Introduction to Test Automation and DevOps: A Case Study

Robert Taylor

rmtiv@comcast.net

Abstract

Test automation and DevOps is necessary to increase test coverage, boost efficiency, and improve the quality and quantity of a delivered product. But have you ever wondered how to begin implementing any kind of test automation or DevOps if little or none exists? Or understanding what test automation or DevOps entails?

This paper explores the challenges that beginning test automation and DevOps Engineers may face when beginning their Automation Engineer career, especially within .NET projects in an agile setting. This paper discusses a case study about test automation and DevOps separately, then the integration of test automation into DevOps, and lastly addresses common challenges beginning Automation Engineers may face. Key topics will include tools used through the case study, understanding the purpose of test automation, determining and articulating why testing needs to occur, understanding why automation is important, and learning how to integrate test automation with DevOps.

This knowledge can help beginning Automation Engineers' determine which tools may add value to their company, increase automation coverage, gather effective analytics, and increase efficiency of the Continuous Integration/Continuous Deployment (CI/CD) pipeline. The purpose is not to dictate how an Automation Engineer should do their work, but to teach or expand on what they know and understand what steps can be taken.

Finally, this paper concludes by describing how to use the information presented to quantify necessary changes and solutions to use for next iterations. That change is not static, but everything created is a living breathing thing that needs to be continuously monitored and improved upon each iteration.

Biography

Graduated from Oregon State with a B.S. in Computer Science in the summer of 2017. During my senior year and beyond, I worked on independent projects using Unity and C#, participating in Portland Indie Game Squad (PIG squad). In October 2017, I worked for Experis-Manpower Group with a focus in black-box testing on major label video games. At Experis, I learned how to use a multitude of testing methodologies and software development cycles. After a year at Experis, I joined BlueVolt in November of 2018 as a Junior test automation and DevOps Engineer. At BlueVolt I continue to build upon my knowledge of testing by adapting it with scripting to automate the testing process. Taking over the test automation I built out a Regression test suite when none-existed, implemented some performance testing, participate in all UAT testing, and managed and directed interns on where they can begin and how to begin help with test automation. I've also increased my DevOps knowledge and experience by setting up lower environments, creating a CI/CD pipeline to speed up development and testing, setting up gated check-ins, and using a wide range of tools to monitor and configure BlueVolt's production environment.

1 Introduction

Test automation and DevOps are becoming important roles through all varieties of industries. and for beginning or aspiring Automation Engineers it can be difficult on how to get involved with the tools, practices, and cultures of test automation and/or DevOps. Learning how to work with test automation and DevOps always starts with the customer, as the main objective is to server the customer effectively, efficiently, and increase the quality of the product. To achieve this, it is important to learn and understand the product's development cycle, agile processes, goals of the company, and the pain points for the different groups involved with the product (such as the product team, support, developers, QA, and customers). Coming into BlueVolt's, there were a lot of manual steps for deployments through a complex system of NAnt [18] scripts with limited DevOps implementation. There were outdated unit tests, limited code coverage, and developers were using their time to manually run regression tests at the end of each sprint. But before I could begin automating tasks it is important to get familiar with what the product is and the goals of BlueVolt.

Before creating any kind of automation, it is important to take the time and effort to understand the problem that the product is attempting to solve, what is the metrics determine the product's success , the products development cycle, and what pain points are preventing the product from being more successful. These 4 ideas can be broken up and act as signposts to help guide successful automation implementation. The first signpost is to learn about the company's product to understand what it's trying to achieve and how to serve the customer. At BlueVolt, we are a Learning Management System (LMS) that provide customer with a platform to host and create a dynamic learning system. The second signpost is to identify the goals of the product that the company's creating. At BlueVolt, the company's primary focus is the user experience, what the user can and cannot do throughout the platform and helping create a product that will satisfy different types of user's needs. Knowing this emphasize the importance of user experience for the LMS and helps focus tasks towards improving user scenario. This included expanding upon the regression suite through automation, putting tests with pipelines, create code and product quality feedback loops, create not tests to increase functionality coverage, and using performance analytics to find performance improvements.

The third signpost is about improving the CI/CD process by automating tasks so that developers only need to complete a pull request to see their code get deployed to test environments and then after testing automate processes to get the code into production. Improving these processes allow for efficient development and could decrease the downtime of the application during updates and deployments. At BlueVolt, a few improvements to the CI/CD was accomplished through automating regression tests, creating deployment pipelines, and setting up gated check-ins. These improvements have helped alleviate development pain points, improved testing, sped up the deployment process, and improved the quality of the product.

Lastly, the fourth signpost is about improving the quality of the code and product. Being able to provide effective feedback helps provides the information needed to figure out how to fix broken code, improve code coverage, and increase performance. Using automated tests to provide information as soon as a pull request is completed can help maintain the integrity of the build. Other methods to improve code and product quality can be through test pipelines, integration tests, unit tests, stress testing, setting up SonarQube, or any other tools that provides feedback about quality and performance. At BlueVolt, test automation was combined with pipelines that would check to ensure that the product could always build, database migrations always work, and code coverage of unit tests are analyzed with SonarQube. Once the product is pushed to production, Pingdom and New Relic are used to track and measure the performance.

Following the guidance of the above signposts is an excellent way to start figuring out where the problems exist. From there the problems can be broken down into investigations which will lead into actionable items for automation. Another important factor when creating automation is understanding the development stack of the product. Knowing the development stack can help provided answer into what

tools to use to develop tests, test environments, and software to use so that the automation will work well with the product.

At BlueVolt, our stack is C# .NET backend with an Angular frontend where we work closely with Azure to host and deploy our product. Becoming familiar with the stack has helped direct me towards focusing on automation through Azure DevOps and Azure Portal as much as possible. Also used Jenkins and SonarQube which are used to create testing and analytic pipelines to quantify and automate code coverage, regression testing, code deployments, and other useful tasks. Automating test have been done through Protractor, JMeter, and custom scripts to test the different components, functionality, and simulate user scenarios throughout the web application. Another key component at BlueVolt is the use of a content delivery network and performance software to help deliver our product effectively. As an LMS we work with Akamai, which is a content delivery network (CDN), to improve the caching of our content to speed up our page loads and use Pingdom and New Relic to monitor performance of the production environment.

This paper will explore test automation and DevOps independently by discussing the different types of tools that have been used at BlueVolt. Each tool will be broken down into how they function and how they fit within the culture of BlueVolt. The paper will then discuss about how to use tools from test automation and tools from DevOps together to create automated testing. Lastly the paper will finish with questions that beginning Automation Engineers may face. The purpose of this paper is to show the steps taken to integrate automation at BlueVolt, introduce different combinations of test automation and DevOps together to increase efficiency, and to inspire or provide useful knowledge for others for their Engineering career.

2 Test Automation

Test automation is taking manual testing and tasks and then, "… automating the tracking and managing of all those testing needs, including how much of the system different tests cover and what other types of testing might be required to cover all the moving parts." [1] Where the goal is to use automation to create a quick, efficient, reliable way to get results. These results are valuable as it provides Quality Assurance and Developers with feedback towards improving the product. Automating tests should help find issues quickly and efficiently so that developers can resolve found issues before the next release. Another benefit of automating tests is that it allows other testers to focus on areas of the product increasing the overall test coverage.

To increase testing coverage is important to understand a wide range of testing techniques. Knowing different testing techniques will increase test coverage across the system and help find the weak points within a system. Some examples of different testing techniques are:

- Golden Path (Happy Path) – The default scenario of known input producing a known or expected output throughout a system.
- Testing Edge Cases – Testing outside of the base assumptions the expected use of the function or feature within the system.
- Smoke Test – Check basic functionality of a system, ensuring that the product's main feature work as expected.
- Combinatorial Testing (All-Pair Testing) – Testing all the difference combinations of parameters in a system to achieve 100% test coverage.
- Performance Testing – Testing the speed and effectiveness of the system.
- User Testing – A test plan to run through user simulations to ensure that functionality will work for users before a product is released.

- Stress Testing – Test the stability and reliability by putting a system under an extremely heavy load condition.

Knowing different testing techniques and integrating them into the development cycle will increase the product's quality. Another benefit is that knowing how to test will narrow down where to start automating and help determine which testing tool should be used.

2.1 Test Automation Tools

After knowing about the different type signposts to follow for test automation, it is time to determine which tools to start using to build out those tests. At BlueVolt our stack is a C# .NET backend, an Angular frontend, and our database lives within Azure. The focus of our product is through customer success using user interfaces, performant page response times, quick content delivery, and providing an effective learning platform that the customer can use. This information helps determined to create tests that will monitor application performance, check that UI functionality is correct, automate regression tests, and preventing breaking changes from being released into production. Knowing what test need to be created, the tools that help met these requirements were Protractor for frontend testing and a mix of JMeter and custom scripts for backend testing.

2.1.1 **Selenium/Protractor**

Selenium/Protractor are frontend UI testing tools to do end-to-end testing, simulation how an end-user would use the system. Selenium is a Java tool that interacts with browser page elements of a web application. This can be used to verify content is loading correctly, check that elements are interactable, simulate use navigation, combine UI commands, and to test different kinds of frontend functionality. Whereas Protractor is a test framework for Angular JS and uses Selenium to automate the browser interactions. Since BlueVolt's platform uses angular for the frontend we built out our end-to-end tests using Protractor. Along with building out a Protractor test suite, time was spent deciding where and how the Protractor test will run.

End-to-end testing should run against a clean test environment. The test environment should also be updated with completed features, that are planned to go out with a future release, so that test have a chance to catch a breaking change. For example, if a feature exists on a dev environment and introduces a broken functionality, and end-to-end tests are being ran against a different environment, those tests will never catch the breaking change until it's possibly too late. By automating test to check existing functionality, in an updated clean environment, can help expose any breaking changes quickly and efficiently so they can be fixed (or reverted) before the next release.

2.1.2 **JMeter**

JMeter is a "...Java test framework provides facilities for load, stress and performance testing, but it can also be used to perform regression tests and a limited set of functional tests." [2]. The tools focus is to simulate multiple users hitting an endpoint that could test the endpoint's durability, reliability, and to find its limitations (where it will break, what load it can handle, and where the stress/weak point reside within that endpoint). Another use of JMeter is the possibility to run a suite of regression tests or other tests for functionality due to the tool's ability of: having a variety of samplers that allow any kind of request simulation, assertions for verifications, and a diversity of listener to help debug the tests. [3]

At BlueVolt, JMeter is used to test our performance of our Web API endpoints, databases interactions, and perform load and stress testing. JMeter has also been used to check the configuration of our different development environments (dev, quality assurance, staging). An example of using JMeter at BlueVolt, was comparing our performance between using a SQL Server database and an Azure database. Using JMeter, a suite of tests would run to hit endpoints that reads/writes from a database tracking response time, throughput, and deviation using large amount of simulated user. Afterwards, the metrics from each were compared and analyzed to determine which database would be best for our product. JMeter was not the only determining factor for which database, but the metrics did help justify and predict how BlueVolt's product would function in the different environments.

2.1.3 **Scripting**

Creating custom test script through scripting language can be a powerful way to create flexible automated tests, some example of different scripting languages includes JavaScript, Python, Ruby, PowerShell, or Batch. A test script is a set of instructions that is performed on a system to quantify if the system is performing as expected.[4] Creating custom scripts allows a test to be completely customizable. A few examples used at BlueVolt are increased scalability by using any available library, determine which target(s) to test, what dataset and amount of data to use, and alter the load the test will put on the system. Custom test scripts can be dynamic for different environment, where the test is adjusted to avoid destructive testing (create, update, or delete data) if the test saw that it was running against the production environment, or can include destructive testing when ran against dev, qa, or a staging environment.

At BlueVolt, currently we do not have an abundance of custom scripts, but have made a few that check the response header, response code, and response time of our different nondestructive Web API endpoints in production. These custom scripts help check the functionality of those endpoints and can monitor the response time and other performance metrics of those endpoints. Another use of these test scripts is when we setup up other environments (through Azure Portal or on a VM) the script can be used to check Web API endpoints to ensure that the system was configured correctly.

2.2 **Applying Test Automation**

Test automation is finding repetitive manual testing, are of the application that can't be automated, or speeding up testing while increasing the quality of the product. Successful test automation should help improve product quality by finding bugs or broken functionality that may have been caused by a code updates and be easy to maintain/run. [5] Test automation needs to be efficient and save time it would have taken to manually test allowing other testers to focus on different areas of the product to increase overall test coverage. The goal of test automation is the serve the customer and ensure they experience a high-quality product.

One way of applying test automation at BlueVolt was that at the end of each sprint (once every two weeks) running manual regression tests were run to check the functionality of the product before being released to production. This was done to check if any breaking changes were being introduced and to fix them before the release. But by performing manual repetitive test, the tests were taking away time that the rest of the team could be using for other tasks. Also, since the tests were running once every two weeks, a breaking change may not be found until it was too late in the development cycle. But by using Protractor, an automated regression suite was built and turned something that was ran one once every two weeks manually, into something that runs on a nightly basis. This also helped relieves the manual testing from the team and finds issues the night of rather than at the end of the sprint.

Another example is when migrating from a SQL Server Database to an Azure database, tests were creates using JMeter to check product performance. Then these metrics were used to determine which type of database the company should use to improve the quality of the product. These same JMeter test were repurposed from performance and are now used to check Web API functionality of our production and test environment using non-destructive (update, add, or delete data) testing.

In this section a lot of different testing tools and techniques were explored and discussed throughout BlueVolt. The next section will go into detail about the different DevOps tools and purposes they have served at BlueVolt. Afterward the paper will discuss how DevOps and test automation can be combined to create powerful features.

3 **What is DevOps**

DevOps is a multitude of tools, practices, and culture shifts that expand across the agile process within a product's lifecycle. DevOps can be described as a collection of ideas and agile practices that are used between dev and ops engineers throughout a product's lifecycle. Starting from design features, through

the development cycle, through test, and through production. Where DevOps "…extends the Agile principles beyond the boundaries of the code into the entire delivered service." [6] Bringing together all the different groups involved to value building quality throughout the development process.

The primary objective of DevOps, much like test automation, is to satisfy the customers wants, needs, and concerns. This can be accomplished through increased deliverability, quality, improvements to the development cycle, helping developers work efficiently and effectively, convey information about performance and metrics, promote sustainable development through agile processes, making things simple, and architecture improvements. [7] A few ways this was achieved at BlueVolt was through setting up pipelines, automating tasks, setting security policies, creating environments, working with monitoring tools, and relieving other pain points throughout the company. Improvements throughout the company means improve the deliverability of the product which then increase the service to the customer.

3.1 DevOps Tools and Applications

Like test automation, when implementing a DevOps tool, process, or cultural shift it is important to understand the issue being solved, how the implementation will solve that issue, and how can the process be iterated upon for future updates. For BlueVolt the problem DevOps is trying to solve is how to improve the deliverability of the product while improving its quality, integrity, and health. Many different tools, processes, and ideas have been implemented to help improve the product across the whole company. DevOps is used to speed up the development cycle, improve our agile practices, increase the amount of product that can be delivered, enforce a higher-quality product, and to track performance to find improvements. This can't all just be achieved with one or two tools/applications but with several different tools. Some of the tools and applications that BlueVolt uses are Azure DevOps, Azure Portal, Jenkins, SonarQube, Pingdom, New Relic, and Akami. The reason why some of these tools were chosen was due to our stack (C# .NET backend, Angular frontend, and Azure database), current issues that the company faces, and to increase future planning to prevent issue from arising.

3.1.1 Setting Up Pipelines

Azure DevOps and Azure Portal offer a wide range of DevOps opportunities to speed up the CI/CD process, provide analytics and metrics, allow the user of pipelines and agents for deployments and gated check-ins, and has a system to create, alter, and track Public Backlog Items. Azure DevOps also provides flexibility in build, deployment, and release pipeline by providing a wide range of extension from the Visual Studio Marketplace. Whereas the Azure Portal provides the capability to setup different resources that can be used for all aspect of our application. Due to the nature of the .NET project, the Azure Portal is primarily being used to setup different WebApp's providing us the ability to migrate away from Internet Information's Services (IIS) on virtual machines (VMs) to host and run different environments. Like other monitoring tools, the Azure Portal offers analytics of the WebApps to show the performance of each on being used in the different environments. Another important pipeline application that we use is Jenkins.

Jenkins is a useful pipeline application that can be configured to run all different types of pipelines. At BlueVolt, Jenkins has been used to turn our test automation into automated testing. Where we integrate our tests with Jenkin pipelines, set specific triggers for the pipelines to automatically run, and then output the results via email after the tests are complete. This has been beneficial as the team went from running manual regression tests once every two weeks to a nightly automated test suite with emailed results. This has helped us catch and address issue well before the night we release to production.

3.1.2 Improving Delivery and Quality

Setting up Jenkins, and other pipelines, continue to help us track, find, and resolve issue well in advance, but there are other ways to help improve the quality and deliverability of the product. The product's quality and deliverability can be improved through task automation and settings up software to gather performance metrics. Some performance metrics that we gather to quantify the integrity of our product is response time, uptime, and the overall health. This is accomplished at BlueVolt through the user of Azure DevOps and SonarQube. BlueVolt uses Azure DevOps to transform our manual task into pipelines that run from triggers to improve the speed, quality, and quantity of the deliverability. Whereas SonarQube is

used to help track our code coverage to find areas where we can improve our testability to improve the quality of the product.

Azure DevOps has provided us with the tools to transform our manual task into automated pipeline that will run when triggers are hit. These pipelines help us deploy to our managed test environments, release our product to production, check database migrations, and speed up the development process. To run these pipeline, Azure DevOps agents must be either installed on a self-hosted VM or there are a few that are Microsoft Hosted that can be used. The few purposes of these pipeline agents (self-hosted or Microsoft hosted) are that they can pull down a repository, build the solutions(s), run tests, run custom script, create build artifact, chain pipelines together, and then deploy the code to a VM instance or the Azure Portal. These pipelines have been effective for increasing product production and has sped up deployments to our test environment and production. Where each sprint the CI/CD automation process is continuous refined to improve the deliverability of the product.

Another way we use Azure DevOps is by setting up quality to check that no breaking changes are introduced into our managed code source. Some simple quality gate pipelines have been setup to triggers so that before a pull request is completed the changes to the different C# .NET solutions are checked to ensure they can build, that our legacy code still functions, and that a new database migration will not break the build. The purpose of these pipelines is to ensure that the quality of our product is not negatively affected, prevent bugs from being introduced into the system, keep our source control functional, and increase the deliverability of our product.

SonarQube is another product we use to help track and monitor our unit tests and code coverage. At BlueVolt, SonarQube is primarily used to analyze our code quality, find deprecated code lines, find vulnerabilities, and analyze if there is any tech debt. We combine Jenkins and SonarQube to bring automation to the process. Jenkins will start and stop a SonarQube docker container (to manager VM resources) and run the unit test through the SonarQube Scanner to gather daily reports to track the trends and health of our product.

3.1.3 **Setting Up Environments**

Azure Portal is used to setup our managed test environment and production. As our company grows (in clients and employees) we have been moving away from inhouse managed services and moving to the cloud. Using Azure Portal our applications have been transformed into WebApps and our static SQL Server was changed into an Azure DB with an elastic pool for dynamic scalability. As BlueVolt continues to improve our production resources, we continue to try to mimic the same types of resources to our test environments. Our test environments have moved off IIS and now use WebApps, each environment has a scaled down Azure Database, and our deployments to the managed test environment mimic production releases. Bugs and other issues can be prevented by having test environments setup similarly to production.

Setting up managed test environments are critical to test new features. A test environmental allows testers, developers, and QAs to test functionality, bug fixes, incomplete features and other agile updates without having negative effects on the production environment and database. [8] Having a test environment allows all testers to find bugs, issues, glitches that may have been introduced throughout the sprint. But there is some limitation of a test environment which prevents it from being an exact replica of production. A test environment will not have the stress of an active userbase on the system, third-party integrated systems, equivalent resources (it can be too expensive to have a test environment with exact same resources as production), or any other outside influences. Regardless, having "… a confined environment where outside influences on the code are limited, gives those involved in the production of a system confidence that their code works as expected." [9]

3.1.4 **Monitoring Tools**

After a product has been released to production there needs to be monitoring setup so that performance and health can be continuously tracked. The purpose of monitoring the production environment is that a healthy product:

- Protects the image of the business
- Keeps customers happy
- Prevents losing sales (if applicable)
- Obtain better search engine results ranking
- Detect hackers, ddos, and attacks
- Track performance metrics
- Provide a quick response if the site goes down [10]

BlueVolt believes that customers of the product should always have an excellent experience while developers continuously find solution to improve the product'. To accomplish this, we use a combination of Pingdom and New Relic to help track real-time performance and health metrics. These metrics help us find way to improve the performance of our product by breaking down the slow parts and making it apparent where exactly the pain point exists. Metrics are a good way to provide information about the product to the different group involved with the product to help develop the DevOps culture.

3.1.4.1 **Pingdom**

Pingdom is a monitoring tool that helps analyze and gather information about the uptime, transactions, page speed, and visitor insights of our product. Custom alerts can be created for uptime, transactions, and page speed tracking the response time and outages that have occurred. These alerts can be customized to monitor specific URLs or create custom scripts to monitor certain features of UI integration, specific endpoints, or third-party integrations. Then if an alert is trigged, Pingdom can be setup to send information to the team so that the performance issues can be immediately addressed.

3.1.4.2 **New Relic**

New Relic is a monitoring tool that uses New Relic agents to track the different performance metrics within a system. To track certain information New Relic agents can be installed on VMs, WebApp, databases, mobile devices and is compatible with C, Go, Java, .NET, Node.js, PHP, Python, and Ruby projects. At BlueVolt, we use New Relic as our primary way to track the performance throughout our VMs, WebApps, and Azure Database. It offers a wide range of customizable analytics that helps pinpoint and track down where and what transactions are cause slowdowns on the production environment. We also use this tool to create custom analytic dashboard to track the overall health and integrity of different aspects of the product. The goal of using New Relic is to find the slowdown within the .NET project and determine where performance updates could be introduced. New Relic also allow custom alerts to be created and can send alerts when a violation occurs. At BlueVolt we have created transaction speed New Relic alerts to determine if a transaction is taking longer than it should, and if a transaction is performing too slow an alert is sent to the development team to address that performance issue.

3.2 **Applying DevOps**

DevOps is not just tools or practices, it is applying the tool, practices, and culture changes of DevOps to every aspect of the product life cycle. It is about the different group involved with the product to become aware with the entire development cycle to improve the products quality and overall better server the customer. DevOps should be implemented from the product team figuring out what content needs to be created or adjusted to serve the customer, to the agile and scrum process of meetings, at standup, throughout planning, to the integration of different tools and application to gauge the metric of the product and help speed up the CI/CD process. "DevOps doesn't confine itself to technical implementation and execution of testing and development activities. It is a cultural shift that is needed to ensure that the strategy is collaborative and closely monitored. Developers and operations need to work together to reduce inefficiencies and bring speed to the development activity." [15]

DevOps can be extremely overwhelming for anyone involved with the cultural shift. DevOps is about bringing together the different groups that have been separated from the software development process. [17] It is challenging to influence and provide solutions for each of the different groups, as the focus is not going to be the same between them. But different DevOps tools and practices can help build out the culture by finding solution to overlap datapoints to involve the different groups.

An idea of DevOps is to provide a feedback loop where each group involved (QA, testers, developers, product) can get key information about the product quickly and effectively. Pipelines are an excellent way to speed up the processes while proving key information about the product's lifecycle. One example is by creating and managing gated check-in, important information about the quality of the code, automated tests, and other metrics can be retrieved quickly. Pipelines can be triggered from one another to create a deployment and a release pipelines to managed test environments or to production. By creating and managing pipelines to the test environment, this allows testers to start testing new features, updates or bug fixes as soon as the developer completes the pull request. Aside from pipelines, other software tools (Azure, Pingdom New Relic) can be used to track and gather performance metrics that can be used throughout a company to quantify the healthy and integrity of the product. DevOps provides the ability to quantify the work done through the agile process and communicate throughout the organization. But DevOps cannot be just implemented, it needs to be continuously refined and build upon it's becoming more and more effective. But it all starts with taking that first step to help bring different groups together to build an effective development workplace.

4 Using DevOps to Create Automated Testing

Test automation and DevOps can be extremely powerful when combined. As DevOps can transform test automation (that runs are a push of a button) to automated testing that runs automatically. The purpose is to create a process that will help improve the agile process of the deliverability, increase scalability, and increases the quality of what is being delivered to the customers. The power of automating tests is only as good as the effort put forwards and always needs to be continuously improved upon with each iteration of new test, DevOps tools, practice, features, or anything that would work with test automation.

The purpose of testing it to increases the quality through code coverage. Some example where testing may help the produce is through exposing existing issues and help catch issues that may have been introduced with new code changes. With the application of DevOps tools and practices, the quickness, reliability, and effectiveness of test automation can be improved. The goal is to implement test automation and DevOps at different stages of the CD pipelines so that "…one or more test suites run. Each test should provide feedback. This feedback helps people understand CD pipeline status and the quality of code moving through it." [13] As a warning, this should not be confused with making every test automated as automation will not catch every issue, adhoc testing needs to be manually involved. Adhoc testing is an unstructured process that's conducted by a tester with strong knowledge of the software testing areas of the software through error guessing.[19] By having a feedback loop throughout the CD process developers can get quickly notified with faults and are able to fix the issues while the current code is still fresh within the sprint. Testers and automated testing work together to increase test coverage helping improve the overall product by finding issues before a release to help the company serve a higher quality product to the customer.

The goal of a product is the figure out how to best serve the customer. A company cannot see increase on return on investment if the customers' needs are not met, and consumers are always looking for the what works the best. If a company decides not to focus on their products deliverability, they run the risk of being passed by another competitor or will be unable to catchup to a competitor that's ahead of them. When improvements are made to the CI/CD process with automation the deliverable product becomes an improved version of itself that better serves the customers. But the deliverability of a product is not just determined by improvements to the CI/CD process, but by influencing the agile, testing, scrum processes with the DevOps culture. It is necessary to integrate DevOps ideas into the agile process to help refine the existing practices that adapts to change to better satisfy the customer through early and continuous delivery of valuable software. [14]

Automated testing should help increase the deliverability of the product by speeding up tests while maintaining or improving the test quality. Then, further improvement to product deliverability can be accomplished by adding test automation to DevOps. Doing so will help find issues effectively and quickly which can become cost effective, increase product quality, and help save the company time it would take to run test manually. [16] Creating automated testing will allow developers, product, testers, and QA to direct their focus to other agile aspects to increase the deliverability to production.

At BlueVolt we try to integrate test automation into all aspects of our DevOps tools. An example is the regression suit, where automation transformed manual test that happened once every 2 weeks into something that could be ran through a shell command. Through DevOps test automation was automated further. The DevOps tool Jenkins allowed us to remove the need of having someone starting and reporting on the test suite and turned test automation into automated testing. The ideas of adding test automation to DevOps expanded past the regression tests and endpoint tests were setup to run nightly on Jenkins to verify functionality and ensure that nothing broken throughout the day. This practice was also applied with the Azure DevOps but with different types of tests.

Azure DevOps has provided me with the resources to create validation pipelines that we use as our gated check-ins, pipelines the ensure certain tests pass before the pull request is able to be completed into a branch. This combined with branching policies has helped me and the developers ensure that no breaking changing would be entered and ensure that our branching processes are being upheld. What this means is that ensure that branches go through the correct methodologies, practices, and quality gates before making into the Release branch and ensure that our agile process is upheld. Where the gated check-ins are continuously being revised to ensure excellent code throughout the development process.

Even the monitoring tools has test within their framework that allows the developers and I track issues and send alert when certain threshold are being violated. Pingdom allows alerts to be setup for Uptime, Transactions, Page Speed, and allows custom reports to be made. Where BlueVolt determines what is an acceptable performance, and if something breaks that performance then alerts are sent out and leads into tickets to be resolved during the current or a future sprint. Just like Pingdom New Relic function similarly with the type of alerts that can be setup. For BlueVolt New Relic is used more for digging deeper into the transactions which helps single out slow performant operations and setting up alerts accordingly. Even with what BlueVolt has setup, we are continuing to expand upon each agile process either adding, creating, or refining upon what we currently have.

5 Challenges Faced by Beginning Automation Engineers

Whether working in test automation or DevOps, Automation Engineers will face a learning curve as they continue to problem solve each automation hurdle. This could be through learning about a new tool, a new process being introduced, another automation practice, or a cultural shift. To help overcome these hurdles I'm going to answer questions that I ask myself to break down most automation challenges and hurdles I have faced. These questions may not be applicable to everyone but the intent it to help show how I helped myself in learning, implementing, iterating, and maintaining the tool, processes, and tests that have been introduce throughout BlueVolt.

1. **"Where to begin?"**

   Get involved with the product and start learning how the product functions, who the target audience, what is the issues the product is trying to solve, what is the stack it's developed in, and where are the pain points for groups within the company. Take in information about the product and its lifecycle. It is important to take the time and focus to understand what the company is trying to deliver, what issue this automation will solve, when the automation will be used, and how does it affect the deliverable of the product. Try to fit the mentioned tool above to see how they can help introduce new ways to test, deliver code, bring out analytics, or build out the DevOps culture. Find what needs to be tested and bring automation to it, what automation can be used to increase delivery of the product, or maybe find ways to speed up workflow effectively. Sometime old implementation needs to be taken out and completely reworked because it did not fit within the agile scope of the project. Or old code, tests, tools, or process need to get revisited and upgraded to better fit future planning. Even time needs to

be spent writing up documentation about different way to test, explain what certain features do, build out a roadmap, or anything else that explain in detail about the product that can be used by the different groups involved.

2. **“After figuring out what needs to be happen, how do I even begin working on an automated test or automated pipelines or combining the two?”**

   This is another hard question that a lot of beginning and aspiring Automation Engineers face and continue to face with each new project. Breaking down the task into smaller tasks can help provide steppingstones towards speeding up the creation or integration of a test or pipeline. Figuring out what the purpose of the task, where this task needs to live, what resources are needed to host the task (if it needs a VM, use a WebApp from Azure Portal, be setup in ISS, or just exist on your local desktop), and what tools may be needed to accomplish this will help guide towards a good starting point. Sometimes the first step is the hardest, but it’s important to realize that nothing must be static, and it’s healthy to make beginning mistakes to strengthen yourself for future tasks. Also, when applicable, go back and build upon previous tasks to improve upon them to make them better, scalable, more robust.

3. **“After finishing one task, what do I do next?”**

   Figuring out what to do next can always be a tricky question. It is easy to move on and forget what was just implemented, thinking that the task is completed and never needs to be revisited but that cannot be correct. After a task is complete whether it is a new tool, process, improvement, or test it continuous needs to be iterated upon to maintain its upkeep and ensure that it is performing accurately. Looks for ways to expand upon what was just built and figure out how to use it If an automated test was created, test it in all kind of environments and start getting information out of the created test. If an automate pipeline was created, start using the pipeline and see what it’s doing well and where it can be improved. Never complete one task and think it does not need to be revisited, each task should always be iterated upon into a better version of itself. Another thing to keep in mind, is that the next task could be the next step of the development cycle and how-to best build upon what’s there and what needs to be created.

6 Conclusion

To reiterate, test automation is the process of taking manual ran tests and automating them though tools or applications and that they can be ran from either a button click or through a shell. Where the created test automation should run faster, accurate, maintains or improves the quality of the tests. Additionally, test automation can achieve testing that that manual testing cannot such as different types of performance testing, network testing, or testing in parallel. Whereas DevOps is a multitude of different tools, practices, and cultural shift made throughout the company. This could include building out a more robust CI/CD process, improving the agile process, and bringing the DevOps culture to all groups involved with the product. As companies move more towards automation, the test automation and DevOps practices continue to play important roles throughout product lifecycles with the goal to better serve the customer.

Test automation helps take away manual repetitive tests and turn them into s test suite that can be ran from a push of a button or easily from a Shell. Where the goal of test automation is to increase the quality of the product through increase test coverage to help create a higher quality product for the customer. Before creating and implementing test automation, it is important to look for where manual testing exists, where there’s a lack of testing, carefully create a test plan, and find tools that will work well with the product’s development stack. Putting the time into planning out how and what tests to automate should be just as important as the creation and implementation of those tests. Also, after automating a test it is important to continuously build upon and refine those tests.

DevOps is the implementation of tools, practices, beliefs, and ideas that evolve the culture to bring awareness for each group, involved with the product, about the product’s development and lifecycle.

DevOps's culture is about find the best way to covey the different parts of the product development so that it gets all the different group to be involved with most (if not each) aspects of that product. They can be accomplished through different ways such as:

- Creating automated pipelines in Azure DevOps and Jenkins to perform automated testing.
- Creating deployment and release pipelines to setup managed test environment or push new code to production
- Set up gated check-ins to ensure code quality using SonarQube
- Analyze product performance through Pingdom and New Relic

Building out a robust DevOps culture is crucial; it allows all parts of the agile sprint to be applicable to each group involved. Where developers, QA, tester, product, managers, and anyone else can be exposed to the product's roadmap. Where the roadmap can be broken down into tasks then spread across different sprints to provide continuous implementation and development to the production product.

Beginning and aspiring automation engineers face all kinds of tasks and challenges, and it can be complicated finding the answer to solve the task or overcome the challenge. I continue to face those challenges myself, which is why I broke down my problem-solving process into a few questions. The first question was figuring out how to begin. Before the creation and implementation of automation it is important to understand how the automation will better serve the customer. Get involved with the product and understand what's problem it's trying to solve, how it functions, and where pain points exist for each group involved with the products development. Find tools that will help introduce new ways to improve quality, deliverability, show more metrics, or ways to improve the DevOps culture. The second question was how to break down a task. It is easy to get lost in planning and the semantics of the problem, but it's important to understand that the solution is going to be an iterative process. Interact with the different groups involved with the product and find extra information to help figure out the problem that is trying to be solved with automation. From there break the task into its simplest form and find what needs to happen first and then build upon that until the automation is complete. The last question I discussed was determining how to build upon an existing tool, practice, or application. Find where the existing tool, practice or application is lacking by talking to the people who use it the most. This can help expose the weaknesses that will then show how it can be improved.

References

1. McMeekin, Kyle. “Test Automation vs. Automated Testing: The Difference Matters.” QASymphony. https://www.qasymphony.com/blog/test-automation-automated-testing/ (accessed June 15th, 2019)

2. McKenzie, Cameron “5 Java test frameworks and tools JDK developers must know” The Server Side. https://www.theserverside.com/video/5-Java-test-frameworks-and-tools-JDK-developers-must-know (accessed June 16th, 2019)

3. JMeter. “Regression Testing With JMeter – Learn How“ https://www.blazemeter.com/blog/regression-testing-with-jmeter-learn-how/ (accessed June 17th, 2019)

4. Software Testing Fundamentals. “Test Script” http://softwaretestingfundamentals.com/test-script/ (accessed June 17th, 2019)

5. Mexazros, Gerard. “xUnit Test Patterns: Goals of Test Automation” Pearson. http://www.informit.com/articles/article.aspx?p=759702&seqNum=3 (accessed June 17th, 2019)

6. Mueller, Ernest. “What Is DevOps?” The agile admin. https://theagileadmin.com/what-is-devops/ (accessed June 17th, 2019)

7. Mueller, Ernest. “A DevOps Manifesto” The agile admin. https://theagileadmin.com/2010/10/15/a-devops-manifesto/ (accessed June 17th, 2019)

8. Tetanich, Amanda. “Test Environment Benefits and Best Practices” omatic https://omaticsoftware.com/blog/benefits-of-a-test-environment (accessed June 19th. 2019)

9. Brown, Jordan. “The Importance of a Test Environment” Edge Testing. http://www.edgetesting.co.uk/news-events/blog/the-importance-of-a-test-environment (accessed June 19th, 2019)

10. 99tests “The Importance of Web Application Health Monitoring” 99tests . https://99tests.com/blog/the-importance-of-web-application-health-monitoring/ (accessed June 18th, 2019)

11. WineHQ https://www.winehq.org/ (accessed June 18th, 2019)

12. Pataki, Daniel. “How to use Pingdom to Measure (and Monitor) the Speed and Performance of a Website – Effectively!” WinningWP https://winningwp.com/pingdom/ (accessed June 19th, 2019)

13. Bailey, Joe. “Good Automated Tests are Critical to Agility” OSG IT Solutions. https://blog.osgcorp.com/automated-testing/good-automated-tests-critical-agility/ (accessed June 19th, 2019)

14. Buchanan, Ian. “Agile and DevOps: Friends or Foes?” Atlassian https://www.atlassian.com/agile/devops (accessed June 19th, 2019)

15. “4 ways to Automate Testing in a DevOps set-up“ Cigniti Trvhnologies. https://www.cigniti.com/blog/4-ways-to-automate-testing-in-devops-set-up/ (accessed June 19th, 2019)

16. Moragn, Herman. “Scalability in the Age of DevOps: A Must for Success” DevOps.com https://devops.com/scalability-in-the-age-of-devops-a-must-for-success/ (accessed June 19th, 2019)

17. Wilsenach, Rouan. “DevOps Culture” MartinFowler.com https://martinfowler.com/bliki/DevOpsCulture.html (accessed June 19th, 2019)

18. https://github.com/nant/nant

19. “Adhoc Testing: A Brief Note With Examples” testbytes https://www.testbytes.net/blog/adhoc-testing/ (accessed August 13th, 2019)