Continuous Testing Integration with CI/CD Pipeline

Junhe Liu

Junhe.liu@carrier.com

Abstract

When the software development community started moving toward Agile methodology from Waterfall by the turn of the century, a critical phase of the software development model - deployment – was still stuck in Waterfall time. From the conflict of Agile and Waterfall deployment practices came the idea of DevOps. At the same time, continuous testing -- automated tests integrated within a CI/CD pipeline -- has taken software testing one step further, and it is a direct benefit of the DevOps transition.

In this article, we give a brief overview of the path from Waterfall methodology to Agile, the rise of the DevOps practice, and the benefit of continuous testing. We also show an example of how to integrate automatic testing into the CI/CD pipeline.

Biography

Junhe Liu is currently a DevOps engineer at Carrier. He has over 30 years of technology and technology management experience. At Carrier, he integrated various automatic tests into the CI/CD pipeline. Junhe Liu studied Computer Science at Portland State University and Stanford University. He's also a co-author of Enterprise Java Developer's Guide.

1. Waterfall methodology

The Waterfall Model originated in the nineteen seventies. In Waterfall, a project consists of a few phases, namely requirement analysis, system design, coding, testing, deployment, and maintenance. Each phase needs to be completed before next phase can start, the cost of going back to a previous phase is prohibitive, hence the name waterfall.

In the requirement analysis phase, a product manager would write a paper detailing market opportunity, product capabilities, dependencies, competitive analysis, etc. This paper is often called product requirement documentation

With the product requirement documentation, engineers will write a corresponding system design that meet the business requirement. The system design will include technical architecture, data model, business logic, user interface, etc.

Then there are the coding, testing, and deployment phases. The whole process could take years. When the customers finally see the product, not all features in the product requirement documentation work as anticipated. In addition, some needed features are not in the product, either because the product manager didn't have the foresight, or the requirement have changed over time.

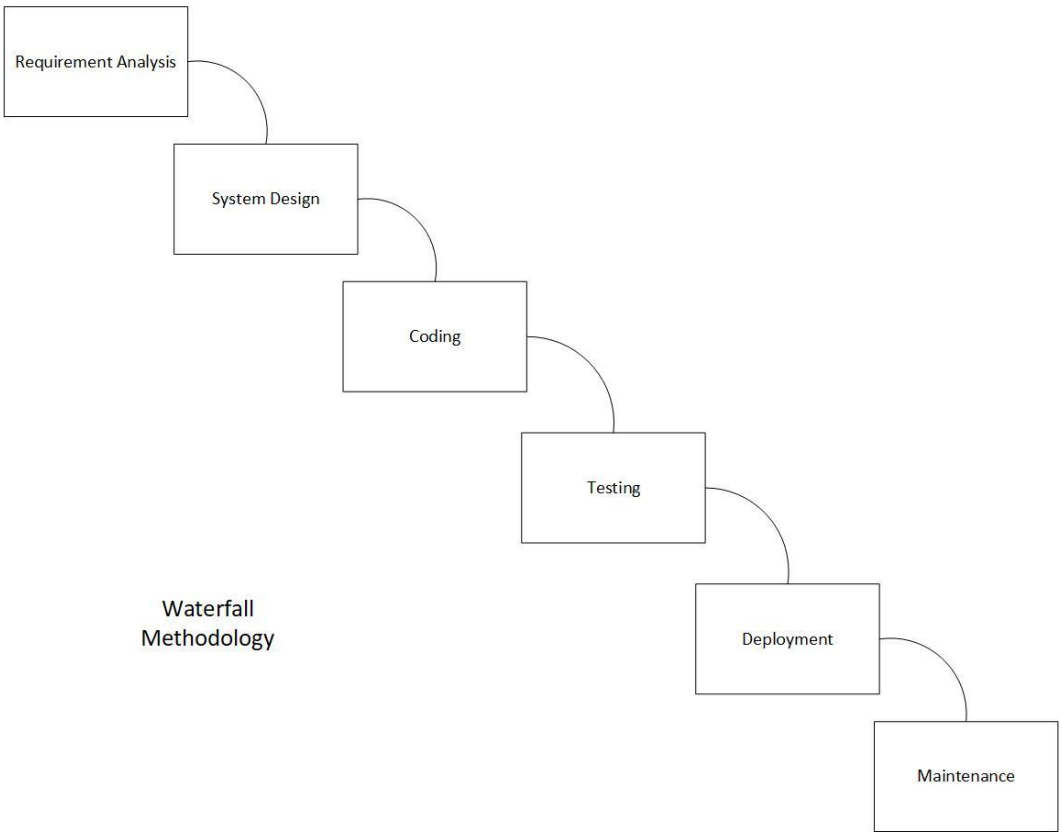

Fig 1: Waterfall methodology

Waterfall was successful before the Internet age because at the time, there were few third-party libraries, and many applications needed to be built from the ground up. Therefore, it takes a long time to develop a new product. The first commercial mobile phone, as an example, took ten years to develop [5]. The first commercial Java implementation took 5 years [6]. However, the rigidity of

Waterfall makes it unfit for the Internet age where time to market is measured in months and customer input has to be implemented as soon as possible. [1][2][3][4]

2. Agile

By the turn of the century, as the Internet became more prevalent, the problems of Waterfall became more apparent. The product would take years to finish and, by the time it was released, the nature of the requirements have changed and product likely failed to address market needs, which have changed or become unimportant. Also, there is no way for developers/product managers to continuously glean changing market needs and feed back to product development.

Agile was invented in 2001 to address the short comings of Waterfall methodology and meet the fast-changing market conditions. Unlike Waterfall, there was no heavy upfront requirement documentation in Agile. If the development team is not sure of the final requirements, it delivers the first approximation of the product and then listens to feedback. The process is iterative, like what the following picture shows:

Fig 2: Agile methodology (Source: Krusche & Company)

A development team uses a series of “sprints” to incrementally finish the product, and in each sprint (usually 2 – 3 weeks), there are phases like plan, design, develop, test, deploy and review. Customer feedback is the key to Agile, and feedback is usually addressed in later sprints.

Agile has nicely addressed the issues of Waterfall, including long product development cycle and no customer feedback during development. The finished products are more aligned with changing market needs. However, there was still one problem – deployment. Agile calls a new deployment for each sprint, therefore customers can see the newly released product. However, nothing significant has happened in the deployment process – in other words, deployment process hasn’t changed much since Waterfall methodology. In the early days of Agile, there was conflict between development teams and operation teams, who usually handled deployment. That’s where the idea of DevOps was created.

3. DevOps

In the golden days of Waterfall, operations teams deployed software releases once every several months. As Agile became popular, such a deployment frequency no longer satisfied the need of development teams. There were constant conflicts between the development team and the operations team. In 2009, during O'Reilly Velocity Conference, technologists John Allspaw and Paul Hammond from Flickr gave a talk titled: "10 + Deploys per Day: Dev and Ops cooperation at Flickr".

In their talk, they described the state where developers and operations blamed each other and how that attitude could be detrimental. They suggested instead that developers and operation should collaborate and make deployment as frequent as possible. That was a defining moment in what was later called DevOps movement.

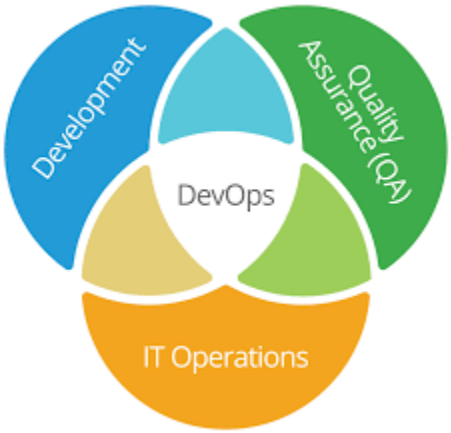

Fig 3: DevOps (Source: Smartsheet)

The idea of DevOps is to fix the inefficiencies of deployment process in Waterfall. In addition to more frequent deployment, DevOps is also set to achieve:

- Shorter lead time for a new release
- Minimized downtime
- Significant improvement in product quality
- Automation in repetitive tasks

Organizations that implement DevOps tend to have more satisfied customers and are more capable of building the "right" products because of more frequent deployment.

In the beginning, DevOps appeared to be the intersection of Dev and Ops. Since then, it has evolved to becoming more than just deployment. Now it also encompasses continuous testing -- the testing methodology that allows developers to continuously receive testing feedback during the coding phase. [7][8]

4. CI/CD Pipeline

If you search the Internet for the definition of DevOps, the result could be quite abstract. It is a set of practices, tools, and cultural philosophies. In practice, it can be also viewed as a CI/CD pipeline. CI stands for continuous integration. It is the practice which developers of a team merge code together into a repository. CD stands for continuous delivery (or continuous deployment).

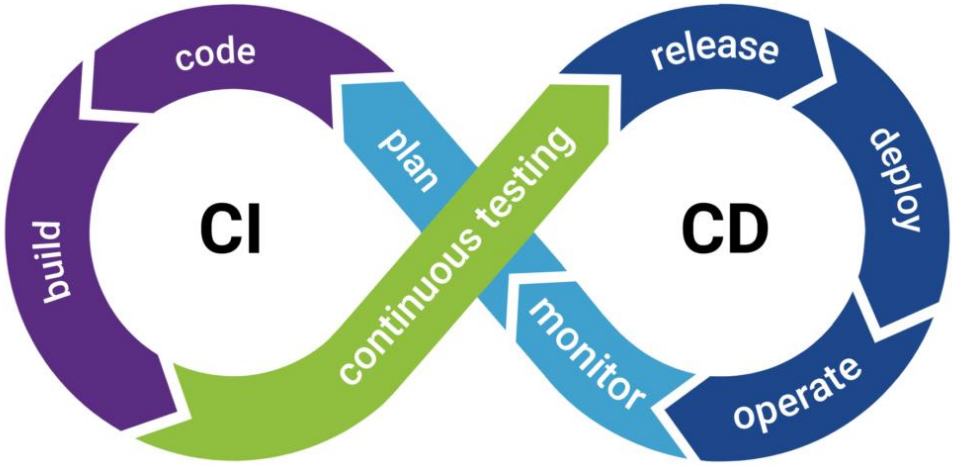

Fig 4: CI/CD Pipeline (Source: Synopsys)

Stages of CI/CD include:

- Code

  The first stage of CI is coding, where developers generate source code according to requirement and design. Tools used at this stage is usually an integrated development environment (IDE). After source code is created, a developer would check the code into a code repository and version control system such as Git.

- Build

  At this stage, the CI tool (sometimes called a build pipeline) would check out source code from the code repository, combined with relevant libraries, internal or external modules and dependencies, and compile all components into artifacts. Developers will be notified of the build result, whether successful, possible error code, and build logs.

  The build process may also include extra tools such as scripts to generate executables or package the artifacts into runnable instances for deployment.

- Deployment

  There are usually several environments for deployment, some examples include development (DEV), QA, Staging, and Production. The first three environments are for internal purpose and Production is what the end-users see. If a build is successful, the runnable instance is usually deployed into the first environment – in this case DEV.

  Depending on the company and the application, an environment may either be in a data center or in the cloud.

- Continuous testing

Because of the impact and cost of software failure, continuous testing has quietly become the centerpiece of DevOps.

5. Continuous Testing

In the Waterfall model, testing only happened after coding is done. Continuous testing is Agile's approach to software testing. Testing happens throughout the CI/CD pipeline. We don't have to wait until a large chunk of coding is done and then test. Test is done during build, and there are tests after each deployment.

The benefits of continuous testing include [9][10]:

- Expose defects early:

  The later a defect is found in the development process, the higher the cost of fixing it is. By moving tests early in the development stage, we can also find defects early.

- Be able to improve the test process

  Unlike in the Waterfall model, where testing is done at a specific time. In Agile development, product becomes more complex over time, so are the tests implemented. By setting up tests in every stage of the development, we can gather more data on testing and improve the process over time.

- Immediate feedback

  Because all tests are automated, the development team will immediately receive feedback in all stages of testing. This saves time and eventually means faster product release times.

6. Implementation Example

I have been a DevOps engineer in the Supra organization of Carrier for the past several years and have had the opportunity to implement continuous testing integrated with our DevOps pipeline. Let me describe how we did it in this paper.

6.1. CI/CD Pipeline – TeamCity/Octopus Deploy

We use TeamCity as the CI tool and Octopus Deploy as the CD tool.

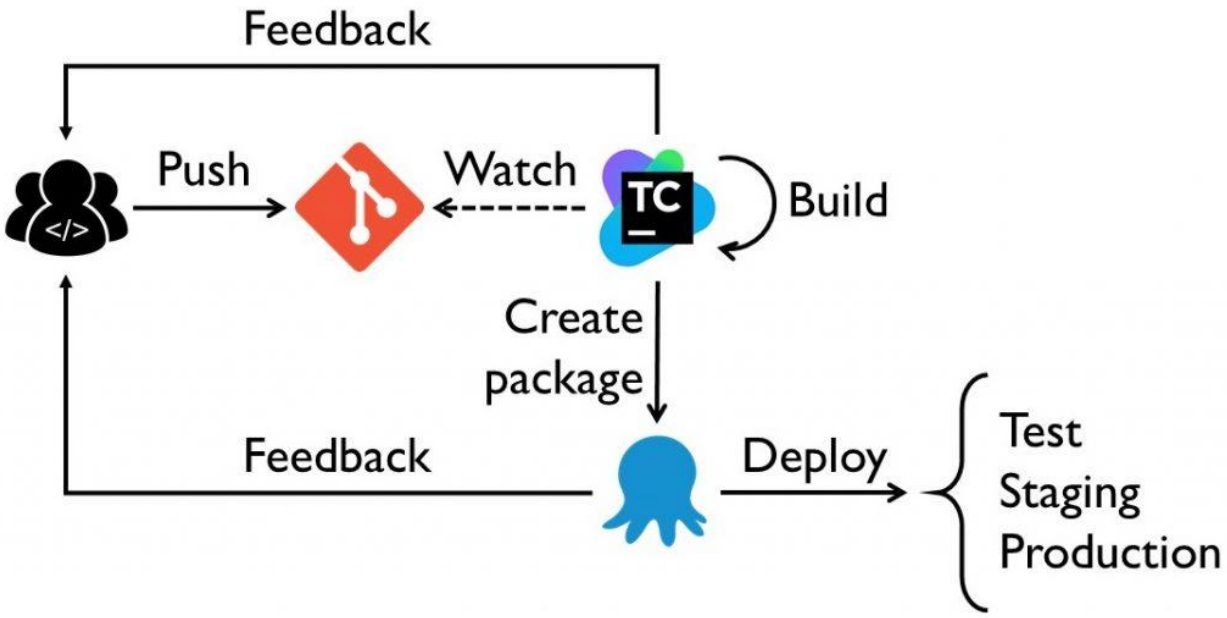

Fig 5: CI/CD Example (Source: boyan.io)

TeamCity is a popular continuous integration server that supports all major IDEs and version control systems, including Git (which we use). It has native support for languages such as Java, .NET, Python, and Ruby. Many other software vendors have written numerous plugins so that their software can work with TeamCity, including Octopus Deploy, Smartbear, Jfrog, etc.

Octopus Deploy is a continuous deployment tool for .NET applications. It works with CI tools such as TeamCity to securely deploy applications to different environments, whether they are on premise or in the cloud. Octopus Deploy enables DevOps engineers to bring deployment related automation into a single place.

6.2. Test Integration

Many types of tests can be integrated with the CI/CD pipeline, including unit tests, integration tests, API tests, UI tests, etc.

6.2.1. Unit Test

Unit test is about low-level tests performed close to the source code. Individual methods, subroutines, functions, or properties are tested. We integrated unit test with our CI tool – TeamCity. TeamCity has built-in support for multiple types of unit tests including nunit and vstest.

6.2.2. Integration Test

Integration test is about whether individual modules work as expected when connected. The test may also include whether the modules can interact with external databases or

APIs properly. Because the test requires other applications to be up and running, it is run in a particular environment. Therefore, it is integrated with the CD tool – Octopus Deploy, which runs tests against the deployment environment.

If the environment is on premise, the test is quite straight forward, including parameter passing.

If the environment is in the cloud, the implementation is a bit more complicated. First, the script that runs the integration test has certain parameters that needs to be passed in during deployment. Those parameters may include description about external resources and what tests to run. Special care must be taken for the cloud deployment. Secondly, because the key user interface is Octopus Deploy, the test report must be passed from Azure to Octopus Deploy. Mechanisms must be established for Octopus Deploy to find out when the test finishes and then download the test report from the cloud to Octopus Deploy.

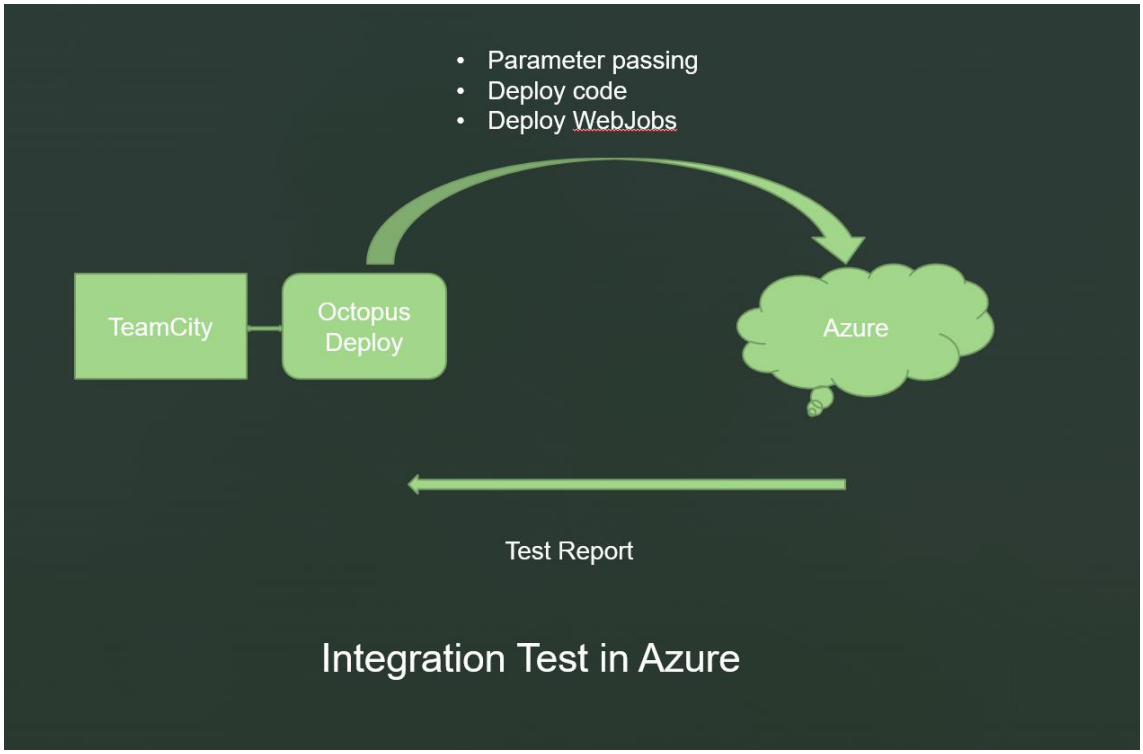

Fig 6: Integration Test integration

6.2.3. API Test

API tests validate the Application Programming Interface (API). Usually, a tool is needed to automate API tests. In our case, we use ReadyApi from Smartbear to run the test. Here are the steps to integrate ReadyApi test into our CI/CD pipeline:

1. Install SOAP UI's TeamCity Plugin on the TeamCity server.
2. Extensively modify the main plugin script so that parameters such as test environment, test type, script repository branch, etc. can be passed into the script.

3. Invoke TeamCity build configuration associated with ReadyApi test from Octopus Deploy using a REST API call.
4. Octopus Deploy keep polling TeamCity until the test finishes.
5. Octopus Deploy then invoke a REST API call to download the test result.

A key consideration in this implementation is job queuing. If multiple tests are invoked at the same time, there should be a mechanism to queue test jobs one by one. Without job queuing, ReadyApi might return unexpected results. To provide this synchronization, the TeamCity build agent has automatic job queuing capability. Therefore, our implementation can handle situations where test jobs are invoked simultaneously.

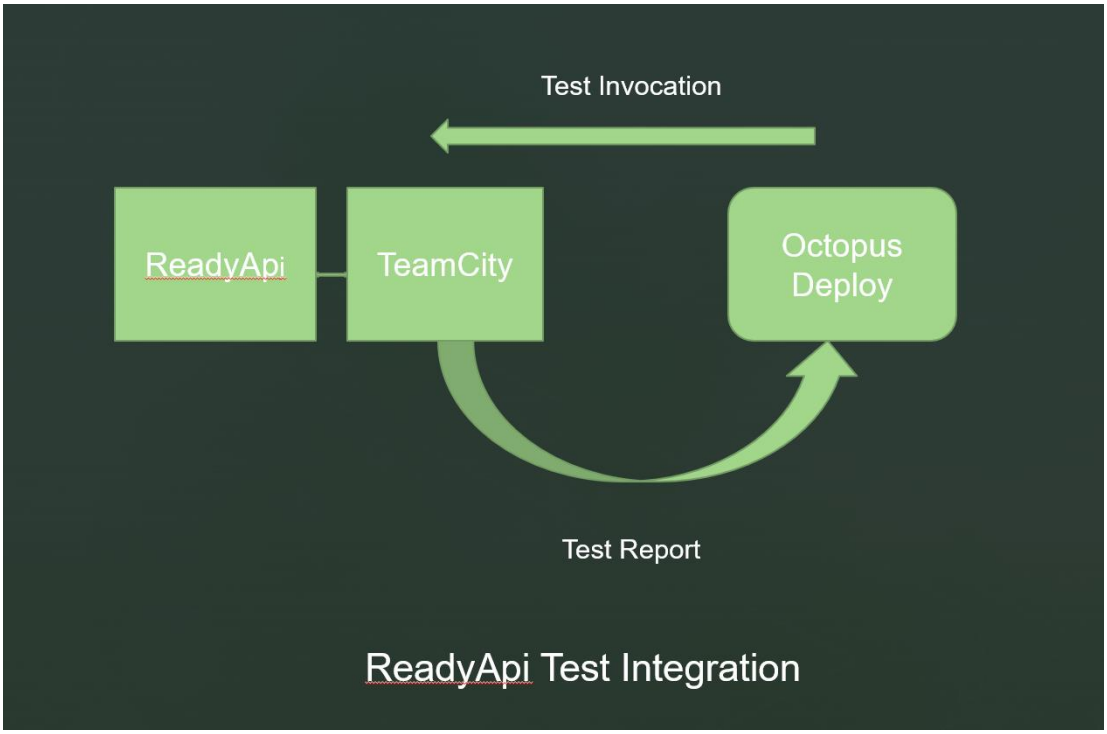

Fig 7: API test integration

6.2.4.UI Test

Automated UI tests can also be integrated post-deployment. We have done a Web application UI test integration with TeamCity and the same test can easily be integrated with Octopus Deploy. We are also in the process of integrating a Mobile Application UI test utilizing both Selenium and Apium.

7. Continuous Testing Process

With automatic tests integrated with the CI/CD pipeline, the development process has changed. After new code is checked into the repository, builds are automatically started, and then unit tests are run. After deployment to a certain environment, say DEV, the integration tests, API tests and other tests such as UI tests can be invoked automatically against the DEV environment. As we promote the application through DEV, QA, Staging, and Production environments, multiple tests are run against each environment, as the following diagram shows:

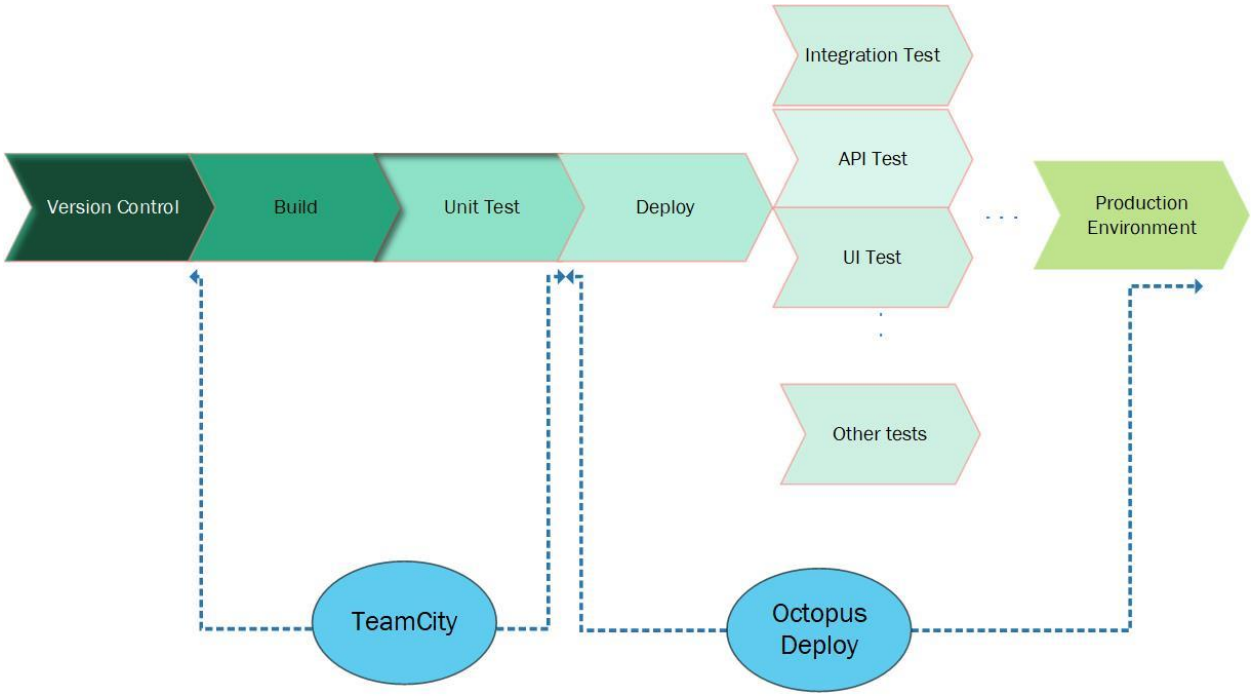

Fig 8: fully automated testing integrated with CI/CD

With test automation integrated with the CI/CD pipeline, Agile team deploys more frequently, tests more frequently, test early, receive feedback early. Not only does the team have better ability to build the “right” product, but also build a quality product.

8. Conclusion

Traditionally software testing starts after developers finish coding. With the fast-moving Agile development method, there is a need for developers to receive immediate testing feedback with new code commits. In addition, after the advent of DevOps practice, there is a need for automatic testing against important deployment environments such as staging and production. Both needs are addressed by continuous testing – automatic testing integrated with a CI/CD pipeline. In other

words: Agile is just the beginning; the road to quality software goes through DevOps and continuous testing.

References

1. To Agility and Beyond: The history—and Legacy—of Agile Development (https://techbeacon.com/app-dev-testing/agility-beyond-history-legacy-agile-development)
2. An Introduction to Agile: The history of Agile and project management (https://www.cambermast.com/agile-blog/2020/05/07/an-introduction-to-agile-the-history-of-agile-and-project-management.html)
3. The Complete History of Agile Software Development (https://agilemania.com/history-of-agile-software-development/)
4. A brief history of the agile methodology (https://www.infoworld.com/article/3655646/a-brief-history-of-the-agile-methodology.html)
5. Inventor of cell phone: We knew someday everybody would have one - CNN.com
6. Java (programming language) - Wikipedia
7. The history of DevOps – How has DevOps evolved over the years (https://www.harrisonclarke.com/devops-sre-recruiting-blog/the-history-of-devops-how-has-devops-evolved-over-the-years)
8. A Brief History of DevOps (https://www.knowledgehut.com/blog/devops/history-of-devops)
9. Continuous Testing in DevOps Best Practices and its Adoption (https://www.xenonstack.com/insights/what-is-continuous-testing)
10. What Is Continuous Testing In The DevOps Pipeline (https://insights.daffodilsw.com/blog/what-is-continuous-testing-in-the-devops-pipeline)