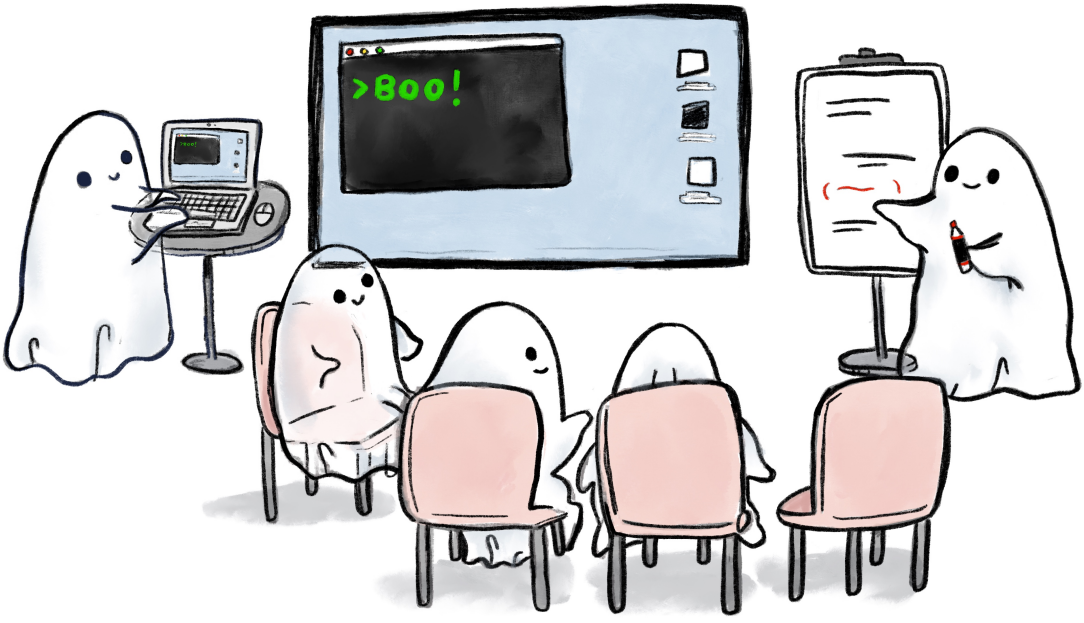
ENSEMBLE PROGRAMMING GUIDEBOOK

MAARET PYHÄJÄRVI



ENSEMBLE PROGRAMMING GUIDEBOOK

This book is available for download at

https://ensembleprogrammingguidebook.xyz

It is work in progress to be published at a later date.

This version was published at July 29th, 2020.

THANK YOU

To the person reading this book in progress,

Thank you. For considering ensemble 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 Ensemble 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 Ensemble 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. You can tweet about the book. I use the hashtag #EnsembleProgrammingGuidebook on Twitter.

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*. Recognizing the feeling leading me to new insights, I had to try it.

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.

Later, with years of experiences working in a group like this, I tuned in to hear a message that was around all along: mob has unnecessary negative connotations for something as inclusive as the method community had joined in discovering. Following a suggestion from Denise Yu, as learning all the words groups could be referred to, I settled into me calling this practice Ensemble Programming.

At my work after inception, we continued ensemble programming as a way of learning, having a session every two weeks. Ensemble 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 ensemble 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. Ensemble programming - and ensemble testing - have become invaluable for me for growing the next generations of professionals as well as keeping the current generations continuously learning.

I still face resistance to ensemble 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 ensemble programming in personal learning, but enabling learning of others.

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

CHAPTER 1

WHAT IS ENSEMBLE 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

Ensemble 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 shared work they are doing.

Image: Ensemble 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*. In this book, we consider programming as a shorthand for all activities in development needed to build value in software at hands of customers.

Since the activity with ensemble programming does not have to be programming, you will find we often use the term ensembling, or may talk specifically about ensembling for testing as Mob testing.

When the team works together using one computer, this guides them 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 or vanish. There's less back and forth and a shared drive to deliver consistently, and a group will improve whatever is slowing them down.

Before we talk more about benefits, let us look into the basic dynamics of what ensembling looks like.

The Basic Dynamics

What separates mob programming from any group activity are the basic dynamics of *roles* and *rules*. A mob is a group of more than 3 people, usually 5-8. The way the group is set up is not that one is working and others are watching. Instead, the actions on the keyboard flow through one person and everyone in the mob contributes.

Image: Roles for Mob Programming Starting Setup

ROLES

A new mob starts 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.
- **Learning or contributing**. Everyone should be either learning or contributing, perhaps both.
- **Kindness, consideration and respect**. We're working closely together and we are all valuable contributors. Make space for everyone to learn and contribute.

What Does It Look Like?

The group sits down on chairs facing a big screen with one computer connected to the screen. They discuss what they are about to do and start doing the work.

The designated navigator voices what should happen on the keyboard, and driver follows that guidance transforming the communicated intent and details into details of code.

Every four minutes an alarm goes off encouraging all of them to switch places. If four minutes feels fast, they try two minutes to first learn to work together as a group. If they

are an established mob, the timer goes off every fifteen minutes. When switching, designated navigator becomes the driver and the driver returns to the mob.

It looks like a game of musical chairs, without music, with focus on getting work done as a group where everyone is either learning or contributing.

Group of 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 a 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 different way. Typing the code is not programming. Programming is about learning to translate business needs into working software solution that stands the test of time in production.

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.

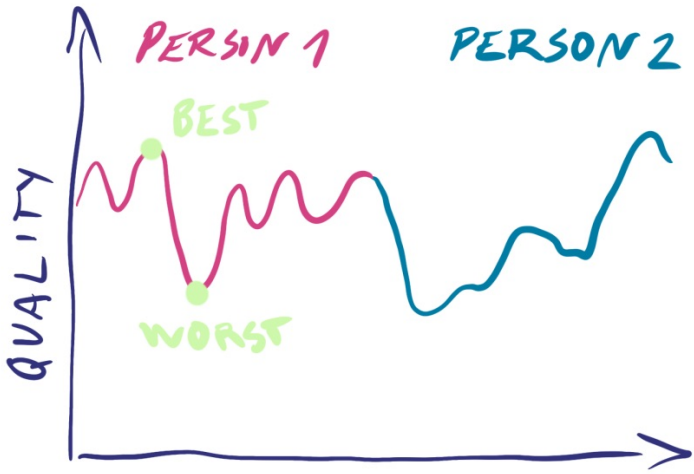


Image. Working Alone.

When we work alone, the delayed feedback allows us to create structures that are hard to undo, and create unwillingness to change when we realize they were not optimal.

When we work as a mob, 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 individually, 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, hands-on skills, tools and techniques. 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 based on people's perceptions.

In company chat, a team member shares an image with some overlaying text on a screenshot. Someone exclaims: "Windows Snipping Tool!" as they had just few days earlier learned of its existence as someone noticed them struggling with taking screenshots. The screenshot looked like it was handcrafted. "Greenshot!" exclaimed another as sharing of tools started. Before finding the tools evident to others, everyone had done screenshots the hard way and had no idea there was different individual ways to learn from.
Years of missing out on ideas and tools are typical when teams are not mob programming.

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 we make a mistake working solo and don't notice it ourself, we will compound the mistake and build more on top of it until it

gets corrected. And if we have used significant time building on top of that mistake, correcting it makes us 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

FACILITATORS GUIDE TO MOB PROGRAMMING

From first chapter, you have an idea of what mob programming might look like and why ensembling can make sense. This chapter gives guidance to people who take upon themselves to facilitate mobs.

Facilitator is the person who ensures we have something you want to do together, the physical space, the right people, and the necessary rules to have a good experience mob programming.

Image. Large Group Mob Testing at Training in UK

Understanding the Facilitator's Work

Not all mobs, even beginner mobs, have a facilitator. Facilitator is usually needed when:

- The mob participants can use help in recognizing patterns of interaction to grow to work better together
- The facilitator has a teaching agenda in relation to the work the mob is doing
- Coming together to work benefits from support of one individual focusing on improving the experience of everyone

Introducing mob programming for your team, you might want to step to side to facilitate, or introduce rules of how to work and then participate fully in the mob.

As mob programming enthusiast, you may want to create space for mob programming to continue or move from being occasional experience to a way of working all day. Getting to that place requires facilitation and advocacy.

Coming into an organization as a consultant or a trainer, you would want to use mob programming as a tool to see what people do (as opposed to what they say they do) and move them to better place through doing. Role of a

facilitator is the natural role for a trainer transferring habits, skills, knowledge and tools to a group of people where teaching can happen through people in the mob discovering the best they can do as a group, or through facilitator taking a navigator role when teaching with an example is necessary.

Facilitating a mob does not require expertise in the work the mob is doing. As long as the mob has someone who knows how to do the work, the facilitator can focus on everything but the content of the work.

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 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 this 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.

An Overview to the Work

To have your first mob programming session successfully run, you will need the following:

- **Problem to Work On**. You need to choose work to complete that is simple yet interesting. First time mob programmers are primarily learning to work together and you want the problem to illustrate they need the group while being simple enough to leave space for learning about collaboration.

- **Time and People**. You need a common time, usually 60-120 minutes slotted on every participant's schedule for your first experience. While you can mob remotely over the internet, separating people with time breaks mob programming. You will not want to force people to participate.

- **Physical Space**. Setting up a physical space for mob programming has its own considerations. There are considerations from visibility to the work happening on the screen, to ensuring movement in the room and using physical space to ensure everyone hears one another.

- **Rules and Working Agreement**. You don't need many rules, but the ones you need are important. Roles, rotation and working together are important.

- **Recognizing Problems**. First time mobs can turn chaotic, end up hijacked by individuals setting the mood and benefit from some facilitation. Knowing what to watch for is good.

- **Retrospective**. No experience is complete without taking time to discuss it with the whole group. Make sure you leave time for this and rewrite people's individual experiences through the eyes of the others.

Problem to work on

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.

Time and People

Finding two hours from people's schedules is sometimes hard, and our recommendation is to pay attention to setting up the expectations:

- This is optional not compulsory. We find that strongly encouraging people to join who do not want to creates a bad experience for everyone. Opting out in an option.
- When joining, stay for the whole duration including retrospective. The first times new groups work together, showing commitment through being there for whole duration of the session is important.

Many mixes of people can work. A natural unit to work on is a team that already works together.

The group size drives the time you should reserve. The time allocated should fit minimum of 3 full rotations of everyone in 4 minute timer. For a group of 8 people, we recommend 1,5 hours plus half an hour for retrospective.

Physical Space

Your basic setup is a meeting room that allows for people to move for rotation and see the screen. Meeting rooms allowing for movement are not always obvious, so try to ensure the table is not fastened in a way that encourages fixed seating. When we rotate, everyone should be able to get up and move one notch in the rotation. If there is a physical obstacle, we recommend finding a more open room.

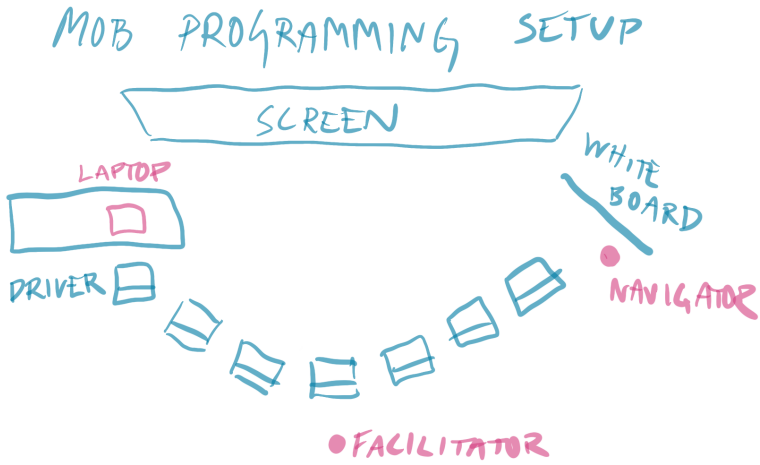


Image. Typical Mob Programming Setup

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.

Rules and Working Agreement

INTRODUCING MOB PROGRAMMING

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 ensembling 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.

ROLES

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.

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.

ROTATE ON TIMER

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.

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?

BIAS TO ACTION

NO DECISIONS ON THE KEYBOARD

NAVIGATE ON HIGHEST LEVEL OF ABSTRACTION

KINDNESS, CONSIDERATION AND RESPECT

An important rule for mob programming is one of *kindness, consideration and respect*. As a rule, it talks about how we

work with each other. It is also one of the the more difficult rules and requires a little elaboration.

We will treat everyone with kindness, consideration and respect.

— Hunter Mob

This rule helps people discover a good way of working over a long period of time, serving as a guiding principle for which one's own behaviors can be attached. Our aim is to be our authentic selves in the mob and still treat each other better and learn to work together well.

Kindness may appear self-explanatory, but is far from it. Keeping challenging, even negative perspectives to oneself can be *nice* but it isn't *kind*. Being kind requires *radical candor*, the idea that you both care personally and challenge directly. Being kind means you will share your perspectives even when they are in conflict with what others expect to hear from you. Kindness is about constructive feedback.

Consideration is about listening actively, hearing what others mean to say over what they are saying, understanding the meaning they are trying to convey. Listening is particularly important when on the driver seat, and to listen well, you must let go of your own idea of what

should happen and trust in the navigator. Listening to everyone in the mob and making space for them to contribute is necessary. As a facilitator, you may need to call our missed contributions and remind people on the rule of "no decisions at the keyboard" to enforce consideration. Consideration means making space for everyone in the mob to express themselves and bring forth ideas, and often shows up as more senior people yielding to give space for less seniors to grow through navigation. Consideration also shows through trying other people's ideas even when we think they are not right.

Respect starts with believing that the work that happened before ours and shows in the codebase we are dealing with did the best job they could under the circumstances we may not recognize. While criticizing a piece of code created by someone not in the room may feel like a bonding exercise, it is also a show of how we talk about code created by these people when they are not around. Disrespect is corrosive. We need to feel safe to experiment, safe to fail - remembering that FAIL is a First Attempt in Learning. We will not know everything, and come to the work in the mob with our weaknesses and vulnerabilities, and should be respected as our whole selves. Mob programming exposes a lot about each individual in the mob.

When we through experience gain trust on what we get heard on, it changes how and what we speak on. As Google's project Aristotle finds, the most productive teams have everyone contributing for existence of psychological safety - the ability to trust that no matter what you say, you will not be ridiculed.

- Speak honestly about the good and the bad, aim to turn up the good
- Avoid decisions at the keyboard while driving
- Acknowledge other people's good ideas, listen to notice them and when needed, amplify them assigning credit
- Seniors yield in navigation when they can
- With multiple ideas, try them all, the least likely first
- When in mob, focus on the work. Leave whenever you need a break.

Recognizing Problems

As groups of people come together for mob programming, many things can unfold. Spending time in the same room, working on the same thing surfaces existing conflicts, starts

off by rubbing some people the wrong way and requires significant effort to help overcome.

A good way of recognizing problems for a facilitator is to paint a picture of what good looks like when mob programming.

- Everyone contributes and learns.
- Everyone is treated with kindness, consideration and respect.
- Mob follows the basic pattern of decisions happening off the keyboard.

If the group is unable to work together on a problem but it becomes one working others watching, you will want to try reinforcing the basic rules:

- No decisions (thinking even) on the keyboard!
- Navigate on highest level of intent the driver can work with and still do what the navigator intended.
- Designated navigator channels the rest of the group, and rest of the group makes space for them to navigate.

We provide more suggestions on what might go wrong when discussing Special Mob Programming Setups, specifically ensembling in front of an audience.

Retrospective

At the end of your ensembling, sometimes also in the middle, make sure to do a retrospective to collect and discuss people's observations. Having everyone write their observations on post-it notes allows for thinking time for those who don't think of their feet.

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.

CHAPTER 3

CONTRIBUTING 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.

Contributions for a Driver

Here are some things for the **Driver** to do:

- 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 voice others did not hear from the Navigators group

- 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).

Contributions for a Navigator

Here are some things for the **Designated Navigator** to do:

- Ask for ideas from everyone
- 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).

Here are some things **Other Navigators** can do:

- 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).

Contributions Have Dimensions

A lot of our experiences in contributing in a mob programming come from the foundation of being a less privileged voice when it comes to programming, and a highly privileged voice when it comes to testing.

Our experiences as foundation guide us to appreciate contributions come in many forms:

- Being the slow one in a mob made others more clear and thoughtful in response. The somewhat random way of doing whatever came to someone's mind that everyone had hard time following turned into agreed steps.

- Knowing from the history why things may be the way they are proved useful on several occasions.

Remembering what was already there and conceptually similar turned code from technical debt to technical assets that we could build on.

- Suggesting changes in browser we test in or test data we use revealed problems at early stage of programming a user interface component.

- Knowing customer priorities and past functionalities helped us avoid mistakes that would require complete rewrites.

- The group looking at the tester in the mob remembering they are not satisfied with the level of testing for a feature without a word being spoken.

We learned that we had great ideas we had not been able to turn into code when working in separate workflows, that when brought to a mob, could translate into changes in the way we work that save a lot of time and effort.

We learned to appreciate that some contributions in a mob are about being *productive*: doing a task, contributing a relevant piece of information. Some contributions are about being *generative*: having an impact on how others do their tasks. Both contributions are valuable.

CHAPTER 4

LEARNING IN A MOB

CHAPTER 5

FROM TIME-BOXED TO CONTINUOUS

When we start with mob programming, we usually time-box it to take place in a meeting room for a limited amount of time. When done with the mob programming session, we ask when we should do it again.

If our mob programming sessions within the time-boxed mode are heavily geared to everyone learning with little impact on contributing more effectively, our experience is that people can continue mob programming for years in a time boxed mode. The sessions alone change how people relate to one another outside those sessions and can serve as a great productivity boost.

As the group of people sees the benefits of mob programming to the ability to deliver, some groups opt in beyond regular time-boxes, turning into mob programming continuously. There are teams in the world who have opted in to do all their production code in a mob, which means they usually come to work same time and leave same time,

mob programming with what ever combination of people is available at that time.

CHAPTER 6

SPECIAL MOB PROGRAMMING SETUPS

We've covered setting up first mob programming sessions under regular conditions. A lot of times we find that our organizational constraints may not allow us for the basic 5-8 people in the same room all working together on the same thing.

As a consultant and trainer, I show up in places where there are too many people in relation to the time available for everyone to be in a mob and still be learning or contributing. Layering people into two groups - the mob and the audience - is a helpful pattern to work around that.

With the purpose of teaching for 1-3 days, mob programming with a large mob has its own special characteristics.

Working in global organizations, not all mobs I would want to facilitate can take place locally. Some organizations are born distributed, and still can mob.

Ensembling with an Audience

Mob programming with an audience is a special setup that is useful tool especially to someone teaching mob programming, teaching any skills in software development in a hands-on style making new kinds of sessions available for conferences, or generally running demo sessions with partial session participant involvement.

As a conference speaker and a trainer, a lot of our 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 ensembling, we use ensembling with an audience.

THE SETUP

For ensembling with an audience, you split the room to two groups:

- **The Mob**. For the most effective mob made of complete strangers is small. You want to have a diverse set of mob programmers. These are the people doing the work.

- **The Audience**. The rest of the group sit in rows as audience. The role of the audience is to watch and make observations, and their participation is welcome when doing a retrospective.

For the mob, you will set up a basic mob setup in the front of the room with chairs for each person, whiteboard furthest away from the computer to ensure speaking volume for the designated navigator through the physical setup.

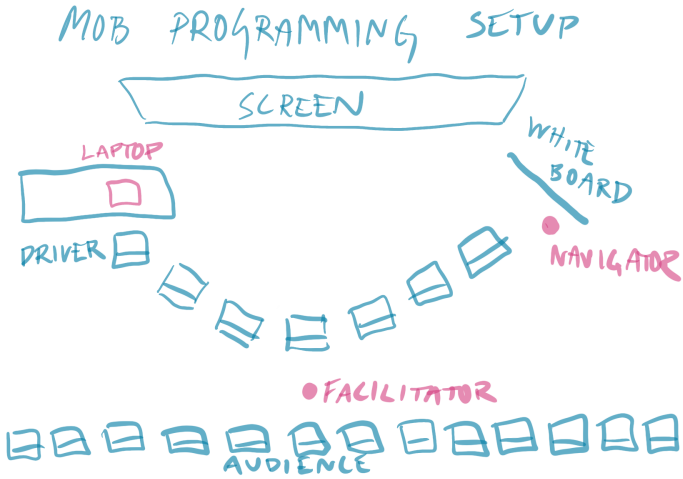


Image. Mob Programming with an Audience

For this setup, you will need a room with chairs that are freely moving. Make sure text on the screen is big enough not only for the mob to see, but the audience to follow as well.

TIPS FOR THE FACILITATOR

As we have run some hundreds of sessions with various groups in this format, we have had things go wrong in many ways.

Things you can do in advance to ensure less problems

- If the room is big, ask for a microphone for both the driver and designated navigator. It is essential that people in the room can hear their dialog. While there are no decisions allowed on the driver seat, speaking back to the navigators pointing out things you see and they don't is often necessary.

- If you have only one microphone, give that to the designated navigator. Even in smaller rooms, the microphone can work as a talking stick the designated navigator passes around for other navigators and can help create an atmosphere where everyone in the mob gets to contribute.

- Make sure the text on the screen is visible from the back row. Avoid dark theme, it does not serve you well for live coding and testing in front of an audience.

- When selecting the diverse mob, what you need to do for this depends on who you are. If you are a white man facilitator and want women, start with inviting women or

facilitate mob member selection in a way that gives you a diverse set of mob programmers. As a white woman, women volunteer for me in ways they don't for the men and I need to work and I need to work on other aspects of diversity.

- For a demo mob, you may want to demo a group with experience working on the problem and even together. If that is your aim, invite the people you want for the mob in advance.

- A new mob with different experiences highlights many powerful lessons around collaboration and people helping each other and your goal to set up a fluent demo is probably infrequent. The new programmers exclaiming "they now know how to do TDD" as equal contributors is a powerful teaching tool.

Things you can do while ensembling to improve the experience

- Encourage people in the audience who want to be navigating from the audience to join the mob. To be more exact, demand that or holding their perspective that can be very disruptive.

- If you want to introduce who is in the mob, you can do that on first round of rotation. If you want deeper introduction, you can have a different question to tell about themselves on each round of rotation.

- When people rotate, ask them to tell what they continue on. It helps to enforce the yes and -rule and is sometimes necessary when nervous participants have been building their private plan waiting for the hot seat.

- When driver is making their own decisions at the keyboard, simply call that behavior out referring to the rule: “No decisions at the keyboard.” or “Looks like there’s decisions happening at the keyboard.” Calling it out when you see it gets the group to a place to adapt.

- When group is stuck, ask questions. “Does it compile?”, “What should you do next?”, “Did you run the tests?”, “What are your options now?”. Your goal is not to do things for them but get them to see what they could be doing.

- Repeat ideas from more silent mob members. Sometimes people only hear the designated navigator and don’t pay attention to the other people and their contribution. When this happens, stop the group and make space for

the more quiet people's ideas. You can amplify them by calling attention to them.

- When group is stuck in not knowing how to do a thing, say "Let me step in to navigate" and model how to do a thing for short timeframe. Expect the group to do that themselves the next time.

- When a group gets stuck in planning, move them to quickly listing options and introduce a rule: do the least likely first, be open to do them all. A lot of times people are happy with the option they would fight indefinitely after it has been implemented. Doing things in many ways over arguing about them moves the needle from ideas to action.

- If someone in the mob starts taking up all the navigation as they know what needs doing while others are learning, enforce a stronger designated navigator pattern to allow everyone space to turn their ideas to words that get turned into action and code.

Things you can do in retrospective to save up a messy session

- Facilitate a retrospective towards discussions around reasons we could learn from for lack of progress

- Introduce theories or ideas of how you could try doing things different the next time.

- Find your own style of facilitating groups of strangers. Having seen multiple people facilitate, there are style differences where one person's approach would feel off on another. Strong-handed "supporting progress" and light-handed "enabling discovery" will result in sessions that are different.

Training in a Mob

A special case of mob programming that has been very beneficial for us is to deliver a training in a mob. Anything you could teach in a class room format, you could also teach as an experiential session where you lead people to learning through doing and reflecting on doing.

Before mob programming, our experience was to teach experiential contents through pairing and group work where the facilitator set up the task, and the pair / group was on their own on doing the task. Facilitator would again bring the whole training together to learn across groups. This format worked well, but missed out on one thing a trainer could do: them seeing what exactly is going on, and them teaching over just facilitating learning that happens.

Mob programming as a training tool for us has provided many frames in which to work in:

- 1-3 day courses where problems we work on are designed to teach a curriculum for a group of 12-25 people. This is a drop-in model for a traditional training.
- Visiting coach model where training is time boxed for each team, usually in two kinds of learning experiences: working on your own production code or working on designed teaching problems. This is a model for coaching and teaching while coaching and replaces talking about perceptions with doing things and building habits and knowledge.

Remote Mob Programming

CHAPTER 7

RECOMMENDED PEOPLE AND MATERIALS

For coaching using mob programming model, *Emily Bache* does great work on structuring and popularizing what she calls *technical agile coaching*. Her talk from Agile Greece is available on YouTube: https://www.youtube.com/watch?v=9tfGt5ToBfI

For working as a programmer in a full-time mob, I look for inspiration from Åsa Liljegren. I recommend reading her blog post about 4 years of constant mob programming: https://reallyshouldblogthis.blogspot.com/2019/07/4-years-of-constant-mob-programming.html

Mob Testing is a special case of mob programming and has taken the testing communities as a mechanism of learning. Reading what *Maaret Pyhäjärvi* has to say about it is worthwhile: https://www.ministryoftesting.com/dojo/lessons/mob-testing-an-introduction-experience-report

For using mob programming as a tool to transform organization from within, *Lisi Hocke* works with multiple teams starting them with mob programming and learning

with them. Lisi's sessions are golden. Her blog is full of experiences, starting with: https://www.lisihocke.com/2017/04/our-teams-first-mobbing-session.html

A special integration of ensemble programming to a product development I find inspiring is reported by *Anssi Lehtelä*. He has spoken of their experience at Visma in conferences and as a practitioner, appears on conferences fairly irregularly.

Sal Freudenberg and *Matt Wynne* and the Cucumber Pro team did remote ensemble programming. They discovered that individual work did not give them what they could get through ensemble programming over the internet together and reported some of their findings here: https://cucumber.io/blog/inclusive-benefits-of-mob-programming/

The "father of ensemble (mob) programming" *Woody Zuill* continues to evangelize and teach the approach. He has written a book on mob programming available on LeanPub: https://leanpub.com/mobprogramming and I recommend searching for one of his talks. Usually the latest has condensed his insights best.

The original Hunter Mob members continue advocating for Ensemble Programming and I recommend checking out the

work from *Christopher Lucian* on scaling ensemble programming at Hunter and bringing ensemble programmers together through a podcast, *Jason Kerney* on experiences of years of ensembling (https://www.agilealliance.org/resources/experience-reports/mob-programming-my-first-team/) and A*aron Griffith* on his work on ensemble programming for the introverted (https://www.agilealliance.org/resources/experience-reports/mob-programming-for-the-introverted/).

The ensemble programming community is large, and comes together annually in Mob Programming Conference organized by Agile New England in Boston. The best way to find people and experiences is twitter hashtag: #MobProgramming and #EnsembleProgramming

CHAPTER 8

MISCELLANEOUS THEMES

As our vision of this book is evolving and there's a need to write pieces down, this is a place for these miscellaneous themes.

Attribution for Work Done in a Ensemble

When we ensemble at work, the rights to the work belong with our employer. When we create something of relevance even if the rights belong somewhere, we want to know we are remembered for our contributions.

Even before ensemble programming came along, we knew it takes a village to build a software product. While we could see our individual contributions in the changes we made and types of changes we made, ensemble programming makes that more fuzzy.

Just look at Git, a core tool for version control. We have only recently been able to tag code contributions for multiple authors and while it is technically possible, we usually use someone’s account.

The attribution becomes more tangled with ensemble programming. Like when baking a cake, we can list the individual ingredients: sugar, flour, eggs, vanilla, butter. But when we mix them up in just the right proportions, the cake we get is better. We can say a missing ingredient makes things worse, but we can't attribute good to one of the ingredients. In an ensemble, people inspire one another, and end up creating things together they would not create alone.

Similarly, people in an ensemble change. A visitor in an ensemble two years ago may deserve private credit for their remaining impact, but would not be considered one of the people an award of recognition for today's level of excellence in software.

BEST IDEAS WIN WHEN YOU CARE ABOUT WORK OVER CREDIT

We often advice people who work in ensembles 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.

When working individually, claiming credit happens by idea and prototype. The finishing work rarely gets same level of recognition. We thinking that crediting ideas is wrong, and

that turning ideas into running, working software should be the target of attribution. 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 an ensemble, we seek a situation where everyone is either learning or contributing. Learning is something that stays with you when you leave the ensemble. Contribution is something you share with people you ensembled with, that is a product of its time where fair attribution and remembering it realistically is important.

Fair attribution does not happen automatically.

THIS BOOK AS A CASE EXAMPLE

Learning eventually leading to this book was a paired work at its time. As two authors could no longer collaborate on an unfinished book, we learned we did not share a vision of what collaborative crediting looks like.

The author of this Ensemble Programming Guidebook took the learnings, but recreated the whole book as a single author book after being kicked out from LeanPub for the book she had written. With two authors and this books

author as the main author, a lot of the text was written by her and checked in from the other authors computer, marking commits in GitHub for the second author only.

With a version of two author book unfinished, this book is a complete restart of the vision to create an Ensemble Programming Guidebook.

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. They felt differently and the dispute will be discussed in court of law. Regardless of the conclusion, the collaborator blocked this book from being published on LeanPub.

Attribution for work done in an ensemble 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. Ensembling 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 difficult prioritizing credit.

A famous example talks about identifying types of dinosaurs based on fossils. Researchers around the wold identified 12 types of dinosaurs, and everyone who identified one, got to name their discovery. Later, a researcher asked a relevant question: “Why are there no baby dinosaurs?”, only to discover that 5 of the 12 identified types are actually younger versions of same types. As crediting works, identifying a new type was prioritized over grouping new characteristics to an already identified type. Being the first mattered. Being the one with the idea matters.

APPENDIX

Ensemble Programming Timers

Problems to Ensemble Program On

Test-Driven Development

FizzBuzz Kata. Create a program that outputs Fizz for numbers divisive by 3, Buzz for numbers divisive by 5, and FizzBuzz for numbers divisive by both 3 and 5.

Roman Numerals Kata. Create a program that turns numbers into Roman numerals.

Legacy Code:

The Gilded Rose Kata by Emily Bache.
https://github.com/emilybache/GildedRose-Refactoring-Kata

The Tennis Kata by Emily Bache.
https://github.com/emilybache/Tennis-Refactoring-Kata