Zero to Metrics-Driven Hero

COURTNEY SHAR, RxSavings Solutions

I will take you on a journey with a new team that went from being unable to even plan a sprint in sixty minutes to a metrics-driven planning machine that estimates with confidence and delivers on promises. I hope that by sharing our journey from Zero to Hero, you gain ideas and inspiration you can take with you and make your own heroic team.

1. INTRODUCTION

I will begin by going over the first years of a particularly unique team. No one was experienced on this team when it was created, and they were expected to quickly become experts in several areas. I will talk through what the team was like before I was their PM and the struggles they went through. This will walk through how the team went from being unable to keep their commitments for their quarterly Big Room Planning, also known as Program Increment planning [1], and their transformation. I will demonstrate how, as their next project manager with 7 years of experience and an agile champion, took them through the four stages in the Agile process. This went from organizing the flow of work to sophisticated forecasting models. In addition, you will see how the team learned to bring metrics into each part of their own development lifecycle, including how to measure success with their consumers.

2. BACKGROUND

Like any good story of a hero's journey, we must start at the very beginning. The team lead, Shawn, had joined the company in 2012 as a Software Engineer and ended up on a backend services team where he spent the first five years of his career. He enjoyed working on this team, and after five years you can imagine that he was an expert in how the team operated, including how they practiced Agile methodologies. He was then approached to become a team lead for a new team. He started thinking about what he liked about the team he was on and how he could bring that experience to the new team; however, he also started thinking about the things on his team that he wished were different. He was excited and motivated to make this team great. The next day, he decided to take the opportunity and in July of 2017 the team was born.

On day one, the so-called team was Shawn in a brand-new role and his manager Amber. It wasn't long after, however, that they were given two software engineers and a project manager, all three of whom were brand-new. They began work on a couple of small projects, creating some JIRA tickets along the way. They decided on two-week sprints and scheduled their planning/retrospective meetings. Shawn didn't give too much thought to any of these decisions; instead, he carried forward what had been done on his previous team. Everything seemed like it was going well until they reached the first planning meeting. There he learned that as no two projects or teams are the same, what will work for his previous team wouldn't necessarily work for his new team.

There are two major differences between Shawn's team and his previous team I'd like to discuss. The first difference was the code. All of the other teams in our organization were what I would consider typical teams—they own code and concepts where they have become technical experts in their area. They have a high familiarity with the projects they work on, and don't have to branch out into different concepts too often. This team, on the other hand, was created to be more of a floating team. Instead of being a floating individual, the team was used as a collective loan. Instead of owning their own concepts, they would primarily be working on loan to other teams and making changes in that team's code—code they didn't own and were not very familiar with. This meant that the team needed to operate in a very collaborative manner, and venture into unfamiliar concepts on a regular basis. This situation would lead us to implement some unique processes that would help us in this situation.

Courtney Shar, RxSavings Solutions; email: courtneymshar@gmail.com

The second major unique challenge for the team was how they fell in the company's organization. They belonged in one organization, but were completing projects for another. Their organizational alignment was where the asks went to; however, the funding for the team came from outside. This is because the work aligned well with the organization they were in, but the requests for work came from the funded one. The challenge was that they were responsible to both to follow their different processes. The development processes needed to follow their own policies, but needed to align projects and planning with the other part of the company. The latter was very difficult for the team to get used to.

3. THE FIRST PLANNING MEETING

When they walked into the planning meeting, they didn't do it as a team. Shawn walked into the meeting with the expectation that things would go the same way they did in planning meetings on his previous team. His manager, Amber, also walked in assuming it would be the same as on her other team. The two new engineers came in not knowing what to expect, and their new project manager came in prepared to lead the meeting based on what he thought from his own observations on other teams. As a result of the lack of preparation, things started falling apart almost immediately. When they went to look at the stories, they had to pause to discuss whether or not to do story pointing. Once they had decided to do story pointing, they had to figure out how. Would Shawn or Amber participate? What does a '3' mean? They didn't know how to move forward. Eventually, they decided to try pointing, which wasn't successful. Upon further investigation, the team's stories were poorly made. They were vague and broad, such as a story that simply said "Code". Then, instead of pointing a story, they were now spending time figuring out what the scope of the story was, what was our "acceptance criteria", and how to story point when it was certain to take many iterations. They ended up abandoning story pointing, and because the stories were so vague and broad, any attempt to slot the stories failed too.

As you may guess, there were additional issues. The project manager was new, and was on a new team, resulting in a tug-of-war between three different needs—the need for Shawn and Amber to help coach him as he grows into his role, the need for him to be a recognized leader on the team by leading the agile ceremonies—including the planning meeting, and the need for the team to sort out all of the issues they were running into. The result was that they did a mixture of trying to satisfy all three; but in the end they didn't get anywhere. For example, the team needed to have their own ground rules for retrospectives. Such as focusing on what you can change and not on the negative. While "rant" sessions can be helpful for the team to feel listened to, the need to have actionable items outweighs this. In addition, Shawn believed that the project retrospective questions that he used with his previous team would be successful as they were copied from an already successful team. As they began to work through the questions, the team didn't understand them all. They hadn't written them, and were stuck in translation. Also, not all of the questions applied to the team since they were going through the motions. So, after all that, they decided to end the meeting, think through things, and regroup at a later time.

3.1 The Aftermath

In the aftermath of the initial planning meeting, Shawn first began giving the processes the attention they deserved. It was time to stop blindly carrying forward what the previous teams had done and to start thinking of what was needed. There was no perfect template. Shawn was by no means an expert at Agile development methodologies, but did recognize some things that needed changing, such as story breakdowns. It was then that he took the team's very first step towards greatness. He took the current projects and re-organized the tasks into much smaller bite-sized pieces that could be completed in a single two-week iteration, and showed these smaller tasks to the team as an example of how to organize things moving forward. It wasn't a change he was able to make easily, as he had spent years working with tasks that were vague and broad, and had grown to accept that as an appropriate way to track projects. Then, the positive results began to show.

In the second planning meeting, the team came in knowing that they were going to point the stories, and that they were now broken down into more manageable tasks, so they began the first story pointing session. It was rough, but this time around it was possible. They were able to understand each story's scope and give something more than a wild guess as to its complexity. Now, they had the typical difficulties for a new team when story pointing—coming to a common understanding of what a particular number of points means, as well as, not being afraid to disagree with other team members and voice a unique opinion. These difficulties would prove to naturally smooth out over time as they all continued to story point together as trust within the team grew.

Another step the team took in the right direction was to take the Agile Bootcamp course as a team, where I, coincidentally, was the teacher. This helped them to level set on the Agile processes, giving all of the new members an idea of how things should ideally be done. After this, the whole team was able to understand where they were in our processes and where we wanted to be, and could help contribute to minimizing that gap. The team had started making some progress towards functioning well and in an Agile way. Things were still rough as they were learning; however, they were story pointing, breaking out tasks, and retrospecting. They started to get into something resembling a successful rhythm for their first two months together, until their first Big Room Planning session.

Their first Big Room Planning session was stressful and confusing. As a new team joining an organization that had been planning for a while, they were missing context that was key to getting value from the session. Instead of understanding why they were going through this process, they found themselves scrambling to check all of the boxes, with no time to understand why. The result was two days in a meeting room struggling to pull together some sort of educated guess as to what could be accomplished in the next quarter to be communicated out to the organization for commitments. They survived their first Big Room Planning session with a plan for the next quarter of work that they were a little confident in, but it was clear they were a long way from where they needed—and wanted—to be. What they would eventually realize was, all they had been doing so far was trying to fit themselves into their processes, without understanding why they were doing these things and if they even made sense for the team. Unfortunately, no one on the team had the knowledge and experience to really understand this yet, or where they needed to go from here. They were stuck, and needed help. They needed an Agile Hero!

4. ZERO TO HERO AND THE IMPLEMENTATION EVOLUTION

I was pulled aside by my manager and asked to come in and look into the team. We had realized that a new PM with a brand new team wasn't the best idea. At the time, I already had five agile teams so I had thought it would be a quick fix as other teams had been previously. I did a one-week observation of the team and immediately came up with quick fixes. However, I noticed that with a team that constantly had a changing scope, we needed to start pulling numbers. I looked into both ActionableAgile [2] and KanbanSim [3] with the intention of getting the best of both. While doing so I took them through 4 different phases.

In phase 1, the team focused on establishing Work In Progress (WIP) limits, ensuring the board visualizes the workflow by removing the blocked status, and ensured that each column had a clear definition of done before moving to the next status. This helps with context switching by limiting WIP and ensuring quality and prioritizations are maintained. For this team, we removed the blocked status and instead added a flag to blocked items as they could become blocked anywhere in the workflow. We also made sure that we had clear done criteria; such as you have to have testing in your code before you put it out for review. The team has the column on their board turn red when we exceed the WIP limits put in place. When this happens, we discuss with the team during standup to evaluate what happened. Our team's limits are based on 1 piece of work per engineer and we used them on "In Progress" and when the code was "In Review" for work that was peer reviewed by the team. Our WIP limits on these two columns were 1.5 x the # of engineers on the team. This was determined by the idea that no engineer should be focusing on more than two stories. If they wanted to do more, they needed to work as a team to keep the stories moving through the board to completion in order to keep the board from having any "red".

In phase 2, we began measuring and using metrics in earnest. Metrics increase transparency and help the team to think of ways to improve. Since the team was more open to this when we were at risk for deadlines, we could focus on ideas for improvement. Metrics generated a safe space for the team to experiment, as feedback was seen as a gift. Specifically, we needed to focus on measuring and understanding the team's cycle time and throughput. Measuring the data wasn't enough; we needed to be discussing what we measured in our retrospectives. Outliers such as tasks that took more than ten days needed to be discussed, and if our cycle time was getting higher we needed to discuss why that was, and how to improve it. We used Actionable Agile but this can be done in other tools, as well. The cycle time scatterplot is a graph representation of all of our stories and how long it took for them to close. For our team, we decided that any story taking longer than ten days needed to be discussed during our retrospective, so when preparing for that meeting we check the data for stories that fall in this category. Cumulative flow diagrams were used to monitor the queue of the team. This is particularly helpful when practicing Lean, where there is a focus on keeping Work in Progress items and cycle time at a minimum. For this report, our team focused on the average amount of time all the stories during a specific time span spent in each state. When looking at the data, our team focused the most on our time spent

"In Review", as that was a status that we found we had a tendency to get stuck in for too long. Also, we had the most control over this state as the dependencies were always within our team. With the aging work in progress graph, we can see how the team is doing throughout the project and see if there are any stories that go over the threshold goal. The team had a goal to finish all stories under ten days. With this chart, we would review all the stories that are close to the ten-day threshold and be sure to discuss them before they become an issue.

In phase 3, the team focused on prediction models for forecasting and deriving estimates with Monte Carlo simulation. Monte Carlo simulation is named after the city in Monaco, where the primary attractions are casinos that have games of chance. Using Monte Carlo with your teams by giving a probability of risk. It runs thousands of simulations as to when you would finish a certain number of stories by confidence score on a specific date. Within your teams you may decide that you'll have a goal with a 75% chance that you'll finish a milestone by the deadline. To your executive, you'll communicate with 85% chance. Then for a client, you'll work off of a 95% chance date that you'll finish on time. This is all due to how comfortable you feel about communicating risk acceptance to your stakeholders (team, executive, client). This relies on dedicated and stable teams, which requires little turnover. You will also need to have a history of cycle times in order to run the simulation. This is often thousands of simulations requiring many data points. This means that the larger and more accurate the data set, the better your forecast. You'll also need cycle time for each status transition as your board is laid out like we discussed in phase 1. Understanding what is in your backlog means that you'll have the project broken down or the idea of how many stories to expect. Also, that the stories are broken down to approximately similar sizes. Having the large cycle time history will take care of natural variations in size. For the first Monte Carlo simulation we would run a "How Many" simulation to talk to the team about how many stories we would be able to complete until the next retrospective to make sure everyone was on the same page. We also looked at a "When" version of Monte Carlo to see if we were still on track for our end of quarter or release commitments. This was pulled every retrospective so that if the team needed to stay late, we discovered this sooner and could stay a few hours late instead of several days.

In phase 4, we introduced a new tool to the team. We started looking at what we can track and making the simulation more intelligent as we learned more about the team. What the Monte Carlo tool before could not show me was if I should be worried earlier in the project. For example, during December there are always many engineers taking vacation or traveling. My previous simulations will only tell me how we've been doing so far and not that I need to account for having fewer engineers for an entire month, which would put our commits at risk, before even starting the project. KanbanSim from Focused Objective handles this. For Shawn's team I focused on scope creep, as there were specific issues around what was assumed when starting a project increasing inspection and adaptation for the team. I then start to gather numbers around how much scope creep we take in on average and use it for running my simulation, which ActionableAgile's Monte Carlo simulation did not run through. I'm also able to calculate any vacations or holidays that come up and take those out of the simulation. By using these tools, I was able to find targeted areas for improvement.

4.1 Zero to hero and the retrospective

In addition to the normal bi-weekly retrospective, I worked with the team on two others. We first focused on a project retrospective where I would ask the team questions and they would rate themselves on a scale of 1-10, which then ended with a grade. As the team was constantly working with outside organizations, we came up with polling the teams we worked with every quarter. By doing this we were able to recognize ways to improve things for future projects and fix any issues that we had for long-standing projects, such as those that we knew we would be working on for more than a quarter. Here were the themes that we came up with for projects:

1. Project Page: Did we update the project page in a timely fashion?
2. Pro-active Data Analysis: Did we use real-life numbers (metrics) to drive project scope and feature decisions?
3. Collaborative Engagement with Other Team: Did we design, develop and test collaboratively?
4. Frequent Deployment and Early Integration: Did we deploy frequently and integrate early?
5. Planning and Execution: Did we plan and execute the project well and without last minute all-hands-on-deck effort?
6. Agile Ceremonies: Were our Agile Ceremonies well run and effective?
7. Passivity Enforced: Did we design for passivity/Did we design without clients needing a downtime?
8. Validate Passivity: Did we validate passivity before deployment? Did we validate that clients will not need a downtime?

9. Released on Time: Did we meet our release milestone?
10. No Known Issues: Did we resolve all identified functional issues prior to release?
11. Tech Debt Introduced: Did we tie up all loose ends after the project was "finished"?
12. Technical Debt Resolved: Did we resolve any previously existing technical debt?
13. Well Documented: Did we document project scope, leftovers, requirements, hazards, reference pages, concept pages, schema documentation, and diagrams?
14. No Negative Impact on the System: Did we measure performance impact on the system and create no harm?
15. Feature Enabled on Time: Did we build a product that was good enough to make available to end users right away?
16. Time Management: Did we use our time effectively?
17. Meetings: Did we invite all the appropriate stakeholders to the meetings?
18. No Break/Fix Cycle Post-Deployment: Did we avoid the need for a break/fix cycle for issues uncovered by early adopters?
19. Does What User Wants (Early Adoption): Did we identify and implement success metrics to track early adoption? Are they showing success?

Here were the items that we came up with for the quarterly survey sent out to the teams that we worked with:

1. Unresolved Issues: Did we leave any functional issues unresolved prior to release?
2. Meeting Deadlines: Did we meet our release milestones?
3. Response: How well did our team respond to your requests for changes or updates in a timely manner?
4. Process: How well did we document project scope, requirements, design, diagrams, and discussions? Were scrum / sync meetings held regularly? How effective were scrum / sync meetings? Did we plan and execute the project well and without an all-hands-on-deck effort? What other feedback do you have for the project's process? How well did our team follow your documented development processes?
5. Kickoff: How well was project scope communicated / understood? How well were project roles communicated / understood? What type of collaboration occurred (developed functionality that my team will own long-term/both developed functionality that needed to integrate together)?
6. Implementation: How well did our team design for passivity? Did we leave any loose ends after the project was "finished", or create any technical debt for you? Did you have adequate time to give attention to this project? How well did we deploy frequently and integrate early and often?
7. Feedback Loop: What other feedback do you have about the collaboration effort on this project? Should we schedule a retrospective meeting to discuss this project?

5. RESULTS

By making the retrospectives metric-driven, we were able to start measuring and putting action plans based on feedback and collaboration. By doing this, we were able to move the team towards becoming lean by focusing on turnaround time and amount of work completed. Our standups turned into updates and daily planning. The team changed to story-pointing everything at a “3”, as that was determined to be the smallest amount of work that we could finish in two week iterations. We used this breakdown to continue to have the story-pointing discussions that we had while using scrum, but we turned the talking points into focusing on how we can make our stories even smaller. Then we looked into how we were doing our estimates. Since we didn’t own the code base, we often had to depend on the team we were working with to do the estimating. Of course, there was a learning curve or things that they thought were implied. We started to measure how many new stories and tasks were logged outside of our initial estimate to come up with an average of scope creep or learning curve stories logged per quarter that we then used to make our overall project estimation even better.

The thing that we learned while going through these changes, is that our team was prone to scope creep. Instead of talking about why we weren’t estimating correctly, we shifted to measuring how much scope creep we would have every quarter. We then used that to work through the average and kept that in mind when making commitments. As we were constantly working in other teams’ code bases, we needed to recognize that making estimates wouldn’t be accurate if we weren’t familiar with the subject matter. By catering retrospectives to results, we included our numbers and discussed them with the team so that they could have full visibility, as the quarter went on, and not be surprised by anything that we were tracking.

By including the engineers on the team in coming up with a retrospective format, we were able to get better feedback and cut-down on the amount of stories that were > 10 days. This helped to build a safe space for the team to talk about issues that were going on with them constantly contributing feedback to make the team better. By ensuring there was a safe space for idea generation, we were able to get more buy-in to any experimentation that we tried as a team. Overall, I think we did a good job!

6. WHAT WE LEARNED

Use metrics to drive your decisions when planning sprints and in Big Room Planning. But first, find the metrics that your team needs to use to estimate. As we said before, our team is prone to scope creep, so we track our scope creep stories and add the percentage of the average scope creep stories from our previous quarters to our estimates for future quarters. Then, be persistent in your pursuit of becoming an Agile hero. Not everything that you try will work, and not everything that works for the teams around you will work for your team. Last, remember to ensure that you and your team understand why you are following the processes that you are, or else you will risk not receiving value from them.

7. ACKNOWLEDGEMENTS

I wish to thank Shawn Scrivner and Amber Beerends for allowing me to experiment with their team. I also wish to thank my Shepherd, Krish Srinivasan, for all of the feedback throughout the process. I wish to thank my cat, Pepper Jack, for being cute enough to distract me from working. Finally, I wish to thank my husband, Jason, for making sure I remain sane at work and at home.

REFERENCES

[1] Scaled Agile Framework, webpage: https://www.scaledagileframework.com/pi-planning/
[2] ActionableAgile website: https://actionableagile.com/
[3] KanbanSim http://focusedobjective.com/kanbansim-and-scrumsim-v1-1/