Retrospectives: 548 Days to Fix One Line of Code

RYAN LATTA

A year and a half of retrospectives where the same problem evaded a solution. We changed course. We tuned our retrospectives to focus on experiments and staying singularly focused on the problem. The problem was one line of code.

1. INTRODUCTION

Flaky automated tests plague teams all over the world—as do ineffective retrospectives. I was on a team that suffered under flaky automated tests and a year-and-a-half of retrospectives that failed to improve them.

Every retrospective we brought up our automated tests as a problem, but for a year and a half, we couldn't fix them. It wasn't until I pushed for some changes that we eventually saw a path forward. We changed our retrospectives, got the insight we needed, and made the right changes to get past our problem.

This was a hard path though. I was an engineer and still discovering what it meant to affect change. My approach alienated those around me and jeopardized the results that we all wanted so badly. In the end, we navigated that and found our path forward.

This is a story about changing the way we viewed and executed our retrospectives to finally fix the one line of code that plagued us for a year-and-a-half. That one line removed the inconsistencies in our tests, led to better releases with less production defects, and caused the team to reconsider how they would use retrospectives in the future.

2. BACKGROUND

Today I work as a consultant that helps build high-performing software teams. At the time of this story, I was an engineer that was hired, in part, to help my employer achieve their goals of test automation.

Two colleagues and I wound up on the market at the same time, and a local company was more than eager to hire us. The company was looking for talent and in particular talent with experience with test automation. The three of us joined with the hopes that we would usher in their future of automated testing.

The company I was hired into was around six years old and had grown to around 120 employees. In many ways it still embraced the cavalier attitude of a startup. We had an interesting mixture of technologies and products, but at the heart of it was a massive monolithic web server built in Java. This single server was responsible for the entirety of the business running, and was built to be fast. Within its tangled code were the numerous pivots the business had and the countless features that were barnacles within its codebase.

The three of us began to teach the techniques and tools of test automation to the rest of the company. We focused primarily on acceptance level testing using Cucumber and Ruby, but also spent time updating the unit level tests as well. It took some time, but within a few months, everyone was writing tests. Unfortunately, they didn't pass.

We were also fortunate to have a great agile leader who was our director and Scrum Master. I considered him my role-model and I'd spend all the time I could with him. I'd sometimes spend days ruminating on what he'd say before I'd realize the wisdom in it. Agility was his job, retrospectives his responsibility, yet as an engineer, I sought to change our retrospectives.

Ryan Latta, 803 Roaring Springs Dr, Allen, TX, 75002 email: latta.ryan@gmail.com
@recursivefaults https://ryanlatta.com

If you've been in a position where you wanted to affect change well outside of your job, you know the struggle that I endured to change our retrospectives. I was foolish, lucky, and it worked.

3. STARTING OFF

We began by establishing a few tests using the tools and techniques that we had mastered in our previous job. This was broadly accepted, but there were some resistance and skepticism. Some of the quality assurance people were unsure what this automation would mean for their career. One even quit as they thought automation was flawed as a strategy. Several senior engineers were concerned that the tools we chose were inadequate to the task as they were different from the core technologies in use elsewhere. This was a Java shop, and we were introducing Ruby.

We worked quickly to make our case and show that we could build tests quickly that gave confidence. Thankfully the three of us were hired in to set the direction, so we had support from management to push forward. Even though we lost one person, and several engineers were concerned about the new non-java technologies, everyone was willing to try.

Our task then turned to teach people how to write automated tests. This started slowly at first, but engineers were quick to pick up the new technology. Quality assurance struggled a bit more as they had always relied on manual testing, but we pulled them in to help engineers understand what to test. As the team wrote tests we saw a problem emerge. The tests failed inconsistently.

We held retrospectives every sprint, and in every one of them, we'd be asked to identify what went well, what didn't, and what we wanted to change. "Flaky automated tests," made its first appearance in what would become the next year-and-a-half. In our retrospectives we'd generate ideas and choose the top 3-5 actions. The team decided that the resolution to those flaky tests was that they hired the three of us as experts, so we would fix the issue soon enough.

3.1 After Three Months

Three months later and after numerous retrospectives, the flaky tests still made its regular appearance. In retrospective we saw that while now everyone was developing their skills and learning the tools, it makes sense that things are imperfect. We therefore needed more time for everyone to become skilled at testing before we could expect the tests to be stable.

3.2 After Six Months In

Six months in the attitude shifted. In our retrospectives, the flaky automated test issue would make its appearance as always, but the team would avoid the topic. We'd vote on other things to talk about. The team was tired of trying to fix the tests. By now, the team knew how to write tests and things should be better than they are. The team had tried relying on us as experts, thinking the database was a problem, networking was inconsistent, shared systems caused problems, and that the technologies themselves were buggy. This was a problem that the team had no more energy to give and so silently it was ignored.

Throughout these six months, even as our tests continued to fail we had some interesting side-effects. We operated with a definition of done that required that we write automated tests and that all the tests pass. These two items created an interesting challenge for the team to navigate. How do we consider our work done if the tests pass inconsistently? Well, the team would watch the testing dashboard. Every so often the tests would pass. As soon as it did, everyone would claim victory and consider their work done. Then the tests could go back to failing. The other interesting side effect was that we didn't stop shipping to production or stop developing new features. We just didn't have much confidence in our testing to show that we were shipping high quality. This led to a lot of production support. Our team leads took that as their responsibility and after six months their entire job day and night was fielding support issues.

3.3 After One Year

Near my year anniversary I attended the Lean Agile DC conference and during the closing keynote by Jabe Bloom, my mind was blown. I was introduced to several concepts from lean manufacturing and that sent me on a path to learn as much as I could as quickly as I could. Through hours of reading books like, *The Toyota Way*, *This is Lean, The Toyota Kata, and, The Lean Startup*, I arrived at several conclusions. First, we were attempting to fix too many things at once and that we should focus on one problem until it was solved. Also, in our retrospectives, we were creating too many actions and we'd benefit from one single action that had whole-

team focus. Also that our disposition to arrive at an action would be better replaced by an experiment or a question to answer. Assuming we had the answer was getting us nowhere.

I didn't know what to do with them, but I was sure that I was on to something. My Scrum Master was also the director and my role-model. He was brilliant and made everything seem effortless. I'd find any excuse I could to pick his brain on any topic related to teams and agility. Often I'd finally realize his point several days after our conversation. We'd spar regularly on certain points, and I considered myself truly, fortunately, to be near him.

Normally, I'd bring something to him, we'd debate, and ultimately he'd have some wisdom that would help me see that I wasn't right. When I brought my conclusions to him this time, he had to think about it.

I brought my conclusions to my Scrum Master. I wish I could say this was a conversation where I was consulting my role-model and seeking advice. In truth, I brought these learnings to him as a demand that he do something with them in our retrospectives. I saw him as the path to make the changes that I wanted and brought my request to him. I didn't care what he was dealing with or what his thoughts were or his opinions. I wanted him to say yes to my request and deliver.

Three more months passed.

3.4 After One Year, Three Months

These three months were agony for me. I was aware enough to realize that in making demands of my director and Scrum Master that I had jeopardized what I wanted at the same time. If I checked in with him I may accidentally make my request too much of a burden compared to everything else and that would be the end of it. I had to wait and hope that something would change. My demands and lack of empathy put everything at risk.

A year and three months have passed and we walked into our retrospective. We were prepared to put our sticky notes about what went well, what didn't, and what we'd change. Instead, we were greeted with a picture of a sailboat. Our Scrum Master went on to explain, "Imagine we are this sailboat out at sea. The wind in our sails that push us forward are the things that are going well for us. The anchor beneath the waves are the things holding us back." We identified the things that worked well for us and the things that held us back. He had us prioritize those anchoring items and our flaky tests emerged as the biggest issue that we had. This was the issue that for months now we were ignoring in our retrospectives and now was in the spotlight.

Then my Scrum Master asked us to vote on three to five action items. I looked at him incredulously. He looked back at me confused. In my requests to him, I had brought up that we needed to focus on one single experiment instead of three to five actions.

I stepped into the conversation. Throughout this year and a half, the team had split into camps that had theories about our tests. One camp believed that the problem was in that we were using unproven technology in our tests which were the source of the instability. Another camp believed that it was due to all the tests running against a single shared system. Truthfully I was in this second camp, but I took a neutral position in this conversation.

I wanted each group to consider the assumptions behind their approach. For each group, I proceeded to ask a series of questions to tease out the assumptions. For the camp that believed the tech was bad, I asked how they'd know that in re-writing in a new tech would result in anything other than moving the problem over to the new technology. For the camp that believed it was a database issue, how would they know that the issue wasn't in the test code? While both camps believed they were right I asked each a question, "Would you bet your job on it?"

Asking this question is highly confrontational and I advise against using it unless you're sure people trust you. I developed this question after reading some material on confidence. Asking this question forced everyone to consider how confident they were in their solution. The answers to the question revealed that they were much less certain than they let on. This question, while effective has the potential to destroy the trust you have in your relationships by questioning the validity of their employment.

From here I asked what we could do to find out what the problem actually is. The team was quick to point out that we could track the problems and see if a pattern emerged. We decided to have our testing tools track all failures in a database and build a small web page to visualize the failures.

3.5 After One Year, Six Months

Over the next three months, our retrospectives stayed singularly focused on the problem of flaky automated tests. In each retrospective, we'd review what where we were with gathering data and what our next steps should be. It was during this time the team and I experienced something that troubled us.

Even though our retrospective had only one action for us to take, time and time again, we would come into our retrospectives to report that we not only didn't make progress, but we didn't take any action. It turns out that as a team we were really bad at staying focused on that one item. We let any and all other work take priority and it wasn't until several retrospectives passed that the message hit home for us. We have to make improvement a priority or nothing changes. We had to choose to act on our single improvement instead of moving on to the next task or story. We had to choose to improve instead of doing more work. Here we were, a team of smart people who were incapable of taking action that would lead us out of our problem.

Finally, a day came and someone tapped me on the shoulder. He asked me to come to look at the results of the data collection. By now we had collected data for just a few weeks, but the data couldn't have been more clear. Almost every single problem was focused on one tiny bit of shared test code. We looked at the code that our data pointed to and our jaws dropped. We instantly saw the problem in the code. I remember my co-worker asked, "Should I get the team together to show them this?" My reaction was simple, "Fix it now, we can talk about this forever!" He made the change, pushed it, and we watched as all of our tests suddenly turned green and stayed green.

There was no celebration yet, nor was their applause. There was dread. Normally when the tests passed more than once it meant something truly catastrophic happened. The team was beginning to wonder what critical part of the system went so bad that it messed this up. We told them how we found the problem and fixed it.

We stopped work and celebrated.

Everyone looked at the data and the change in the code. There was no debate or second guesses. No other alternative existed. It was painfully obvious to the entire team that this was, in fact, the core problem and we did, in fact, fix it.

From that day forward the team looked to me for advice on retrospectives. I didn't stay long for unrelated reasons. I remember during the retrospective we had after we fixed this problem the question on everyone's mind was, "Why did it take so long to fix that one line?" I offered the team this, "Maybe improving is a muscle and it's one that isn't very strong with us yet. It took us three months this time around, I wonder if we've grown enough to tackle the next problem sooner."

The team liked that thought, but I wouldn't be around to see what impact it had. I put my notice in shortly after. I wanted to be a Scrum Master myself, and while I was surrounded by great leaders where I could have learned a lot, there was no opportunity for me to transition away from development.

4. WHAT WE LEARNED

Throughout this experience, there were numerous things that I learned that I'd like to share. These lessons have re-taught themselves to me over and over in the years since this story.

4.1 Stay focused on a single problem until it's fixed

Previous retrospectives we'd start with a clean board each time. A new set of ideas and problems. This ability to clean the slate allowed us, in time, to censor our one key problem out of the picture. Staying focused on one problem holds the team to a clear purpose of improvement.

4.2 Try 1 action item instead of 3-5

Similar to the first learning, reducing the number of improvements was key to our success. In our early retrospectives, we'd leave with a list of actions. We'd rarely complete any of them, and we could easily reconcile that as other people didn't finish theirs either. When we reduced our efforts to one there was nowhere to hide. We had to look at ourselves and our inability to do one single action. That stark reflection eventually spurred us to action for the first time.

4.3 Different retrospective activities bring out different things

Our early retrospective format was the common three-column, what went well, what didn't, and what would you change. While simple and at times effective it brought only certain items out in certain ways. Eventually,

we were able to exploit this format to avoid talking about issues. It wasn't until we did the Anchors and Engines retrospective that we saw that the issue that hurt us the most was also the one we were avoiding.

4.4 Questions and hypothesis are great action items

It is easy for a team to believe they have the solution to any problem. It's far rarer for that to be true. By acknowledging this truth and focusing our efforts on learning and testing what assumptions we have we can unlock our abilities to improve. Today I often frame action items to be a hypothesis where they must answer three questions: What will we do, how will we know it worked, and when can we check? These create a testable solution that doesn't leave the team too much room to be lost in the land of solutions.

4.5 Empathy matters

When I ravenously learned what I could about lean manufacturing after attending the conference I took my conclusions to my Scrum Master. I never understood his world, problems, or needs. I made my demands of him and put everything at risk. I became a nuisance and risked nagging him to the point of taking no action. Later, when I dismantled the team's ideas for how to fix the problem I asked confrontational questions. I was inexperienced and didn't fully understand how this question would feel. Thankfully I worked with people who were better than me and worked through that to find a way forward. Know the lives and worlds of the people you work with. Work with them through empathy. Be their partners and you can avoid so many of the mistakes I made.

4.6 If the problem is known enough the solution becomes obvious

When we came up with our ideas of how to fix things we were guessing. Our guesses were full of assumptions that we weren't going to verify. It wasn't until we acknowledged those assumptions that we realized we didn't really understand the problem. It was only after we gathered data that we saw the truth. Everyone was wrong, and there was one line of code that needed our attention. When we found it the answer was so obvious that even our team who couldn't agree on an approach all agreed. There was no debate or questions or alternatives. It was the obvious work to do. Understanding the problem clearly makes the path forward obvious.

5. ACKNOWLEDGEMENTS

This story exists because I was surrounded by wonderful people. I specifically want to mention my Scrum Master and role-model at the time, Matt Philips. He is a fast-talking sage and I'm truly grateful to have worked with him. Also, I worked with a wonderful facilitator and someone that is gifted with the ability to listen, Laura Burke. I also want to thank Jabe Bloom, whose closing keynote at Lean Agile DC blew my mind and put me on a different path.

REFERENCES

Liker, Jeffrey. *The Toyota Way.* McGraw-Hill Education; 2004.
Rother, Mike. *The Toyota Kata.* McGraw-Hill Education; 2004.
Ries, Eric. *The Lean Startup.* Currency; 2011.
Modig, Niklas. *This is Lean*. Rheologica Publishing; 2012.
Hubbard, Douglas. *How to Measure Anything.* Wiley; 2014.