Growing the Mob

CHRIS LUCIAN, Hunter Industries, Inc.

Mob Programming was conceived of at Hunter Industries in 2011. We were a small, highly successful team. Then we grew. This report is about how over the past year we grew our department from one Mob of 5 team members to a department of 30 people supporting multiple streams of work. We've had to overcome challenges of incorporating new members at a pace we can sustain while preserving our team culture and focus on quality.

1. INTRODUCTION

I am Chris Lucian, part of the team who started Mob Programming at Hunter Industries in 2011. Several members of our team have published experience reports based on the results and experiences our team has had: Woody Zuill [Zuill], Aaron Griffith [Griffith], and Jason Kerney [Kerney]. This report is about how we have grown our team from one Mob to eight. In September 2015, I was asked by Hunter Industries to take a new software development director position to begin a new department and to grow the number of Mob teams. My role has been to help expand and grow our team while keeping its culture and values intact.

2. BACKGROUND

Before our most recent expansion, we had been Mobbing for four years with a team of 5 people. During that time, we experienced a one-and-a-half-year period where we received no new bug reports for any software we had developed while releasing working software to production sometimes twice in the same day. This success led to executive management asking us to expand the practice of Mob Programming into a larger portion of the organization. For one year now we have been scaling Mob Programming from one team to eight.

Our team Mob programmed for 4 years between 2011 and 2015 with no real change to team size. We had some people leave and new members join the team, continuously improving as we progressed. Our supervisor at the time, Woody Zuill, felt that a manager was no longer needed for the team due to how well it was operating. Woody then left the organization 6 months before we had started scaling. Our Mob then reported directly to our Director of Information Systems, Marc Kase. In this model we identified that we did not need to explicitly have a manager. This proved important to the growth of our department, and we continue to find that not having managers works well for us. We did a better job of setting objectives for each other and of building our skills than a manager could have ever done in a traditional top-down role.

For about six months after our manager left, our Mob programming group continued to work as it had, however, we self-organized around management tasks rather than any one person in charge. We handled management-related tasks within the team and many times as the entire Mob. Ultimately, when I became Director of Software Development and we decided to scale to multiple Mobs, we as a department decided to move forward without managers and continue to use this model.

In the summer of 2015 we also began to experiment with multiple Mobs by bringing on two interns. This experiment was a cost-effective way to allow us to add people to the team and evaluate the benefits. These experiments were successful and influenced the decision to expand the team further.

Author's address: Chris Lucien, Hunter Industries, 1940 Diamond St, San Marcos, CA 92078 USA; email: chris@chrislucian.com

3. GETTING BUY-IN TO GROW

As with any new process, while we developed Mob Programming within the organization we experienced both support and skepticism. We knew Mob Programming was working for us and we believed it could help with other difficulties in the organization. It is hard to get buy-in from managers as well as individuals within the company when people don't understand how it can be more effective to have four people working on one computer instead of a single developer. In order to overcome this, we had to address everyone's concerns one by one. It is important when spreading a practice like Mob programming to establish good relationships with the people who are least likely to believe in the effectiveness of the process. A good relationship will help facilitate the use of good metrics in order to convince them otherwise.

The most interesting comparison I've seen between a group that is doing Mob programming and a group that is not, came from within our organization. During the time, we have been practicing Mob programming on internal applications, our external product development team was going through some of the same problems that were relevant to us 4 years earlier. During an external software development process audit the two teams were evaluated for their practices. Our team was performing Mob programming and Agile in general, while the other team was operating under an increasingly unsuccessful process. It was clear that the Mob process needed to grow and spread within the organization. Yet it was only a state of emergency on the other team that caused rapid and drastic change.

Unfamiliar with agile software development, Arne Pike, our vice president of engineering's curiosity was piqued about the process after hearing about our results. I described to him the many aspects of Mob programming that I believed made us successful. I told him that the foundation of Mob programming was our learning sessions and retrospectives. I also described how overtime is not an indicator of dedication and instead is an indication of failing processes. I mentioned, however, that release frequency is an important metric and needs to be monitored regularly. We discussed how bad practices can cause a breakdown in both release frequency and manifest over time. I also introduced the concept of technical debt and the importance of continuous integration, continuous delivery, Test-Driven development, and behavior driven development. Finally, I described to him the positive effects of having no bugs in production. Quickly, the reasoning I presented helped convince him that our Mobbing and agile process was a good one to adopt within the software that he was overseeing.

With both the vice president of engineering and the director of information technology convinced, the president of the company, Greg Hunter, gave us the ability to scale and create an environment in which great software becomes inevitable. We got buy-in from all major areas of the organization and we now only had to address the small patches of skepticism within the organization rather than having to fight a large up-hill battle.

At this point it was decided that the external product team and the internal product team should merge. Our department would report to both the VP of engineering and the Director of Information Technology as a centralized software development organization within the company. At this point, all of the team members from both groups began Mobbing, allowing us to grow to our second Mob. Since the organization as a whole identified the importance of software development, leadership wanted to further grow 6 more Mobs. The number of 6 Mobs eventually became 8 as we took on interns and added additional projects to our catalog.

I felt that the new product owners were the most taken aback out of everyone. Their previous experiences with software development groups were entirely based in Waterfall practices. Since we wanted to work closely with our product owners, we explained the concepts that had helped us succeed in the past with our product owners for internal applications. This meant explaining minimum viable product development, automated testing, single item work in progress, and Agile in general. One of my greatest successes in helping convince our product owners to work with us the way we desired, was through one-on-one conversations over a set of PowerPoint slides that we use discussing minimum viable product feedback and other systems we use.

4. GROWING THE MOB

Once we had decided to grow Mob programming within the company, we needed to identify how each Mob would handle work as it came in. In the past, with our one Mob team we used Kanban and it was very effective for us. We described our work as a stream of work that could be utilized for any project, depending on whatever priority was highest. With our new department, each Mob was identified as a stream and we allocated streams to the projects that we wanted to accomplish. Each Mob on average consisted of four people. In order to grow our capacity, we decided to begin hiring four people per mob. Initially, we liked four members

in each Mob, however, we found that five typically is better in order to facilitate people taking vacations and dealing possible distractions. With the assumption that any feature will have four people working on it, the developers self-organize around available work. Membership of a Mob is purely negotiated and any developer can move from one Mob to another as long as each Mob has four people. A long-term effort may consist of two full time Mobs, however, those Mobs could re-form or decompose based on the self-organizing nature of the department.

4.1 Preserving the Culture

We knew that adding people to the team would dilute the culture we had established. Since the organization decided that we needed to have six Mobs, we decided to begin hiring for twenty-four people. Since twenty-four is an enormous number of people to add to a small team, we decided to define a growth rate that was not to be exceeded in order to preserve our culture. This number was two people per month.

Quickly, we found that we were able to hire at the rate of two people per month. We found that through community interaction at user groups and agile conferences, we were able to find the type of people we were looking for to join our Mob programming teams. In fact, because of the benefits that we receive from Mob programming, we found that many developers wanted to work with us for the experience with a new style of software development. We had Woody Zuill come in to facilitate workshops in Mob programming, helping us tremendously with hiring and spreading the word about our openings.

To integrate each pair of new hires into the team, we developed a way to onboard the members and get them familiar with the team's core beliefs. We hired at this rate for several months (see Figure 1). We experimented with hiring four people within the same month and found that it was a much larger disruption to the culture and practices. Since then we have decided to stick to two people a month.

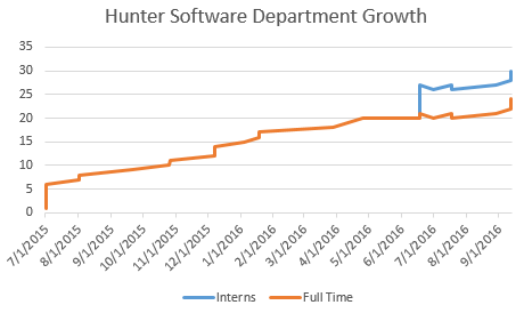

Figure 1. Growth was limited to two people per month

4.2 Environment

Our facilities were another concern we had to address. In our prior space we only had room for two Mobs. In order to accommodate six Mobs, we needed a new facility. This was great because we were able to identify everything about the facility that we wanted to include to make our growth successful. One of the first items that was apparent was wanting everything on wheels, including our 80 inch 4K resolution monitors. This would allow us to rearrange the office at any time. Another change that we made to our new space was the removal of walls to make more of an open area for us to collaborate more. I made sure that no desks assigned were to people, including myself, and instead gave each developer a locker. I set up pairing stations that laptops could dock at in case anything non-Mob related like expense reports and other administrative needed to be worked on. Finally, I wanted to make sure that there was office space for any product owners that would like to be relocated near us interact with the teams more frequently.

4.3 Interviews

As a team, we identified that our interview process needed to accommodate Mob programming as well as technical excellence. As a group, we designed our interview to include questions about test driven development, debugging, legacy code re-factoring, and databases. All of the questions were asked to candidates as if they were in the Mob. Team members were there to aid them, but those team members did not know the solutions to the problems. We had some successes in this process and we retrospected frequently making adjustments to our hiring process based on our successes and failures. There were the few times that we had candidates too nervous to continue the interview with a Mob. We believe that the nervousness came from the fact the candidate was having to program in front of four people, which is unusual. Subsequently, Jason Kearny

wrote a blog about our interview process (http://jason.teamkerney.com/post/a-better-interview) and we began to send every candidate this link to help prepare him or her for the interview.

5. MAINTAINING TECHNICAL EXCELLENCE

Attending conferences and a conference budget were two of our highest priorities when we started as one Mob. Typically, in one year we would go to two or three Agile Open conferences as well as a major conference like the Agile Alliance. We found that going together as a team helped build camaraderie and trust. This is a pretty large budget for individual developers to be attending conferences. There was a negotiation around budget that we went through when I became director of the department. I had asked the organization to keep most of the training budget intact for every new developer that we brought on. This was extremely important in helping develop a highly cohesive team within the timeframe that we wanted to become effective.

Another item we negotiated for when we began to grow was onsite training. I wanted to make sure that the technical excellence of the team was kept to a high standard even with the influx of all the new people. This meant that we needed a large amount of technical training upfront. We decided to do a monthly training for the first year and subsequent years would be quarterly.

In the past, with only one team, a single click manual deployment was all we needed to practice continuous delivery. However, as we grew, we identified the need for rigorous, continuous integration and a continuous integration tool. This need came from the fact that some people knew how to deploy while some did not, and many would forget over time as they worked on new or different projects. Rather than risk team members forgetting how to deploy, we fully automated deployment. This ultimately led to an increase in delivery speed, as well as an increase in quality.

As the teams were growing, we discovered that we had naturally emerging technical leaders, which helped us to define core elements of our coding practices that would continue to grow and evolve over time. One of our Mob Programming heuristics is “everyone on the team should be navigating about an equal amount”. Referencing this heuristic, we determined these technical leaders should be navigating less and teaching more. This action reinforced technical excellence within the team and allowed the senior developers to build up their teammates.

We continue to encourage self-organizing teams so the team composition for each project is based purely on negotiation. This means that if two developers on separate teams agree, they can exchange places on project teams. This is done with the understanding that a Mob will consist of mostly people that can continue to move the project forward while someone new is getting up to speed. Retrospectives in the past have identified that sometimes the team is switching people too quickly and some were switching too slowly. This has been a great way to help regulate the people that have social difficulties with each other since they can switch teams at any time.

The only structure that is imposed on the department around team composition is the idea of a minimum number of developers per project. When initially planning our growth, we had three major development paths or projects we wanted to focus on with the fully staffed teams. As we grew, we asked that a team always end up with eight people on one project team, eight on another, and five on the last. This allows the team members to self-organize without the organization risking a loss of interest by the developers in the project.

With rapid growth, comes drastic change. Our goal is not to prevent that change but instead to steer it toward a positive direction. We were extremely aware of areas in which we needed to pay attention to this fact, especially with culture, quality, and structure.

6. CULTURE

With a large influx of new people, we also identified the need to maintain our culture. We decided to start practicing retrospectives around culture and explicitly design the culture of the team as a group.

In our new emerging culture, one of the things we implemented immediately was a review of our core beliefs. Number one was the idea that we should be treating everyone with kindness, consideration, and respect. With all new employees, we also reviewed the Agile Manifesto, the Software Craftsmanship Manifesto, and the concepts valued by the No Estimates movement. We found that this review worked well and helped to immediately reinforce some of the areas that we had deemed important. As we grew, there was a dip in quality and technical excellence, we also did see some failures as well as successes. We did not expect to maintain our high level of quality the entire time we were growing, so it was important to make the failures evident so we could retrospect back to a high level of technical excellence.

The culture of the different project teams also began to diverge from the beginning. Today we have four distinct groups working on what we would consider projects. Since membership of a team is negotiated, I have noticed there is a personality spectrum that has surfaced within the different teams. The people who are passionate about new technologies have coalesced around what I call the popular projects. Others passionate about the Agile process and maintainability of Legacy Code have organized around the hard-refactoring project. Then there are those that are amiable and willing to take on anything that will help the department who ended up on the important but less appealing project. These cultures have solidified around these projects and over time people have moved between them. When the transition happens, the new member seems to adopt the culture of that team. To facilitate the transition, however, we as a department have identified core working agreements that the entire department agrees on. These working agreements include lofty goals and standards like Gherkin across teams. It is also important to note that the teams are all working in different languages and development stacks, however, Gherkin is the unifying language between teams for clarity of direction and requirements. When someone switches between teams they can always get up to speed quickly by reading the Gherkin and Unit Tests. Finally, we hold department retrospectives about once per month to help develop a unified department culture.

6.1 Flat Department Structure

Another interesting culture shock was the implementation of our flat department structure. As a director I do not play the role of a manager. With a group of twenty-four people in a typical department at Hunter Industries there would be multiple managers. I am ultimately responsible for the success of the department to the rest of the organization; however, I try and distribute as much of the actual management related responsibility into our flat structure. In fact, the only management responsibilities I personally do by myself are related to wages and reviews. We decided to continue with our current situation, which was to distribute the management responsibility to everyone in the team. Any management task, including budgeting and team performance metrics, is handled from within the team. We quickly established a management Kanban board and used that to quickly delegate work. Finally, we created a rotation for people to attend the internal department meetings that occurred on a regular basis. In that implementation, we used a simple round robin technique to allow each person to experience the meetings and contribute in his or her own way. We have also been experimenting with the idea of an impromptu Mob that takes on management tasks instead of having those tasks be done by only one person. So far, the team has been favoring the management Mob instead of a management task Kanban.

The self-awareness of a team of four working together is significantly greater than people working by themselves. I believe that a group can make better decisions event with the most minor choices such as the sentence structure of an email. Ultimately, having somebody either micromanaging or middle managing a team working in this way becomes redundant or wasteful. I feel that my role as the director is to provide everything the team needs to be successful in both a technical and psychological sense. The team can progress quickly, and make good technical and interpersonal decisions without my intervention. I try and influence the system only in ways that support the overall direction of the team.

My biggest challenge in this regard has been our rate of growth. Since the team is growing so quickly, it is hard to guide so many new influences on the system over time. When hiring our last four members, I also made the mistake of letting them all start at once. Having four people start at the same time violated our two people per month rule, which proved to be chaotic, as it was extremely difficult for me to avoid getting directly involved in the team's incorporation of these new people.

I believe my greatest contributions to the team has been around communication of our practices and performance out to the rest of the organization, guiding the team's interpersonal communication, and helping the team develop coaching skills to guide each other to a better outcome. If the organization can communicate well with the team members and the team members can communicate well with each other, then I believe we will have a foundation for continuing to make great things happen.

I started as a developer Mob programming with the team and I still try to continue to employ these skills. I can have a discussion technically or from the business lower level. These skills are also important to develop in every member of the team at least on some level. We have created an environment where every member of the team has some level of technical expertise as well as coaching, managing, and teaching skills.

To communicate well with the rest of the organization, I have created several presentations around what we expect from our different counter parts. I have also created several automated reports around the performance of the team, which helps the organization be more aware of the decisions we make. Examples of

these reports include metrics on releases to production per day and an automated historical Gantt chart that depicts the complete Gherkin scenarios created over time. This has led to a high level of transparency, which in turn has created a high level of trust.

As a technical leader in the department, I do get involved at a technical level with teams, but it is important to note that we try and avoid the HiPPO (Highest Paid Person Opinion) effect. Before I begin Mobbing or participate in an architectural discussion, I explicitly state that everyone should only take my technical advice as recommendations and not a command to execute my version of the plan. I participate in technical discussions as a Mob member and not the director of the department and I have people in my team hold me accountable to these rules.

7. CHALLENGES

It is easy to talk about all of the successes we continue to experience and I would like to also address some of the challenges that we have incurred. With growth, there will always be stress on the system. That stress will warp and manipulate the system in order to accommodate that growth in whatever way necessary.

7.1 New people all the time

The first big challenge has been the continuous growth for about a year. With that growth, we have experienced difficulties primarily around conflicting prior technical experiences. This is both a benefit and a challenge. People with different values and opinions come into the team and have to work with the team to define what their working agreements around technical style will be. These working agreements defined at a department level are guided by our lofty goals.

With each new developer, we have found that they have different technical backgrounds and different experiences with quality and automated testing. Much of the challenge has come from standardizing the level of quality that we wanted to achieve. Behavior Driven Development was not a common practice for us until we had so many people that we needed to do more integration testing. In the past, comprehensive unit testing was all we needed to get to the point where we had zero bugs. It is important to note that we increased the amount of testing in part specifically, to address the difficulty of working with more people in multiple Mobs.

We write BDD tests as a result of product owner input and maintain Gherkin for our automated acceptance tests. Since many more people are on the team, Gherkin has become an outstanding communication tool between members across teams or to new people working on projects. The number of bugs we have seen has increased with our growth and the need for better communication between Mobs. However, our BDD tests have decreased those numbers. We are now going longer and longer periods without reported bugs on the teams that have been incorporating automated acceptance tests into their development.

7.2 Grow and Split

As we grew the Mobs, we did not explicitly set out to separate people and instead used a heuristic for when to form new Mobs. We typically tell everyone that a Mob only is too big when someone in the Mob is not contributing or learning. This means as we grow, we allow the Mob to become larger and larger until there is a key moment in which the team decides to split. We always have one more Mob station available that is needed for the teams to operate independently. This provides an avenue for suggesting that the Mobs split since there is nothing in their way of becoming a separate Mob. This caused us to divide original Mob team into multiple teams to seed culture.

As we divided the teams and brought on new people, our most experienced Mobbers and developers spread amongst the new teams that formed. We found that if we did not get enough of a knowledge transfer before moving, an experienced person would constantly be asked questions until the Mob built up enough of the common knowledge that they could move forward without that person's knowledge. We called this effect, when a group's collective knowledge was not complete before they lost any single team member, "creation of a tribal knowledge silo." We had two solutions to this problem. First was the practice of keeping teams intact and adding the new person before someone else left the team. Second was the incorporation of Gherkin as acceptance tests throughout all our projects. The combination of these two solutions helped us bridge the knowledge gap.

When the merge of the Internal and External product teams happened, we found that there were silos in each team that needed to be distributed across the other teams. We encouraged the start of movement between Mobs because we want to avoid developing silos within the overall group. We wanted to make sure that there are at least a number of people that can handle any particular task that would otherwise be done by

a single developer. A great example of this, was when the teams merged, there was an already an established silo on the team that had to share their knowledge in order for the team to become more effective. This was an effect that we saw both when we started Mobbing in 2011 and when the teams started to grow in 2015. Each time we grew, we found that certain developers had developed deep silos of knowledge around some specific, crucial functions that we determined needed to be distributed across the other members of the team.

Due of the nature of Mob programming, distributing that knowledge was simple and learning the information needed to take care of those tasks was easier as well.

7.3 Interruptions

Another issue that new Mobbing teams typically have to endure is the nature of the interruptions created by people getting used to Mob programming. Examples of these issues are background noise, and side conversations. The Mob programming etiquette that is developed can be accelerated by retrospectives and team building exercises. Eventually the team needs to get to a point where they are all comfortable giving each other honest and candid feedback. Yet, at times it was still an issue as we grew. Today, however, I have seen the same groups of people performing without any issues and it just appears to come with time. When noise levels become too high, the teams can now turn off the lights in the office when the collective groups of Mobbers are too loud. For side conversations, there are taped perimeters around the Mobbing areas where people having a side conversation must move away from.

7.4 Retrospective Saturation

Another area that we struggled with is the issue I like to call "retrospective saturation." This is when you have so many retrospectives that people on the team become sick of them. This effect reduces the value that can be gained out of retrospectives. In the end, we reduced and then increased the number of retrospectives that we had in order to accommodate this. It was made clear quickly that we would want to do more impromptu retrospectives so there was less to talk and time spent at the more general retrospectives. There needs to be some breathing room to allow the teams to get to know each other a bit more before the next retrospective. Now, at times, the teams' retrospect daily which is a large improvement but as a team grows, giving feedback to people you just met is difficult and requires both social and structural development before deep issues can be discussed.

7.5 Bi-Annual Goal setting

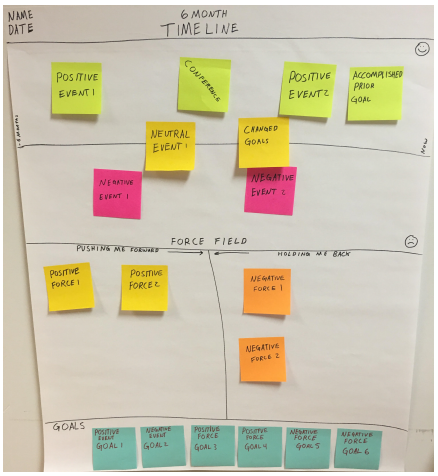

Figure 2. Timeline & Forcefield Retrospective for Bi-Annual Goal Setting

Traditionally, our goal setting and evaluation practice was to come up with a few goals for developers to meet and the manager would evaluate them on their performance in these areas. We decided after retrospection that we wanted to make better use of the biannual goal setting process and the time the company was expecting us to spend on them. We ended up with a process in which each developer helped facilitate another developer in two retrospective formats (see Figure 2). They did a timeline retrospective and a force field diagram, to isolate everything that happened to the developer regarding their career in the last six months. The force field diagram retrospective format is one in which the participant identifies things that are holding them back and moving them forward. The idea is that these are forces pushing a field back and forth. In other words,

the time line yields insights into events that were positive and negative for the developer and the force field retrospective provides insight into every day behaviors or elements of that person's life that are contributing to or taking away from that person's work experience. We identified two things from each section of the retrospectives that we could count as goals and then I, as a director, had one-on-one meetings with each developer. In a later retrospective, a developer said that this was the only time they had ever had a relevant goal be produced from their yearly corporate goals in their entire career.

8. CONCLUSIONS

Overall, growing Mob programming at Hunter Industries has been a wonderful experience. I have learned a great deal about the different ways that Mob programming can aid in scaling Agile at an organization. I believe many problems of scaling were not as much of an issue as they would've otherwise been had we been working as individuals. We began to grow the Mob and when the Mob was too big we split into two Mobs. After reaching six people in a Mob we would split into two Mobs of three. Typically, we normalized at four people per Mob. Since we treat each Mob as a Single Item Work in Process development stream our goal was to be able to reach eight streams of work. We continued the style of growth and split until reaching eight teams.

We kept a limited hiring pace of two people per month, which helped us better integrate new people into the process quickly. Our focus has been on technical and social excellence and our interviews reflect that. We practice continuous integration and continuous delivery; we also train for agile coaching and automated testing within the teams.

In order to maintain a close pulse on the progress of the team, we perform frequent retrospectives and typically consider it a problem if we have gone more than a week without a retrospective. We designed our new offices specifically to foster collaboration and Mobility, especially the requirement that everything in the office had to be on wheels. We have a flat department structure meaning we distribute the management responsibility and self-organize around management related tasks and maintain a management Kanban. We have requested from the organization that our product owners be engaged and always available and we have made it clear to everyone in the organization that we practice no estimates. The metrics that we currently value as a team are: the time since last production release for each project, the time since the last retrospective, and the number of bugs in production, which should always be zero. Finally, as a director I am practicing servant leadership in order to provide the best environment possible for the team to be successful without being commanding in my management practices. Since a team of this size is so new, we can only get better as people in the team form new bonds from their experiences working in Mobs.

I also find that the evolution of our department structure has played a great role in maintaining buy-in from the developers working on the teams. Ultimately, we have found that Mob programming has scaled well for us and we are currently working on projects with few to no bugs. We have had struggles along the way, but our learning sessions and retrospectives have played enough of a role that these problems faded as we progressed.

9. ACKNOWLEDGEMENTS

Hunter Mob, I could not have had this experience without you. Your dedication to experimentation and awesome culture has made great things able to become even better. Carolann Lucian, thank you for dealing with my focus on my career, editing this paper, and being the sounding board for my ideas. Hunter Industries, Marc Kase, Arne Pike, and Greg Hunter I appreciate your support and your unwavering trust in our team and process. I have heavily relied on Jason Kerney, Dexter Baga, Gordon Pu, Aaron Griffith, and William Getz in order to help establish and propagate our culture to our new hires, thank you all for investing so much into our ideas. Thank you Woody Zuill for putting on amazing workshops and spreading the idea of Mob Programming to the world. I really appreciate the time that Rebecca Wirfs-Brock has put into shepherding this experience report it really has been a pleasure.

REFERENCES

[Griffith] Griffith, Aaron, "Mob Programming for the Introverted" Agile 2016 Conference, Atlanta, GA
https://www.agilealliance.org/wp-content/uploads/2016/03/Mob_Programming_for-the-Introverted.pdf
[Kerney] Kerney, Jason, "Mob Programming – My first team" Agile 2015 Conference, Washington D.C.
https://www.agilealliance.org/wp-content/uploads/2015/12/Mob_Programming_-_My_first_team.pdf
[Zuill] Zuill, Woody, "Mob Programming – A Whole Team Approach" Agile 2014 Conference, Orlando, Florida
http://www.agilealliance.org/files/6214/0509/9357/ExperienceReport.2014.Zuill.pdf