Joining the Mob at Clearlink

TORREY POWELL, Clearlink
CRAIG ANSLOW, Victoria University of Wellington

Software engineers historically have largely worked alone and in a vacuum on key projects. This has caused problems with transparency, creates knowledge towers, increases technical debt, and stifles innovation. Mob Programming has given Clearlink solutions to all of those problems and created benefits that have been unforeseen during our first two years of adopting the practice. From our experiences, we outline some best practices that will be beneficial to all those who wish to also adopt this technique.

1. INTRODUCTION

Clearlink—a SYKES company headquartered in Salt Lake City, Utah, serves many of the world's leading brands through intelligent marketing, sales, and data science solutions. In early 2017, Clearlink adopted a new technique called mob programming [1,2] to improve team processes. Almost two years later, Clearlink's entire engineering team has implemented this transformative approach to software development. Other departments and teams outside of software engineering are even catching collaboration fever and are running their own experiments with mobbing.

Mob programming is a concept featuring the driver/navigator model used in pair programming [6], mob programming evolved from a commitment to quality and innovation. Instead of two engineers working side by side, as is customary in pair programming, a larger group of developers, testers, product experts, coding experts, and managers come together on a single project. They communicate and collaborate while working collectively on a product.

The results of the experiment have been undeniable: Software engineers are engaged, the code is clean, and the products are delivered bug-free. All of these benefits result in a major reduction in technical debt and an increase in innovation. Other benefits include: a reduction of knowledge towers, higher technical expertise, and cross-team collaboration.

In the words of Agile coach and programming consultant Woody Zuill, Mob programming is "a whole team approach" where "all the brilliant minds work on the same thing at the same time in the same space on the same computer" [1,2].

2. MOB PROGRAMMING - HOW IT WORKS

At its core, mob programming is a grassroots movement that still hasn't garnered the industry attention it deserves—partly because of its unorthodox nature. It is easy to understand why some software engineers and companies might hesitate to give mob programming a try, and easier still to predict the questions they might ask. Why have three to five developers work on a single task? Can't they accomplish more if they divide the task between them? Why shrink your resource pool? How is this method cost effective? These similar style questions were often aimed at the practice of pair programming [6,7].

While it may seem counterintuitive to have an entire team dedicated to one task, research shows that having multiple developers working on a single task is effective. In 1995, famed software engineer Larry Constantine noted in his book *Constantine on Peopleware* that pair programmers were developing bug-free code more quickly than ever [3].

Author's address: Torrey Powell, Salt Lake City, UT, USA; email: torrey.a.powell@gmail.com
Second author's address: Craig Anslow, Wellington, New Zealand; email: craig@ecs.vuw.ac.nz

A few years later, a seminal study on pair programming conducted at the University of Utah discovered that while it took pair programmers an average of 60% more time to complete an assignment (compared with developers working alone), that number dropped to 15% after an adjustment period [4]. After that adjustment, "all students reported they had overcome the constant urge to grab the mouse or keyboard from their partner's hands." Researchers also found that 85% of students involved in the experiment "indicated a preference for pair programming," and those same adjustments and preferences could apply to mob programming. Knowing this, it helps to think of mob programming as a long-term investment in enhancing the quality of a software product.

For far too long, companies have judged the productivity of a team by only the amount of development time it takes to create a feature. Not taking into account the total lifecycle of the feature. The time to develop, test, and fix or in other words technical debt. Fixing, especially post-release is quite expensive. Mob programming might increase initial development time, especially when people are new to the approach. However, the total development time, including testing and fixing decreases dramatically with the reduction or even elimination of technical debt.

Mob programming empowers individual developers to work more efficiently as a team. When they're stuck, time is not wasted looking for answers. The collaboration of a team effort solves problems faster and progresses the development past any individual hurdles.

Developers working in this way can bond with each other, learn together, grow as a team, and watch massive projects take shape in real time. What's more, mob programming dovetails with agile frameworks and in particular Scrum and XP methodologies many companies now rely on [8]. Sprint planning, sprint reviews, and daily stand-up meetings can all be applied to mob programming if you operate in teams of mobs. Otherwise, these rituals are self-managed and are not even needed.

The value of mob programming helps to accelerate software delivery while keeping software engineers firmly on track makes it indispensable in an agile office environment.

3. IMPLEMENTING A MOB PROGRAMMING MODEL

At Clearlink, our adoption of mob programming was somewhat serendipitous. Our team was collaborating on a project when one of our software engineers suggested we work together to make sure we were all on the same page. Instead of everyone retreating to their individual workstations with plans to regroup later, we set up a big screen TV and took turns driving and navigating through the task.

At the same time, our Marketing Technology Department had a project in the works that was taking longer than expected. Director of Marketing Technology, Nate Wixom, had brought in a consultant to help refine their programming process. This client had heard of mob programming, looked at the way we were working, and pointed out that we'd inadvertently become mob programmers ourselves.

Intrigued and eager to maximize our involvement, Wixom and I decided to investigate the concept further. We attended a small mob programming conference in Boston—the only one of its kind at the time—where we met several developers from Hunter Industries, where mob programming was first conceived in 2011 under the guidance of Woody Zuill [1].

That meeting led us to San Marcos, California, where Hunter Industries is based. We gleaned additional information about mob programming, its methodology, and its benefits. We were already practicing an adapted version of mob programming at Clearlink at that point, but knowing it had a formal name and industry backing—and seeing how it could work when refined to a sophisticated degree— gave us the vision to propel it forward at Clearlink.

3.1 How We Got Buy-In

When we returned from Hunter Industries and based on our new garnered knowledge, Wixom and I went to our Chief Technology Officer, Bruce Westenskow, with a proposal: we would try mob programming for one year to demonstrate the benefits of this approach and prove its value to the company. Twelve months would give us enough time to get our engineers comfortable with the idea, prove the return on investment in the back end, and illustrate the positive impact that mob programming could have on our products.

The developers outside our small test team had not yet been exposed to mob programming. Naturally, they were still unsure and a bit unreceptive at applying it. Our challenge wasn't just to get buy-in from our top executives but also to persuade these software engineers into adopting a collaborative development process. We knew they needed to experience mob programming fully to appreciate the efficiency and to catch the vision

the way we did. So, we went to an electronics store, bought a TV that became our mob programming screen, set up a few desks across from it, and put our engineers to work.

3.2 Making Our Case

The attitude at Clearlink was that we needed to come into the new age and undergo a renaissance of sorts in our technology department. Our stakeholders were open to the idea of mob programming. They were hopeful it would improve our team processes. With a little persuasion and encouragement, our engineers were also willing to try it. However, getting full buy-in wasn't easy. Because mob programming is often broached by IT or by the engineers themselves, it's up to those individuals to convince the executive team that we needed to implement this new process.

Getting managers on board helped, and because we proposed testing the concept for a year to prove its worth, we felt like we had a better chance of converting upper management. It simply wasn't possible for us to fully experience the efficiencies mob programming can create in a matter of weeks or even months. We needed time to build a project from start to finish and time for our developers to grow comfortable with this new model.

After a month or two with four engineers completely working in a mob, which consisted of a daily routine of seven hours in a mob and one hour of personal learning, they reported that they enjoyed mob programming and they realized it was helping them hone their skills and perfect their craft by allowing them to learn and grow.

4. RESULTS

Implementing mob programming wasn't about having a shiny new toy or going outside the box simply for the sake of trying something new. We knew there was tangible research to support this way of working and that the benefits we'd experience would be tangible, too.

The intellectual level of our team has risen exponentially since implementing mob programming. Engineers know new languages and are practicing coding techniques that they have learned from their peers. Our engineers are constantly learning from each other. Their skills are sharper than ever, projects are being completed more quickly, the quality of the end product is superior, and the team has increased its efficiency. Instead of releasing products built in silos and backtracking to fix issues or make improvements, we are able to test new iterations seamlessly and quickly complete products. This allows us to move on to others with confidence and provides freedom to innovate.

Implementing mob programming requires communication, open minds, and patience on the front end. But the outcome is worth the exertion. Over the past two years, our team has completely rewritten our flagship application from the ground up. This application supports our entire sales organization. The application has been solid and effective with very few known issues and nearly zero technical debt.

4.1 What Worked and What Needed to Change

Because of mob programming, we've streamlined our development work to become more productive, produce fewer bugs, and remain more focused on the project at hand. Our new workflow uses the Scrum framework—the product owner works with a team of developers collaborating on a single project. We're seeing less waste, fewer delays, increased communication, enhanced trust among team members, and more knowledge overall.

Prior to adopting mob programming some of our products and projects, which consisted of 3rd party API's and our internal CRM, were being developed by individual engineers in a silo, isolated from the rest of the team. This created knowledge towers that prohibited scalability within the team and knowledge loss when attrition occurred or when engineers went on vacations. While our reliance on a siloed, waterfall project model provided a structured approach to software development, it also led to wasted resources if developers moved in a direction that didn't meet the stakeholders' needs. With so many means of communication and collaboration at our disposal today, the value of working in the same room is sometimes forgotten.

Our use of agile software development, specifically Scrum, combats issues like lack of direction with feedback, iterations, and accountability. But there was a point where we were making so many changes to address those issues that our team became fatigued. With mob programming, they felt better. They responded positively to the social stimulus it provided, the opportunity for learning, and the ability to work without being tied down.

The software engineers also felt a new found freedom. They could go on vacation without taking their laptop. They also became aware that this freedom was also an insurance policy of sorts. We have an engineer

whose wife was expecting twins. She gave birth to them prematurely and they ended up needing to be hospitalized for several weeks. It was very reassuring to him to be able to leave and spend considerable amounts of time away from work and to be with his family. Nate Wixom, his manager at the time, was able to just say "Go, we have you covered." This was a tremendous lesson that all of the team, managers and engineers alike, learned. With mob programming, flexibility is gained. But it is not the only benefit. In just a short period of time, we had completely cross trained the team. Siloed roles could now be filled by various individuals. Liberating is the word that I feel summarizes the feelings of both the engineering team and the business.

We also implemented Hunter's suggestion to have the junior software engineers lead the technical discussions to really get them thinking about the problem that needed solving. They may not have known the right answer, but putting those junior developers in the spotlight gave them an opportunity to write the code that produced a solution. More senior software engineers made suggestions for modifications, allowing the junior developers to observe another way of addressing the problem. In the end, the whole team came up with the best possible solution, which increased team member engagement.

Our junior developers are now more comfortable in the spotlight because of this approach. One of the core characteristics of mob programming that worked for us is that it's a safe and open atmosphere built on respect for where everyone is in their careers. Our senior developers know they're teachers, while the juniors know they're learners. However, there are plenty of instances where the senior developers can also learn from the juniors.

We feel our team has done an excellent job of maintaining this characteristic. When we have brought on new employees, it has been interesting to see how they adapt to our culture, as many aren't used to working with their peers in a mob programming approach or even in a culture of respecting your peers.

There are some aspects of mob programming, though, that we now know we should have handled differently. Among the takeaways we got from Hunter Industries was that we shouldn't immediately convert everyone to a mob way of working—that we should do it over time instead as a gradual approach.

When we looked at our department's makeup early in this transition, it made sense to merge two teams into one. My team was initially part of the Data Science Department, but we combined it with the Development Operators team, including the engineers that worked on our legacy systems, and suddenly, my five developers grew into a group of 12. Instead of rolling out the new approach little by little to give our developers more time to adjust, we ended up jumping in with both feet.

With this jumping in with both feet approach, we learned some valuable lessons very quickly. One lesson that has constantly remained in my thoughts is the experience of one engineer. He was hesitant at first to the thought of working side by side with others all day long. However, he was open to giving it a shot. I placed him on a very mature mob that consisted of mostly senior engineers. After a couple of weeks his response was one of pure excitement for mob programming. He was completely engaged with the product and his mob. Shortly thereafter, we had the need to adjust the mobs. With his excitement and energy, I decided to give him the opportunity to mentor a couple of less experienced engineers. He now found himself as the only senior engineer in the mob and the great opportunity to shape how that mob functioned. He performed at a very high level for the first few weeks but he began to feel fatigued and lost his excitement as he felt he wasn't progressing past the storming phase. He felt like he was carrying the load, solely responsible for completing tasks in a timely fashion, and responsible to teach the engineers that were less experienced. Ultimately he lost his enthusiasm for mob programming and left the team. We quickly learned that the composition of the mob needed to have a balance.

In hindsight, I wish we could have implemented mob programming more slowly, because some of my developers felt unsure about this new method. As is the case in most office environments, we knew some individuals were more introverted and others more comfortable in a mob dynamic where their actions were more visible to their colleagues. Not everyone was content with our new method of developing software, but I met with my team of 12 engineers one-on-one to identify and address their concerns so we could move forward with projects.

5. BEST PRACTICES

Our experience implementing mob programming at Clearlink has led us to many conclusions. The best practices we developed through our experimentation made the integration process as smooth as possible.

5.1 Mob Research

At Clearlink, we made a point of learning as much as we could about mob programming before putting it into practice. Attending conferences, networking, reading mob programming experience reports [9,10,11,12,13,14], and speaking with those who've successfully implemented the approach was crucial because it helped strengthen our argument for implementing the process and prepared us for what lay ahead.

5.2 Teamwork – Whole Team Approach

For mob programming to be effective, each member of the team has to be willing to work together toward that common goal. Ultimately, focusing our team's power on each project produced better outcomes, faster delivery on projects, and a better customer experience overall.

We established a rule that nobody could be on their phones at a mob station, as this was distracting and disheartening to the team; in a mob, everyone has to pull their weight. If someone needs to make a phone call or step away to answer a text message, they temporarily remove themselves and pick up again once they get back.

Likewise, if a developer needs to take a break from the intensity of the experience and reset their mental and social muscles, they're free to do so. We encourage individuals to take frequent breaks to combat burnout and fatigue. With practice, though, handling the process becomes much easier for everyone.

During mob programming sessions, each developer has anywhere from 4 to 15 minutes at the keyboard. For engineers that are new to mobbing this time is shorter, before rotating to another driver so each engineer can contribute. We rotate to ensure each driver and navigator experiences every part of the process. To further build camaraderie and an effective work rhythm, our engineers stay on their teams for at least six months. However, after that time, they can request an assignment change.

5.3 Recruitment - Mob Sessions

Once our mob programming model was in place, we knew it was vital to work closely with our company's recruiter to prepare new hires for the work they could do with us. Our interview process now includes having candidates sit in on an active mob for an hour, ensuring there are no surprises after they're hired.

Without the need to set up a new developer's environment, train them, and get them up to speed on current projects, onboarding new engineers is far quicker and easier. New hires are welcomed into the mob and immediately contribute to the best of their abilities. They learn on the job by watching others and doing their part alongside our most senior engineers.

5.4 Giving Credit

The idea with mob programming is that the focus shifts from giving and taking credit for development work to concentrating on producing better work for the good of the company. That said, individual team members still need to be recognized for their contributions; their job satisfaction and career progression demands this kind of acknowledgment on a regular basis.

Effectively recognizing our developers'' good work was important to us. We hosted internal contests and award competitions to keep our team motivated over time. Sprint retrospective meetings were opportunities to recognize specific achievements, look back on the week, and share what we learned as a group.

6. DISCUSSION

At Clearlink, we've fully adopted mob programming, and we always have a seat available for anyone who wants to join in. Our mob spaces involve more people in each project than before, and everyone brings something unique to the keyboard. It's part of our culture now, and that's only going to continue.

Today, we have one mob member who works remotely, and we're exploring the possibility of adding more. The key will be finding a way for spontaneous conversations—the kind in the breakroom after everyone has walked away from the mob station—to occur even if multiple team members work outside the office.

We're now up to 11 mobs, averaging 3 to 5 people per group. Other departments within Clearlink, including our digital PR/outreach team, have taken notice and incorporated the principles of mob programming techniques into their work.

Our advice to companies interested in pursuing this method of software development is to be persistent. Even if you're given the autonomy you need to set up your first mob station, success will take time—but don't be afraid to experiment with mob programming. Give your developers the opportunity to increase knowledge

while improving the quality of your products. At Clearlink, we wouldn't have it any other way as mob programming has increased productivity and team morale.

7. ACKNOWLEDGEMENTS

I would like to give my sincere appreciation to my team at Clearlink that has been extremely patient with the process of adopting mob programming. We have constantly iterated on our processes and will continue to iterate so we can become the best that we can be. Thank you for your patience and excitement toward change. Likewise, I would like to express gratitude to the senior management at Clearlink for giving me the opportunity of making drastic change and supporting it to the point of making it successful. Everyone at Clearlink has been supportive and excited for the opportunity to become better and mob programming has accomplished that.

I want to give a big thank you to my shepherd, Johanna Rothman, for her knowledge and insights. I could not have completed this report without you.

Lastly my wife Jennifer. Thank you for always being supportive and for giving me extremely valuable feedback and suggestions. I can always count on you.

REFERENCES

[1] Zuill, Woody, "Mob Programming – A Whole Team Approach" Agile 2014. https://www.agilealliance.org/resources/experience-reports/mob-programming-agile2014/
[2] Zuill, Woody, "Mob Programming – a Whole Team Approach" GOTO Copenhagen 2017, https://www.youtube.com/watch?v=SHOVVnRB4h0
[3] Constantine, Larry (1995), "Constantine on Peopleware," Yourdon Press Computing Series
[4] Williams, Laurie, Kessler, Robert, Cunningham, Ward, Jeffries, Ron, "Strengthening the Case for Pair Programming," IEEE Software 2000.
[5] Ashkenas, Ron, "Jack Welch's Approach to Breaking Down Silos Still Works," Harvard Business Review, https://hbr.org/2015/09/jack-welchs-approach-to-breaking-down-silos-still-works
[6] Williams, Laurie, Kessler, Robert, (2002) "Pair Programming Illuminated", Addison-Wesley
[7] Cockburn, Alistair and Williams, Laurie, "The Costs and Benefits of Pair Programming," XP 2000.
[8] Kropp, Martin, Meier, Andreas, Anslow, Craig, and Biddle, Robert. "Satisfaction, Practices, and Influences in Agile Software Development." In Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE), 2018.
[9] Boekhout, Karel. "Mob Programming: Find Fun Faster." In Proceedings of the International Conference on Agile Software Development (XP), 2016.
[10] Wilson, Alexander. "Mob Programming - what works, what doesn't" In Proceedings of the International Conference on Agile Software Development (XP), 2015.
[11] Buchan, Jim, and Pearl, Mark, "Leveraging the Mob Mentality: An Experience Report on Mob Programming." In Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE), 2018.
[12] Kerney, Jason, "Mob Programming – My first team", In Proceedings of Agile 2015.