Quality Delivery Pipeline

Joseph W. Yoder, The Refactory, Inc. – USA
Ademar Aguiar, Faculdade de Engenharia, Universidade do Porto – Portugal
Hironori Washizaki, Waseda University – Japan

There are many proven practices that are used during development to help sustain delivery at a good pace while maintaining confidence in the system. Some of these are related to Agile or Lean practices such as short delivery cycles, good testing, clean code, and continuous integration. Small delivery size with regular feedback through incremental releases has proven itself over the years and has become the de-facto standard for most agile practices. Having a good delivery pipeline can help assure the delivery meets the requirements by ensuring that proper validation and checks are done before integration or release. This paper will focus on the "Quality Delivery Pipeline" as a practice that can help sustain delivering with confidence by addressing important qualities in the pipeline.

Categories and Subject Descriptors
• Software and its engineering ~ Agile software development • Social and professional topics
• *Software and its engineering ~*

General Terms
Agile, Quality, Sustainable Delivery, Patterns, Confidence

Additional Keywords and Phrases
Agile Software Development, Continuous Integration, Continuous Delivery, Quality, Reliability

ACM Reference Format:
Yoder, J.W., Aguiar A., Washizaki, H., "Quality Delivery Pipeline", 2018. Procs. SLPLoP'18, November 20-23, Valparaiso, Chile. 7 pages.

Author's email address: joe@refactory.com, ademar.aguiar@fe.up.pt, washizaki@waseda.jp

Introduction

Continuous Integration and Delivery has become the de-facto standard for most Agile and Lean processes. Many of these development processes help teams respond to unpredictability through incremental, iterative work cadences and through empirical feedback. Regular delivery of reliable working software that has the highest value to those using or benefiting from the software is becoming more important for the success of many companies. Actually, the first principle behind the Agile manifesto is: *"our highest priority is to satisfy the customer through early and **continuous delivery** of valuable software."* That has led to the acceptance of DevOps, which advocates automation and monitoring at all steps of software construction, from integration, testing, releasing to deployment.

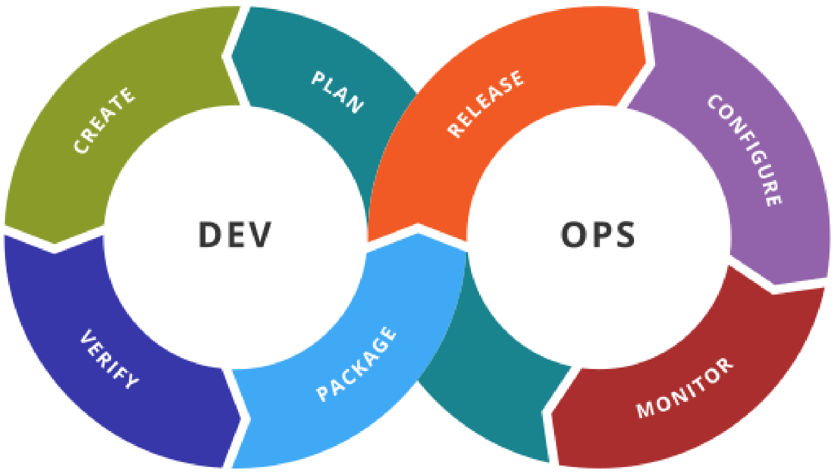


Figure 1: DevOps[1]

However, blindly following Agile practices isn't sufficient to help sustain delivering quality software at a good pace with confidence. Quite often system qualities, such as reliability, scalability, or performance, are overlooked or simplified until late in the development process, thus causing time delays due to extensive refactoring and rework of the software design required to correct quality flaws. There are other practices (patterns) that can help with this, such as reflection time, finding ways to improve, and building quality into the process and product from the start [YWA, YW, YWW14, YWW15, YWW16a, YWW16b].

The goal is to be able to safely and quickly release our software in a sustainable way. Continuous Delivery and DevOps focus on both automation and team practices that can help with delivering more quickly and reliably. Automation and a good pipeline that includes quality steps has proven invaluable for raising confidence in the release. This paper will focus on the "*Quality Delivery Pipeline*" as a practice that can help sustain delivering with confidence by addressing important qualities in the pipeline.

[1] https://en.wikipedia.org/wiki/DevOps#/media/File:Devops-toolchain.svg (CC BY-SA 4.0)

Quality Delivery Pipeline

"We've got the greatest pipeline in the company's history in the next 12 months, and we've had the most amazing financial results possible over the last five years, and we're predicting being back at double-digit revenue growth..." — Steve Ballmer

Figure 2: Pipeline[2]

Continuous Integration[3] has become a key factor for most software development teams as a means to safely evolve the code ensuring that we have not broken anything that we have to interact with. A step to be successful with this is to setup servers and scripts for continuous integration. Continuous Delivery [Humble] is an extension of this by taking the automatic or semi-automatic promotion of builds and process them into something that can be delivered.

The goal is to take changes from version control to production by making deployment to different environments as automated and as reliably as possible.

How can we create a process that helps us deploy while ensuring the desired qualities are met?

❖ ❖ ❖

Automating the testing and deployment process is challenging and requires a lot of expertise.

Building and releasing into production environments can be tedious and error prone.

Businesses can get very busy trying to meet the requirements where there is barely time, resources or people to do what is needed to keep a project afloat.

Lack of quality validation or testing of your release can be dangerous and costly.

❖ ❖ ❖

Therefore, develop and automate a delivery pipeline composed of steps to get and build the current version, run various quality validations on the build, and when successfully validated, push the build into a staging or production environment.

Figure 3 outlines the core of what a minimum pipeline should be. In a sense it is analogous to pipes and filters where each step in the pipeline is a quality filter such as code quality,

[2] https://pxhere.com/en/photo/776653 (CCO Public Domain)

[3] https://www.martinfowler.com/articles/continuousIntegration.html

technical debt, security, etc. It is important that the pipeline be small and fast, instead of long and slow. *Automating As you Go* [YWW16b] makes the process quicker and more reliable.

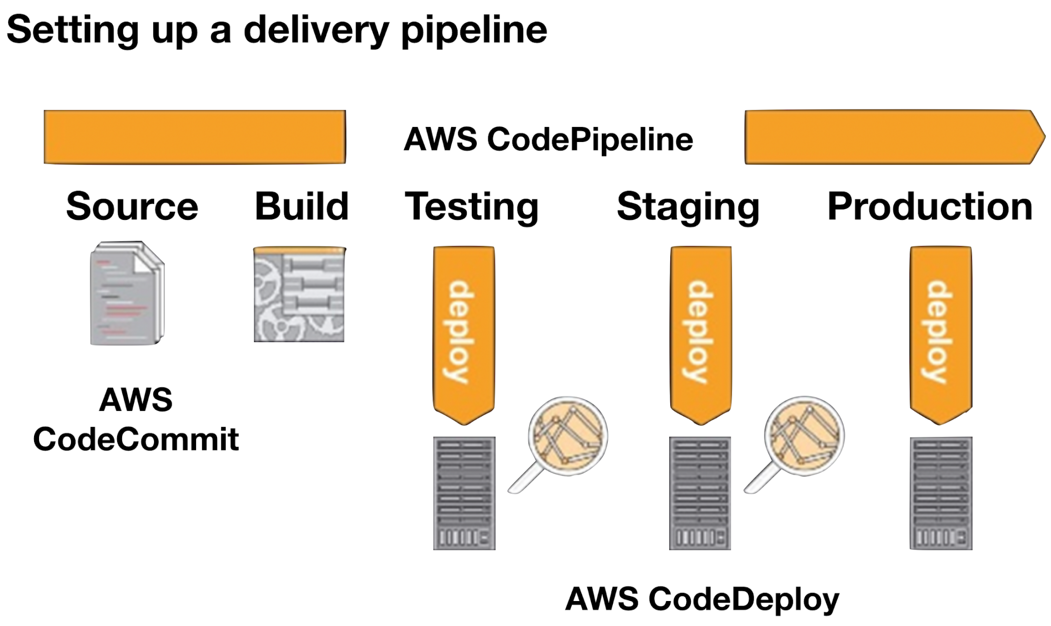
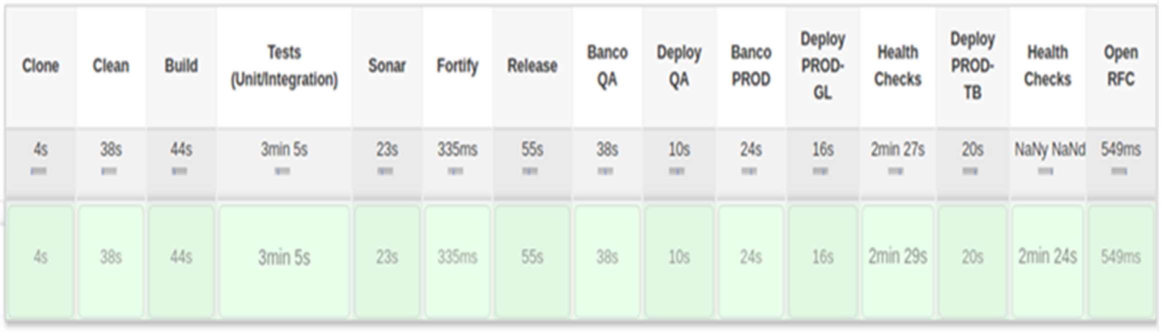


Figure 3: Minimum Delivery Pipeline[4]

The minimum pipeline must first get the code and build it. Then there is a step for some form of minimum quality testing such as unit and integration testing. If the build and test steps are successful, the system can be pushed into a staging environment for QA testing. Finally, if all steps were successful, the system can then be pushed into the production environment. Although this minimum pipeline works, it may not be "ideal" for gaining confidence as there can be many other critical qualities that need to be validated before releasing into production.

"Ideal" Quality Pipeline

A pipeline that offers more confidence would add more involved testing steps and also validate many other important qualities. Some of these qualities might be code quality while others might be reliability, security, performance, or other architectural qualities. The following (Figure 4) is an example of a more complete quality pipeline.



Clone	Clean	Build	Tests (Unit/Integration)	Sonar	Fortify	Release	Banco QA	Deploy QA	Banco PROD	Deploy PROD-GL	Health Checks	Deploy PROD-TB	Health Checks	Open RFC
4s	38s	44s	3min 5s	23s	335ms	55s	38s	10s	24s	16s	2min 27s	20s	NaNy NaNd	549ms
4s	38s	44s	3min 5s	23s	335ms	55s	38s	10s	24s	16s	2min 29s	20s	2min 24s	549ms

Figure 4: Ideal Quality Pipeline

[4] Adapted from Julien Simon - https://www.slideshare.net/JulienSIMON5/aws-codecommit-codedeploy-codepipeline

This pipeline has steps for building the system (clone/clean/build), unit and integration tests, security checks through Fortify[5], and other code quality checks through Sonar[6]. Additional architectural checks can be included in Sonar such as those outlined through *Continuous Inspection* [Merson]. Validating the dependencies are healthy through a pre-health test before release minimized potential issues when going into a production environment. Building monitors as part of the release and include as part of the pipeline is key and can help teams "monitor" the results of the release.

The following is a set of related practices to gain confidence and assist with a quality pipeline:

- Small Incremental Releases - keep releasing small changes into production.
- *Automate as you go* - automate what you can as soon as you can.
- *Continuous Inspection* - getting regular feedback.
- *Blue-Green Deployment* - separate deployment from release.
- Staged Releases - first in a safe environment and spread out.
- *Canary and Rolling Deployment*s - gradually roll out your release..
- Health Checks - Smoke Tests to make sure things are ok, etc.

* * *

The following are some advantages of *Quality Delivery Pipeline*:

- Freeing teams up from needing to think about delivering quality makes them more productive as important quality validations are built into the pipeline;
- Some important critical tasks can be done more quickly and with confidence.
- You can ensure that what gets delivered meets a minimum quality.

There are also some potential disadvantages to *Quality Delivery Pipeline*:

- Automating all quality checks might be very difficult.
- Creating a quality pipeline can require a lot of technical expertise.

Automate as you Go helps ensure the *Quality Delivery Pipeline* is run frequently, and error prone steps are avoided. The *Quality Delivery Pipeline* becomes a form of *Continuous Inspection* during the release cycle.

[5] https://www.microfocus.com/en-us/products/static-code-analysis-sast/how-it-works
[6] https://www.sonarqube.org/

Summary

DevOps has become popular as a methodology that combines software development with deployment and operations which has an overall goal of delivering value more frequently and with confidence. Having a *Quality Delivery Pipeline* is key to being able to reliably deliver quickly and with confidence. This paper outlined the beginning principles that are part of a good delivery pipeline.

A good “quality” pipeline breaks down the software delivery process into various stages that focus on verifying the quality of the system before release. This can be from validating functionality to verifying code quality and ensuring certain system qualities such as security and performance are being met. The pipeline provides visibility to the team and everyone involved in delivering and maintaining the system. A minimum pipeline should include: build, test, quality checks, and deployment verification. A more ideal or complete pipeline should also include other steps that validate other qualities such as security, reliability, performance, and health tests.

Acknowledgements

We’d like to thank our shepherd Foutse Khomh for his valuable comments and feedback during the SugarLoaf PLoP 2018 shepherding process. Finally we’d like to thank our 2018 SugarLoaf PLoP Writers Workshop Group, Audreas Seitz, Eduardo Guerra, Monica Michelle Villegas, Hernán Astudillo, Joelma Choma, and Paulina Silva for their valuable comments and suggestions.

References

[Humble] Humble, J., Farley, D., *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Boston, MA: Addison Wesley, Pearson Education, Inc., 2011.

[Merson] Merson P., Yoder J., Guerra E., and Aguilar A., "Continuous Inspection: A Pattern for Keeping your Code Healthy and Aligned to the Architecture," 3rd Asian Conference on Patterns of Programming Languages (AsianPLoP), Tokyo, Japan, 2014.

[YWA] Yoder J., Wirfs-Brock R., and Aguilar A., "QA to AQ: Patterns about transitioning from Quality Assurance to Agile Quality," 3rd Asian Conference on Patterns of Programming Languages (AsianPLoP), Tokyo, Japan, 2014.

[YW] Yoder J. and Wirfs-Brock R., "QA to AQ Part Two: Shifting from Quality Assurance to Agile Quality," 21st Conference on Patterns of Programming Language (PLoP 2014), Monticello, Illinois, USA, 2014.

[YWW14] Yoder J., Wirfs-Brock R., and Washizaki H., "QA to AQ Part Three: Shifting from Quality Assurance to Agile Quality: Tearing Down the Walls," 10th Latin American Conference on Patterns of Programming Language (SugarLoafPLoP 2014), Ilha Bela, São Paulo, Brazil, 2014.

[YWW15] Yoder J. and Wirfs-Brock R., and Washizaki H., "QA to AQ Part Four: Shifting from Quality Assurance to Agile Quality: Prioritizing Qualities and Making them Visible," 22nd Conference on Patterns of Programming Language (PLoP 2015), Pittsburgh, Pennsylvania, USA, 2015.

[YWW16a] Yoder J., Wirfs-Brock R., and Washizaki H., "QA to AQ Part Five: Being Agile at Quality, Growing Quality Awareness and Expertise," 5th Asian Conference on Patterns of Programming Languages (AsianPLoP), Taipei, Taiwan, 2016.

[YWW16b] Yoder J., Wirfs-Brock R. and Washizaki H., "QA to AQ Part Six: Being Agile at Quality, Enabling and Infusing Quality," 23rd Conference on Patterns of Programming Language (PLoP 2016), Monticello, Illinois, USA, 2016.