Creating Quality with Mob Programming

Thomas Desmond

Thomas.Desmond@HunterIndustries.com

Abstract

Mob programming is a software development approach where the whole team works on the same thing, at the same time, in the same space, at the same computer (Zuill 2016). Mob programming is all about collaboration, teamwork, helping others, and developing great software. With it, we get improved communication, consistent exchange of knowledge, enhanced idea generation, and more learning.

In my organization, we mob program on every aspect of our production code. Every line of code, every test, every release is developed using mob programming. Because of the high quality of our software developed with mob programming, we went a year and a half without any bugs reported in our production software. The software we release is developed drastically different because of mob programming.

In this paper, we will explore how the mob programming software development approach produces quality software in my organization.

Biography

Thomas Desmond has been developing software utilizing mob programming for over three years. He is passionate about learning, teaching, and helping others perform their best. Mob programming has become his preferred method for developing software. Thomas has also incorporated mob programming concepts when teaching university courses.

1 Mob Programming Fundamentals

1.1 What is Mob Programming?

"All the brilliant minds working on the same thing, at the same time, in the same space, on the same computer." (Zuill 2016) In my organization, we mob program on every aspect of our production code. Currently, we have nine separate mobs working full time.

So, what does mob programming look like in my organization? We have up to five developers all working together at the same time on the same thing driving the software forward. We do all software development practices together: architecture design, code development, testing, deployment, kanban, prioritization, EVERYTHING. It may help to think of mobbing as a more extreme version of pair programming. Instead of two people working together, we generally have up to five members all working together.

We may be at a single computer, but we are not huddled closely together around one small screen. We have one desktop computer attached to 80-inch monitors with multiple desks surrounding the monitors. We have plenty of space to sit, work, and move around. The image below is an actual mob set-up.

1.2 How did Mob Programming Begin?

"Having an environment where awesome things can happen, pretty much guarantees that awesome things WILL happen." (Zuill 2016)

Mob programming did not start overnight. Mobbing has continually improved over the past seven years, starting slowly and expanding into what it is today.

It began with a teammate that needed help with a feature they were tasked to complete. They did not feel like they would meet the deadline, so they called upon their peers for help. They all got together in a conference room. They collaborated and worked together to come to a solution and did end up meeting their deadline. The team enjoyed working together and continued it. They held a retrospective every day to discuss their progress.

The collaboration, working on the same task and daily retrospectives continued. With each retrospective, they found ways to improve their processes and work better together. Eventually, they were given a permanent space to mob program.

This was the very beginning of mob programming. It has now, seven years later, expanded to nine mobs in my organization. We want to always find ways to turn up the good and mob programming worked well for the team. Mob programming is still evolving today but began because a group of people saw it as a beneficial way to develop software.

1.3 Driver-Navigator Model

When people first think of mob programming, they often think of it as one person at the keyboard writing the code while the others watch. It is not like that at all. We take a much stricter paradigm coined "Driver-Navigator". The person at the keyboard is the driver while everyone else is a navigator. The driver is not allowed to implement their ideas. They are taking the input of the navigators and translating that into code.

Llewellyn Falco expressed the statement that "For an idea to go from your head into the computer, it MUST go through someone else's hands". The navigators describe their ideas in English to the driver, who then translates that into code. If an idea cannot be explained in plain English and understood by the driver, then it would not go into the code.

For the driver-navigator model to work, it is crucial to ensure the driver is never talking through their ideas and coding at the same time. It is the responsibility of everyone on the mob to ensure that this does not happen.

We have found that the driver-navigator model forces people to have to explain and discuss ideas with others. At a minimum, one other person, the driver, has to understand the idea enough to put it into code. The shared understanding of the code and being forced to explain code in English brings out quality code.

The driver-navigator roles are also not static. They change often. Using a timer, we rotate roles between driver and navigator, so everyone gets a chance to present ideas and navigate through the solution. Depending on the team, rotations are done each five to fifteen minutes. The forced rotation gives everyone the chance to navigate and prevents one person from navigating the code endlessly.

While the driver and navigator are essential roles in mob programming, there are many other roles that can be present. For more information on the driver-navigator roles and the other roles that can be seen in a mob, take a look at Mob Programming: The role-playing game (Larsen 2016) or the IBM experience report on mob programming patterns (Keeling 2019).

1.4 Zero Bugs

In my organization, we have a set of lofty goals. The lofty goals are meant to be statements that we would want to be true if we were in the perfect software development environment. One of the lofty goals is to have zero bugs in production.

Zero bugs may seem like an unattainable goal for most teams, but the first mob team at my organization was able to improve the quality of the software being released to the point of having zero reported bugs in production for a year and a half. Having zero bugs was made possible because they were mob programming.

To this day, we still have very few bugs. When a bug is reported, it becomes the top priority for the team. We only return to feature development once the bug has been resolved. We have so few bugs that we do not use a software bug tracking tool. Each project has a physical whiteboard to be used as a kanban board to keep track of tasks. The board is broken into three sections: To Do, Doing, and Done. When a bug is reported it goes immediately into the Doing column. The kanban boards gives us and interested stakeholders visibility into what we are currently doing, what has been completed, and what still needs to be done.

2 Ways Mob Programming Creates Quality

2.1 Communication

The number one benefit of mob programming is communication. We are face to face with our teammates for the entire day. We are always talking to one another. The driver-navigator model forces us to be continually collaborating, working together, debating ideas, etc. The immense amount of communication that occurs in a mobbing environment breeds quality software.

Fewer formal meetings need to occur when mobbing. If we need to discuss something as a team, we are already together. If a product owner wants to see our progress, we are all together ready to give a demo. Mob programming as defined by the Agile Alliance, “Ironically through the use of ongoing, "working meetings” Mob Programming ensures that work, knowledge creation and decision making are joined together thus leading to much better alignment and more effective action.” (Mob Programming)

Questions get answered immediately. If we do not understand why we are doing something in the code, we can stop and get our questions answered. We do not need to struggle through confusing code on our own. We do not have to queue up questions to ask a coworker once they are available. Mobbing allows us to ask for help and clarification from those around us immediately.

2.2 Knowledge Sharing

When you have a group of three or more people all working on the same feature at the same time, knowledge is shared amongst the individuals. We code through the solution together, sharing how and why we made our decisions.

Jason Kerney describes how mob programming benefits from systems like transactive memory. If multiple people share the same experience, they are then able to remember different aspects of that experience. They can recall the information better and put all the pieces together (Kerney 2015).

It is not possible to remember every piece of a solution. However, with mob programming, the shared experience of coding through a solution means we can, as a team, better remember how and why something was done.

We have a shared redundant knowledge of the software among the team. Quality is increased because of the transactive memory and shared experiences in a mob.

2.3 Idea Generation & Development

With mob programming, we are no longer working alone to come up with solutions to our architecture, implementation, and testing problems. We have multiple team members to discuss ideas with face to face immediately. One study came to the following conclusion that mob programming led to a more creative problem-solving environment. They came up with higher quality solutions because of the mobbing environment that they were in (Aune 2018).

Having a group of people to work together with through a problem together also helps to reduce stress. I never feel as if the entire responsibility of finding the right solution lies solely on me. I can leave at the end of the day not worried about how I would take on the next day’s tasks. I know I have my teammates to help tackle the hard problems together.

Quality is heightened when we generate our ideas together as a team. We come up with more creative solutions to our problems and experience less stress around having to solve every problem alone.

2.4 Elevated Learning

When programming in any setting, we are always learning. Learning how to use a new tool, framework, language, or technique. In a mob, we are always sharing what we learn with each other. Knowledge is quickly disseminated among the team.

Karel Boekhout studied a mob programming team and found that the team had grown much faster in development skills and development processes while mob programming, compared to traditional individual development (Boekhout 2016). The group setting of the mob is conducive to learning. We gain the knowledge of the others on the team.

I joined a new project team that used a mobile development technology that I had never used before. Nine months after joining that team, I accepted a position as a lecturer to teach that very same technology at a local university. Going from zero knowledge to teaching a university-level course on the subject would not have been possible for me had I not been mobbing.

We are encouraged to share and encouraged to bring everyone around us up to the same level. The interactions and navigation in mobbing lead to amplified learning. Quality is increased when everyone is learning more and at a fast pace.

3 Mob Programming Things to Consider

3.1 The Work Environment

The mobs in my organization are all collocated together in a single building. We do all of our work on-site and do little to no work remotely. Having the teams together, face to face is an important aspect that has made mob programming successful for us.

It has been studied that there is a correlation between the proximity of teams and the amount of collaboration on the teams. By having our mobs close all the time, we can achieve the highest level of collaboration (Kiesler 2002). Mob programming is all about collaboration and is a big part of how we create quality software.

Some teams perform mob programming remotely; INNOQ is one example. They have put together an excellent resource for how they successfully mob program remotely (Harrer). Remote mobbing may work for some teams, but at my organization, we do almost all our work onsite.

A time-lapse video published to YouTube titled "A Day of Mob Programming 2016" gives an overview of our work environment and what we do every day. (https://www.youtube.com/watch?v=dVqUcNKVbYg&t=1s).

3.2 Mob Team Size

Having an infinite number of team members could bring quality to a whole new level. However, we know this is impossible, and there must be a maximum number of people on a mob team to prevent benefits from declining or stagnating.

As of writing this paper, few formal scientific studies have been conducted revolving around mob programming, especially team size. Yet, there has been a large number of studies surrounding pair programming. Woody Zuill breaks down mob programming, using data from pair programming studies, to provide estimates of the correlation between team sizes, costs, and the quality of work produced.

It is hard to provide an exact number that makes up the perfect mob. However, based on the work completed by Zuill and my empirical evidence, mobs should be made of four or five members. Zuill's extrapolations found diminishing returns with team sizes any larger than six. For a full breakdown of Zuill's calculations, see section 3.10 titled A Simple Cost Analysis of Mob Programming in his book "Mob Programming A Whole Team Approach" (Zuill 2016). Zuill goes into details of costs effectiveness, expected quality metrics, and effects of the mob team size.

3.3 Developer Fatigue

Mob programming requires a lot of focus. Working with a group of people and trying to focus on development tasks all day can be very draining. One study found that mob programming, "strengthens the team, but can also be draining on energy and mood because of the intense focus required". (Aune 2018)

In an experience report by Aaron Griffith, he talks about how mob programming can be draining and over stimulating especially for introverts. Being among a group of people, and having little to no alone time can be demanding (Griffith 2016).

Having dedicated breaks and time to complete personal tasks can help alleviate fatigue. In my organization, we mob for up to 7 hours a day. We then have one hour of learning time a day that can be done individually or in a group. This learning time can be the perfect opportunity to be alone, learning something and recharging.

Mob programming is still something that takes some getting used to. In order to keep the quality high and consistent, we need to ensure we do not overwork ourselves.

3.4 Team Psychological Safety

For mob programming to be successful, teams need to be psychologically safe. Team members must feel comfortable to challenge one another, be able to share constructive feedback, and trust each other. Just being kind to one another is not enough. Feeling comfortable enough to give candid feedback and have a healthy debate is vital to creating quality software.

In a study completed by Google, they found that psychological safety was the number one factor in how successful their teams were. They described psychological safety as “Team members feel safe to take risks and be vulnerable in front of each other” (Rozovsky 2015). Psychological safety is not about just being polite, but instead being able to be vulnerable with one another. Google has proved it is what leads to truly outstanding teams.

Psychological safety can be built up over time by framing work as a learning problem, being able to acknowledge fallibility, and modeling curiosity (Edmonson 2014). In order to create quality software, we need to feel safe with one another.

4 Conclusion

In my organization, we have found mob programming to be a successful methodology for developing quality software. The nine mobs work full time every day on production code because of the results we have experienced.

The quality that comes with increased communication, spreading of knowledge, creative idea generation, and heightened learning has led us to mob on all aspects of our production software. Mob programming is very different from traditional programming and has its obstacles like the work environment, developer fatigue, and requiring psychological safety. However, when mob programming is done well, it can be very conducive to creating high-quality software.

If you are interested in learning more about mob programming or getting started with mobbing yourself, Woody Zuill’s book, Mob Programming A Whole Team Approach available on LeanPub is a great first resource. It goes deeper into what mobbing is, how to start mobbing on your teams, and the problems mobbing tries to alleviate.

References

Aune, Ole Kristian & Echtermeyer, Christian & Sørensen, Elias. (2018). Mob Programming: A Qualitative Study from the Perspective of a Development Team. https://www.researchgate.net/publication/328150167_Mob_Programming_A_Qualitative_Study_from_the_Perspective_of_a_Development_Team.

Boekhout K. (2016) Mob Programming: Find Fun Faster. In: Sharp H., Hall T. (eds) Agile Processes, in Software Engineering, and Extreme Programming. XP 2016. Lecture Notes in Business Information Processing, vol 251. Springer, Cham

Edmondson, Amy. Building a Psychologically Safe Workplace. May 4, 2014. https://www.youtube.com/watch?v=LhoLuui9gX8.

Falco, Llewellyn. "Llewellyn's Strong-style Pairing." June 30, 2014. https://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html.

Griffith, Aaron. "Mob Programming for the Introverted." Agile2016 Conference, August 2016. https://www.agilealliance.org/resources/experience-reports/mob-programming-for-the-introverted/.

Harrer, S., Christ, J., & Huber, M. (n.d.). Remote Mob Programming. https://www.remotemobprogramming.org/

Keeling, Michael, and Joe Runde. "Harvesting Mob Programming Patterns: Observing How We Work." Agile2019 Conference, August 2019. https://www.agilealliance.org/resources/experience-reports/harvesting-mob-programming-patterns-observing-how-we-work/.

Kerney, Jason. 2015. "Mob Programming - My First Team." Agile2015 Conference, August 2015. https://www.agilealliance.org/resources/experience-reports/mob-programming-my-first-team/

Kiesler, Sara. Distributed Work. MIT Press, 2002. https://mitpress.mit.edu/books/distributed-work

Larsen, Willem. "Mob Programming: The Role Playing Game." https://github.com/willemlarsen/mobprogrammingrpg.

"Mob Programming." https://www.agilealliance.org/glossary/mob-programming.

Rozovsky, Julia. "The Five Keys to a Successful Google Team", November 17, 2015. https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/.

Zuill, Woody. "Answering a Few Questions." Mob Programming. November 2012. https://mobprogramming.org/answering-a-few-questions/.

Zuill, Woody, and Kevin Meadows. Mob Programming A Whole Team Approach, October 29, 2016. https://leanpub.com/mobprogramming.