

# Design Patterns for Continuous Delivery: Restructuring DTAP for DevOps

WRITTEN BY OLAF MOLENVELD CTO & CO-FOUNDER OF VAMP  
AND JOEP PISCAER ARCHITECT AND PATHFINDER

## CONTENTS

- > Introduction
- > What Are All These Continuous Practices?
- > What's Wrong With Continuous Delivery?
- > DPAT: The Continuous Release Cycle
- > Benefits of Continuous Releasing for Software and Business
- > Conclusion
- > Summary
- > Key Takeaways for Practitioners

## Introduction

Organizations are adopting microservices and DevOps to stay competitive and increase the speed of releasing new functionalities with improved scalability, quality, and (cost) efficiency.

Current processes, tooling (continuous integration/testing/packaging/deployment), and Development/Test/Acceptance/Production environments (DTAP) are typically designed for traditional codebases and applications, manual and time-consuming testing, separate organizational silos and handovers, and “big bang” deployments to production.

With the move to DevOps, these traditional environments, processes, and tools no longer serve in reducing risk in the software delivery lifecycle (SDLC.) Instead, they act as a block to the full potential of increased efficiency in software delivery.

Because of the DevOps mantra of “you build it, you run it,” development teams are allowed a lot of autonomy and control, and are expected to also handle quality assurance (QA) and releasing tasks. The reality is that these are specialized and complex tasks that need expertise and dedicated tooling provided to software delivery teams. In addition, business outcomes are ignored as they are hard to validate for development teams, and QA becomes mainly focused on technical aspects.

In this Refcard, we're going to give you a guide on how you can rethink your software delivery methodologies for modern software development — one that reinjects the business function of QA and takes advantage of continuous releasing in order to lower the risk of releases, shorten development cycles, and help software support business.


You will learn:

1. How testing is handled in continuous integration, delivery, and deployment
2. The promise of these three “continuous practices”
3. How continuous delivery falls short on these promises in practice
4. How DTAP stands in the way of continuous delivery fulfilling its promise
5. How to restructure DTAP for continuous releasing to gain the advantages of continuous delivery for software and business

## What Are All These Continuous Practices?

Various practices have been developed to optimize parts of the software development and delivery process. In this Refcard, we'll consider each specifically as it relates to testing.

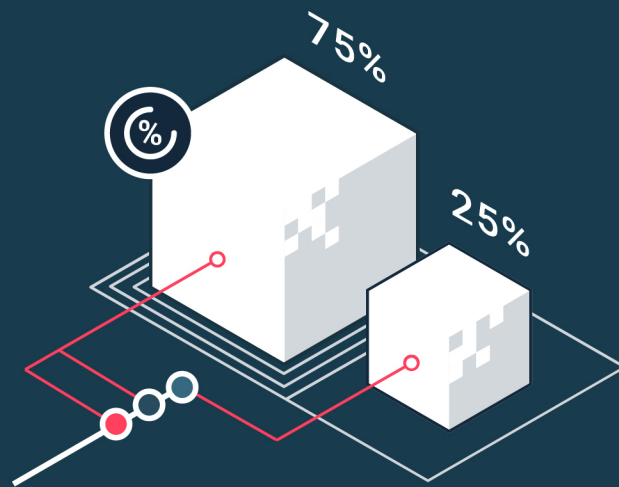


 **vamp**

# Deliver

software with confidence

[learn how](#)



## **Release often and with confidence – even on a Friday afternoon**

**Tired of hasty final approval chaos in your release train?**

Learn how automating your release pipeline can make all the difference to stress-free software delivery

Discover how to gain step-by-step control over reliable delivery

Take the first steps to release simply, safely and at scale

**Learn how**

*A Note on Terminology:*

*In this Refcard, we'll be using terminology with some overlap. To prevent confusion, let's define it here.*

*The acronym DTAP is short for Development, Testing, Acceptance, and Production. These four separate environments are used in a phased approach of software development.*

*In this guide, we refer to each of these environments using capitalization: Development, Testing, Acceptance, and Production.*

- *A Development environment is often a developer's laptop.*
- *A Testing environment is where completed and built code is tested to verify it works as expected. Testing should closely resemble Production.*
- *Acceptance is where successfully tested code is deployed for final verification, sometimes involving the end user in a pilot group.*
- *Production is the environment where code is made available to all users.*

*In addition to the environments, we refer in lower case to development and testing as the processes where engineers go through different phases of work.*

## CONTINUOUS INTEGRATION

Continuous integration, or CI, is a set of practices during development of code.

CI is essentially the automation of a set of housekeeping rules for developers that makes sure that the Development environment and code repository remain in shape. The basic practices are to use version control (like Git) and to regularly check in all code. Developers make sure that the code they check in passes quality tests and builds into an artifact successfully.

The promise of CI is that work from individual developers moves into Testing more quickly and without major coding errors, freeing up developers and testers to pick up new work.

The focus of the testing done during the integration phase is on high code quality.

## CONTINUOUS DELIVERY

Where CI makes sure that the code in the repository is always ready to move to the next environment, continuous delivery, or CD, is a set of practices that builds on CI to make sure code is always deployable to Production.

CD implies the automation of all steps needed to take a built artifact and deploy it into Testing, Acceptance and Production environments.

The promise of CD is to have a pipeline that completely automates the deployment (and infrastructure provisioning and configuration).

The tests done in the CD pipeline are focused on catching regressions and checking performance before moving to Production.

## CONTINUOUS DEPLOYMENT

Continuous delivery and continuous deployment are very similar concepts. Where continuous delivery promises deployable code, potentially put into Production, continuous deployment always automatically puts the code into Production.

The technical act of deploying involves copying deployable code to the Production environment so that it is ready to receive user traffic.

The technical act of releasing means making the features that code implements available to users.

However, whereas deployment remains only a technical act, releasing has both a technical and business — decision-making — function in the process of software delivery.

## THE CONTINUOUS PROMISE

Each of these continuous practices has evolved to optimize parts of the software delivery process. Together, they promise:

Quality of code. The primary goal of continuous integration is to make sure that code works and doesn't regress between releases.

Low risk releases. The primary goal of continuous delivery is to make the deployment of new software painless.

Faster time to market. The overall goal of the continuous practices is putting software to work as quickly as possible to achieve business goals.

## What's Wrong With Continuous Delivery?

In the ideal world, CI/CD promises to shorten the feedback loop between Production and the development process and to allow developers to optimize performance without long wait times or switching context. This is not surprising, since 70% of digital transformation projects focus on optimizing CI/CD tooling.

In practice, the DevOps approach still has shortcomings. Testing is focused on automated testing — on technical checks of developers' output (the code) — rather than checking for improvements in business outcomes (conversion, revenue) or impact (market share, profitability, customer satisfaction).

Relying on automated testing in CI/CD leaves behind exploratory testing. As a result, the net effect of many CI/CD efforts is that business-oriented QA engineering is cut out by automating just the low-hanging fruit in CI/CD; often skipping out on the more valuable, but more complex QA testing.

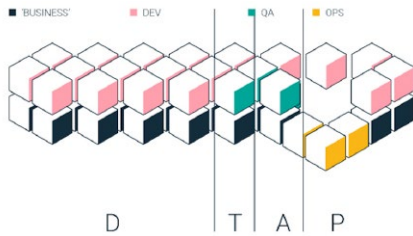
Or even worse, Testing and Acceptance are skipped entirely by letting developers write code and put it into Production based on developer-led testing, cutting out the checks and balances QA would provide. (This is inefficient, as it results in 20-50% of developer time wasted on non-functional tasks.)

This happens because there are systemic issues with Test and Acceptance environments of a typical Development-Testing-Acceptance-Production cycle.

While Testing and Acceptance are great for doing many technical tests, some testing simply requires real users. No amount of synthetic testing in pre-Production, automated or otherwise, can identify all the potential causes of user experience degradation.

**TRADITIONAL DTAP DOESN'T WORK FOR DEVOPS**

Since we're no longer building traditional monolithic applications using traditional methods and silos, we cannot use static pre-production Test and Acceptance environments to ensure our application will work as intended. This is because of the independent nature of microservices and their deployments.



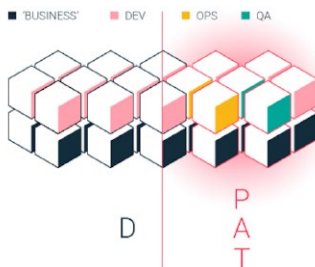
Representative QA engineering that traditionally takes place in the Testing and Acceptance environments should be done on real user traffic in Production.

The solution is rearranging the release train so that the business-oriented QA engineering “acceptance and testing” steps occur in Production.

This gives back control to the QA engineer, and once again opens up the opportunity of testing in a meaningful way: in Production, with segments of real user traffic, to validate new features and optimize the business value of software.

**DPAT: The Continuous Release Cycle**

DPAT stands for Development, Production, Acceptance, and Testing. It is a reordering of the traditional DTAP. The rationale behind DPAT is simple: QA testing yields the best results when done in Production. In fact, it is only by moving quality assurance to Production that QA can validate new features and optimize the business value of software.



*The right architecture allows control over releases*

Unlike past architectures, microservice-style architectures now allow this type of business-focused representative QA testing to be done in Production. Having loosely coupled, independent services allows teams to release multiple versions of the same service and granularly control the flow of traffic to each version. This allows teams to validate the quality of each version independently in a gradual and controlled way, under real production conditions.

This type of testing is called canary testing, and is not unlike the small-scale trials of testing new versions of software with a group of pilot users.

The major difference, however, is that scaling up a canary release is much more granular and controlled, as QA engineers can tap into a larger array of conditions and users to segment traffic and immediately see results in business outcomes.

**SETTING CONDITIONS TO SEGMENT USER TRAFFIC: AUTOMATING CONTINUOUS RELEASING**

This is what we call continuous releasing: the validation of new versions of software early in the software delivery lifecycle — in Production — by releasing them gradually to a small segment of user traffic. By testing new versions of a service on real user traffic, QA engineers can validate both the technical validity of a release and improve business outcomes, such as conversion, revenue, or basket size.

Continuous releasing allows teams to validate software and business outcomes by giving them control over what is released to users. That control comes through setting release policies. These policies govern the conditions under which a new version is released to users, and in what steps.

Conditions can be as simple as a percentage of traffic, or as complex as a selection of users based on location, device type, or login status, as well as business conditions such as “has ordered Product X before.”

Policies allow QA engineers to automatically test an ever-increasing set of technical and business criteria to optimize new software for business outcomes such as conversion, basket size, or revenue in Production.

**Benefits of Continuous Releasing for Software and Business**

Continuous releasing extends the promise of continuous delivery to improvements in business outcomes by re-inserting quality assurance at the end of the release pipeline, where it can now be carried out safely in Production.

Continuous delivery and continuous releasing are not mutually exclusive. In fact, they complement each other.

CI/CD works well to support development work, allowing developers to create better quality code by giving them the tools to automate an ever-growing battery of technical tests to improve software quality, security, and performance issues.

Continuous releasing, on the other hand, supports the work of quality assurance, giving QA engineers the tools they need to execute business-oriented tests to improve both technical and business outcomes for software.

**Conclusion**

In conclusion, the continuous releasing practice treats quality assurance as a business function, responsible and accountable for releasing new software. This delegates decision-making about feature development back into the hands of the business, giving product owners — the stakeholders most qualified to make business decisions — control over releases, closing the software delivery lifecycle.

## Summary

In this Refcard, we've examined a number of fundamental changes you can make to your software delivery lifecycle to gain the competitive advantage of microservices and DevOps and lower the risk of releases, shorten development cycles, and help software support business.

We've introduced the concept of restructuring DTAP as DPAT in order to create a continuous release cycle. By moving quality assurance testing to the right — into Production — teams can gain the advantage of low risk releases for software and organizations can gain the advantage of increased time-to-value for business.

## Key Takeaways for Practitioners

According to the DevOps Research and Assessment (DORA) organization's 2018 State of DevOps Report, successfully evolved DevOps companies are 44 times more likely to have repeatable and automated software configuration and release automation. After speaking to more than 100 enterprises over the years about their DevOps journey, we've distilled our best practices into these steps for increasing DevOps ROI:

1. Add a continuous releasing stage after continuous delivery  
That will allow development teams to independently validate their code under real world production conditions and observe that other services are not impacted by a new release.

2. Distribute separate roles and responsibilities for both the deployment stage and the releasing stage of your SDLC
  - Assign DevOps engineers control over the technical deployment stage and the first step of the releasing stage that involves technical validation.
  - Assign control over business validation to quality assurance and product owner roles.
3. Move business-focused QA testing to the right, into Production  
Rearrange your release train so that the business-oriented QA engineering steps occur in Production, after the technical act of deploying services.
4. Develop automated release policies  
In conjunction with stakeholders, define straightforward policies for releasing new software based on stakeholder objectives. Codify these objectives as Service Level Objectives or SLOs.
5. Automate Service Level Objectives and Indicators  
Define the health and success factors for releases to enable early detection and real-time mitigation of technical issues, such as bugs and/or degraded customer experience.

Typically, introducing a Continuous Releasing stage will double key KPIs, such as time-to-market, in a few months after implementation. If you are interested in learning more, we offer an example [business case builder](#).



### Written by **Olaf Molenveld**, CTO & co-founder of Vamp

Olaf has over 20 years of experience in IT architecture and management roles. While working as an enterprise architect, he helped teams design, build and release innovative online and e-commerce focused platforms for digital enterprise. In his role as CTO of Vamp.io, he channels that experience into his vision for a DevOps, microservices and container space where testing in production and continuous release allow for flawless software delivery.



### Written by **Joep Piscaer**, Architect and Pathfinder

Joep is a seasoned IT professional, with 10+ years experience as a CTO, head of IaaS and infrastructure, enterprise architect. In his most recent role, he worked as a technical pathfinder, straddling the roles of CTO and coach for scrum teams. For his continued efforts to share, contribute and evangelize in the DevOps space, he received the Veeam Vanguard '15 - '17, Nutanix Tech Champion '14 - '18, the EMC Elect '15, Atlantis Community Expert '16-'17 and the VMware vExpert '09, '11 - '17 (and vExpert NSX in '16 - '17) awards.



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.  
600 Park Offices Drive  
Suite 150  
Research Triangle Park, NC

888.678.0399 919.678.0300

Copyright © 2019 Devada, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.