpful during this. We advice asking people to place their stuff in the corner of the room before starting to keep rotation flowing. If people are uncomfortable with this, it is not a big deal to hold on to their possessions.

A whiteboard where the navigators can express ideas is also important. Place it on the other side of the screen than the keyboard. It should be so that everyone can see it, but the person standing next to it is not standing next to the driver. This is important for speaking volume in the mob so that everyone can follow.

The Facilitator

As you are reading this book, you are a likely facilitator holding space for mob programming to happen. It is good to note that while facilitator is not a default role in a mob, getting started works better id there is a facilitator.

As facilitator, your job is to ensure that all the steps of mob programming are carried out appropriately. This may mean you will not be a part of the

group yourself, but follow the dynamics of the group from the back of the room.

As a facilitator you can pause the mob to introduce ideas of how to work in the mob. If you are a facilitator using mob programming as a teaching method, you may want to pause the mob to introduce new ideas by temporarily stepping into the navigator role. Our advice is to state this clearly when you do.

You do not need to be a good programmer or a programmer at all to be a good facilitator. Facilitator holds space for the group bringing their knowledge to the work they do. When everything is going well, you will be doing nothing at all.

The Work

There needs to be something to work on together. For your first mob, keep the task clear and simple. Learning to work together takes your focus. Tackle hard tasks as a mob after you first learn to work together.

There are a few typical work tasks to begin with:

Simple work task

If your team has a simple work task to do, that is a good candidate for mob programming. Be mindful about the task really being simple. If your group isn't good at expressing intent in the form of tests and examples, a normal work task turns hard in a group when the vision is held inside one person's head.

Refactoring large methods

Many teams have code that is hard to read and understand. A refactoring for improved readability is a great task for first mob programming session.

Choose a method that is troublesome that you would need to work on sometime soon anyway.

We suggest you limit the refactoring in this session to simple extraction of paragraphs to methods and giving them names. If you only use *rename* and *extract method* -refactoring, there is still plenty to do but the team gets to focus on learning to work together.

Also, we suggest you frequently commit the changes. It often reveals interesting dynamics around refactoring. Usually you can commit after each extracted paragraph.

Programming Katas

Katas are simple exercises used to practice programming. The common ones include FizzBuzz and Roman Numerals (creating code, test first) and GildedRose (cleaning up existing code).

Creating test automation

The code you work on can be test code. Adding tests to existing test automation or cleaning it up with refactoring are good options too.

Exploratory testing

Many times the best first task is a task of testing without code. If you use mob testing mechanism, see what problems your group is able to find in your own software or any software you would work on as a practice.

The Computer

While you get away with almost anything in your first mob, there are a few tips to pay attention to.

Keyboard and mouse

When a group works together, having an external keyboard and a mouse helps. It just makes things simpler for the driver. External keyboard and mouse can also allow you to close the laptop so that everybody is always looking at the same screen. This way the driver can more easily follow the cues of people pointing to the screen.

Make sure the language of keyboard is what the majority of the group is used to. Making UK groups write code on a Finnish keyboard takes the attention away from the work to the tools.

Screen

You may not have much control over the screen in the room, but when possible, favor rooms with high resolution screens and projectors. Code has a lot of details bad projector quality can make hard to follow.

Simple Editor with Line Numbers

Line numbers makes it easier to talk about where your focus should be at. Editors that both allow for simple typing but also support the language in question should be ones a mob favors. Use common settings over specialized individual configurations. Make sure the font size is big enough for people to see.

Avoid first mob programming experience with editors like vi and Emacs that add to the cognitive load if not configured the way each individual member would normally use them.

Hygiene

While hygiene or ergonomics are not the main focus of your first mob, you may add to participants comfort by paying attention to hygiene. We suggest cleaning up the shared keyboard and making hand sanitizer available for use.

If you find an external facilitator for your mobbing experience and use their computer, that computer is often part of their traveling life and they may not pay attention to it themselves.

Seating, Roles and Rotation

Driver is the typist - intelligent input device. There should be *no decisions* by the driver. For anything to happen, there should be speaking out loud involved, initiating with other members of the mob. While driver can speak back to the navigators, it is important that they actively listen and trust the group enough to do what they are asked to do.

Navigators are the people programming without touching the keyboard. **Designated Navigator** stands next to the whiteboard where we note the agreed task we are on. While they take insight from the mob, they need to make final decisions on what to do out of the options provided. They should be talking about the task in the highest level of abstraction possible. Usually in the beginning the highest possible level drills down to keystrokes and simple programming structures.

Navigators in the mob are expected to contribute insights when appropriate in support of the designated navigator.

People should not be too comfortable in their seats and roles. We suggest a new mob rotates on **4 minute timer**. In the first mobs, the facilitators phone is the least disruptive timer. The timer forces people in the mob to pay attention.

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.

Facilitator stands in the back and does not rotate with the rest of the mob. If they need to step in, they will pause the mob and assume whatever role needed except the driver.

We will talk more about roles and their expectations in the chapter introducing Strong-Style Navigation and the chapters outlining things to do as Driver and Navigator.

Introducing the Group To Mobbing

Many new mobs start with introducing the group to mob programming, roles and rules. Our experience shows that before the first experience, this discussion is premature and theoretical. We advice to leave time for talking about the experience after the experience.

We suggest you introduce mobbing as *a working and learning together* in a meeting room. As facilitator, you can introduce the roles and rules as you are already working on the tasks selected.

Talk After - Retrospective

After you have had your experience doing this, collect people's observations. Make sure to leave enough time to talk about what people learned in the mob, and how you could improve their experience for the next time.

These instructions should be enough for you to start with your first mob programming experience.

CHAPTER 4

WORKING IN YOUR FIRST MOB

In previous chapter, we talked about setting up the space for mob programming to happen so that you would be ready for trying it out for yourself. This chapter looks into what that experience could be like. The purpose of this chapter is to give you an idea of what it would be like.

Image. A Large group Mob Testing at a training event in UK.

If everything is going right in your mob, rotation will not disrupt the flow of the mob. There's many tricks to pay attention to for keeping the flow in place.

Mobbing in General

Rotate on Timer

In the very first mob, our advice is to not use a Mob Timer but rely on facilitators phone for timed alerts on need to rotate.

There is also a selection of Mob Timer applications available. Their general idea is to alert on who should be on the keyboard and when it is time to switch. Some group find early fondness for Mob Timers and can't imagine mob programming without them.

When your mob programming flows, it does not matter what gets you to switch places. At first you start of with a four minute timer, and if your group finds four minutes too short, our advice is to try a two minute timer to enforce flow. Longer rotations up to 15 minutes should be used only when a mob is established and used to working together.

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. You can do more, but not skip what was going on when you rotated. This allows continuously making progress and keeps people engaged in the group.

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

It also helps calling out what the task was that the mob is working on. When you're adding to that, are you changing the task completely?

No Decisions on Keyboard

Highest Level of Abstraction

Bias to Action

Mob Programming Specifics

Mob Testing Specifics

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 on it from the mob:

> *We will treat everyone with kindness, consideration and respect.*

This rule comes from the Hunter mob and was coined as a result of realizing it helps people find the way of working over a longer period of time and acts as a heuristic of what is expected behavior. Our aim is to get

everyone in the mob to treat each other better and learn how to work together.

Kindness

While kindness appears self-explanatory, it isn't. Being kind is not the same as being nice. Being kind requires radical candor - openly sharing perspectives even if they are in conflict with what others would expect to hear from you. However, kindness entails constructive feedback.

Consideration

Consideration is really about listening. Listening is more than hearing the words, but understanding what the words are trying to convey.

The place where listening shows up clearly is on the driver seat. Often the driver starts off by not listening to the navigator, focused on their own idea of the work.

The facilitator can usually fix this by simply stating "No thinking at the keyboard" or "There's too much thinking at the keyboard". Call it out when you see it, and the group adapts.

Another way that listening will show up is in acknowledging the good ideas spoken by members of the mob. Many times the good ideas will be totally ignored. The ideas that are ignored may come from people who are soft spoken, or who find it hard to time their ideas to the moment the group is listening. As a facilitator, you can amplify those ideas by calling attention to them and making sure that everyone gets heard and finds space to express their ideas.

Over time the mob will learn how each individual member best contributes, and learns to listen and make space. The more senior members show consideration by making space for the less senior members

in active navigation not contributing everything they'd have as others can also bring that in. As we gain trust through experience that we are being heard, it also changes how and what we speak on. We let others shine without needing to show how much we know.

Consideration shows up also as trying other people's ideas even when they appear wrong. In a mob, a good facilitation trick is to make a list of different ways of solving the problem, and choosing one that seems less likely to be right. With many things we implement, as soon as it is done one way we no longer care about the other ways. Sometimes each way reveals more about the way it needs to be implemented. It is ok to try a solution multiple ways.

Finally, one way of not being considerate is for people in the mob to tune out and start checking their computers or phones. It should be ok to step out of the mob any time you need a break. You can try handling this with environmental settings by having chairs without tables to place laptops/ phones on, but this also prevents the group from working out just the right piece of information to navigate that we want to do with more established mobs. Frequent rotation also keeps people engaged better than trying to discipline the group for not paying attention.

Respect

Respect starts with assuming that people who created the code we are working on now did the best work they could under circumstances and knowledge they had at time of writing it.

Showing disrespect to anyone, even through criticizing their output, is corrosive. It may feel like your group is bonding over criticizing a piece of code written by someone who isn't in the room. However, it sets a precedent that your code one day may be treated with similar disrespect.

We want to create a safe space. We want it to be safe to experiment. Safe to learn. Safe to show your weaknesses and vulnerabilities. Safe to improve code without criticizing its author.

Mob programming exposes a lot about each individual in the mob. We need to feel safe to bring our full selves into the work. We need to feel safe and supported.

CHAPTER N

STRONG-STYLE NAVIGATION

CHAPTER 5

BEHAVIORS IN A MOB

When figuring out how to work better in a mob, here are some behaviors you could try practicing. This is inspired by Willem Larsen's Mob Programming the Role Playing Game and is best practiced by playing the game.

Driver

- Ask clarifying questions about what to type
- Ask to be navigated on a different level of abstraction
- Type something you disagree without
- Use a new keyboard shortcut
- Learn something about tooling
- Ignore a direct instruction from someone who isn't the Designated Navigator
- Amplify the unheard voice
- Support contributions of a mobber with least privilege
- Celebrate moments of excellence
- Point out a long line of code
- Point out unnecessary complexity
- Point out duplication
- Point out a misnamed variable or methods
- Propose an action to improve the code
- Point out a problem the mob is not seeing

Drivers take on three roles: from being a Driver (intelligent input device) to Sponsor (supporting others from a unique position) to Nose (noticing things about the code).

Designated Navigator

- Ask for ideas
- Filter the mob's ideas to tell the Driver exactly what to type
- Tell the driver only your high-level intent and have them implement the details while you review them
- Create a failing test, make it pass, refactor
- Find and share relevant information from documentation
- Find and share relevant information from a blog
- Find and share relevant information from a coding forum
- Point out a repeated task in a tool
- Point out a repeated aspect of the team process
- Point out possible unnecessary code
- Propose an automation for a repeated tasks
- Propose automating a repeated task
- Take more than one idea from the other people in the mob and do both
- Get and implement an idea from someone in the mob who has been quiet
- Make a bad suggestion and ask people for options
- Allow the choice of next action for the least privileged voice

Navigators take on four roles: from being a Navigator (translating ideas into code) to Researcher (having better information available) to

Automationist (recognizing repetition) to Conductor (enhancing others contributions).

Other Navigators

- Make room for less privileged voice to be heard
- Contribute an idea
- Ask questions until you understand
- Listen actively ready to pitch in when needed
- Quietly speak into the navigators ear
- Give the smallest cue necessary to move navigator forward through the problem
- Navigate the navigator on highest level of abstraction they can successfully take through
- Record solution alternatives on a whiteboard so they are not forgotten
- Step on whiteboard to express an idea as it is taking shape
- Articulate insights of current task at hand to make them visible for the mob

Other navigators can step in as Designated navigators any time, and this list includes some ideas of what they might try while not actively navigating the driver. Other navigators take on three roles: from Mobber (always contributing in different ways) to Rear Admiral (helping designated navigator do better and learn) to Archivist (improving team visibility).

CHAPTER N

CLOSING A MOBBING SESSION

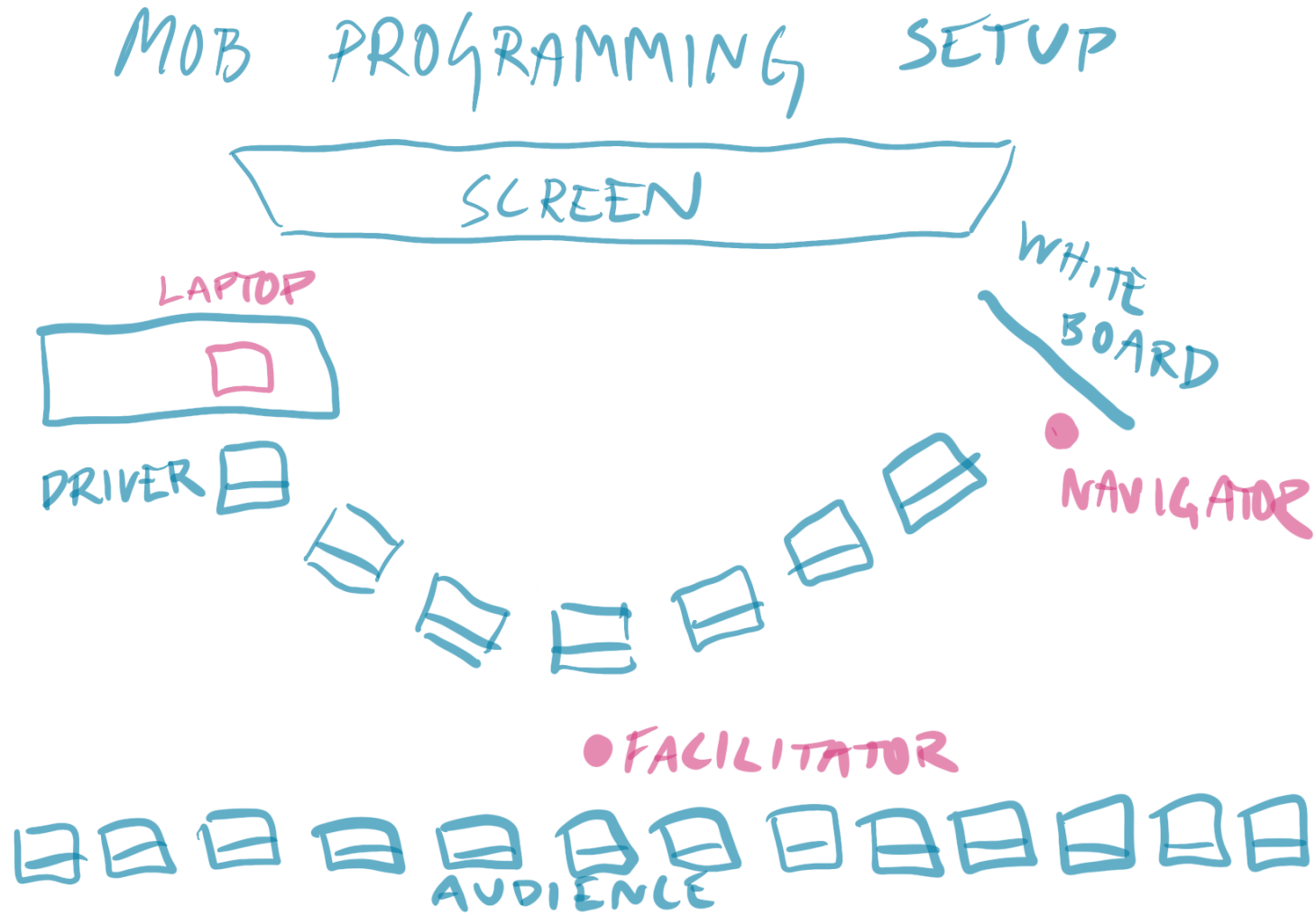
CHAPTER N

WHEN NAVIGATORS GET STUCK

CHAPTER N

MOBBING WITH AN AUDIENCE

As a conference speaker and a trainer, a lot of my mob programming experience comes from facilitating mob programming sessions with various groups. For a training, we usually set up the whole group into a mob where everyone rotates. For conference sessions where time constraints limit participant numbers for effective mobbing, we use mobbing with an audience.



hhhh

CHAPTER N

REMOTE MOB PROGRAMMING

CHAPTER N

MOBBING CHEAT SHEET

RECIPE 11: GROWING YOUR MOB

CHAPTER N

DRIVER THINGS TO DO

CHAPTER N

NAVIGATOR THINGS TO DO

CHAPTER N

BUILDING HABITS

CHAPTER N

FULL TEAM ENGAGEMENT

CHAPTER N

DIFFERENT WAYS TO CONTRIBUTE

CHAPTER N

ATTRIBUTION FOR WORK DONE IN A MOB

When a group of people works together, it becomes difficult to identify what was the contribution of each of the members. Like with baking a cake, you cannot identify if it is the flour, the sugar, the eggs or the vanilla sugar that makes the recipe just right and perfect. People inspire one another, build on one another and create together things they would not create alone.

We hope these ideas help you work out how to figure out giving credit where credit is due, without overemphasizing a mob visitors' contributions.

Best ideas win when you care about work over credit

We often advice people who work in mobs to think of the work in terms of collaborative crediting. When you care about the work over credit, that is how you get the best ideas to win.

Many times you get credit for an idea. We think of this as a wrong way of looking at it. Instead of ideas, implementing those ideas in creative ways is where value is created. Attributing a piece of software to the architect over the developer implementing the reality of the vision would not be the right approach.

In a mob, we seek a situation where everyone is either learning or contributing. Learning is something that stays with you when you leave the mob. Contribution is something you share with people you mobbed with, that is a product of its time where fair attribution and remembering it realistically is important.

This book as a case example

This book is a spin-off from a collaborative writing effort. Writing in strong-style, both authors co-create every single line of the text. Early versions of the book were created so that there was not one author but two authors for every line. For most of it, I was on the keyboard as it is my primary mode of thinking. I could say I wrote the text. But it was a result of a collaboration - strong-style pairing. The text created belonged to both of us.

Before the book was finished, we found reasons to not want to continue our collaboration. The written text did not reflect my vision of the book, so I took it forward. The written text by that time was something I did not want my name associated with in the long term.

There is a contribution from the collaborators in sharing ideas, a little deeper than one you would have with a great reviewer. Not caring for credit would mean that either one can take the book forward as the original idea was.

However, it turns out that the second collaborator does not feel the same way. Their suggestion is to freeze the unfinished book and claim it is done - something I cannot agree with. It is like suggesting that the code we created while mob programming isn't allowed to change further with other people joining the mob.

The second collaborator feels strongly enough that this book is no longer available on LeanPub. The LeanPub edition of the book has been retired and no new Mob Programming Guidebook on LeanPub can be created.

LeanPub model in disputes of copyright agreements leads to retiring books where contributors appear to be in conflict. The old version is available to people who purchased and downloaded it before it was retired, but it is not legal to redistribute the copy.

This book version is founded on the lessons learned on the original writing effort but does not reuse any of the text. It is a new book into the same container of a vision to have a book that helps you get started with mob programming. For this book, the learning that happened with an earlier writing effort provides a foundation, but this book is all written by a single author. The second collaborator was valuable at their time, but they can neither stop the text from going forward within the container, nor ask that everything others are doing later on are attributed to them.

Instead, like with open source, the second collaborator has a fork. They cannot publish the fork with my name but take it forward as the version they feel that holds their vision. They were paid from the little income the book they were part of generated more than I was, and there is no unfairness.

Attribution for work done in a mob is to recognize and appreciate the impact, without minimizing the work of the others coming after.

People crave recognition

No matter what we do, we crave for recognition for our contributions. Mobbing makes us worried that our personal impact cannot be identified. Instead, we need to collaboratively credit a group creating something together that none would have created alone.

We seek recognition to a point where we make collaboration and realistic perception of reality difficult. A famous example is identifying 12 types of dinosaurs based on fossils. Later, a researcher asked a relevant question: "Why are there no baby dinosaurs?" only to discover that 5/12 identified types were actually younger versions of the same types. Finding a new

allowed you to name it, and credit yourself for it. Finding the same was adding to the previous researchers pool.

At work, we remember credit through names in our version control systems, that only recently have started to support collaborative crediting.

CHAPTER N

BECOMING A GREAT FACILITATOR