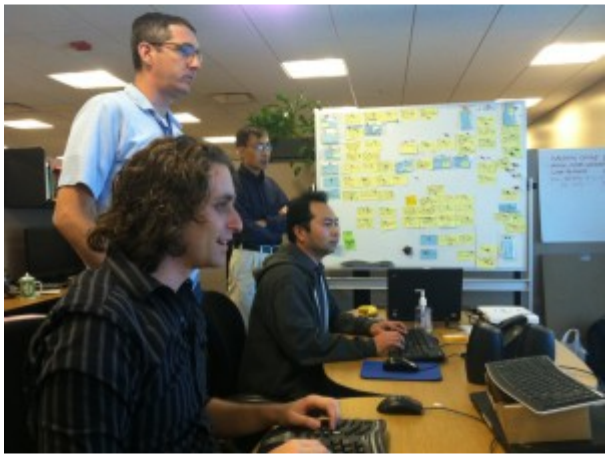
Mob Programming – A Whole Team Approach

WOODY ZUILL, Application Development Manager at Hunter Industries, Inc., Agile Coach



Mob Programming is a software development approach where the whole team works on the same thing, at the same time, in the same space, and at the same computer. This is similar to pair programming, where two people sit at the same workstation and collaborate on the same code at the same time, but with Mob Programming the whole team works together on delivering a single work item at a time, continuously collaborating and inputting code at a single computer.

1. INTRODUCTION

Mob Programming is a software development approach where the whole team works on the same thing, at the same time, in the same space, and at the same computer. This is similar to pair programming [PAIR], where two people sit at the same computer and collaborate on the same code at the same time. With Mob Programming we extend the collaboration to everyone on the team, while still using a single computer for writing the code.

In addition to software coding, the team works together to do almost all the work a typical software development team tackles, such as defining stories, designing, testing, deploying software, and working with the customer (in our environment we call our internal customers our "Partners", which is similar in nature to a Product Owner). Almost all work is handled as "working meetings" or workshops, and all the people involved in creating the software are considered to be team members, including our Partners. We work this way more or less all day long, every day.

In other words, this is an evolutionary step beyond the Extreme Programming [EXT] concept of pair programming. We strive to accentuate and amplify concepts such as face-to-face and side-by-side communication, team alignment, collaboration, whole team involvement, continuous code review, and the "self-organizing team", to name a few.

In this experience report I describe why we work this way, our basic setup, some of the benefits we have seen from Mob Programming, and how we work this way including a few critical practices, such as the "Driver/Navigators" teamwork model that we've adopted that makes it possible for us to work as a "whole team" all day, every day.

Author's Twitter Handle: @WoodyZuill, Email: wzuill@yahoo.com

2. HOW WE STARTED

We did not set out to invent a new way to work, or to extend the idea of pair programming. We simply noticed something that was working well for us and expanded on it.

Prior to discovering and adopting Mob Programming as our general work style, we followed a practice of having frequent retrospectives and continuously working at making things better. We had been learning pair programming and Test-Driven Development [TDD] and holding practice sessions using a Coding Dojo [DOJO] approach (where everyone uses a single computer with a projector and passes the keyboard around) to practice these skills. Several of the team members had started using TDD and pair programming, along with other new skills to some degree in their daily work.

At one point we needed to prepare to restart work on a project that was previously in development but had been put "on hold" for several months while some other more critical work was being taken care of. We gathered in a typical meeting room to take a look at this project and decide on how to take on the work. A few of the team members and a contractor had previously worked on this project, but the rest of the team had not.

During the initial meeting we were familiarizing each other with the project. We started by taking a look at some code, database tables, documents, and a few other details of the project. At some point in this review/refresh meeting we started trying out a few ideas, writing tests, changing code, and discussing how to proceed. As we had all been learning and practicing pair programming over the last few months it was very natural for us start to pass the keyboard around as we worked.

After several hours another group was scheduled to use the meeting room we had occupied, so we quickly grabbed up our stuff and headed off to find an unoccupied meeting room. We held a "mini-retrospective" [RETR] at the end of the day and we all felt the experience was very productive. We decided to arrange for meeting rooms to use the next day so we could continue to gather and work together in the same way.

We followed this same basic pattern of working together moving from meeting room to meeting room over next two weeks. Each day we reviewed how the day went, and each day we decided we wanted to keep working "as a team". We felt we were rapidly improving our ability to communicate well, increase our knowledge, and find better solutions. We were gaining a deep and shared understanding of the project and the technologies involved. About this time we decided to call what we were doing "Mob Programming".

Our biggest problems were things like the disruption of moving from room to room, issues with network connections in some meeting rooms, varying quality and usability of the computers and projectors, and other similar logistic and equipment issues. We also found we were experiencing ergonomic problems such as getting sore backs and necks from poor posture and the bad orientation of tables and chairs in relation to the projected screen at the end of the table. We were also getting headaches from squinting at low resolution or poor quality projected computer screens.

After the third or fourth week we were able to find a "permanent" work area we could use every day for a few months and had started to solidify some of our work practices. We were discovering ways to deal with the ergonomic problems as well, which I'll share in a following section.

We have successfully delivered many projects and enhancements over the last 3 years since we started doing "Mob Programming". While we now have a permanent work area and have made many incremental improvements to the way we work, we have continued to follow the basic pattern of collaborating as a whole team working at a single computer.

3. WHY WE WORK THIS WAY.

We work this way because the team decided to work this way. This is an important concept for us: The team doing the work can best determine how to do that work. We were not told to work this way. We have the freedom and responsibility to decide how we want to work. Having discovered that continuously working and collaborating together all day, every day, was working well for us continuing to do so was the natural next step. We continue to pay attention to what is working, and we frequently "tune and adjust" as needed. Our mantra is to "always turn up the good" [TURN].

4. THE BASIC SETUP

The basic setup isn't complicated, but is very different from the typical cubicle arrangement. While physical comfort and personal preference are easy to accommodate when working solo in separate cubicles, it becomes a bit of a challenge when most of our work is done sitting together in the group work area for extended periods of time. We have found it is very important to be physically comfortable while working relatively close to each other, and using shared monitors, keyboards, computer setup, and programming tools. We need to allow for personal preferences in keyboards, coding styles, work styles, tools, and so on.

Please refer to Fig. 1 to get an idea of our basic floor plan. Our main work area is configured using standard cubicle walls, and is about 16ft x 18ft. There is one computer used by all team members when writing code. We have two projectors to project "dual monitors" onto the wall, and two keyboards so team members have a choice to match their preference. Around the walls we have several rolling whiteboards and areas for flip charts. Several desks and other computers are available for individuals to use when not coding or otherwise using the main team computer.

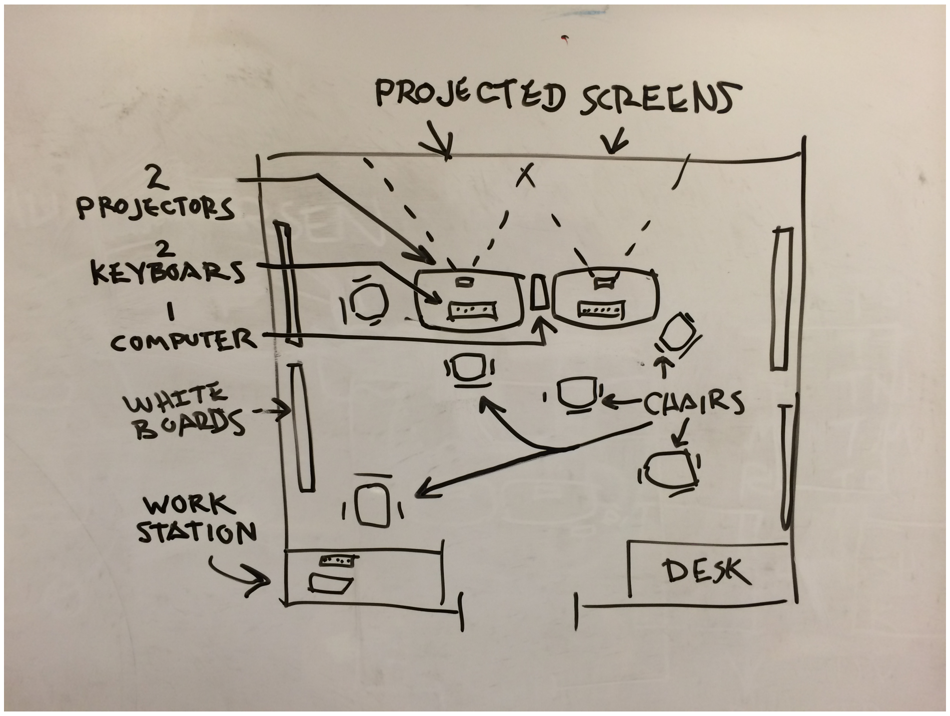


Figure 1 - Early Whiteboard Drawing of our floorplan once we had a permanent work area

4.1 Computers

There is only one computer in use for programming. All code that enters the code base is input through this single computer. This computer is also used for team emails, designing, testing, and other activities that involve the whole team.

We also have other desktop and laptop computers available that we use for researching, independently looking at databases or trying things out, writing personal emails, and other purposes in parallel with the programming. There will often be more than one person searching for information about some problem or new technology we are trying to use. We all stay together and communicate continually about what we are learning.

4.2 Projectors

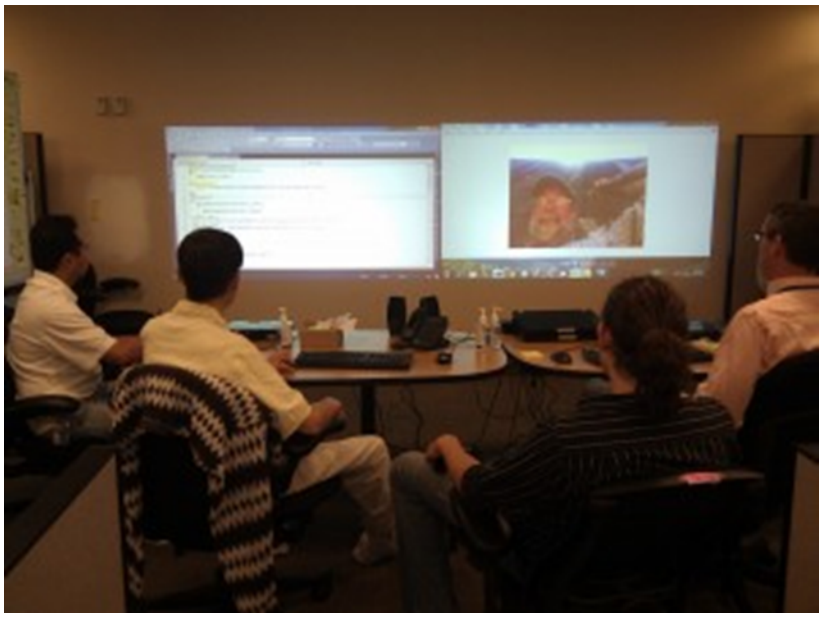


Figure 2 – Dual projected monitors.

There are two projectors which we use as dual monitors. We project onto a wall we have had painted with a special projector screen paint that works well for this purpose. After experimenting with the height, distance, brightness, ambient room lighting, wall paint, and other settings we have adjusted things to work nicely for everyone on the team. We project a lot lower on the wall than the typical meeting room projector as we found this allows us to keep our necks from getting sore. The two projectors are the same model and are high quality. Our goal is to keep the screens at about the same size, general position, resolution, and brightness to make them comfortable to work with all day long.

4.3 Keyboard/Mouse

There are two keyboards and two mice so everyone has a choice to suit themselves. We experimented with four or five different keyboards and settled on two: a "regular" one and a "natural ergonomic" one. We do not use both at the same time; there is only one developer “keyboarding” at any one time as I’ll explain later. When a developer prefers a drastically different keyboard layout, such as Qwerty vs. Dvorak, we find a way to quickly switch between the options. There are many personal preferences to take into account, such as left-handed vs. right-handed programmers, track-ball vs. traditional mouse, and so on. Any set-up issue that requires one team member to work differently from another is not a problem as long as we find a way to make the switch quickly and smoothly when each person takes their turn at the keyboard.

4.4 Chairs and Tables

Each team member has their own chair which is moved around as we take on the different roles (Driver or Navigator). This way we don't need to constantly readjust the chair settings and each person can stay as comfortable as possible. Our chairs are good quality and comfortable, and chosen individually for the team member during an ergonomic assessment. Our work surface is a couple of tables that are comfortable to sit at. The computer, keyboards/mice, projectors, phone, speakers, hand sanitizer, and a few other things we like to keep close at hand are kept on the tables.

One important factor is how we orient ourselves to the projected screens throughout the day. In typical meeting rooms, the projector screen is at the end of the table, so almost everyone needs to turn their heads to see the screen. While this is fine for short meetings, it becomes very uncomfortable when we work this way for several hours or a whole day. In our layout the tables are parallel with the screens so we can face them in a comfortable and stress-free way throughout the day.

We have a rolling magnetic whiteboard we use for keeping track of the work we are doing similar to a typical task board, as well as a few other whiteboards and easels, and a couple of file cabinets and small desks.

4.5 Private Work Areas

We each have our own separate work area to use whenever anyone would like to work alone. We have small desk areas in a separate annex to the main team area. These are configured as either sit-down or stand-up workstations depending on what each individual prefers, and each team member has their own computer, dual monitors, drawers, phone, etc. When in the private area we can still hear and pay attention to the main "mobbing" area if we like, or we can wear headphones or otherwise "tune-out" what everyone else is working on.

5. A FEW IMPORTANT WORK PRACTICES

With Mob Programming everyone is in almost constant communication with everyone else. While this brings a lot of value, it is also a relatively foreign way to work for many doing software development. We've found that we need a few simple principles and practices that allow us to keep focused and collaborating nicely throughout the day. Our goal is for everyone to be able to contribute or be learning at the level they feel is most useful to both the team and to themselves. We actually think of learning as a type of contribution to the team.

Disclaimer: While the principles and practices we have found and are using are working well for us, we are always on the lookout for improvement. We also realize that other groups and teams might find that our approach will not work in their context. We invite experimentation and innovation and would love to hear from others who have found new or different ways that work nicely in their environment.

5.1 The Principle of Treating Each Other with Kindness, Consideration and Respect

Tens and hundreds of interactions between people occur every day in our work. The number of interactions compounds quickly when most of our conversations involve 5 or 6 or sometimes more people, rather than between just two individuals at a time. We express ideas, discuss problems, explore possible solutions, and share thoughts all day long. We are rarely in agreement on most things until we have had a chance to hear from everyone who has something to contribute, and have had the back and forth discussions that expand our understanding.

To make it possible to keep this high level of communication happening throughout the day we have adopted a principle to always treat each other with kindness, consideration, and respect. While this seems straightforward, we feel that expressly acknowledging the importance of this principle provides a foundation for our daily interactions. People like me who are not good at being kind, considerate, and respectful quickly get better at it when everyone is committed to live this principle.

5.2 The Driver/Navigators Pattern of Programming

We use the Driver/Navigators [DRVR] pattern I adapted from Llewellyn Falco's "strong" pair programming style. The basic rule is that "for an idea to go from your head into the computer it MUST go through someone else's hands."

There are two roles: The Driver, and the Navigator. The Driver sits at the keyboard and types in the code. The Navigators discuss the idea being coded and guide the Driver in creating the code. This means the Driver has a much more mechanical job than when coding solo. The Driver listens to the Navigators, and must trust the Navigators. The Driver is focused on the typing/coding. The Navigators are expressing their ideas to the Driver in a slow, metered approach so the Driver only has to focus on the next thing to type at any given time.

While expressing these things to the Driver out loud they are also being expressed to the rest of the people on the team. We discuss and work out the possibilities verbally and at the white board so everyone is gaining a full understanding of the idea. This creates a sort of collective intelligence of the Navigators and the team as a whole.

We use a timed rotation, where each team member works at the keyboard as the Driver for a short period of time (typically 10 to 15 minutes). We use a timer, and the current driver hands the keyboard off to the next driver when their turn ends (explained below in section 5.3).

It is important for the Navigators to speak at the highest level of abstraction that the Driver (and the rest of the team) is able to digest at the moment. Sometimes this can be at a very high level when the Driver understands the concept to be coded and can proceed without detailed instructions. It can also be at a very detailed level if necessary, even at the level of keystroke instructions when needed. This will change from person to person, and will also change for the same person throughout the day depending on the idea being worked on, and the ability of the Driver to understand the instructions.

I’ve found the Driver/Navigator approach to be very powerful. To follow this approach we must become good at communicating and discussing each idea with another person before it can become part of the code base. We automatically get continuous discussion and review of the problem, the solution design, and the code. Everyone stays involved and informed.

5.3 Driver Rotation using a Timer

We rotate the Driver every 15 minutes, so no one is attached to the keyboard for very long. Using a randomized list of who is working that day, we "rotate" through the list: Every 15 minutes the current Driver moves away from the keyboard and joins the Navigators and the next person on the list moves to the keyboard to start typing. As the day goes along whenever we get to the bottom of the list we just start over again at the top. We've written a little timer application that takes care of this for us, and actually blanks out the screens when the Driver’s time is up. While we typically use a 15 minute rotation, we shorten the duration at times, and suggest that those just starting out with Mob Programming might want to use a much shorter setting. When we first started we used a 4 or 5 minute timer, and eventually increased it as our skills and comfort level with this practice matured.

5.4 Telephone and Email

All team related telephone calls and email communications are done as a team. We sign our emails as "The Dev Team" and have a group email address. When we make or take phone calls as a team, we mention to the other party that they are on a speakerphone call: "Hi Mary, this is Woody on speaker phone with a few of the other team members here".

One reason we use this practice is so everyone on the team is aware of all team-related interactions with people from outside the team. This solves some of the common silo problems which occur when there is only one person who is a point of contact. When that person is not available communications break down until they return. Additionally, misunderstandings are less common as the one team members will catch things that others have missed.

6. IMPORTANCE OF RETROSPECTIVES [RETR], AND TAKING ACTION

This is one of the Agile Principles: "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." [APRN] We have taken this to heart and found it brings a great deal of value to us. We frequently evaluate what is working for us, what problems we might be having, and how we can improve on things. An important part of the Principle is to "tune and adjust". By simply paying attention and then "tuning and adjusting" we have been able to choose actions that have led to tremendous improvements and to the discovery of our Mob Programming style itself.

We follow a typical retrospectives pattern for most of the retrospectives we hold. We set aside a half hour to an hour to reflect on the last week or two. In these sessions we gather information on sticky notes, do affinity groupings, dot-voting, and have conversations about the things we have observed and new things we'd like to try [RETR]. We have used a number of techniques, but most commonly we do some version of the "What is working", "What needs help", "What we want less of" pattern where we gather ideas about each of these headings and explore the things that we find are most meaningful to the team.

We always look for "action items", and limit ourselves to only one or two we can use to "tune and adjust" our process. We have found that having more than one or two action items is almost always counter-productive. We take "baby steps", make sure we try changes we think will be useful, and then reflect, tune and adjust.

6.1 We also often do Just-In-Time Ad-hoc retrospectives

Besides scheduled retrospectives, we hold retrospectives at any time we feel it will be helpful to us. When anyone on the team notices something they feel we should reflect on, we simply go ahead and do it while the experience is fresh. These are usually short and focused on one item. The point isn't always to "fix things".
Besides noticing problems, it is just as likely that someone will have noticed something good that just happened, or even something we found interesting. If there is an action item we think we should take on we make a notecard and put it up on our board along with our other action items.

6.2 Lean Coffee ™ [LCOF]

We often use the Lean Coffee method for some of our retrospectives. Lean Coffee™ is an easy way to keep discussions on point and moving rapidly. The basic premise is that those attending determine the agenda at the start of the meeting, and each topic discussed is time-boxed to 5 or so minutes. This keeps us focused and keeps the conversation from running on and on. We've found this to be a very effective approach to covering a lot of topics in a limited amount of time. We typically spend 45 minutes in a Lean Coffee™ session, and strive to find an action item for anything we discuss that we feel requires further action. By applying our principle of treating each other with kindness, consideration, and respect our Lean Coffee sessions are always meaningful to us.

6.3 "Turn Up the Good" [TURN]

We have found that we get a great deal of value from recognizing things that are going well and finding ways to increase the good from those things. We call this "turn up the good". While it can be useful to identify problems and try to fix them, we find we get even more value from finding ways to increase the things that are going well.

6.4 Taking Action

To get any real value from reflecting and holding retrospectives we need to take action. We always discuss possible action items, and choose a few to try as we go forward. Some action items have good results, some don't. We pay attention to these results, reflect on them, and tune and adjust over time.

7. FADING PROBLEMS [FADE]

After doing Mob Programming for a while we noticed that many of the problems we had previously faced were no longer affecting us. While we hadn't directly attempted to solve these problems we noticed they were simply fading away.

We listed each of these "fading problems" and ended up with a list of many specific problems we were previously having and were either no longer affecting us, or had become much less a problem for us. While the specific problems will likely be different from company to company, we found these problems can be grouped into several categories. A few of the main categories are counterproductive communications, decision making dysfunctions, the waste of doing more than what is barely sufficient, the debilitating disease of technical debt, and the burden and associated waste of heavy management techniques (estimating, prioritizing, scheduling, queuing, performance reviews, etc.). Other problem categories include the loss of focus and effectiveness caused by thrashing of programmers through unnecessary task switching and interruptions, the harmful aspect of politics of the workplace, and the harm of "management by meetings" with its inherent separation of decision making from the creation of knowledge. There are likely other categories of problems, and I've included these here just to give you an idea of the sort of things we've noticed.

Agile Software Development addresses all of these common problem areas, and Mob Programming encourages and enhances a heightened Agile work style. We use both the philosophy of the Agile Manifesto [AM] (the Values and Principles) and the concepts of Lean Thinking [LEAN] to evaluate the practices and techniques we are using or want to try. This almost automatically happens by simply "turning up the good" of collaborating and working as a whole team. For example, face to face (and side-by-side) communication and decision making just naturally happens when we all sit together throughout the day. Additionally, the fear and "analysis paralysis" that interfere with effective decision making are lessened or eliminated when we make decisions in an environment where everything is transparent and we have rapid feedback so the results of our decisions are quickly validated.

We attribute the improvements we've seen to the continuous collaboration and communication of our work style, the automatic application of a "limited WIP" and one-piece flow work style, the frequent delivery of software into use and rapid feedback this provides, and holding frequent and meaningful retrospectives. More details of these fading problems have been written up in a recent blog post [FADE].

8. PRODUCTIVITY

One of the first questions people ask about Mob Programming goes something like this: "How can 5 or 6 developers be productive while working this way? Wouldn't it be more productive to have them working on different things?"

While we feel we have become a great deal more productive working in this manner, there is no easy way for us to prove this is due solely to Mob Programming. If we take to heart the Agile Manifesto Principle of "Working software is the primary measure of progress", perhaps we can judge our productivity by comparing the number of projects delivered into use for the year before discovering Mob Programming, and the year after we adopted it: We had an increase of approximately 10 times the number of projects delivered. There are many factors we are not taking into account using this measurement, such as project size or number of features. While this general comparison is meaningful to us internally based on our knowledge of the work involved for these projects, it might not be meaningful to anyone else.

Even though we feel we had a strong increase in productivity, we do not claim that this would be repeatable in any other environment. There were many factors involved. For instance, the previous development methodology in use for this group prior to Mob Programming was the traditional phased waterfall model. Developers were working in a solo manner and on separate projects. Simply moving from that existing methodology to an Agile based approach could be responsible for much improvement.

While productivity is important, it is not more important than working on the right things. Validating our ideas and results, discovering value, and communicating well about the things we are working on increases our chance at valuable results. Maximizing the amount of work not done by focusing on simplicity and finding solutions that are "barely sufficient" eliminates waste. Simply getting a lot of work done has little value if it is of poor quality and can't be maintained, or isn't fit for its intended use, or for which the intended use itself turns out to be of little value to the end user.

9. THE HAPPINESS FACTOR

Another aspect of our work style is that each day we come to work energized and excited to be working together. While few would claim that they "live to work", we find that our lives are greatly enriched by working together as a team. We call this the "happiness factor". We follow a very sustainable approach which keeps us engaged and interested in our work. We pay attention to our physical and mental health and strive to provide an environment where everyone can excel in their work, and excel in their lives. This helps us to do our best thinking and invent the best solutions we are capable of creating. We are all constantly advancing our careers by learning and expanding our skills and capabilities. We have a general sense of happiness and fulfillment in our work.

10. CONTINUOUS LEARNING, INTENTIONAL LEARNING/PRACTICE, AND EXPERIEMNTS

Mob Programming provides an environment where continuous learning occurs. Typical programming skills are easily revealed and learned as we watch each other coding. Everything from keyboard shortcuts and programming language features to design patterns and business concepts are exposed and shared across the team. An interesting side effect is that we are all learning to become better students and teachers; better at asking and answering questions. Regardless the level of experience any one person has, there are endless opportunities for each of us to discover and learn things that are meaningful to our own improvement.

Besides the amplified learning that occurs naturally by working in a highly collaborative manner, we also take time to "re-sharpen the saw" [S&R] daily by spending the first hour of the day in a group study session. Additionally we have an extended study session most Fridays to do a more intense study for 2 or 3 hours. In our daily study sessions we select some aspect of programming that we feel is a weak spot for us, and spend an entire hour studying it. We usually do our study as a workshop and run it as a Coding Dojo similar to our Mob Programming style. We'll use any technique that helps, such as working through a code kata, watching on-line video training, studying a book, or tackling some interesting algorithm or some new technology.

Since we work in very short iterations of a day or two it is easy for us to experiment with various ways to do things. We keep an eye out for any aspect of our work that we can automate or simplify and try any approach that we think might work. This includes both programming and process related ideas. For example, if we have several ideas for solving a problem, but with no clear winner across the team, we'll try a minimal version of each solution and see which we like better. The cost for doing experiments is relatively low, and the payoff for us is often many times the time invested.

One interesting thing we have noticed is that when we add a new member to the team they become contributors almost immediately by bringing new ideas and coding skills to the team. In reverse, they quickly learn the nature of the project we are working on and "learn as they earn"; the new developer is providing value while they learn the ins and outs of the business logic we are developing. This also shows up when we have visitors join the team for a day. We have had many cases where a visitor starts contributing coding and technical ideas within 10 minutes of joining the team.

11. A FEW THINGS TO WATCH OUT FOR

Working as a Mob means all of our weaknesses are exposed. Everything we do is visible to everyone on the team. Some people might find this uncomfortable to such a degree that they will not be able to work this way. However, while it took us a while to get there, everyone currently on our team has been able to adjust to this constant scrutiny and exposure. We all understand that we are committed to treating each other with kindness, consideration, and respect so we are not as vulnerable as it might at first appear. While some people are more skilled or capable than others, we are all less than perfect, and we will all have something we don't feel confident about exposing to others.

To make it easier on everyone, we do not insist that people participate if they don't feel comfortable. Everyone is expected to contribute in the way they feel they best can, but no one is forced to sit and work with the team. It's a personal choice. All are invited to join in but are free to work alone if they so choose.

12. HEALTH AND ERGONOMICS

We realize that working closely with other people can increase our chance of sharing germs or passing illness around. To lessen the chance that we'll all get sick at the same time we have enough room for everyone to sit far enough away from each other so we aren't sneezing or coughing on each other. We also keep hand sanitizer on our table for use when we change Drivers at the keyboard. When one of us is sick, we encourage that person to stay home. If they want to work we have them work from home rather than spread germs around to the team.

It is important to pay attention to ergonomic factors. We work at identifying and dealing with problems that interfere with having a comfortable and stress free workplace. For example, we have various keyboards available to us to accommodate everyone's preference to try and combat repetitive stress injuries. We pay attention to avoid repetitive stress, headaches, and eye strain, and make adjustments as needed. Our rotation helps limit time at the keyboard and mouse. We are set up to be able to stand and work, and change our position often. As I mentioned previously, each of us has our own chair adjusted to suit ourselves. We wheel our chair up to the keyboards when it is our turn to drive so we can maintain a comfortable posture without having to readjust.

13. WHAT IS THE IDEAL SIZE FOR THE TEAM?

This is a frequently asked question, and we do not have sufficient information to answer. We are currently working with 6 people on the team. We have had as few as 3 and as many as 12 working together as a "Mob" and found we were able to be productive. However, we use a heuristic to help guide us in regards to "how many is too many": For each individual, if you do not feel you are either contributing or learning we take that as a sign that this might be a good time to work on your own for a bit, or to split off with a pair or a few others to start your own Mob.

14. DO YOU RECOMMEND MOB PROGRAMMING?

People often ask if we recommend Mob Programming. We have found that it works well for us and it might work for you. However, rather than recommending it we are merely sharing our experiences. We do believe that it is worthwhile to investigate the concept and see if there are parts of it that will work for you. Who knows? It might end up working well for your team.

There are a number of teams around the world that are experimenting with Mob Programming, and some teams we have heard from are working this way on a daily basis, or several times a week. We have often heard from people who use this style of work when they have an emergency or particularly hard problem to solve, and I've included links below to a few articles others have written about their experiences. [LINKS]

One concept I feel is important is that the team must decide if this appropriate for them. It would likely not be appropriate to mandate that anyone must work this way.

For those wishing to try Mob Programming, I would suggest that having previous knowledge and experience with pair programming, one-piece flow, test-driven development, continuous delivery/deployment, retrospectives, Coding Dojos, Clean Code, and other Agile and Lean concepts are all useful. However, anything you need can be learned in the doing along the way, as long as you dedicate yourselves to treat each other with kindness, consideration, and respect.

We encourage everyone to get good at holding meaningful retrospectives, paying attention to what is working for you, and experimenting with ways to "turn up the good." If you decide to try Mob Programming, I'd love to hear from you about your experiences. Also, if you would like some help just let me know – I'd be happy to help any way I can.

15. Acknowledgements

First and foremost I want to acknowledge the members who were working on the team when we first discovered Mob Programming: Dexter Baga, Gordon Pu, Chris Lucian, and DanYeung Wong. Without their dedication to paying attention to what was working and to always "turn up the good", we could not have found this wonderful way to work. I'd also like to thank my boss, Marc Kase, who has the wisdom and foresight to recognize the power of the Agile Principle of "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done." [APRN] I also want to acknowledge Aaron Griffith and Jason Kerney who have joined the team since we first started Mob Programming and have been great contributors and team members.

I owe a lot to Llewellyn Falco for his unselfish sharing of every good thing he discovers, his ability to discover lots of good things, and his undying habit of always looking for ways to help others in any way he can. I learned the Driver/Navigator model of "strong" paring from him, as well as the Coding Dojo concept. I've also learned a lot from Llewellyn about pair programming, test-driven development, how to work with legacy code, ApprovalTests [APPR], and software development in general.

I greatly appreciate Joseph Yoder [YODER] who has been my shepherd on this paper. Without his help and direction I would not have been able to turn my jumbled set of ideas into an actual paper. Everyone should have a chance to work with Joe – he's a real pro.

REFERENCES

[PAIR] Pair Programming: Extreme Programming Explained: Embrace Change, 2nd Edition, Kent Beck
[EXT] Extreme Programming: Extreme Programming Explained: Embrace Change, 2nd Edition, Kent Beck
[TDD] Test Driven Development: Test Driven Development: By Example, Kent Beck
[TURN] "Turn Up The Good" comes from the idea of "I would turn all the knobs up to 10 and see what happened" from Kent Beck: [EXT]
[DOJO] Coding Dojo – I was introduced to the Coding Dojo via Llewellyn Falco, who learned the concept from Laurent Bossavit, who introduce the idea with Emmanuel Gaillot. Emily Bache has written a book on the subject available from LeanPub: https://leanpub.com/codingdojohandbook
[RETR] Retrospectives: Agile Retrospectives: Making Good Teams Great, Authors: Esther Derby, Diana Larsen
[DRVR] Llewellyn Falco have written a bit on the subject here:
http://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html
[AM] The Agile Manifesto: http://agilemanifesto.org/
[APRN] Agile Principles: The Principles Behind the Agile Manifesto, http://agilemanifesto.org/principles.html
[LCOF] Lean Coffee: Jim Benson and Jeremy Lightsmith, http://leancoffee.org/
[FADE] Fading Problems, on the Mob Programming Blog: http://mobprogramming.org/fading-problems/
[LEAN] Lean Software Development, Authors Tom and Mary Poppendieck, http://www.poppendieck.com/
[S&R] Nobody Ever Gets Credit for Fixing Problems that Never Happened: Creating and Sustaining Process Improvement. Repenning, N. and J. Sterman (2001): http://web.mit.edu/nelsonr/www/Repenning=Sterman_CMR_su01_.pdf
[APPR] ApprovalTests are a library that extends Unit Test in many programming languages : http://blog.approvaltests.com/
[YODER] Joseph Yoder: http://www.joeyoder.com/
[LINKS] Here are some links to articles or video about experiences with Mob Programming
Our team blog: http://mobprogramming.org/
Time lapse of a day of work: https://www.youtube.com/watch?v=p_pvslS4gEI
AppFolio Engineering Blog: http://engineering.appfolio.com/2014/03/17/my-experience-with-mob-programming/
Tagged: http://blog.tagged.com/2014/05/mobbing-tagged/#more-2520
Agical: https://www.youtube.com/watch?v=goAMu-XqJts
Per Jansson: http://pichdude.wordpress.com/category/agile/
Marcus Hammarberg: http://codebetter.com/marcushammarberg/2013/08/06/mob-programming/
Kevin Rutherford http://java.dzone.com/articles/reflections-day-mob
Amy Lightholder: http://www.light-holder.com/mob-programming-at-shesgeeky/
Mob Refactoring: http://blog.codeclimate.com/blog/2014/01/30/mob-refactoring/
Agila Sverige, Tobias Anderberg and Ville Svärd: https://agilasverige.solidtango.com/video/2013-05-20-agila-sverige-torget-d2p02
Richard P. Gabriel at the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications on October 19, 2000, in Minneapolis, Minnesota, USA on a related concept: http://www.dreamsongs.com/MobSoftware.html