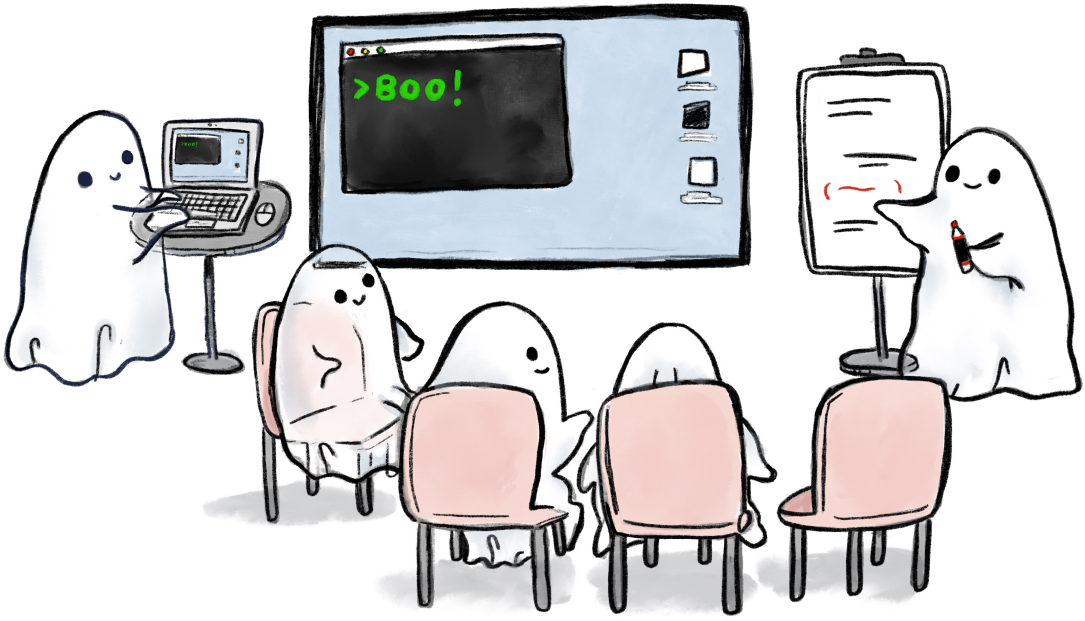
MOB PROGRAMMING GUIDEBOOK

MAARET PYHÄJÄRVI



MOB PROGRAMMING GUIDEBOOK

This book is available for download at

https://mobprogrammingguidebook.xyz

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

This version was published at November 2nd, 2019.

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. You can tweet about the book. I use the hashtag #MobProgrammingGuidebook 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*.

With twenty years of experience in the software industry behind me, I've learned to recognize the triggers that often lead to me to new insights. Resisting something without having tried it is a promise of an insight.

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 me 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 personal 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 shared 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*. 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 mob programming does not have to be programming, you will find we often use the term mobbing, or may talk specifically about mobbing 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 mobbing 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.

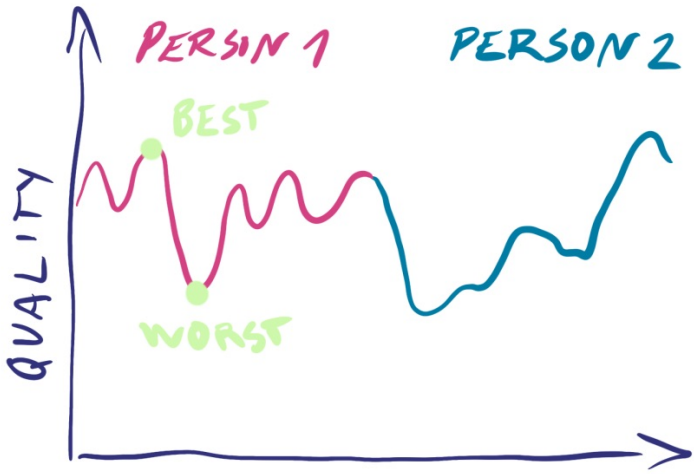


Image. Working Alone.

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.