Learning to Experiment

CHRIS LUCIAN, Hunter Industries, Inc.

The best organizational changes in my career have happened as a result of emergencies. Support for the process of experimentation only seems to become enabled whenever the status quo has completely broken down. In those moments of crisis rebuilding with better has been possible, but why do we wait? In this report I share our experiences with experimentation at Hunter and encourage you to develop a culture of continuous improvement and continuous experimentation that will get you to where you want to be without having to be in a crisis mode.

1. INTRODUCTION

The best organizational changes in my career have happened as a result of emergencies. Support for the process of experimentation seems to become enabled whenever the status quo has completely broken down. In each emergency experience, I have seen estimation to be the first practice to go when prior to that event it was the hardest to contest. In those moments of crisis rebuilding with better has been possible, but why do we wait? I am Chris Lucian, part of the team who started Mob Programming at Hunter Industries in 2011.

Today I am the director of a new software development department and we continue to utilize Mob Programming as well as many other practices. This report is about my journey through software development and specifically the experiences I have had with the process of software estimation. We are currently a department of 27 with 6 full time mobs; each team does not estimate software but instead practices pure Kanban and delivers vertical slices of software daily. Hunter is not the first place I had dropped the practice of software estimation for a project. Back in 2007 I was working in an Environmental Chemistry and Military Contracting organization that had a more traditional waterfall environment. During that time, we created a Gantt chart of how each project would go until one day there was an emergency that prompted us to do things slightly differently. I have come a long way and experienced many iterative steps through my software development journey. Let's start with where we are today and work our way back discussing practices and insights we discovered on the way.

2. ORGANIZATIONAL CHANGE, PROCESS INNOVATION, AND SAFETY

When influencing organizational change, paying attention to the motivations and needs of everyone affected by the change you want to have realized is important. You need to ensure the safety of everyone involved. The organization must also realize that stagnation in its process is extremely unsafe, as the software industry changes so must the practices of the organization to avoid a disruptor taking its place. Developers also need to feel safe running experiments and failing as well. Without the failures we won't have the learning moments that will carry us to a new level of understanding. The best way so far that I have seen to encourage this experimentation while keeping things safe at all levels of the organization has been to budget for that potential failure and have an avenue to pull any new learning into the production work.

In practice, for us this involves a combination of 7 hours of dedicated learning session time per week, retrospectives with action items, and budgeting for software instead of estimating it. The effect of these practices has been awesome. The team has the safety to try radical new ideas during their learning sessions. These ideas can find their way into the retrospectives where the team can decide whether to apply them to their production work. This does not mean that the team does not experiment within their regular work time; it just means there is an additional budgeted time for some of the more extreme ideas to be worked on before flowing into the system. Experiments with high risk of failure are important to maximize learning. However, in

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

the production work, the business may not feel safe running those experiments. This is especially the case when there needs to be visible progress on the project during periods where there is also experimentation. In order to allow these new ideas to flow regularly into the production work, whether it is a process experiment or a technical one, the team needs to not have a product progress estimate hanging over their heads.

2.1 Inception of The Mob

Between the inception of Mob Programming [Zuill] and today I could list our hundreds to thousands of experiments. I want to reiterate that we did not start with these desirable practices in place. Instead we had found our way here through an iterative journey. I think the best example of this is the inception or the catalyst for Mob Programming. Back in 2011, when I had joined the team, it was just in the infancy of an agile transformation that was being guided by Woody Zuill who had been there 3 months before I was hired. Knowing of Woody from the local user groups scene I was expecting things like pair programming, test driven development, and refactoring to clean code.

Instead I was shocked to realize that I had been hired onto a team that was divided up into cubes specially designed to prevent people from sitting next to each other at the keyboard. With my first project at Hunter I was handed a large spec document that contained all sorts of details that were ultimately irrelevant or needed to be updated in some way. Woody was just starting to seed the refactoring and test driven development practices but the team had not adopted the ideas yet. There was a Kanban for each person's work, retrospectives, and learning session. There was also an agreement that the team would work toward better but we wouldn't provide estimates for the things we were doing. Instead we would deliver value continuously.

These new goals were a key paradigm shift. Continuous delivery was incentivized to the team for the first time. The new target was to become deployable. Given that there were no estimates being provided, the conversation was shifted from, "How can we get it out by X?" to, "How could we deploy one new feature right now?" Working on my first project at Hunter I had a great responsive product owner who could get on the phone with me at any time and work through everything that came up. I also had great support from our Ops team. This enabled me to be able to release the first iteration of the application to internal customers faster than anyone on the team thought possible. There was now a situation where all of the developers could learn from each other if they were just given the opportunity.

Our learning sessions were structured and Woody defined the content. He required you to attend the meeting but you did not need to participate if you were uncomfortable. The earliest learning sessions I was a part of were a code kata Randori where the entire team would implement Tic, Tac, Toe using Test Driven Development. We would throw away the solution each time and start over, refining the learning around TDD. There were people on the team who would not touch the keyboard and had never been required to program in front of other people before.

During this time there was noticeable tension for us to stick to the estimates that had been made before Woody hired on with Hunter. Specifically, the most senior developer in the team had a requirement to deploy an application at the end of the year due to tax restrictions. This developer had been working on the application for a little over a year and was having a hard time dealing with the tools they were using. Eventually it was obvious to this developer they would not be done by the end of the year. This was the moment Mob Programming had its catalyst. The developer asked the team for help and went so far as to say, "Can we all get into a conference room and work on it together like we do in the learning sessions?"

How did we continue working this way?

Once we finished that first day of work on the project we retrospected and got formal support to keep going. The team was safe to continue because of the support the business gave us in the current agile initiative. Eventually, we retrospected further on the topic, "How do we make a good thing better?" rather than, "How do we go back to the way it was before?" Once again, the catalyst was an emergency like all the other big changes to which I have referred.

2.2 Mob Growth

Between 2011 and 2015 we practiced continuous delivery but not automated continuous integration. When our team merged with another into a new department [Luci] we decided to begin experimenting with a few different automated continuous integration and continuous delivery tools and ultimately landed on ThoughtWorks GOCD (https://www.gocd.org/). This experiment also stemmed from the desire for consistency, this time across multiple developers. It is important to note that we as a single mob felt capable of handling all the work we did because we had a sort of collective memory. However, as we grew into multiple mobs, we had people moving between mobs and there was a slow information loss about how to build and

deploy the projects. We felt that as a department there needed to be a way to stay consistent in regards to deployments. This experiment was intended to reduce cross mob silos around deployment. It ended up being successful and very quickly become a ubiquitous practice across the mobs.

We later had another divergence in deployment practices where one team decided they needed to hire a DevOps specialist to seed a DevOps mob for our global scale IOT product. Today all of our mobs maintain their own CI except for this IOT project where there is a dedicated DevOps mob because of its significant DevOps requirements. Developers obviously have a lot of agency for change within the team, making decisions about whether to hire someone into a newly created role.

3. SOME EXPERIMENTS THAT WORKED AND SOME THAT FAILED

A culture of experimentation is one of the things that has helped us become successful over the last 6 years. We have viewed it as a red flag when things stop evolving. We utilize conferences and learning sessions as a funnel for new ideas. These new ideas become experiments within the learning sessions and in every day work. These experiments stop stagnation that teams experience when deadlines and catch up are always looming. Investing time and money into these experiments has been extremely valuable. The following sections touch on a few of the experiments that we have attempted on our process, technical approach, and interpersonal interactions.

3.1 Experiment: Management Kanban

The autonomy given to the team is expressed to an even greater extent in how we handle management tasks. We have had a flat department structure ever since we merged the teams. One thing we needed to handle in this structure was how to spread the management tasks throughout the group. An experiment to gain further consistency of responsiveness for the teams was to distribute management related tasks throughout the department using a Kanban board. Prior to this experiment, management tasks were given to dedicated manager—me. But knowing that we would be growing to a team of 27, we had to make some decisions about how we would handle an increased amount of management related work without hiring more managers.

Our idea for an experiment was that every management task, no matter where it comes from, ends up on a Kanban board. And tasks can be assigned to anyone in the group round-robin style as people become available.

This worked very well for a while but we experienced a strange form of Parkinson's Law (https://en.wikipedia.org/wiki/Parkinson%27s_law). I observed that Management Tasks began to come in more frequently simply because we had such a large capacity to complete then. This of course slowed down development and reduced the consistency of the team. Eventually, our Management Kanban was a sort of failed experiment. But it worked at first, allowing the team to grow extremely quickly and handle all of the necessary overhead work. In the end, the idea was a great temporary measure that eventually lost its usefulness. We have since moved on to another experiment, call "Home Owners Associations," which are optional management-related groups that have a retrospective cycle built into the way they work.

At the time of this report, this current experiment seems successful. However, I want to highlight that is as equally important for practices to have a way to gracefully exit the system as it is to be able to perform new experiments. All too often I see teams fall into the trap of adopting new practices and never letting go of any old ones. A great example of this is the idea that we will continue to iterate on our Lofty goals as a form of culture shaping.

3.2 Experiment: Lofty Goals

Our Lofty goals had been formalized more directly as part of the initial department merge. These are not only goals but also a form of working agreement for the team to reference when things are noticeably going in a direction outside of the parameters of what we as a department define as success. Early on these were loosely defined. But as we hired more people, we found that these values were becoming diluted. We then ratified our new set of lofty goals and socialized them within the group (see Figure 1). As we grew, the culture of the team was modified by the growth. While many of our goals stayed the same, some fell away or were replaced. Today the goals fit the department and the team can influence them over time. We did not make it hard to change these, but we do want to gain consensus from the department when they change.

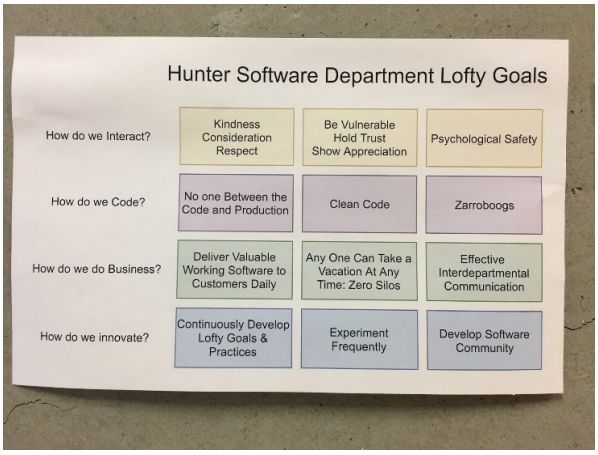

Figure 1. Our Lofty Goals as of today.

3.3 Experiment: Flyway Database Continuous Delivery

Another great place for experimentation is for tasks that could be automated but are not. We had been doing continuous integration and delivery for our code but still had to do all database schema updates by a script run by hand. When we found a deficiency in our continuous integration pipeline we took time to experiment on how to improve our process.

Again, dedicated learning sessions were used to great effect. The team learned about many tools eventually settling on Flyway (https://flywaydb.org/). This tool allowed us to deploy database schema updates to each of our environments through script versioning. Developers set up a folder that contains versioned scripts. As a team member checks in a database script into this folder it will run the script on the environments before deployment happens, enabling the team to do continuous delivery for database schemas.

There is an iterative nature to our experiments and process improvements. Until we as team had so many products to deliver with a lot of associated domain knowledge, there was not much need for an automated continuous integration system. Once we did, then we tried even more experiments.

3.4 Experiment: TSQLT Stored Procedure Unit Tests

In some cases, some sort of problem surfaces the need for experimentation. For example, we only began to see the danger in not having tests for our stored procedures after we had implemented Flyway database continuous delivery. Recently, we bad a bug related to a stored procedure. Reflecting on how to prevent this in the future, the team identified a need for learning about creating automated tests against the stored procedures. Originally the thought was to use integration tests for coverage of stored procedures' functionality. However the run time of integration tests is much greater than it takes to run unit tests.

So the team decided to learn about TSQLT, a unit testing framework for Microsoft SQL. After some learning sessions on the technology, the team decided to introduce TSQLT into the department's production work to great success. This included introducing this into the teams' continuous integration system. These stored procedure unit tests almost immediately provided value and enabled us to confirm this experiment was successful.

Today, stored procedures created by the team are tested with automated tests before they are integrated into the rest of the system.

3.5 Experiment: Automated Acceptance Tests

When the team began to grow, so did the domain knowledge. In fact it grew so much that we began to have a hard time staying effective without more detailed documentation. So, we decided to try behavior driven development (BDD). After looking through some tools and having training long ago on Gherkin, the language used with Cucumber and SpecFlow, we had a retrospective and decided to try to require it for all new features. This outcome of this experiment was interesting because the results were mixed for a long time.

When we implemented SpecFlow in our C# projects we found that this was a great way to communicate business logic to new developers. But we also found that in many other environments it was unreliable. For example, in our Node stack project we found many of the BDD tools to be flakey and there were additional toolsets needed to stabilize them. This difficulty almost made us abandon automated acceptance tests as a way

to prevent flawed releases going to production. But despite these challenges, today, we are still using them and they provide us a lot of value. The automated acceptance tests run on our continuous integration server that was put together roughly at the same time as the inception of the new software development department. Having a continuous integration system is a great way to immediately realize value from automated acceptance tests.

3.6 Learning Guilds And Other Failed Experiments

I wanted share some of the obvious and larger failures we had to show that experiments do not always work out. When I started at Hunter Industries in 2011 the team was already doing 7 hours a week of learning sessions. Through our retrospectives early in 2017, we had modified the learning session requirements so many times that gradually over time we reduced the transparency of how time was spent in these sessions. Also, we were no longer required to do them as a Mob.

After retrospection on the learning sessions, we defined a new experiment around a concept of learning guilds. The learning guild was based around the notion of "Guild Librarians" curating learning material around a particular topic. After completing "Quests" related to that learning goal the guild librarian would hand out badges. Developers in the department could sign up for these quests or become guild librarians. Our theory was that this would provide more visibility on team members learning sessions to everyone in the group. Unfortunately, the whole process felt heavy handed and took too much administration. Ultimately, we determined this guild experiment had failed but in the process trying it, we learned a lot about the place of gamification in our environment.

Today we rely on our bi-annual goal setting process and quarterly lightning talks to help guide people's learning sessions. We make no attempt to monitor or have visibility into any individual's personal learning. Their learning sessions do have obvious results because we continue to integrate new practices into the group, such as continuous integration and stored procedure unit testing based on individual team members' learning.

3.7 Experiment: Product Owner Interactions

For the first 4 years at Hunter we had product owners that were available to us daily or weekly which we found greatly contributed to our success. As we scaled the team, we began to have new product owners from other parts of the business. And we had 4 different interaction patterns with these products for our 4 big projects. First, we had a team who had their product owner stationed in the same building, available to them almost any time for a face-to-face conversation. Second, we had a product owner available for discussion daily with the team about the product. Third, the product owner was available for weekly in-person visits with phone access at any time. Finally, we had a team with a product owner available to them once a month. This opportunity of observing four different patterns of product owner-team interactions was a great way to see different situations play out at the same time.

This product owner interaction experiment occurred organically. I think it's important to notice an opportunity to experiment and observe what happens you are unwittingly forced into alternate ways of working. The results are just as valid as consciously setting up an experience.

We observed a breakdown in the discussions the team was having when they went long periods of time without product owner interactions. Specifically, the team began to make assumptions about what the product owner wanted when there were only vague descriptions of requirements or unexpected edge cases being discussed. This grew even to 6-hour discussions about what needed to be done and how to accomplish these assumptions which ultimately ended up being wrong. We determined that weekly face to face visits and phone calls was the most delay that could be tolerated in the system without the flow of the team falling apart. We saw that product owner availability is a key factor in the ability to be agile in software development and decision-making. As a result of this opportunity and our observations, today we make it a priority to involve the product owner in person at least weekly with a request they be available whenever possible by phone.

Because of an unexpected, but welcome experiment, we are now able to have the buy-in from the business when discussing needs of the team from their product owners.

4. INFLUENCE AND BUY-IN

So, how can we influence change and experimentation without needing an emergency to force us to rethink the way we work?

Early in my career I saw how the software estimation eliminated process innovation entirely from a team. Once a team falls slightly behind on their work panic begins to set in. Estimates start to be treated as deadlines

or another department begins to make assumptions about those estimates. Once the environment loses all slack, practices begin to break down, technical debt begins to pile up, and experimentation with process or technology becomes an afterthought. However influencing decisions around whether to estimate or not feels very unsafe to most potential change agents I speak to. In fact, when I have been fortunate enough to work without estimating during my career, an emergency has always been the catalyst for removing estimations from our process.

After having the freedom to not estimate in my work for 4 years at Hunter, I was asked to become the director of a new software department. This new department gained the attention of nearly every part of the organization. In the spirit of transparency, we disclosed all of our processes during this transition, including our practice of not estimating. Needless to say, there was a lot of skepticism for that practice in particular, as well as for some of our other practices. So even though my new boss was bought in, other parts of the organization with less visibility into our group were more skeptical. That skepticism was an indication to me that parts of the organization felt unsafe about the recent changes. I also had learned in the past that a lack of safety discourages process innovation.

Realizing this, it then became a high priority for me to help the organization feel safer in regards to our practices. Since this sort of widespread influencing and persuasion was new to me, I began a long series of experiments with many failures. Again, in order to successfully achieve my goals, I had to feel safe too. This meant that I minimized the impact of my failures until I had refined our processes.

In our software we use unit tests and automated acceptance tests in order to create safety when we build and check in code. Before unit testing, companies I worked at would create that sense of safety by using staggered releases. If we broke something only 10% of our users would see the impact. Throughout my career this has been how I experimented with making organizational change safe for myself. When communicating about the things people were skeptical about, I sought out people who were least convinced and not directly involved in the outcomes. I asked them what their questions were and how I could prepare presentations to better communicate my ideas.

Communicating my ideas failed many times. A lot of my failures to communicate were related to assumptions I was making about what others thought software development was as a practice. Over time I refined my analogies and descriptions. I also allowed myself to be influenced by my collaborators and colleagues, since we usually ended up in many interesting philosophical discussions about manufacturing versus software development. At the end of these discussions I found more people to reveal my ideas and get their feedback until I was ready to present the information to the entire organization. I was able to socialize the ideas of minimum viable product releases, vertical slicing of tasks, and budgeting for software instead of estimating software tasks this way.

Starting down this path was the result of responding to an emergency; however it could very much have not been that way. If I were to start over and try to influence a new organization to work toward better processes, I would try a very similar approach. I would get to know the people around me and their motivations. Then I would start floating ideas and asking for feedback on presentations. And I would begin to refine my point of view using new information I gained until I had something good enough to use to gain agency for the change I want to create. In most cases I believe my actions would lead to further expanding a culture of experimentation so that change would be possible at even more levels of the organization and outside of software development.

5. SOCIALIZING DIFFICULT-TO-ACCEPT IDEAS

One of the big friction points between the team and the business has the lack of estimation. Socializing these ideas to the rest of the business has been difficult until recently. Specifically, I had developed a new strategy to help others in the organization develop an understanding of how we should move forward from a financial predictability perspective. Earlier this year we did a reorg due to a new acquisition. For the second time in 2 years I reported to a new person. This meant I would be spending time discussing our practices again and getting my new boss up to speed. In looking for a faster way to do this I realized that finding articles from sources he trusts was a great way to help get him on board with what we are doing quickly.

At this time I found this article about budgeting for agile projects instead of estimation from Harvard Business Review [Madden]. This was a publication that many of our executives read that supported some of the concepts we were experimenting with. Along with supporting published articles I resolved to learn more about how the business could interpret our work and budget appropriately for the work we do. With further investigation I discovered that as an organization we green light new projects based on a Net Present Value (NPV) associated with a specific hardware project. The calculation for NPV uses a time series summation of

expected value minus the expected costs. If we included software as a budgeted activity for the life of the product, we would see for the entire 6 years that we created a valid NPV for a project from the release of the Minimum Viable Product (MVP) throughout its lifetime of feature adds. In order to work toward fitting into this model, I proposed a new way of looking at the cost of our development process. Today we are budgeting for development to products with 4.5-person mobs. Meaning that a product that requires 2 mobs had 9 people budgeted to creating new features for that product. Experiments in this area have all been around education of the business in the dangers of software estimation, which in the past always seemed to fall short.

I was fortunate enough to be able to iterate on the description of why we don't do estimates in the group to each one of my bosses. Each time I was successful. But finally, this time I think we have ended up with a lower level of skepticism throughout the business for this idea. Again, I view my explanations as a form of experimentation. The prior iteration of this experiment was to show data around our successes when we removed the constraints that estimation imposed on us. A failure in that experiment was not having enough supporting material from outside sources that the executives widely trusted. Of course, experiments with such an ingrained process as estimation would also require consistency, frequency of delivery, and transparency.

6. WHAT'S NEXT?

We are constantly performing new experiments on our process. An experiment I am looking to start is having an event bus for the department where all source control, continuous integration, and mob timer activity are published and allowing anyone in the department to create micro services to listen to that bus. The hypothesis is that by providing such an unrestricted and transparent resource this will lead teams to create useful and interesting new ways to use this real-time stream of data further promoting measurable experimentation. Today we are running many retrospective-driven experiments on each mob. As a department we are experimenting with new management structures handled by the developers on the team. As an organization we are experimenting with better ways to integrate software development into the rest of the manufacturing business. Some of these experiments succeed and some fail, but the amount we learn from these experiments is amazing. We continue to deliver functionality but we do not avoid changing any of our processes. It is important to not be afraid of failure, but instead find ways to understand the new information a failure delivers. Recently we have learned a lot from the frequency of our product owner's interactions with our teams which leads us to the most recent big shift we have experienced.

7. MAKING EXPERIMENTATION AN ORGANIZATIONAL NORM

Today we have organizational buy-in for the practices we use from all levels of the company. When going to Open Space conferences I have frequently spoken to many coaches and developers who all say they wish they could convince people in their organization to accept a particular practice or process. These questions regularly get me to think back to how we got the point of buy-in. We were able to try big experiments because of emergencies. But we were able to keep the ones that worked through transparency, results, and predictability. However not the kind of predictability people usually think of when talking about software estimation. In fact, for me, supporting the continued utilization of a process has always come from transparency, results, and consistency, and not accurate software estimates.

Don't come away from this paper with "practice envy." Instead I hope to convince you that developing a culture of continuous improvement and continuous experimentation will get you to where you want to be quickly. Technology has not out-paced our team. Instead, from 2011 to 2017 we have gone from being out of date to comfortably ahead of the curve. You too can get here, but don't try to do it all at once. Instead make one small change at a time and evaluate the effect on your performance. Develop the culture of experimentation and you too can be pushing your organization forward technically and interpersonally.

One logical argument for experimentation is as follows: Companies have learned to fear disruptive innovation from the outside. From an executive level there are many reasons to keep an eye on what the industry is doing and when a new company appears and takes control of your market it is already too late. Agility and experimentation are ways to outmaneuver these new organizations by finding ways to disrupt their own business before the competition can disrupt yours. Complacency, on the other hand, is how companies become replaced by these new innovators. At any level of an organization a team member can champion a culture of experimentation by making this argument clear. Investment in experimentation creates safety for both the consumer and developer. Typically the most skeptical people can be motivated to act on this idea when there is data to support the idea in the marketplace.

Sometimes when appealing to the rational mind it might become apparent that your new idea is not being accepted. This is when the iterative subconscious influence becomes important. Whenever I have been given the power to change the organization I was already following the processes of the existing system. This prevented the people I was trying to influence from disregarding my opinion as disruptive. Once in a position of being accepted, incorporating micro experiments has been an easy way to show these people that this was still our process but just a little better. Being inclusive, and sharing credit for the change is also important. Without the help and trust of the people around you it is very difficult to have a larger impact on the system overall. Respecting the current process while iteratively suggesting small improvements can help you gain the trust and small successes for moving forward with a bigger initiative.

Finally, approaching people in order of the diffusion of innovation theory [Rogers] has been a way for me to accelerate adoption of new ideas. I begin the conversation about topics like innovation in process or technology with "innovators" and "early adopters." Then, I work on proving out an idea to the point where the "early majority" would accept it. Finally, I work to convince the "late majority" and "laggards." As each group is socialized to the idea, they have support from the other groups. Eventually the new idea becomes widespread.

8. CONCLUSIONS

Why is it so unsafe in the current environment of today to experiment? What can we do to make our teams continuous innovators and build our own way of doing things well? At Hunter Industries we have found that a combination of not estimating, continuous delivery, learning sessions, and continuous experimentation through frequent retrospectives, we have arrived at a place that allows us to continuously refine our process. When I invite people to spend a day with us sharing our practices I find that the conversation usually goes from, "How do you do this one practice," to, "How did you get the initial buy-in to try any of these practices?" My response to such a question is, "How can we afford to not have the agency to try new things?" The end value of all of the processes that we have retrospected toward greatly outweighs the cost of all of the failures we have had iterating towards where we are today.

Furthermore, we will continue experimenting. We are not perfect today and we never will be. But we can always be investing in improvement in order to keep the team operating quickly with the buy-in from the organization they need to continue to improve and be effective. Let's all take a look at our processes today and ask how safe is it to change something? How easy is it to change a practice? If there is rigidity in the practices, can retrospectives around experimentation help? Lastly, why do we always need an emergency to get us trying new things? If you are in a stable team that is not changing anything, don't wait until the ceiling is crashing down on you to try something new. Even if you have it perfect, through the process of experimentation, you should be able to confirm that. Chances are, however, that everyone has something to learn.

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, Gene Smith, and Greg Hunter I appreciate your support and your unwavering trust in our team and process. I especially appreciate all of the people who were skeptical but also willing to talk to me about the possibilities. I appreciate all of the conference goers who have discussed these ideas with me and ultimately helped me refine these ideas. Thank you Woody Zuill for establishing "No Estimates" as a process at Hunter creating that initial space needed to get movement in the right direction. I really appreciate the time and influence that Rebecca Wirfs-Brock has put into shepherding this experience report it really has been a pleasure. Finally thanks to everyone who will use this as inspiration in the future to create cultures of experimentation and innovation. Without you we cannot make a safe and growth oriented industry.

REFERENCES

[Luci] Lucian, Chris, "Growing the Mob" Agile 2017 Conference, Orlando, Florida
https://www.agilealliance.org/wp-content/uploads/2017/02/GrowingTheMob.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
[Madden] Madden, Debbie, "Your Agile Project Needs a Budget, Not an Estimate" Harvard Business Review
https://hbr.org/2014/12/your-agile-project-needs-a-budget-not-an-estimate
[Rogers] Rogers, Everett M., *Diffusion of innovations*. New York, Free Press of Glencoe 1962