Moving to a Continuous Delivery Culture: Cutting Releases Cadence

Andrew Peterson

andrew.peterson@costcotravel.com

Abstract

You will learn how a growing organization transforms from a release cycle including three sprints lasting five weeks to a single two-week sprint release cycle. You will be guided in the challenges that the testing organization has had to overcome in order to test things faster, earlier and more often as they change their culture to a more continuous delivery model. This was done not only with traditional functional testing, but also the larger non-functional testing such as security and performance. These teams were able to raise the confidence of build candidates as they moved from 10 releases a year to 26 and the other benefits associated with a more frequent release cadence.

Biography

Andy Peterson has spent more than 20 years in IT roles with the last dozen in a QA specific role including Software Development Engineer in Test (SDET), QA Lead, and QA Manager. He is currently the QA Manager for Costco Travel's Products and Markets group. Before that, he was the QA Lead for initiatives such as their platform and international efforts. Originally from Michigan, he moved out to the Pacific Northwest in 2006. Since then he has been through a lot of the regional hobbies like mountain climbing and wine making. Currently, he is busy trying to keep up with two daughters.

1 Introduction

Like so many other organizations, our organization within Costco Travel has been on a path to deliver value to our members and users more continuously with the same or better quality. We have been on a long journey that has had many challenges. Our organization has iterated through changes to develop and release at a faster cadence. There was initially little automation. Longer sprints and stabilization sprints were required to meet the quality bars early on. These stabilization sprints were the biggest challenge for the QA teams. Teams were forced to stop focusing on delivering new features or architecture and instead shift our time and energy on system stabilization, integration testing, performance testing, security testing and preparation of the system for release.

2 Reducing Deployment Timelines

The need for the reduced deployment timelines was due to the need to get feedback from the business faster. If we were going to do something incorrect, we would rather know after two weeks rather than five weeks. There were also many interruptions within the deployment in the form of an overabundance of hotfixes and stabilization sprints. Finally, estimation is easier for a shorter time than it is for a longer time frame, so IT is able to give the business a more confident idea of what it can deliver.

2.1 The Agile Beginnings

Before Costco Travel turned to agile, we used a traditional waterfall software development methodology in which we would deliver one or two releases a year. In 2012, we began the agile transformation. Initially we started with a five-week release consisting of a three-week functional sprint followed by a two-week stabilization sprint. There was no successful QA automation until 2014.

2.2 The First Disruption

In 2015, we moved to two two-week feature sprints followed by a one-week stabilization sprint. In this phase, we were able to make the work done during stabilization smaller, yet we were still not able to deliver more frequently than ten times a year. The deployments required the site to have scheduled downtime during the deployment. Therefore, we continued to keep with a five-week release cycle by changing one stabilization sprint into a feature week and dividing those four feature weeks into two sprints. This also gave us the opportunity to practice between the functional sprints in the release to see if we could get the stabilization work done.

During this phase there were two key steps that occurred that made the following phases much easier to attain for both the business operations and product development: no downtime deployments and feature flags.

2.2.1 No Downtime Deployments

The most important advance needed for this was moving to a model where Costco Travel stayed up during deployments. Our eCommerce site was going dark ten times a year and as we grew, we lost hundreds of thousands of dollars each time. This had to be stopped before we could get out of the five-week cycle cadence and move to the four. The biggest problem was with the database and the content management systems and making sure that we were backwards compatible so that we could continuously deploy throughout the day. Schema changes in the database would need to be phased in through several releases as backwards compatibility would require the database to be read by either the current version of the code or the next version of the code that is being deployed. On the eCommerce site, no new sessions would be allowed on a percentage of servers until it the current sessions were severed. At this point, the new code would be deployed to the application servers and they would be turned on and monitored. The remaining application servers would be updated in the same way.

2.2.2 Feature Flag

A feature flag is a variable that can control whether a feature is available to the system. This will allow features to be tested in a lower environment with the feature flag turned on. It can then be turned off for regression testing and in production because it isn't fully complete or the business is not ready for it. In order to reduce the cadence of the deployments though, these needed to be used in most projects and at a lower level. Feature flags allowed for a feature to be turned on at any time which decoupled the idea of a release of a feature from a deployment of code. Another benefit is that if a severe enough defect was discovered, there was an option to just switch the feature flag off again.

2.3 Un-Stabilization

In 2018, the third phase finally removed the stabilization sprint as the QA and DevOps teams were able to get all of the work we needed to do. Automation and the ability to get work in during the feature sprint made this possible. This change was the biggest and most important move for QA. Because it was still expensive in DevOps and QA effort to deploy more frequently to production, only one production deployment was done every 2 sprints. The two feature sprints would go into production every four weeks. This allowed us three additional deployments into production and it aligned the deployments with the business periods.

2.4 Sprint is a Release

In May of 2019 after eight months without a stabilization sprint, Costco Travel was able to deliver the sprints to production independently. This doubled the frequency that value is delivered to our members and users. These 26 releases allow for faster iteration if there is a needed course correction.

3 QA Implications

3.1 Functional Testing

3.1.1 Adding Automation

There is no surprise that the key to the QA portion of the reduced cadence was the introduction of automation. At Costco Travel, there was none in the beginning, so it was a big effort.

3.1.1.1 Pre-2013 – Automation investigated

The first Quality Assurance team members at Costco Travel were travel agents that were selected based on willingness and aptitude for the application. A QA Manager was brought over from Costco's home office to help develop the effort with QA processes in mind. A few outside QA hires were also brought in from outside. There was still little experience with automation. There were a few trials of automation products but none took off with the team.

3.1.1.2 2013 – Initial volunteer effort

In 2013, two new quality assurance engineers joined and began an effort in their free time to put together a framework using Microsoft Test Manager and Rational Functional Tester. A volunteer group of interested QA came together weekly and made some progress in getting together a suite of 19 end-to-end tests that were scheduled and run a few times a week. These tests tested the base happy path scenarios for login and purchase of our basic travel product offerings. The tests were fragile and took a lot of maintenance and troubleshooting but they were the beginning.

3.1.1.3 2014 – Getting full time SDET support

After realizing how long it was going to take to go from the initial 19 tests, especially with the troubleshooting, management was convinced to hire two six month contractors. 162 tests were initially identified for these two to automate. In addition to doing those tests, they converted to Selenium from Rational Functional Tester and also rewrote the initial 19 tests that were running.

3.1.1.4 2015 – First Full Time SDETs

Following the contractor trial we were able to create three full time and permanent SDET openings, one of which was filled with an original contractor. These three became the start of the Engineering Productivity Team. This team took on the framework, creation of tools for automation, reporting of automated runs, and integration of the automation into the build pipelines. For the first few years, they were the main coders of the automation as well.

3.1.1.5 2017 – Hired performance and security experts

The hiring of performance and QA security engineers to the Engineering Productivity Team was also a key step in the process. Before this time, we had one QA engineer running all of these types of tests and reporting them. The two new experts were joined and were able to take on the traditional testing done during the stabilization sprint as well as spread it throughout the sprints.

3.1.1.6 2018 – SDET on every scrum team

It wasn't until 2018, that the Engineering Productivity was fully staffed and there was an SDET on every scrum team. This was over four years from the time Costco Travel had our first dedicated SDETs. It was a slow process. At this point, the automation of features is being kept up by the Scrum team SDETs and the Engineering Productivity team can concentrate on framework and tools.

3.1.2 Creation of a Dedicated Automation Environment

As the automation suites grew, the time needed for the QA environment grew. The scrum teams needed to get updated features for testing regularly. They would often have to either wait for an automated test pass to finish or start a deployment causing the remaining tests to run in the suite to fail. The opposite would also occur when a functional tester would be in the middle of a test scenario or using specific test data and an automated test would come through and blow away a cache or some other needed test object. A dedicated automation test environment was developed to prevent this from occurring. This also allowed the automation or deployments to run nearly around the clock without fear of interrupting the other teams. This was key in allowing the growth of automation regression to run concurrently with the functional testing going on in the scrum teams. It also created a healthy competition to see whether the scrum team tester or the regression automation would catch bugs first.

3.1.3 Gatekeeping Environments

The development of a dedicated automation environment allowed us to use it as a smoke test environment for a build. This became a key feature to be added into the Costco Travel Jenkins deployment pipelines, which prevents a bad build from blocking testing our standard QA environment. After a successful development build is deployed into the automation environment it is smoke tested. If these basic tests would not pass, the pipeline would stop until a more stable build would pass. This gatekeeping has saved us countless hours of time from broken builds, severity 0 bugs and other problems from sneaking into the normal QA environment.

3.1.4 Getting Automation into Scrum Level Work

Once Costco Travel was able to staff the scrum teams with SDETs and move the majority of the regression test automation from the centralized automation framework team to the scrum teams with the direct feature knowledge. This accelerated the automation development and improved the quality of the automation greatly. Now the Engineering Productivity Team could concentrate on the framework and tools.

3.1.4.1 Reduce Test Debt

There was a reduction in test debt when the regression automation shifted to the scrum teams. This occurred because the scrum teams were able to add the “regression test complete” to the definition of done of the feature. This was not a possibility when it was being done by an external team that was accumulating work from over a dozen teams and those teams were growing at a rate faster than the centralized team could handle, especially while also maintaining test framework and tools.

3.1.4.2 Ownership and Accountability

The other benefit to pushing the automation work to the scrum team was heightened sense of ownership of the tests. Regression tests are being automated by the team that initially tested the feature, the teams would be also analyzing automated results and is ultimately responsible for the quality of that feature. This makes that most knowledgeable team the one that is accountable to those tests.

3.1.5 Manual Testing

Manual Testing has not been completely removed, but it has been drastically scaled back. We have goals to reduce the manual test suite to under three hours. All of the existing Priority 1 manual regression tests have been revisited and triaged. The QA leadership worked with our Engineering Productivity Team and marked about a third of them as able to automate or already automated. There was about another third that were marked as lower priority and we didn’t have to run manually for every release. That final third of the tests that remained ended up being manual fit into a few different categories. These included tests that (1) were too complex to automate, (2) were too risky to automate, or (3) need eyes, emotions or evaluations of humans. In addition, we decided not to run these for every version in the release. This greatly reduced the workload needed for running these tests.

3.2 Non-Functional Testing

3.2.1 Security

Costco Travel’s security tests were initially built to do the basics of keeping Payment Card Information (PCI) and Personally Identifiably Information (PII) compliance. This meant that one tester was creating the scripts in IBM AppScan, running the scan with set default policies, and validating the report. These scans would take the better part of a day to run if nothing went wrong and they were performed in a shared environment. If there were identified vulnerabilities, engineers did not have much time to analyze and potentially fix if needed.

In planning for getting to a more continuous model, the organization realized that it needed to do more frequent scans earlier. Our engineers complemented the regular release runs with daily smaller runs that looked for higher priority vulnerabilities using IBM’s policy called “The Vital Few”. This allowed us to match up new issues with a specific check-in so the new vulnerability bug was easier to assign to the respective team. Also, we were deploying major versions weekly to pre-production. These full versions were still all being fully scanned making more full policy scans run. All of these scans were built into Jenkins, so we did not have to run them manually. We also were able to integrate other tools, like Sonar, a static analysis tool, while checking in code.

3.2.2 Performance

As with the security testing, almost all of the performance testing was performed in the stabilization sprints. This testing was all done in the same pre-production environment as both the functional testing and the security testing. This meant we had to make sure that only performance testing was done in order to have a similar baseline.

The obvious problem was the need for a dedicated environment. A dedicated performance environment was spun up so they did not have to fight with other process resources in the pre-production environment. The new performance environment was also built on non-virtual servers as production was to provide similar performance results. This environment also allowed for the ability to deploy a new build daily instead of the weekly or sprint build which went into the pre-production environment allowing for a daily performance run. This made it much easier to track down performance issues with fewer check-ins or changes to the infrastructure. The dedicated environment also gave the opportunity to have control and administration of analytical tools; such as Splunk and Dynatrace. Access for the QA team was not possible to the pre-production environment. The performance engineers were able to run full tests after each sprint version, which we were doing more frequently. In addition, they were running smaller scenarios daily. Self-service scripts are also now being developed for scrum teams that have specific needs for performance verification so that the teams actually doing the code can see results specific to them. The resources needs of setting up and maintaining the new environment, tools and test runs were made available when we brought on the new performance engineers along with some resource help from devOps from time to time.

4 Demonstrated Benefits

4.1 More Frequent Value Delivered

During Costco Travel's initial agile roll out there were only 10 releases a year in five-week releases. This was the case for the first several years. We were able to deliver value more frequently when we moved to the four-week releases. This four-week release model also aligned with the 13 business periods which synchronized the business and IT organizations. We are currently deploying 26 releases a year. In addition with providing business value faster, it also allows the business to pivot as needed, as it gains more information with the analytics and sales that we are actually generating. See Figure 1.

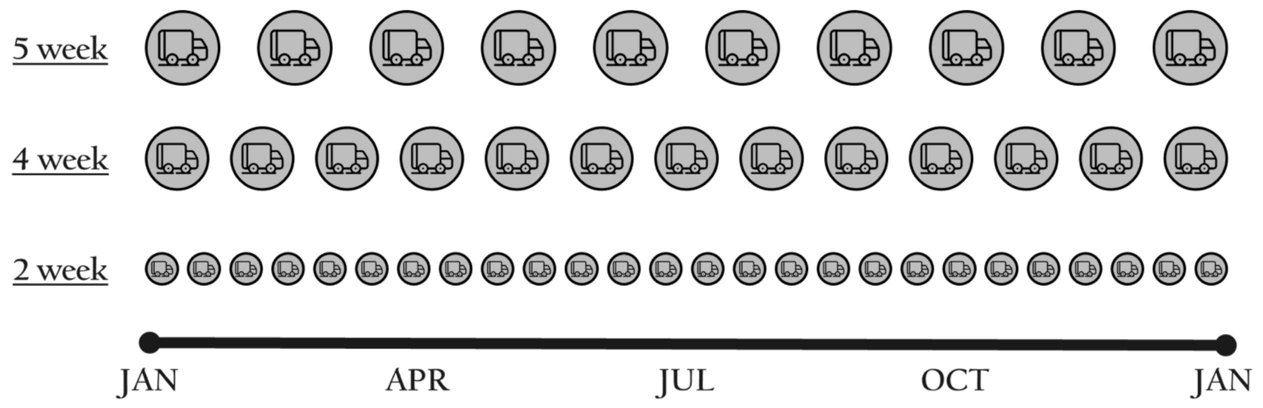

Figure 1: Timeline comparison showing the number of deployments that deliver value to the customer as Costco Travel deployed more frequently.

4.2 Lower Production Defect Fixes Needed

The analysis of the number of escaped defects shows another interesting piece of information that was unexpected. During the timeframe that Costco Travel was releasing under the 2:2:1 week release cadence, there were an average of 15 escaped production defects fixed per week. In the 2:2 week release cadence, immediately after, there was only an average of only 12. If you remember the 1 week in the 2:2:1 week was a stabilization sprint. You would think that if you dedicated an entire week to stabilization that you would have fewer escaped production bugs to fix, but this was not the case. Instead we are now successfully doing a better job of stabilizing the feature work within work sprints.

4.3 Fewer Hot Fixes

One of the most striking improvements has been the reduction of hot fixes. This has been an amazing overhead time savings as many features have had to be pushed out when critical defects have taken precedence over scheduled scrum work. This also causes entire build cycles which takes the time for additional regression testing and devOps deployment activities. These would not normally take place. Finally, any out of ordinary process gets less oversight, testing and monitoring which increases risk compared to having these defect fixes go out with a regularly scheduled release. As seen in Figure 2, there has been a distinct trend downward in the in the past two years. Even the spikes which represent major feature releases trend downwards. There are two general reasons for this trend. The organization is maturing and becoming more stable as automation is developed faster, by testers that are closer to the features and it is run more frequently. Another factor is the amount of time in between releases. A user doesn’t have to wait five weeks for a defect to get fixed anymore. That defect could go out with limited impact in the two-week regular cycle.

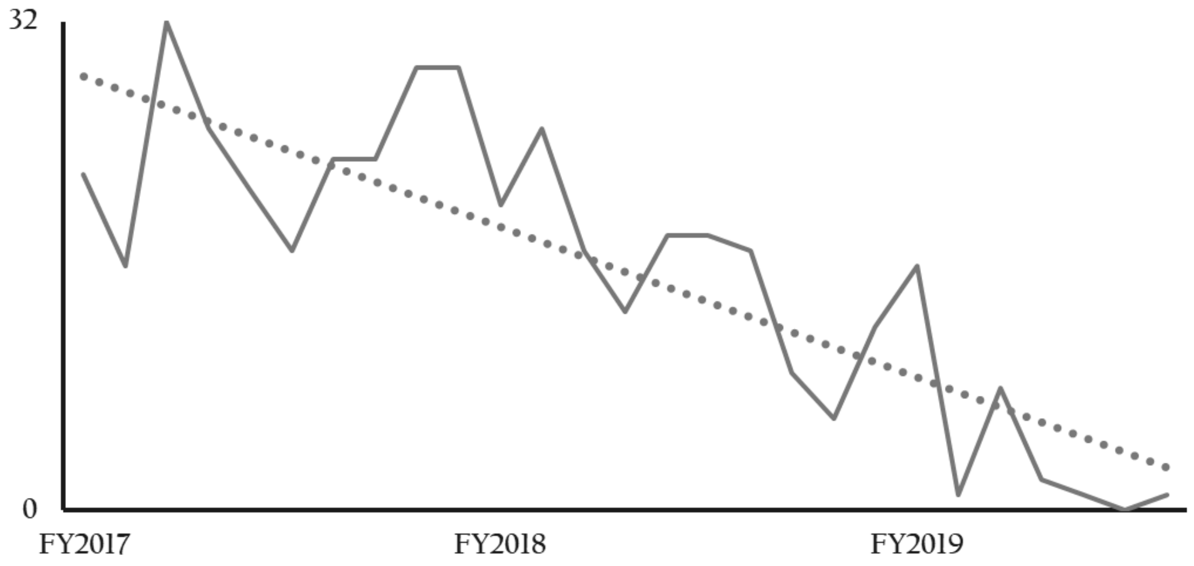

Figure 2: Actual hotfix items (solid) and trend line (dotted) of hotfix items in releases between fiscal year 2017 and fiscal year 2019.

5 Missteps

As with any process change or set of changes, there is bound to be failures. Costco Travel has had its share of missteps as well. There are a few highlighted here to give an idea of what can happen. Remember that one of the core concepts of Lean thinking and Agile is to iterate and continually improve. It is important to keep making small steps that will either produce small failures to learn from or small wins to build off of. This paper will only highlight three of the QA specific missteps while working toward a faster release cadence.

5.1 Use Experience to Start an Automation Effort

The first and most important misstep for QA was to start an automation effort without testers who have automated tests at a large scale. There are several tools on the market that make it seem so easy to automate your application testing. Writing a test may be easy, but developing the infrastructure, reporting, processes, etc. around it and then maintaining them is not really thought about. There were several attempts with different products to get off the ground with varying degrees of success at Costco Travel. Nothing was useful until we hired dedicated SDETs that would focus on the framework and initial tests. Even these were not the end product, but they were the first serviceable automation.

5.2 Tread Lightly Adding BDD to a Mature System

Costco Travel did make an attempt to go to a behavior driven development model (BDD) using Cucumber. This proved difficult because of how mature the systems and documentation was. A couple of proof of concepts were attempted including one that was going to go back and rewrite all of the previous user stories into Gherkin format. This proved to be just too large of a rewrite just for the tests. It also showed how standard writing styles are needed to make the coding clean and easy to maintain. We also tried to do a proof of concept of a small project independently. The idea was to use BDD on projects going forward. This did not move forward because there would have to be more than one process and we still wouldn't have had all of the user stories in one location.

5.3 Don't Let Your Test Run Time Get Away

The final misstep highlighted in this paper is the need to manage test run time. This is one that we still run into here and there. Costco Travel's focus was on the run time of testing. We would regularly get test suites bogged down because tests were continually added and not looked at for efficiency or even need at times. The main problem was the need to focus on the UI testing which by nature are slower. To build on that testers would create an entirely new end to end test using a similar workflow to one that already existed. There were also situations where certain tests did not have the importance to run every day or build. We had test run goals of five minutes for a simple smoke test of an environment. A few months later, a tester would ask why the pipelines were taking so long and part of the reason was because this smoke test was running for 25 minutes – nearly five times longer than it should. We would see this at all levels of our tests. Our Continuous Acceptance Tests (CAT), which was the basic build level certification, would get to the point where it would run for 45 minutes and the team would know we would have to do something to reduce it. Here are a couple of remedies that were employed:

- Actively go through similar workflows and develop permutations that could regression test as many features as possible with as few end-to-end trips as possible.
- Enforce the coding standard of adding a new feature to an existing end-to-end test if it is available over creating another end-to-end test.
- Creating a prioritization in which we could group regression tests by tests that would highly impact the end-user and those that would not impact the end user and we could run the later less frequently and in off-peak lab hours.
- On the hardware side, make sure that we were running as many tests concurrently as possible. There was a significant effort that went into determining how many Selenium nodes could fit onto the virtual machines that were allotted and at a lower level, how many instances of each browser could run on each of those nodes.

6 Next Steps

6.1 Mindset Shift

Costco Travel has gotten to the stage where we cannot simply continue to shrink our sprints or deployments. We need to start thinking about how to get the value out more dynamically. We need to start peeling away from our monolithic core so that we can deploy independent services (Fowler 2019). These independent services must be backwards compatible so that we don't have to depend on one specific release cycle. Costco Travel is already developing proof of concept services that should plug and play into our current architecture including automated test frameworks that will allow for the continuous testing to complement the continuous integration and continuous deployment of these specific services.

6.2 Testing at a Lower Level

Costco Travel has developed an amazing UI testing platform. Even though we are good with UI testing, this type of testing is big, bulky, and time-consuming to run. Code changes causing failures early on in an end to end test may prevent many other tests from getting run until those tests are fixed. These tests also rely heavily on outside influence such as Selenium, individual drivers, browsers, and device hardware. There was no UI test automation strategy early on at Costco Travel and incoming automation testers had to deal with a mature product with little test hooks into lower layers. This left us with more of a goblet looking test model (See figure 3) instead of a traditional pyramid model (Cohn 2009, 311). The future move to a service oriented architecture such as a microservices architecture will allow us for us to work with the architects and developers to get test hooks in place and an accessible service layer to test on.

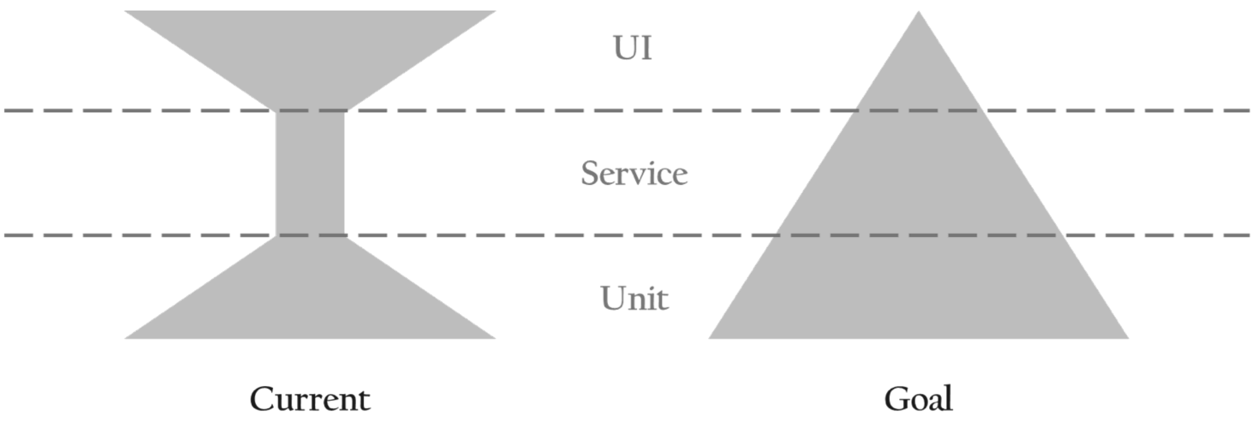

Figure 3: Current model of test distribution with more unit and UI tests and the ideal distribution (right) where there are more service layer tests than Costco Travel has now relative to UI tests.

7 Conclusion

Costco Travel has shown progress in its journey from a manual five-week release with only three weeks of value added work to a much more automated two-week deployment cycle with almost full value capacity. It is easy for existing mature systems to feel that it is too difficult to make progress, but this is not true. A continuous improvement approach must always be in place. In Costco Travel's situation, we focused on getting to closer to a continuous deployment and continuous integration model. They were a large successful monolith with a realization that additional value could be delivered faster. Starting from scratch was out of the question because we did have success. We took small strides. This can be seen in ratcheting up the QA department slowly over the years. The release cycles did not cut to two weeks all at once. It was a slow, continuous improvement. Mistakes were made, identified and reworked before moving on. Victories were celebrated. The roadmap is still in place and the journey is still continuing as Costco Travel is attempting to change the architecture so that it can have more individual independent services. These services will be testable on a lower service level, which will also be independent of the rest of the existing monolith and other services.

References

Cohn, Mike. 2009. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional.

Fowler, Martin. 2019. "Microservices Resource Guide" martinFowler.com, https://martinfowler.com/microservices/ (accessed August 15, 2019).

IBM Knowledge Center. 2019. "Security test Policies", https://www.ibm.com/support/knowledgecenter/en/SSW2NF_9.0.2/com.ibm.ase.help.doc/topics/c_security_test_policies.html (accessed August 15, 2019).