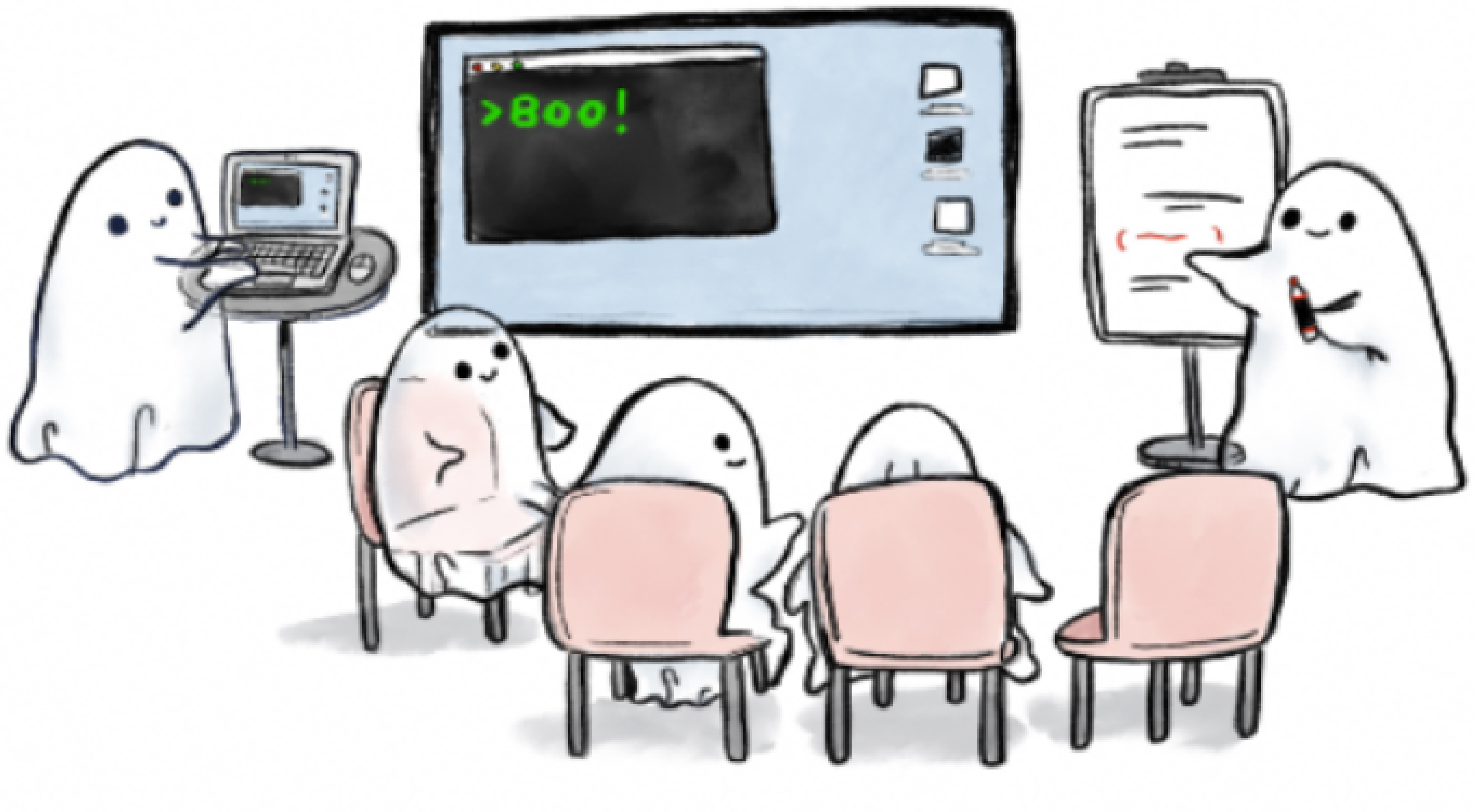
ENSEMBLE PROGRAMMING GUIDEBOOK

Maaret Pyhäjärvi



2022

Ensemble Programming Guidebook

Maaret Pyhäjärvi

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

This version was published on 2022-01-01

Tweet This Book!

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

The suggested tweet for this book is:

Check out @maaretp's #EnsembleProgrammingGuidebook on LeanPub.

The suggested hashtag for this book is #EnsembleProgrammingGuidebook.

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

#EnsembleProgrammingGuidebook

Contents

Ensemble Programming

Ensemble programming is a social software development technique where a group of people works on the same problem on the same computer at the same time. It's not typing that slows us down in our software creation, it's waiting for the right information to be available. Groups do better than individuals at that! When we get the best out of everyone into the work we are doing right now, we create a higher quality solution in the moment, and we improve skills and knowledge of everyone involved in the process. Next time we work solo, we benefit from the lessons and shared understanding only a group can generate.

And in case you wondered about it: ensemble programming is the community rebrand of mob programming to more inclusive language. Since 2020, ensemble programming communities globally have adopted it as a synonym to the software development technique that centers kindness, consideration and respect so that we can bring a diverse group together to create software and have a true bias to action, over worrying about the negative connotations.

Before we go into the dynamics of setting you up for ensemble programming, I will talk about one example.

An Experience in Ensemble Programming

The team of eight developers had been busy, and same was expected further on. Backlog of features was a long one, and everyone was working on something important. The developer with testing emphasis - a tester you might call them - was working with everyone clarifying the shape of each feature and testing for limitations we did not see ahead of time. The team could complete a feature at a time in branch, and on average make a release to production once a day. We worked solo, but together.

One day, the product owner pops in and shares news about a new business opportunity. We need this new kind of a feature, one that is not on the backlog right now. Should we be making space for it and if we did, do we have guesses on how big the effort around it might be? Estimates emerge, summing up happens and total to two months of effort.

Can we split it to multiple people, the product owner asks. Most likely yes, the team concludes, maybe two or three people could work on different components of it, but we would need to set up the architecture and design first to know what really makes sense.

We are not really sure if we're using numbers and rationale to explain that we really had already our minds set up to doing the work we are doing now, or if it is really that big. It has uncertainties, it is not just an extension of something we have already in place.

We have our weekly ensemble programming session, two hours of doing work for learning purposes coming up tomorrow, and we decide in the moment that it wouldn't at least hurt if we learned around implementing this feature, no one is expecting a certain output of those sessions other than learning. Our usual guiding idea of 'learning or contributing' had been quite balanced, but tilted heavily towards learning. But we had learned enough to take this feature up.

The ensemble programming timebox comes up. Seven out of eight of us show up, one of us is not participating as with the six months into these session, they still hate the idea of ensembling. We mutually agreed after a bad experience together they would be welcome back but they would have to follow team rules of no decision-making on the keyboard, continuing on working for the same idea when changing who is on the keyboard, and being on good behavior not making others feeling down on using the little space we had on pointing out how they dislike the technique rest of us enjoy.

We take our seats in the physical meeting room around a round table, facing the screen. One of us brought a work laptop, and has it connected to the screen, with our usual development environment available. First one of us sits in front of the laptop, and the ensemble around quickly agrees something we know already that will need to get done.

We have no one who would facilitate the group, just our mutual agreements. We have agreed we will change who is on the keyboard every four minutes, at sound signal from a phone timer going off. Very low key. The changing of who is on keyboard happens by all of us getting up and moving one seat forward, thus the physical layout in the room tell who is up next. We have agreed we want to all be contributing to telling the next steps, and take turns in assigning the one who would be next up on the keyboard to be the one who will either take the work forward or seek the help on the ensemble. Everyone pitches in when they can, but the rule of next one up on keyboard being the decision-maker ensures we all stay engaged. After all, we don't want to end up watching two people code.

Very soon into the task, we notice we are implementing something that one of the people in the group recognizes as something they have already created. Taking over speaking, the guide us to find a ready extendable implementation of a component we could use here, instead of the component we just started programming together. We do the necessary adjustments, create tests while not test first, and continue on.

Half an hour in, we get stuck with a question of how it should work - is there one of these, or many of these, and how they link to these other things conceptually. The developer with the testing emphasis has a strong model of the concepts, explaining them and group decides on the implementation. Someone cracks a joke: 'that would have been an expensive mistake to fix later on'. Two hours later, no one remembers it happened, expect for the tester who gloats with happiness years later recounting the experience.

Work flows forward, a lot of questions emerge, as well as answers. Everyone knows some bit that others don't. While working, people remind others of keyboard shortcuts, propose libraries that would be useful, rule out issues with licenses we don't want in our code base, test the user interface in different browsers by opening it in a different one every time we want to look, vary test data to test things while looking, and correct mistakes on the fly we can't even call mistakes because a typo isn't a mistake just because you wrote it, it becomes one only if it stays in.

At two hours, our timebox for ensemble programming is over. We recount our results. Another fun session, suprisingly productive. Impressive reuse of libraries. Nice ways of implementing things with less effort. Cutting down the extra from the feature. Correcting mistakes in the moment. But we are not done yet!

Four of the seven decide to continue until the end of the work day, two hours more. Same continues, and we make progress. At end of work day, we are so close to having the feature completed that it is clear we'll finish it. One of the four tells of they drop the other work they were doing to complete this, solo.

We don't talk about how much effort there is to complete, but three days later, the feature is in production.

Instead of 8x40hrs = **320 hrs**, it took us 7x2hrs+4x2hrs+2x2x8hrs = **54 hrs**.

In hindsight, we can argue that we overestimated the complexity. But working in an ensemble gave us **technical assets** we did not know we had without a person guiding us as soon as they realized opportunity for reuse. Similarly, working in an ensemble **integrated testing so that we avoided long loop feedback**. We **scoped the feature down** to the absolute minimum, so we could argue the estimate was not for the same feature. And with all of us in the room answering things in the moment and a product owner at a close proximity even if we did not need their help this time in this team, we didn't have as many cups of coffee as we were **not waiting for the answers** from someone who first needed to get in the context to give us the right answers.

The four hours on ensemble programming, particularly on this feature, left us all a little drained. But at the same time, being able to get the work done, and surprise us all on it was energizing. We were happy continuing with the 2 hours a week with ensemble programming.

Ingredients for Success

There's two main flavors of ensemble programming.

The **stable ensembles** have been working together before, and through spending enough hours in doing the work together, figured out a way of ironing out any wronkles with team member preferences and mutual empathy. Ensembling the 40-hour work weeks gets you here. A year of 2-hours a week also gets you here.

The **temporary ensembles** come together infrequently, or with varying compositions. While basic rules of learning to work together apply, we are still learning to work together.

The story we shared is closer to a stable ensemble than a temporary one. This book is about getting you started with your own ensemble programming sessions where some of your sessions could bring out all the best features of working together when you move from first rehearsals, already bringing value, to stable use of the technique when it helps your team and you the most.

What Does It Look Like?

Over the seven years of Ensemble Programming (and Ensemble Testing), I have been part of a lot of temporary ensembles and a few permanent ones. I have yet to work in a team that would do ensemble programming full time, even if I have visited the Hunter Mob who kindly let visitors slow them down to experience their way of working.

Collated picture of Ensemble Programming / Ensemble Testing

Mob Programming?

You may have seen a synonym for this style of social software development around - the word the original group started with was *mob programming*. Even with the intent of joking name for a technique focusing on kindness, consideration and respect, the impact is that it does not communicate inclusiveness. People opted out in sessions at companies and while in, chose to use significant time on discussing the need for a better name.

Currently ensemble programming is a synonym also in Wikipedia, and the new term is popular enough to find its way to many groups practicing this style of collaboration globally.

The new name came about in early 2020 when Maaret Pyhäjärvi collected proposals from multiple groups, organized a voting on proposals on twitter. The selected term 'ensemble programming' was proposed by Denise Yu, a senior programmer working at Github at the time.

Ensemble Testing

In 2020 with her book Technical Coaching with the Samman Method, Emily Bache moved to talking about ensemble working, dropping programming from the word. In roots of this guidebook, there is an equal amount of experience in ensemble testing (doing exploratory testing, system level test automation, getting code under tests on unit level) to things that are easily identified as programming (two latter of the three examples on testing). It is my belief in collecting the patterns for this book that in many ways, both words are helpful and I use the word programming in the outcome oriented sense - it includes knowing what we will build and verifying we built what we were supposed to in respect to the customers problems.

Encouraging focus on testing oriented ensembling, calling it ensemble testing helps break the team limits of who would be testing and invites developers with testing emphasis to lead through examples.

This book is equally applicable as an ensemble testing guidebook. Discussing ensemble testing separately is particularly useful for ensemble exploratory testing, an approach to testing hard to teach as it is a timely collection of many specific testing techniques. Exploratory testing is best learned by doing the work of software testing in an ensemble, seasoned with a regular reflection.

Getting Started with Ensemble Programming

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Choosing Your First Ensemble Activity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Understand Roles and Rotation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Reserve and Set Up the Space

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

The Driver-Navigator Pair in Ensembling

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Getting Set Up For Pairing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Who Should Drive and Who Should Navigate?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Driver: Things to Do

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Navigator: Things to Do

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Programming style matters

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

The abstraction level dilemma

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Mining the to-do-list

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Express a in-a-nutshell idea of what you're doing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Immediate feedback

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Closure Activities

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Recognizing time to switch

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Step back and think about what you're doing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Retrospectives

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ensembleprogramming.

Ending this with a word of warning