St. Cloud State University

theRepository at St. Cloud State

Culminating Projects in Computer Science and Information Technology

Department of Computer Science and Information Technology

12-2020

DevOps impact on Software Testing Life Cycle

Jigar Singh
singhjigar1391@gmail.com

Recommended Citation

Singh, Jigar, "DevOps impact on Software Testing Life Cycle" (2020). *Culminating Projects in Computer Science and Information Technology*. 36.
https://repository.stcloudstate.edu/csit_etds/36

DevOps impact on Software Testing Life Cycle

by

Jigar Singh

A Starred Paper

Submitted to the Graduate faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Science in

Computer Science

December, 2020

Abstract

DevOps is a software development practice where the focus is on automating repetitive processes [1]. It has brought a change in the way organizations develop and deliver software products. The main purpose of this paper is to investigate the impact of DevOps on the Software Testing Life Cycle (STLC). There is a lot of ambiguity and confusion as to what is DevOps and how it is practiced and implemented and what change it has brought to the software development and the testing process. In this paper, I have investigated how DevOps has benefited the testing process through automated execution of unit, integration, and workflow tests in the build pipeline. This was achieved through a literature review of studies on test automation and with the help of a case study which is used to list the qualitative benefits of Continuous Testing. The results of the case study show that DevOps has qualitatively benefited the software testing process and has shifted the testing process to earlier phases of the software development cycle.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

This chapter presents information about DevOps and how the papers is structured and what to expect from the paper. The process of developing a software has gone through many changes over the last decade. None more significant than the introduction of DevOps [1] to the fold. The idea of bringing in end-to-end automation in software development and delivery has caught the imagination of the Information Technology world. There have been many studies on how it has changed the way software is developed and tested. This paper is an attempt to understand its impact on the Software Testing Life Cycle (STLC).

What is DevOps

The processes that combine to make up DevOps are a broad set of principles and Continuous Testing is one of those. DevOps is based on lean and Agile method of software development and it gives an advantage by quickly incorporating customer feedback into the delivery cycle and helps businesses adapt faster to new technologies.[2]

Benefits of DevOps

High Velocity. Under the DevOps model teams can add updates to the software at higher speeds than as compared to Waterfall model. Which helps gain business advantage capturing the market faster.

Reliability. It ensures the quality of application updates as the practice of Continuous Integration and Continuous Delivery are used to test each update is functionally safe to release.

Scalability. It is easier to scale the application in the DevOps model as most of the processes are automated and change of systems can be achieved efficiently with reduced risk.

Enhanced Collaboration. The DevOps approach helps to build better functioning teams and gain a competitive edge.

The motive of this paper is to investigate and clear some of the confusion that exists in the industry today as to what is DevOps and what is the change that it has brought to the software development process. As part of that effort to investigate how DevOps has changed the software development process my focus is on the testing cycle. Over the course of this paper, I have performed literature review on DevOps, impact of test automation on software quality and cost benefit analysis of test automation to find the areas of future work and to analyze what is missing. Then with the help of the case study I have attempted to clear the air over what are the tools and practices in DevOps and how it has changed the way we test software today.

As part of this approach to analyze the impact of DevOps practice on the STLC, I start with a differentiation between Software Development Life Cycle (SDLC)[3] and Software Testing Life Cycle (STLC). After which I discuss how Continuous Testing has changed the way software is tested and how there is a shift in the quality gates from the end of the software development cycle to the left of it. This is done in Chapter 2.

To analyze the change in software testing process I look at the qualitative benefits and the quantitative benefits of automation particularly software test automation in Chapter 3 by doing a literature review. The literature review of studies done on cost benefit analysis of test automation throw a problem where Continuous Testing can be used as an answer. In Chapter 4 with the help of a case study, I try to visualize the change in the STLC brought by the DevOps practice and implement a CI pipeline where Unit Test and Service Tests are run for every instance of code commit to GitHub. At the end of this paper, Chapter 5, offers the conclusion.

Chapter 2: Continuous Testing

This chapter narrows down our scope to one key concept of DevOps which is Continuous Testing. The DevOps process is automation driven and automating the execution of tests on code release to source code management system can be characterized as the Continuous Testing process. In first half of the chapter, I will describe where Continuous Testing sits in the Software Testing Life Cycle and in the second half of the chapter, I will theorize the impact of DevOps on the Software Testing Life Cycle by discussing the Shift Left movement.

SDLC vs STLC

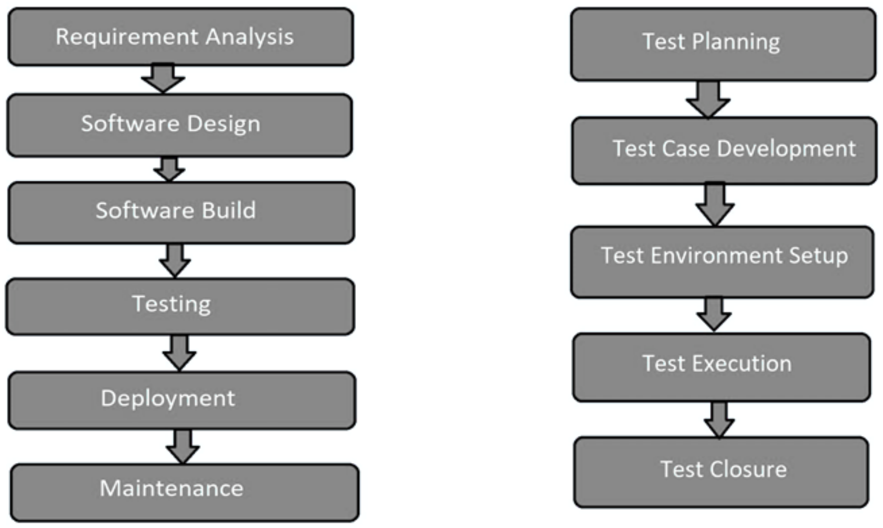

Figure 1. SDLC vs STLC phases

SDLC [4] depicts the phases involved in developing a piece of software and releasing it to the customers. And traditionally STLC has been a subset of SDLC and begins after the development phase ends. But with DevOps the major shift has been involvement of quality in all phases of the SDLC for which a good analogy will be that SDLC and STLC begin parallelly in DevOps practice which is based on the Agile model of developing quality subsets of the whole functionality in iterations. For e.g., Requirement Gathering phase of the SDLC which can be mapped to the Backlog Grooming [3] phase in Agile, traditionally only focused on gathering requirements for developing the software but under DevOps and Agile during Backlog Grooming even scenarios for how the particular functionality will be tested are gathered and documented. This helps the developer in writing Unit tests and Service tests with maximum code and functionality coverage thus bringing quality gates at early stages of SDLC and have the STLC start along with the SDLC.

Now as testing the functionality has also become part of the Development phase of the SDLC this change brought to the STLC is termed as Continuous Testing which we will have a look at in the next section.

Continuous Testing

Continuous Testing is the process where automated test execution is a part of the CI/CD pipeline. The motive of this process is to gain continuous feedback on the quality of software being delivered with every release. It evolves and extends test automation to address the increased complexity and pace of modern application development and delivery under DevOps approach.

Continuous Testing can be summed up as providing the right feedback to the right stakeholder at the right time. Traditionally testing was delayed to the end of the development cycle. Which would create a situation where either time or quality gets compromised. With Continuous Testing, it provides feedback earlier in the development cycle which helps the team deliver better quality software in stipulated amount of time.

DevOps is all about releasing differentiating software as efficiently as possible Continuous Testing process is part of the DevOps approach of Continuous Integration and Continuous Delivery and has brought a change to the traditional STLC, we can now deep dive into what involves Continuous Testing. The test cases that are considered as less flaky and can be effectively run consuming less resources. In the coming section we will see why Continuous Testing has an advantage over the traditional model of running User Interface (UI) regression tests as part of the Testing phase of the waterfall approach. We will start first by looking at the Test Pyramid.

The Test Pyramid

Mike Cohn came up with this concept in his book Succeeding with Agile [5]. It is a great visual metaphor telling you to think about different layers of testing. It also tells you how much testing to do on each layer.

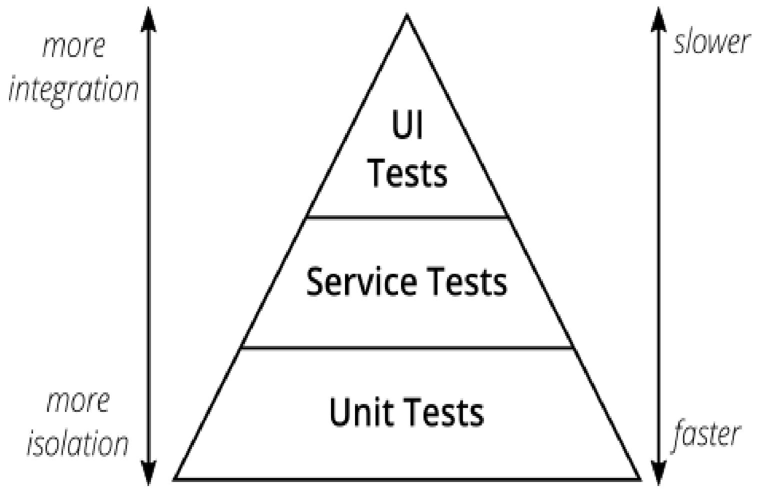

Figure 2. Test Pyramid

The test pyramid serves as a good rule of thumb when it comes to establishing a test suite. Mike Cohn's original test pyramid consists of three layers that your test suite should consist of (bottom to top):

- Unit Tests
- Service Tests
- User Interface Tests

Unit Tests

Taking from his work we can create a good comparison on how testing has evolved and in the Modern approach (Figure 4) the Unit Tests are the base and rightly so. The Unit Tests are less resource heavy and reduce the probability of a defect being passed on to latter stages of development and integration.

Service Tests

Service tests verify that different modules or services used by your application work well together. For example, it can be testing the interaction with the database or making sure that microservices work together as expected. These types of tests are more expensive to run as they require multiple parts of the application to be up and running.

User Interface (UI) Tests

Also known as GUI testing, UI testing is the process of testing the visual elements of an application to validate whether they accurately meet the expected performance and functionality. By testing the GUI, testers can validate that UI functions are free from defects. It involves testing all visual indicators and graphical icons, including menus, radio buttons, text boxes, checkboxes, toolbars, colors, fonts, and more. We want this because user interface tests often have the following negative attributes:

- Brittle. A small change in the user interface can break many tests. When this is repeated many times over the course of a project, teams simply give up and stop correcting tests every time the user interface changes.
- Expensive to write. A quick capture-and-playback approach to recording user interface tests can work, but tests recorded this way are usually the most brittle. Writing a good user interface test that will remain useful and valid takes time.
- Time consuming. Tests run through the user interface often take a long time to run.

Traditional vs Modern Approach

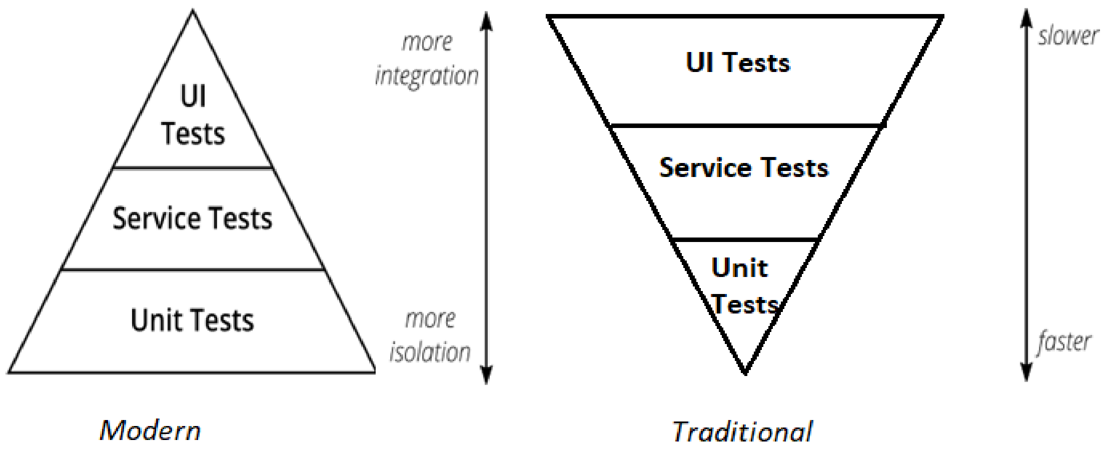

Figure 3. Modern vs Traditional approach

Traditional Approach

In general, before DevOps concept came in, test automation was focused more towards automating the UI Tests which were basically regression test cases that were repetitively being performed manually. Those tests are called Functional Automated Tests and replicate the behavior of the user. UI Tests in general are resource heavy, flaky, and maintaining them is expensive in terms of the hours of work that goes into updating a test and how frequently those updates are required. The flipped test pyramid as seen in Figure 4 shows how the service tests and the unit tests received less attention when it came to automation as the concept of continuous testing had not evolved then.

Modern Approach

As DevOps principles and practices picked up pace the industry moved towards a more sustainable and effective approach of continuous testing where the unit tests are run for every branch merge along with the integration/service tests. This approach where automation of tests script is done at a more predictable level- the code level- has resulted in better product delivery and reduction in the number of bugs being reported in the later environments, which are more expensive to fix. Also, in the modern approach there is less focus on the automation of the dynamic side of the product that is the UI. The UI or the exploratory tests are left in the hands of the experienced individuals who have an eye for detail and their expertise can be used for User Acceptance Test (UAT).

Due to the shift in approach from the Traditional way of test automation towards the more modern way where the focus now is on automating execution of less resource heavy and predictable unit and integration tests, there is a movement developing in the industry. We will discuss about this Shift Left movement in our next section.

Shift Left Movement

The shift left testing movement is about pushing testing towards the early stages of software development. By performing tests early in the development cycle and performing them

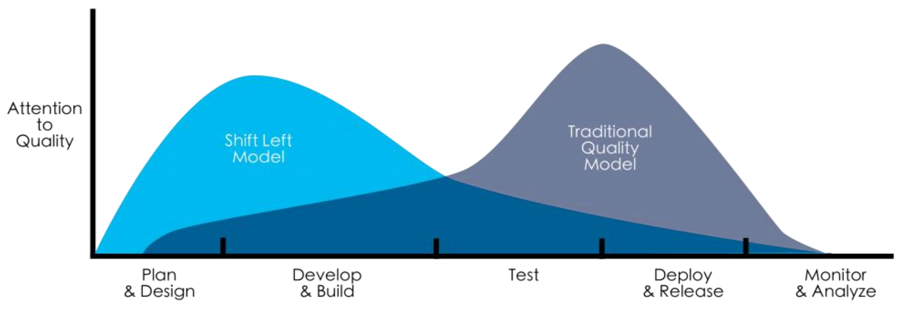

Figure 4. Shift Left Movement [6]

regularly can reduce the number of defects in the software application being developed. Design issues, coding error and architectural errors can be caught and solved early. Shift left is more about problem prevention than problem detection [6].

Shifting Left and Agile

Under Agile model of software development, testing every story before it is merged is essential and when compared to the traditional model of software development where testing occurred after development was done, in Agile model “shift-left” movement means having a quality driven approach during backlog grooming and development [3] scrum phases as well.

In the Agile methodology of software development testing the functionality being developed in the short cycle is an integral part of the sprint which is generally 2-3 weeks long. The quality gates can be shifted further left by using static code analysis tools which perform pattern-based analysis of code flag potential bugs and quality issues. A static analysis tool helps to identify problems with parameter types or incorrect usage of interfaces [7]. Thus, improving the overall product quality.

As seen in this chapter, there is a paradigm shift in how tests are written and executed. In the grand scheme of things, it can be said that DevOps has changed how we test software.

Chapter 3: Cost Benefit Analysis

This chapter is composed of a literature review, the problem description arising from it and the possible solution. The topic for literature review is cost benefit analysis of test automation. I will critically analyze the studies in this area and make a point for inclusion of Continuous Testing in the cost benefit analysis.

Literature Review

The software industry, over the last decade or so, has been moving towards test automation [8] and Agile method of software development has brought to the fore the advantage of having a test automation framework in place for effective collaboration. Many developers work in tandem to develop a particular application or let us say a particular feature in case of Agile. All the developers push their code through local branches. These local branches need to be passed through a quality gate that will ensure the standard of quality being maintained all over the project and for all local branches. The DevOps practice has changed the process of testing an application under Agile model of software development. What has been the impact and how to quantify it?

Dobles et.al in [9] discussed the positive impact on cost and quality that appear after automating a set of test cases, the context was two different set of test cases and regression test cases. The result showed that initially the cost of setting up the automation scripts and executing the scripts is equivalent to manually testing the test cases, thus no significant gain. But then when the Software Under Test (SUT) is in production and the automated tests are run for regression scenarios there is significant benefit in cost and quality during the STLC. This could be expanded to take into consideration the Unit Tests and the Integration Test/Service Tests that are run as part of all local branches under the DevOps practice via build management tool thus achieving better automation and feedback loop.

Another study by Divya Kumar and KK Mishra in their paper [10], focuses on positive effects of test automation and gives a cost benefit analysis in scenario of regression testing over multiple versions of the software being released incrementally. It also provides mathematical equations to compute the advantage of automated tests over manual tests. The authors have attempted to enumerate test automation’s impact on software’s cost, time and quality. The study clearly establishes the benefit of having a test automation regression suite over not having one. This study could be enhanced by inclusion of Continuous Testing approach and estimating its impact on cost, quality, and time of software product delivery.

The industry is moving more towards adopting Agile [11] as the model of software development and DevOps as the preferred practice of software delivery. “Quality deliveries with short cycle time need a high degree of automation” [1]. So there needs to be research to measure quantitatively the positive impact of Continuous Testing (part of DevOps) on the STLC. Having a mathematical model to quantify the benefit gained from setting up a test automation framework for Unit tests, Service tests and the UI tests discussed under the Test Pyramid [5] as part of the DevOps approach of continuous testing and having a continuous feedback on the quality of code being developed will help management make informed decisions on how to enhance the quality of software that is being delivered.

Problem Description

A lot many studies have been done proving the positive impact of test automation on the cost and delivery time of the product. But there is also a need to include the automation of test execution which happens in the DevOps build pipeline and the impact of unit test and service tests in the overall cost benefit analysis of software test automation. Case in point, software test automation is much efficient and cost effective when unit tests and service tests provide the maximum code and functionality coverage.

Proposed Solution

In this paper I will attempt to theorize the cost benefit of integrating the test execution of unit and service tests in the software delivery pipeline using the DevOps concept. This will provide a holistic view of the effectiveness of test automation and how it is the way forward to achieve better quality of software. It will be done by developing a Currency Converter software application incrementally and writing unit and service tests for complete code coverage. It will make sure the code being committed is of the highest quality and hence ensure overall software quality when executed automatically through the DevOps build pipeline in Jenkins.

Chapter 4: Case Study: DevOps impact on STLC

As we have discussed in the earlier chapters that DevOps approach and the Agile model have brought a paradigm shift in how organizations are developing and delivering software products, as part of this chapter we will be looking at how that change can be visualized and how the STLC is different in the DevOps world. The aim of this chapter to is to utilize the DevOps tools and create a Continuous Integration Process Flow do discuss the qualitative benefit of the DevOps approach of software development from the standpoint of STLC.

As part of this case study, I created a CI pipeline for the development of a Currency Converter software which will be developed in two cycles. The cycles resemble features [12] in Agile and are developed over a period of 2 weeks each. The project can be found on GitHub at this URL: https://github.com/jahapanah/Currency_Converter

Feature1(F1): Create a basic currency converter

Feature2(F2): Add the functionality to access audit history record of the conversions.

Each feature required two user stories to deliver the functionality. One to develop the backend and the other to develop the front-end.

Feature 1- Story 1(F1S1): Develop the backend of the converter functionality

Feature 1- Story 2(F1S2): Develop the frontend of the converter functionality

Feature 2-Story 1(F2S1): Add audit history functionality backend infrastructure

Feature 2- Story 2(F2S2): Add audit history functionality frontend infrastructure

While developing the currency converter software I took the DevOps and Agile approach to display the qualitative advantage gained while testing as compared to the Waterfall model approach. I had a pipeline configured for Jenkins build that will run the build remotely and provide real time feedback on the quality of code being developed thus “shifting left” the STLC towards the development phase.

Cycle 1/Feature 1: The basic currency converter is developed

- Story 1: Develop the backend of the converter functionality: As part of this story the backend functionality is developed with Java as the language and the Spring Boot framework. As part of this user story, I developed the backend and committed the code from my branch to GitHub. The GitHub webhook triggered the Jenkins build for my pull request. As part of the build the unit tests were run assuring the code quality is maintained.
- Story 2: Develop the front end of the converter functionality: As part of this user story the frontend i.e., the User Interface (UI) was developed. Along with the UI service tests were written as part of the test automation framework to verify that the functionality is working as expected. The code when committed to GitHub again triggered the webhook on pull request and the build was run with unit tests and service tests thus incrementally testing the application for quality as it is being developed.

Cycle 2/Feature 2: Add audit history functionality for tracking conversions

- Story 1: Develop the backend of the audit history functionality: As part of this user story the backend code is written and committed to GitHub and via webhook the Jenkins build is run. Also, the unit tests, service/integration tests are run assuring code quality and regression testing the previous functionality is still working as expected and the addition of a new feature has not broken the application.

- Story 2: Develop the frontend of the audit history functionality: Once the backend development is complete the frontend is developed with the UI for accessing the audit history of the currency conversion transactions. The service/integration tests are written to update the existing automation framework. Once the code is committed to GitHub the webhook triggers the Jenkins build process and the code is tested for quality.

Continuous Testing Process Flow

The process of software development over the two cycles viz Cycle 1 and Cycle 2 was aided heavily with the Continuous Testing approach as depicted in Figure 5. During story development after each commit the build was triggered to give a continuous feedback on what could have been broken due to addition of the new functionality. In industry projects this a very

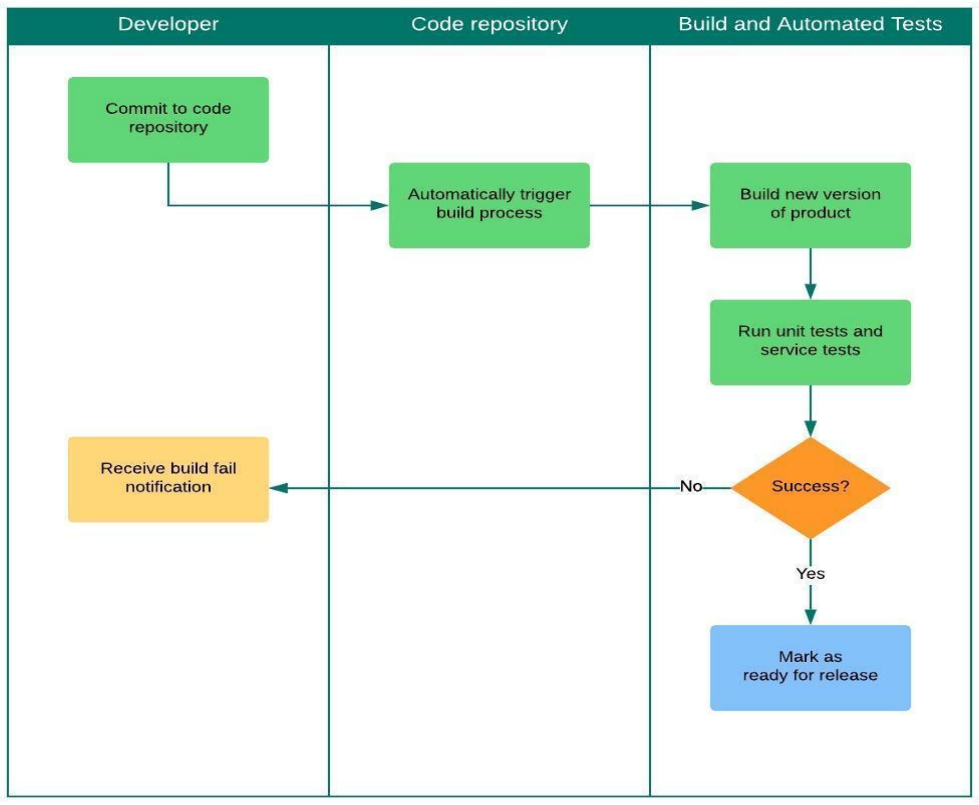
Developer
Code repository
Build and Automated Tests
Commit to code repository
Automatically trigger build process
Build new version of product
Run unit tests and service tests
Receive build fail notification
No
Success?
Yes
Mark as ready for release

Figure 5. Continuous Testing process flow

helpful process as even a small change in a line of code in an upstream project can break many functionalities in downstream projects.

While developing F1S1 only the unit tests for the controller and the service was run as part of the build process then while developing F2S2 the service tests were added which were again run along with the unit tests as part of the Jenkins build thus ensuring overall code quality at an early stage which would not have been possible in a traditional Waterfall based model as there could have been integration defects in developing the basic currency converter functionality which would only have been unearthed after all development was completed for the F1 and the product would have been tested after all software development deliverables were signed off.

Similarly, while developing F2S1 and F2S2 once the unit tests were added for the functionality and code was committed, the Jenkins build process triggered the execution of unit tests and service tests which helped regression test the F1 functionality already existing in the environment. The overall process can be summed with the help of table below.

Table 1. Continuous Testing matrix

F1S1	Unit tests run through the build process
F1S2	Unit tests and Service Tests run through the build process.
F2S1	Unit tests of F2S1 run and Unit and Service tests of F1S2 run as regression tests to ensure F1 is working as expected during the build process
F2S2	Unit tests, Service tests of F2S2 run to ensure F2 is working as expected and Unit and Service tests of F1S2 run as regression tests to ensure F1 is working as expected during the build process

Chapter 5: Conclusion

The Continuous Testing process brought in by the DevOps approach has benefited the quality of software being developed. I attempted to investigate that by performing a case study and by simulating industry work environment under the DevOps setup using build and automation tools. My observation is that it was significantly better to develop the software with the Jenkins CI build pipeline running automated tests for every build and verifying the code quality. Also, the impact of DevOps on STLC is prominent as the testing phase has now moved in the development phase of SDLC and testing has become more automated under DevOps.

This study can be used by organizations to understand how DevOps can be utilized to affect the quality of software being developed by changing the STLC. Moving quality gates left in the software development lifecycle is the way forward.

References

[1] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016.

[2] P. Perera, R. Silva, and I. Perera, “Improve software quality through practicing DevOps,” in *17th International Conference on Advances in ICT for Emerging Regions, ICTer 2017 - Proceedings*, 2017.

[3] M. Mahalakshmi and M. Sundararajan, “Traditional SDLC Vs Scrum Methodology – A Comparative Study,” *Int. J. Emerg. Technol. Adv. Eng.*, vol. 3, no. 6, pp. 2–6, 2013.

[4] “Difference between SDLC and STLC - GeeksforGeeks.” [Online]. Available: https://www.geeksforgeeks.org/difference-between-sdlc-and-stlc/. [Accessed: 09-Dec-2020].

[5] C. Ashbacher, “Succeeding With Agile: Software Development Using Scrum, by Mike Cohn.,” *J. Object Technol.*, 2010.

[6] “What Is Shift Left Testing? A Guide to Improving Your QA.” [Online]. Available: https://www.testim.io/blog/shift-left-testing-guide/. [Accessed: 09-Dec-2020].

[7] A. Trautsch, “Effects of Automated Static Analysis Tools: A Multidimensional View on Quality Evolution,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 184–185.

[8] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects,” in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 426–437.

[9] I. Dobles, A. Martínez, and C. Quesada-López, “Comparing the effort and effectiveness of automated and manual tests,” in *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, 2019, pp. 1–6.

[10] D. Kumar and K. K. Mishra, “The Impacts of Test Automation on Software’s Cost, Quality and Time to Market,” in *Procedia Computer Science*, 2016.

[11] Digital.ai, “14th annual STATE OF AGILE REPORT,” *Annu. Rep. STATE Agil.*, vol. 14, no. 14, pp. 2–19, 2020.

[12] N. Kurapati, V. S. C. Manyam, and K. Petersen, “Agile Software Development Practice Adoption Survey,” in *Agile Processes in Software Engineering and Extreme Programming*, 2012, pp. 16–30.

[13] H. V. Gamido and M. V. Gamido, “Comparative review of the features of automated software testing tools,” *Int. J. Electr. Comput. Eng.*, vol. 9, no. 5, pp. 4473–4478, 2019.

[14] A. Mustafa, W. M. N. Wan-Kadir, and N. Ibrahim, “Comparative evaluation of the state-of-art requirements-based test case generation approaches,” *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 7, no. 4–2 Special Issue, pp. 1567–1573, 2017.

[15] A. Stouky, B. Jaoujane, R. Daoudi, and H. Chaoui, “c,” in *Big Data Technologies and Applications*, 2018, pp. 87–96.

[16] B. Gonen and D. Sawant, “Significance of agile software development and SQA powered by automation,” *Proc. - 3rd Int. Conf. Inf. Comput. Technol. ICICT 2020*, pp. 7–11, 2020.

[17] A. Bertolino, C. Town, and C. Town, “Annotated Buzzwords and Key References for Software Testing in the Cloud,” no. May, 2017.

[18] J. G. Quenum and S. Aknine, “Towards Executable Specifications for Microservices,” in *2018 IEEE International Conference on Services Computing (SCC)*, 2018, pp. 41–48.

[19] T. Laukkarinen, K. Kuusinen, and T. Mikkonen, “Regulated software meets DevOps,” *Inf. Softw. Technol.*, vol. 97, pp. 176–178, 2018.

[20] C. H. Kao, “Continuous evaluation for application development on cloud computing environments,” in *2017 International Conference on Applied System Innovation (ICASI)*, 2017, pp. 1457–1460.

[21] L. Riungu-Kalliosaari, S. Mäkinen, L. E. Lwakatare, J. Tiihonen, and T. Männistö, “DevOps Adoption Benefits and Challenges in Practice: A Case Study,” in *Product-Focused Software Process Improvement*, 2016, pp. 590–597.

[22] D. Stahl, T. Martensson, and J. Bosch, “Continuous practices and devops: beyond the buzz, what does it all mean?,” in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp. 440–448.

Annotated Bibliography

[1] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016.

The editor of this article talks about the processes that define DevOps approach and gives a brief introduction to the tools used for build, CI and deployment of software products. The authors agree that DevOps is a cultural shift from more traditional way of software development and enhances the Agile methodology of software development by automating most of the processes. The benefit of the DevOps way of software development is that updates can be brought faster to the market and thus increasing the overall productivity.

I have utilized this paper to compare what tools I could use for the case study. The authors have presented a tabular comparison of the different build, deployment and operations tools that are used in the industry. As the Continuous Testing process is part of the build phase of the DevOps cycle, Maven and Jenkins suited my requirements of the case study.

The paper also introduces microservices and how companies are moving from large monoliths to many smaller microservices which can be built and deployed independently giving flexibility and agility to cater to market needs. This falls under the umbrella of cultural and architectural shift discussed in other papers as well.

[2] P. Perera, R. Silva, and I. Perera, “Improve software quality through practicing DevOps,” in *17th International Conference on Advances in ICT for Emerging Regions, ICTer 2017 - Proceedings*, 2017.

Perera et al. in this paper have studied the impact of DevOps in software's quality. Their conclusion is that there is a positive relationship between DevOps and the improvement of software quality. The results of the survey conducted by them on working professionals in Sri Lanka displays significant correlation between practicing the DevOps culture, automation, measurement and sharing initiatives and improvement in the quality of software.

The findings of this paper rally with the case study I performed as the practice of DevOps improved the quality of builds buy introducing integration and unit test execution as part of the Jenkins build pipeline. This benefitted in two ways. First by reducing the manual effort of testing regression scenarios and second by giving a feedback on the quality of code in the early phase of software development lifecycle.

The caveat pointed out in this paper regarding Automation and the framework has been rallied in other research work as well. The authors have indicated that Return on Investment (ROI) is a critical factor in determining whether to go with test automation as part of the DevOps process or not. Some test frameworks are difficult to maintain without dedicated technical resources and this is an additional cost factor to consider when deciding whether Continuous Testing will be a good call for a particular project.

[3] M. Mahalakshmi and M. Sundararajan, “Traditional SDLC Vs Scrum Methodology – A Comparative Study,” *Int. J. Emerg. Technol. Adv. Eng.*, vol. 3, no. 6, pp. 2–6, 2013.

The authors in this paper have discussed which model serves the best purpose for software development. The models compared are the traditional SDLC model called the Waterfall model and the Agile model called SCRUM. The authors have listed the advantages and the disadvantages of each model and although the disadvantages of the Waterfall model outweigh its advantages it is still widely used because of the simplicity. On the other hand, the

SCRUM model is complex and requires higher team collaboration but also provides more flexibility and room for innovation.

This paper was helpful to understand how the Agile model of SCRUM is efficient for developing a product which needs constant updates. This is what DevOps is built upon, having a continuous integration (CI) and continuous delivery (CD) pipeline built to rollout updates as efficiently and quickly as possible. This paper also touches upon how the Software Development Cycle is different for the Agile when compared to the traditional Waterfall model as Quality of software being delivered varies on in the Agile model depending upon time and cost variables.

The authors have given a terse comparison which can be used by the industry managers and other software companies to decide upon what is the best software development model that would suit their needs based on cost and time to market.

[4] D. Kumar and K. K. Mishra, “The Impacts of Test Automation on Software’s Cost, Quality and Time to Market,” in *Procedia Computer Science*, 2016.

Kumar et al. have formulated mathematical models to quantify the impact of test automation on software’s cost, quality and time of delivery. They have noticed through the mathematical models that automation of regression test cases benefits the software delivery process and improves the quality of software being delivered.

In this paper the authors have proposed a solution to better measure the impact of test automation on software’s cost, quality and time to market. They indicated future work can include other factors like automatic test data generation, which is part of service and integration tests run in the build pipeline. With my case study I included other factors like service and integration tests run through a build pipeline which further reduce costs and time by reducing manual work. The authors also concluded that test automation benefits the entire testing process and building on that the shift left movement and testing for every build helps to better the testing process further.

[7] A. Trautsch, “Effects of Automated Static Analysis Tools: A Multidimensional View on Quality Evolution,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 184–185.

The author of the paper has attempted to evaluate the effects of Automated Static Analysis Tools (ASATs) like FindBugs on the quality of software. The ASATs help the developers find bad coding patterns and possible pitfalls. These tools can be used in Continuous Integration pipelines to detect early code issues which can turn into defects. This helps the DevOps approach of moving testing in the early phase of software development.

Static analysis tools help move the quality gates towards the left by pointing out potential code pitfalls which may result into bugs and defects later. The author has also touched upon the code quality of the test scripts written for automated test execution. This area is relatively ignored when it comes to static analysis of code quality. It affects the overall cost and the quality of software being delivered to the market and can be included as a factor in other cost benefit analysis studies.

[8] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects,” in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 426–437.

Hilton et al. in this paper have studied the usage of Continuous Integration (CI) in open-source projects. They have pointed out that although DevOps and CI are widely used there is comparatively less research in the field that talks about its benefits and cost and usage. The paper is a call for the research community to engage more with this topic. Their study has found out that the usage of CI is growing in the industry since 2000 when it was first introduced. Automation plays a key role in facilitating CI. And that CI will have much greater impact on how software is developed in future.

The three major research questions that the authors studied in this paper are which CI systems are used by developers, how developers use this system and why projects use or not use CI or DevOps. The authors have found that the use of CI is profound in the GitHub open-source project community. The key conclusion from their research is that projects that use CI release product updates twice as often, accept pull requests faster and the developers on those projects are less worried about breaking the builds. Which ensures overall cost benefit as well.

[9] I. Dobles, A. Martínez, and C. Quesada-López, “Comparing the effort and effectiveness of automated and manual tests,” in *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, 2019, pp. 1–6.

Dobles et al. in this paper have compared the effort and effectiveness of manual and automated tests. In their study they found that automation requires initially more effort but that effort cost gets amortized over time and then the automation reaps more benefit. The study overall establishes the benefit of having automation in software quality assurance.

The two main research questions in this paper established that automated testing has an advantage over manual testing on both the effort and effectiveness front. In the long run automated testing requires less effort and is more effective in finding defects. This research further helped my attempt to show the impact of DevOps automation on the software testing process, as automated execution of the integration and unit test ensured better quality of software being developed in an Agile development model.

The conclusion of this industry-based research is consistent with other papers on this topic where the overwhelming evidence of automation benefitting the software's quality and cost of delivery.

[13] H. V. Gamido and M. V. Gamido, "Comparative review of the features of automated software testing tools," Int. J. Electr. Comput. Eng., vol. 9, no. 5, pp. 4473–4478, 2019.

Gamido et al. have compared the various test automation tools that are used in the industry. The comparison is on different parameters like platforms and browsers on which these tools can be used, the ease of learning and the programming skills that are required to use the tools and various other parameters. They concluded that their study is helpful for organizations that are looking to leverage a tool for test automation. This study is helpful to understand which tool can be picked based on the parameters and the software under test.

The testing tools and libraries used in this paper cover a broad range of use cases. For my use case JUnit and Mockito were the testing libraries I utilized when creating the test suite of unit and integration/service tests. Service tests mimic the user behavior by calling rest endpoints through a service client. This helps test the functionality without the added baggage of creating flaky UI tests.

The comparison between different test automation libraries and tools based on criteria like knowledge of programming and licensing can be useful for other managers and industry stakeholders to decide whether automation and more so DevOps will benefit the software testing process for their project.

[14] A. Mustafa, W. M. N. Wan-Kadir, and N. Ibrahim, “Comparative evaluation of the state-of-art requirements-based test case generation approaches,” *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 7, no. 4–2 Special Issue, pp. 1567–1573, 2017.

Mustafa et al. have provided a comparative evaluation of prominent test case generation approaches. Efficient test case generation is key to testing the software application effectively. The authors have considered five evaluation criteria namely, inputs to the test case generation, the transformation techniques, coverage criteria, time and the tool’s support. These criteria are defined for various test case generation techniques considered in the survey. The result of the comparison shows that there is no particular way that fulfills all evaluation criteria and the specification-based approach is more mature and effective compared to the rest.

This paper establishes a comparison between different test case generation techniques and case could be made for the UML based test case generation technique being used for unit test generation in DevOps environment. This could further benefit the cost and time to market of the software by reducing manual effort and intervention. There is room for future work in this area.

[15] A. Stouky, B. Jaoujane, R. Daoudi, and H. Chaoui, “Improving Software Automation Testing Using Jenkins, and Machine Learning Under Big Data,” in *Big Data Technologies and Applications*, 2018, pp. 87–96.

Stouky et al. in this paper discuss how Jenkins can be utilized to run builds which includes automated tests. They consider Jenkins as a testing tool and compare it with Selenium and JMeter which does not appear a valid comparison as Jenkins is just runs builds and does not provide libraries like Selenium and JMeter for writing automation code. Their conclusion seems to propose Jenkins as the solution to test every commit and make testing more effective.

The paper touched many areas including different technologies to write and execute automated tests and inclusion of Big Data but doesn't properly tie them together. There is still room for utilizing Big Data and Machine Learning to highlight recurring failures of builds in Jenkins which can be used to improve the quality of service and unit tests being written for a particular project.

The common theme of the paper is consistent and reiterates the findings of other research work in the area that automated tests can save cost and time to market of the software product.

[16] B. Gonen and D. Sawant, “out,” *Proc. - 3rd Int. Conf. Inf. Comput. Technol. ICICT 2020*, pp. 7–11, 2020.

Gonen et al. in this paper have discussed how Agile software development process has changed the way software is developed. In traditional way the analysis, design, code and test were one of activities, but in Agile they are a continuous activity. The authors also touch open where automation can be applied in the testing techniques in the Agile development process. Their conclusion is that all organizations benefit from Agile and automation and depending upon the requirement the organizations should select an Agile or a non-agile approach for software development.

The authors introduce ‘Agile Genome’ which can be summed as a strategy to select the best way to develop software using Agile methods and available software test automation tools. The authors subdivided the testing in Agile model into 4 quadrants based on different approaches and ability to be automated. The two techniques discussed in the paper are the Test Driven

Development (TDD) Technique and the Behavior Driven Development(BDD) Technique.

In my case study I utilized the TDD approach. The TTD technique to write unit and integration tests helps the developer to deliver quality code by testing the code first fails and then the developer fixes the failing test by writing the correct code.

The authors have concluded that the Agile software development process can be best utilized by using test automation and that there is need for organization to train and develop resources who can implement this model. The paper iterates that the testing process has changed in the last decade with advent of Agile and automation tools.

[17] A. Bertolino, C. Town, and C. Town, "Annotated Buzzwords and Key References for Software Testing in the Cloud," no. May, 2017.

Bertolino et al. in their paper have provided a literature survey on software testing in cloud. They have provided information on different types of cloud and the different tools that provide cloud testing services. In this paper they have found that there is a need to propose a novel testing strategy for software testing in the cloud.

The authors have studied the intensity of work in the field of software testing in cloud over the period of 2011-2016 and have subdivided the study into five main sections: main sources, research intensity, studies, research method, and application tools. This study is a good reference to understand how much work has been done in this field. The authors at the end conclude that there is need for proposal of a novel strategy to test in the cloud. The Jenkins build and test execution can be included in this work and an enhancement to the cause of testing in the cloud. The orchestration of docker containers and running unit and service tests on those containers can be dubbed as a testing in cloud strategy.

There is room for future work that can branch out of this study where Jenkins and Docker can be utilized as the cloud environment to test a particular program build and generate report for the developer whether the code is stable or has defects in it.

[18] J. G. Quenum and S. Aknine, “Towards Executable Specifications for Microservices,” in *2018 IEEE International Conference on Services Computing (SCC)*, 2018, pp. 41–48.

The paper discusses new approach for microservices testing. The authors have used intelligent agents to create specifications for unit and acceptance tests. The agents are used to capture and specify complex behavior through abstraction of the microservices which implement the functionality. The paper recommends to generate tests through the intelligent agents to improve relevancy and reduce ambiguity.

Quenum et al. have highlighted the difficulty in writing automated tests for microservices. The lack of a common dataset due to the data being managed in a decentralized manner and independence of teams involved in building the microservice are some of the challenges in writing automated tests for microservices. The authors suggest intelligent agents to overcome this problem in a Domain-driven Design (DDD) approach and by deriving unit and acceptance tests from formal specifications.

This approach can be combined with DevOps to run the unit tests generated automatically in a build pipeline where the generated unit tests will be executed automatically at the time of the build in Jenkins. The authors have suggested to work on the implementation of this novel approach and build an engine that will derive tests from formal specifications and run them automatically as future work.

[19] T. Laukkarinen, K. Kuusinen, and T. Mikkonen, “Regulated software meets DevOps,” *Inf. Softw. Technol.*, vol. 97, pp. 176–178, 2018.

The paper discusses how DevOps can be integrated for development of regulated software like that of the medical industry. The regulated software development requires more documentation and compliance whereas the DevOps approach based on lean and Agile focuses on quick delivery. The authors have tried to propose a middle ground where DevOps can be utilized for development of regulated software. They have studied the pitfalls in design and inhibitors of DevOps in domain of medical software and proposed how to overcome some of them.

This paper introduced an interesting question in terms of testing software when it comes to compliance standards set by certain industries for e.g. Medical and Health industries. The compliance of medical data standards and combining it with Agile model of software development is a challenging prospect to deliver DevOps automation that will be regulation compliant give the frequent requirement changes that occur in Agile software development model and the rigid Health industry acceptance standards.

The role of Continuous Testing in this process will be critical as it will ensure thorough regression testing is being administered which will reduce the number of defects introduced in the software in the long run and build confidence of the compliance authorities in the Agile and DevOps method of software development.

[20] C. H. Kao, “Continuous evaluation for application development on cloud computing environments,” in *2017 International Conference on Applied System Innovation (ICASI)*, 2017, pp. 1457–1460.

The paper introduces the concept and design of continuous evaluation for application development on cloud environments. Continuous evaluation is another name for continuous testing, and it uses the DevOps principle of automating build and test cycles. The continuous evaluation framework facilitates the evaluation of both the application and the cloud computing environment parallelly as the development task is performed. There can be future work done to implement and evaluates the continuous evaluation framework.

The author has included setting up of a continuous evaluation build pipeline in Jenkins which will run tests and perform similar functions as the Continuous Testing build pipeline in the case study that I presented as part of the paper. The cloud environment development is analogous to building and testing on different docker containers. This is consistent with other papers I studied as part of literature review to find how the DevOps approach along with other technologies has brought a change in the software testing lifecycle and cloud computing can be another variable added to the mix as it has been recently added to the software development world.

[21] L. Riungu-Kalliosaari, S. Mäkinen, L. E. Lwakatare, J. Tiihonen, and T. Männistö, “DevOps Adoption Benefits and Challenges in Practice: A Case Study,” in *Product-Focused Software Process Improvement*, 2016, pp. 590–597.

Riungu-Kalliosaari et al. in this paper have conducted a qualitative multiple case study on benefits and inhibitors of DevOps. The research questions were posed to the representatives of 3 different software development companies in Finland. Some of the perceived benefits were more implemented features and frequent releases, improved quality assurance and enhanced collaboration and communication. Some of the challenges were industry constraints and cultural difference.

Their findings state that DevOps improves frequency of releases and test automation practice. This could add value to the software development process by increasing communication and enhanced occupational welfare. The pitfalls in adoption of DevOps were along human aspects of resistance to change and insufficient technical knowhow. The authors have called for future work to be done so as to determine how does DevOps impact and benefit the end user.

As suggested by the author, there is a need to determine DevOps impact and one of the keyways to measure impact on the end user is to evaluate the quality of software being delivered to the end user through the change brought in by the DevOps process. In my study I have focused on this particular area of DevOps impact where I have attempted to describe the qualitative benefit of DevOps over the traditional software testing process.

[22] D. Stahl, T. Martensson, and J. Bosch, "Continuous practices and devops: beyond the buzz, what does it all mean?," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp. 440–448.

Stahl et al. in their paper have discussed how there is a lasting confusion in the industry on what constitutes DevOps and how it should be practiced. The authors have done a systematic mapping study of literature written on the topic and after review they have come to this conclusion that there is high degree of confusion and contradiction about in the often in the same resource. Hence, they have called for future work to discuss which practices, tools and methods are used together in DevOps and how they are implemented.

The claim in this paper is that DevOps and continuous practices are used interchangeably even though they are inherently different. The authors have taken a three-step approach where they initially claim there is this issue. Then they propose recommendations to bring clarity on the topic and then they propose a set of definitions which help disentangle DevOps and continuous practices from one another.

The paper describes DevOps having a set of values and principles which are utilized in different methods and practices with help of enabling tools. A future work in this area is how all this can be put together to implement a functioning process which is where I have attempted to highlight the benefits of DevOps by implementing the continuous integration and continuous testing bit of the process.