DevOps Testing

The Primary Key to DevOps and Continuous Delivery

By Marc Hornbeek, “DevOps_The_Gray”

1

Why is Testing Important to Dev and Ops?

DevOps testing is important to Dev and Ops for the same reasons DevOps is important to Dev and Ops.

As indicated in *State of DevOps Report*[1], when implemented according to best practices, DevOps delivers major and simultaneous benefits of dramatically improved agility, stability, efficiency, quality, security and employee satisfaction.

While some benefits can be achieved at the expense of others, benefits cannot be achieved altogether without DevOps test engineering. You simply cannot get the impressive results of DevOps and continuous delivery without engineering testing to fit the rapid pace of DevOps! It simply won't work. Sure, you can put in place some new tools, automate some more tests, and make some local improvements to different portions of your pipeline, without implementing a complete DevOps test strategy.

A comprehensive test strategy in which testing is fully integrated and balanced across the end-to-end Dev-through-Ops pipeline, is essential to realize the full benefits of any of The Three Ways of DevOps - continuous flow, feedback and improvement, as described in the *Phoenix Project*[2] and *The DevOps Handbook*[3].

While some benefits can be achieved at the expense of others, benefits cannot be achieved altogether without DevOps test engineering.

It can be said ***DevOps test engineering is the "essential" ingredient that enables DevOps*** because the verdict data from properly engineered testing at each stage in the pipeline is essential for assessing the artifacts produced in each stage. Assessment data from tests are essential to promote artifacts to the next stage in the pipeline without causing bottlenecks.

The old waterfall idea in which the bulk of testing is the job of an "independent" QA team, performing tests after the development phase, but before release, simply does not work in the world of DevOps. With software changes happening at the speed of DevOps, everyone in a cross-functional team must assume responsibility for the quality of the entire product, not just the component they are working on.

To be a successful Dev, QA or Ops individual or leader, you must understand the fundamentals of continuous testing and DevOps test engineering best practices. Without that knowledge you will fall into the trap of implementing local optimizations which fall short of accomplishing an effective end-to-end continuous delivery pipeline.

Under DevOps, everyone is a DevOps test engineer!

DevOps test engineering is the "essential" ingredient that enables DevOps

2

What is the Difference Between DevOps Testing and Traditional Testing?

DevOps testing is different than traditional QA!

Quality guru W. Edwards Deming understood the fundamental difference long before DevOps became a movement "Quality is Everyone's responsibility" and "You cannot inspect quality into a product".[4]

"Quality is everyone's responsibility."
Dr. W. Edwards Deming

Focus on customer relevant defects, not quantity of defects.

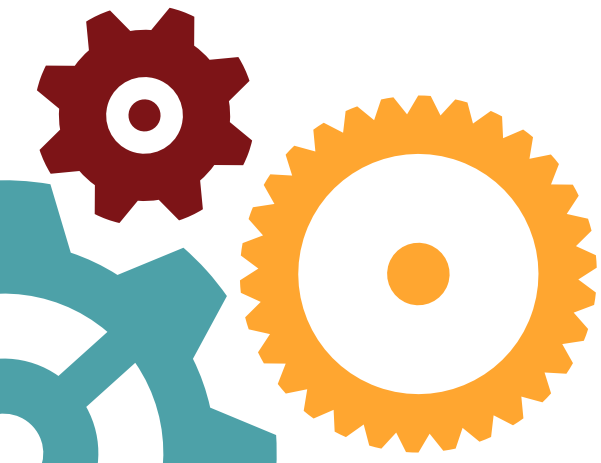
Traditional QA testing is a waterfall process in which a distinct phase following software development is conducted by an SQA test team that is "independent" of the development team. The idea was espoused by Glenford Myers in his seminal books "Software Reliability – Principles and Practices" and "The Art of Software Testing" published in 1976 and 1979 respectively.[5,6] According to Myers' "4th Axiom [of testing] "It is impossible to test your own program". Myers argued that programmers shouldn't do any testing, even unit testing of their own code. He considered it a "conflict of consciousness" to assume a mental role to find defects for the software that the same person designed to work.

While the application of Myer's 4th axiom to unit testing was not followed by most practitioners in the software industry, the idea of independent software functional and system level testing has been a widely accepted practice for many years. The underlying rationale is that it is "impossible" for the same person (the developer) to objectively test his/her own code due to the inherent bias to want the code to work, while a good QA tester must have the opposite attitude to want to uncover bugs in the code. If you follow this logic only "unusual" developers would be able to perform both roles successfully and therefore building a team based on this rare personal quality is essentially not practical.

Despite the logic of the "independent QA test" concept, many very complex software projects without an independent QA test team resulted in extremely high-quality products. There are many high quality open source and commercial software products that had no independent test organization behind each release version. Meanwhile there are countless examples of software projects that had massive independent test resources that resulted in products that failed miserably. How can this be? Perhaps it is not "who" is doing the test that matters but rather it is the strategy used for testing that is important.[7]

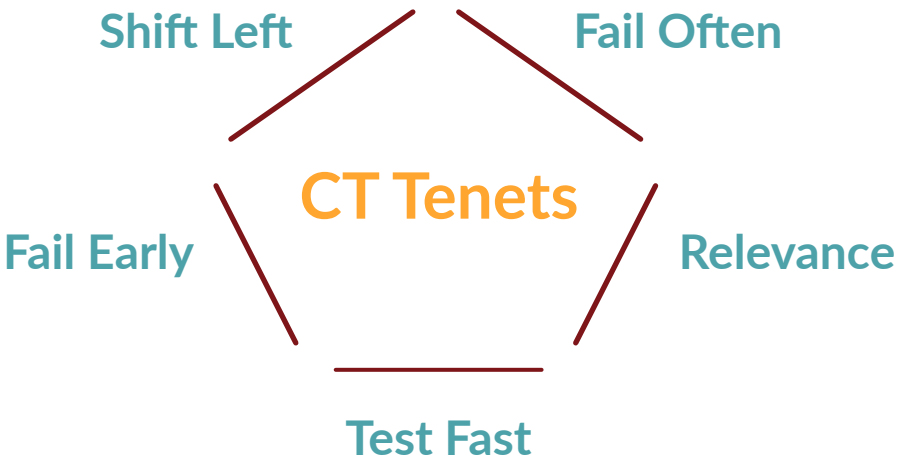
From QA to

Testing maturity is a key differentiator of best practices for DevOps.[8] Many organizations automate their integration, build and delivery processes, but still have trouble with the subtleties of test design, as well as integrating with test orchestration and automation tools and frameworks. There is a vital role for testing engineering expertise, test design, test automation and test case development with DevOps.



3 What are the Best Practices of DevOps Testing?

DevOps test engineering requires a complete re-think of software QA test strategies to align with Dev-through-Ops pipeline stages. Fortunately, best practices for DevOps testing are available and can be learned and applied to any product or service. There are five tenets that underlie the best practices for DevOps test best practices indicated in the diagram below.



- **Shift Left:** Conduct each test as early in the pipeline as possible
- **Fail Early:** Arrange the tests so that the most likely problems are found in the earliest possible stage in the DevOps pipeline
- **Fail Often:** Run tests frequently and with many different conditions
- **Test Fast:** Arrange tests to run in quick cycles.
- **Relevance:** Focus on the most important tests and results.

DevOps test best practices can be categorized as follows:

- DevOps test culture
- Continuous test strategies
- End-to-end test integration
- DevOps test infrastructures
- DevOps test frameworks
- DevOps-ready test tools
- Test creation
- Test Acceleration
- Test Analytics
- Microservices and Containers
- Database DevOps testing
- DevOps Security testing
- Continuous testing antipatterns
- Continuous Test Management
- Advanced continuous test topics

It is beyond the scope of this article to detail all DevOps test best practices. A summary of each of the DevOps test best practice categories are described in the following paragraphs.

1. DevOps Test Culture

A fundamental trait that distinguishes DevOps testing culture is that responsibility for attaining high quality software products is shared by everyone on a cross-functional team. Quality control occurs as an integral part of every phase of the pipeline, and all team members are involved. Testing and quality control is not something concentrated at the end of pipeline, by a separate group. Quality is everyone's concern. A continuous test strategy determines the amount and extent of testing activities throughout the entire development, delivery and deployment process.[10]

To accomplish this everyone involved across the end-to-end pipeline takes responsibility for testing and test results. Here are other characteristics of a DevOps Test Culture:

- A culture of collaboration around tests and test results analysis is encouraged rather than confrontation between testers that find failures and those who fix the code to repair a failed test. Policies for test creation and test coverage are agreed by the cross-functional team.

- Leaders sponsor testing as a strategic element of their budgets and operations rather than viewing testing as a cost to minimize. This includes budgeting time and money for DevOps test training, test resources, test frameworks, test tools, test management and establishing policies for assessments. The assignment of a DevOps test architect is also a best practice. The test architect has an overall perspective of DevOps test practices helps cross-functional teams understand and apply them.

- Strong Dev teams embrace the creation of good tests and an analysis of test results. Strong Ops teams are involved in cross-function test execution and planning.

- Policies for test creation and test coverage are agreed by the cross-functional team.

2. Continuous Test Strategy

The old waterfall V-Model test strategy in which large amounts of software changes are tested in a "big bang' test near the end of the software development process by an independent QA team does not work with DevOps. In DevOps, small changes are tested incrementally by a cross-functional team over all stages of the continuous delivery pipeline.

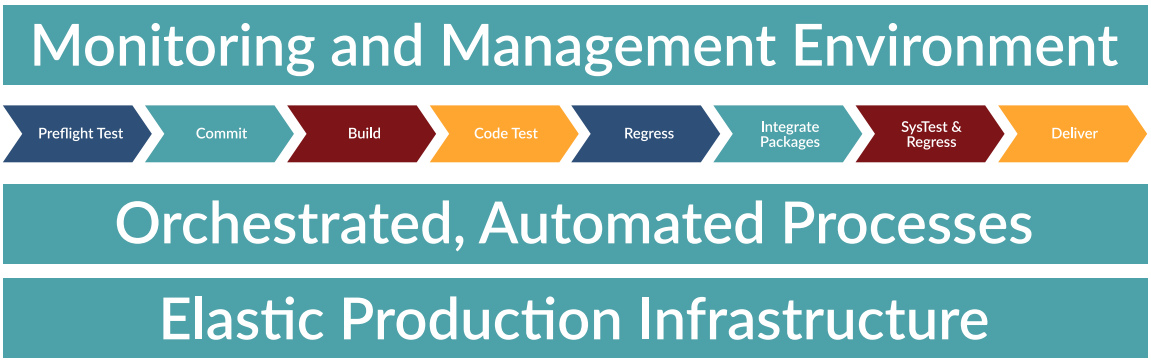
Agile test methodologies are more compatible with DevOps testing because with Agile small changes are tested incrementally in by a cross-functional team. However Agile does not define the infrastructure for continuous testing, the need to integrate continuous testing into a toolchain or the methodologies for extending testing into deployment. Continuous test strategies required for DevOps are more completely defined than with waterfall or Agile.

Continuous testing strategies include the integration of testing at all stages in the pipeline and include strategies for testing during deployment such as A/B testing. Green/Blue deployments, Canary testing and Chaos Monkey for testing the resiliency of the deployment infrastructure.

3. End-to-End Test Integration

In DevOps, tests are integrated horizontally at each stage of the end-to-end pipeline, and vertically at all layers of the continuous delivery toolchain and infrastructure. Each organization may have its own definition of pipeline stages and vertical layers.

The example in the diagram below explains this category of DevOps test best practices.



In this situation, tests are conducted across the horizontal pipeline as follows:

- Prior to committing software changes into the integration or trunk branch pre-flight tests are conducted on the software change integrated with the tip of the integration branch within a private instance, to ensure the changes will not break the integration branch. Pre-Flight tests include unit testing, static code analysis scanning, functional, regression and performance tests. During this stage feature flags or toggles may be installed in the software and each setting of the flag is verified to make sure the flags work. New automated tests that validate the changed software are created and tested, for use in later stages of the pipeline and subsequent regression tests.

- During the code commit stage, assessments are conducted on the committed code to validate the pre-flight test results satisfy acceptance criteria for integration into the integration branch.

- During the build stage a select group of tests are run on each build to ensure the integrated build satisfies build integration acceptance testing.

- During the code test stage functional and performance tests are conducted to ensure the images resulting from the build or set of builds satisfy functional and performance assessment criteria.

- During the regression stage a set of regression tests are conducted to ensure the images resulting from the build or set of builds satisfy regression test assessment criteria.

- During the integrate packages stage a set of tests are conducted to ensure the image packages resulting from the image packaging stage satisfy package assessment criteria.

- During the system test and regression stage the entire application consisting of one or more packages from the integrate package stage are tested to ensure it satisfies system test and regression package assessment criteria.

- During the delivery stage all software that will be is used to deploy the software is tested to ensure it satisfied delivery test criterion.

In the example depicted in the diagram, tests are conducted across the vertical layers as follows:

- Infrastructure tests are conducted to verify the infrastructure variations for deployment satisfy infrastructure assessment criterion.

- Orchestration and automation tests are conducted to verify the software that supports orchestration and automation layers satisfy their assessment criterion.

- Monitoring and management tests are conducted to verify the software the supported monitoring and management of the pipeline satisfy their assessment criterion.

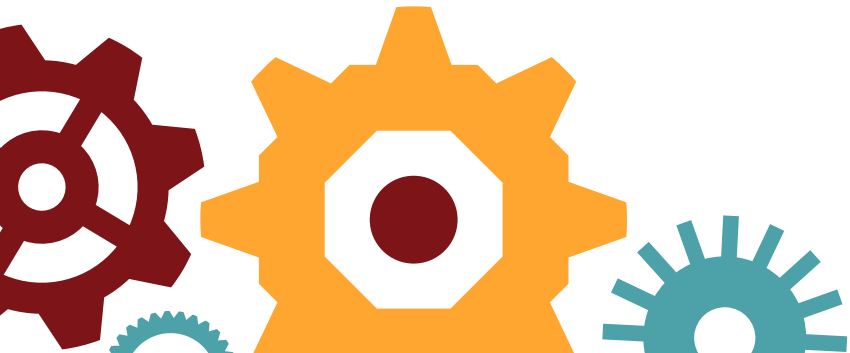
4. DevOps Test Infrastructures

The underlying infrastructure for testing includes all configurations of hardware, software stacks and tools that are necessary to operate tests sufficient for quality assurance of the application and its deployment to production.

The application being tested may be a large monolithic application, 3-tier application configurations or code packaged modularly in separate containers using service-oriented or micro-service oriented architectures.

One of the key best practices for DevOps testing is that tests are to be conducted in a production-equivalent test environment. This ensures tests cover all software configurations and use case variations that the application is expected to operate in, once deployed to production. Therefore, it is essential for the test infrastructure used for each stage to be a replica of production configuration variations.

One of the key best practices for DevOps testing is that tests are to be conducted in a production-equivalent test environment.



The target deployment and test infrastructure may require running software stacks on bare metal servers, virtual machines or containers running in a private data center, cloud or multi-cloud environments. It may require configuring one or more Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) or Software-as-a-Service (SaaS) environments.

The variations may also require a variety of network, cluster, load-balancing, storage and database configurations that the application will be used in during deployment and production use.

Disaster recovery and restoring service infrastructures and procedures may also need to be tested.

Many organizations find that it is too expensive to have dedicated replicas of all possible variations of the infrastructure configurations available for testing prior to deployment.

A key best practice for DevOps is to arrange for an elastic infrastructure for testing. The ability to stand-up and release (i.e. orchestrate) infrastructure configurations as needed during the pipeline for use with specific tests is usually more feasible and cost efficient than dedicated infrastructures. Typical techniques that are employed to orchestrate test environments include dynamic infrastructure configuration tools, infrastructure-as-code strategies, cloud services, Test-as-a-Service providers and use of tools to stand-up and release applications and test resources packaged in immutable containers.

A key best practice for DevOps is to arrange for an elastic infrastructure for testing.

5. DevOps Test Frameworks

A DevOps test framework is a system operated within a continuous delivery toolchain that controls the orchestration and automation of resources for test setup, test execution, and test results. Test resources that a test framework supports may include tests, test tools, applications, test infrastructures, and test data. A test framework can be invoked, controlled and released through an Application Programming Interface (API) so that it can be integrated into a DevOps continuous delivery toolchain. Example features provided by test frameworks include capabilities to orchestrate the test infrastructure, select tests, control the execution of tests, monitor tests in progress and report test results, logs and verdict data.

6. Test Creation

There are a variety of test creation techniques and associated tools available to create high quality, reliable tests. The most common test creation techniques are scripting, capture-replay tests, Keywords, Behaviour-driven and Model-based test creation methods. Each of the methods vary in the cost of tooling, complexity and ease of use. Test-driven development (TDD) is a method in which the tests are created ahead of the code to be tested. TDD is most often applied for unit level tests but may also be applied for functional testing. A key advantage of TDD is that it ensures working tests are available for use in all pipeline stages. Without using TDD there is a tendency for the creation of tests to fall behind the code. Another advantage of TDD is that tests are created without the bias of knowing how the code was written…because it hasn't been written yet.

7. Test Acceleration

Faster continuous testing cycles are preferred because failing early and often is a DevOps goal. There are multiple reasons for this, but it should be evident that faster tests reduce the wait time to get results and since there are typically many development team members looking for results the total developer time savings is huge for every minute of test time saved.[11]

There are many techniques used to accelerate tests such as employing vertical and horizontal scaling of test.

8. DevOps-Ready Test Tools

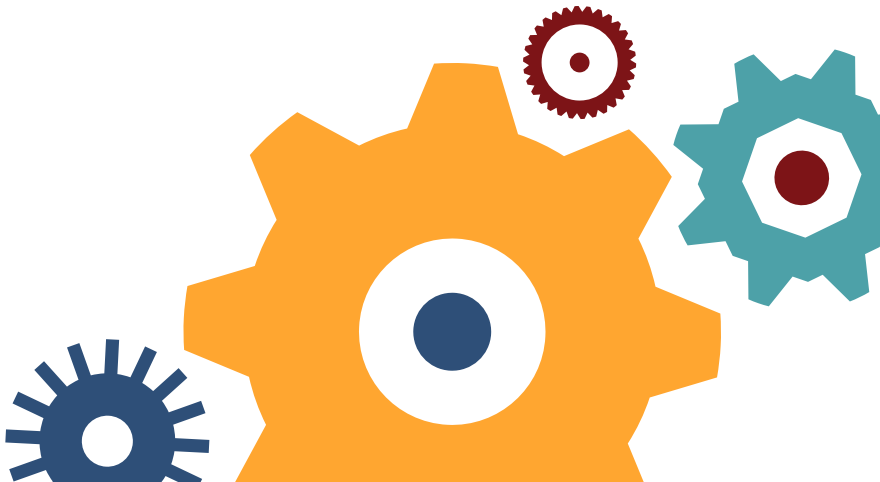
To work well with a continuous delivery toolchain, test tools must provide capabilities to drive an application under test and report results data needed to assess the verdict of a test. Examples of tools may include unit test tools, functional test tools, protocol test tools, performance/load test tools, API test tools, user interface test tools and database simulators. Test tools may be white box tools, gray box tools and black box tools. To be DevOps-ready, the test tools need to easily integrate into a test framework and toolchain and elastically scale on-demand vertically or horizontally, to match the varying capacity and workload demands of tests of software changes going through the continuous delivery pipeline. DevOps ready tools can be orchestrated, scaled, invoked, controlled, and monitored from a API.

Resources, employing fail-fast test design techniques, test framework configurations that accelerate test results monitoring, and test tool configurations.

9. Test Analytics

Fast continuous testing is counterproductive if the analysis of test results does not keep up with the speed of testing. Unless test analysis is as fast as the tests, all that will happen is an accumulated debt of test results to be analyzed resulting in, at best, no net gain in time savings and, at worse, valuable results going overlooked and mass confusion that actually slows down the overall CI/CT cycles.[13]

There are many techniques used to ensure test results analysis keeps up with accelerated tests. Test design techniques, analysis tools built into test frameworks, test results dashboards, test results analysers that run in parallel with the tests are some good examples.



10. Microservices and Containers

Software architected as micro-services and packaged in containers present both challenges and opportunities for DevOps testing.

The microservices approach aims to produce well bounded, networked, service-oriented modules. This is attractive to software application architectures because the individual micro-services can be built and deployed as independent units. The smaller modules are easier and quicker to test compared to large monolithic applications. Packaging a microservice into its own container is attractive because the service can be deployed immutably across any number of infrastructures, scale easily and can be reverted easily in the case of test failures.

From a testing point of view microservices introduce the requirement to validate the contract of each services with any other services that use it. The independence of the microservice or any inter-service dependencies need to be well tested. Performance and reliability considerations of operating services over a network need to be verified. If the application changes affected microservice, or dependent group of microservices need to be regression tested.

Containers offer the opportunity to package test resources in special test containers so they can be conveniently and immutably invoked on-demand for testing changes with elastic scaling benefits.

11. Database DevOps Testing

A strategy is required to verify that changes to a database or changes to an application that uses a database perform as expected, during each stage in the continuous delivery pipeline. Orchestration of the infrastructure, the test framework, and the test creation strategies all need to consider the database and the application that use it. Test data to validate changes need to be available at each stage in the pipeline to avoid bottlenecks. Customer sensitive data needs to be obfuscated from people involved in testing. Tools to replicate production level data volumes are needed to ensure that the tests are conducted on a production equivalent dataset prior to deployment. Database migration, replication and restore tools need to be tested also to ensure they can function quickly enough to keep up with the speed of the DevOps continuous deployment pipeline cadence.

12. DevOps Security Testing

DevOps with Continuous Testing (CT), when implemented according to best practices, is an opportunity for organizations to systematically and affordably integrate security assurance into product and service development. Using DevOps techniques, continuous security testing can be built into software change and deployment operations.[13] Many refer to the application of DevOps practices to security assurance as **“DevSecOps”** or **“Rugged DevOps.”**

13. Continuous Testing Antipatterns

There are multiple antipatterns that arise in DevOps continuous testing because speed and quality run into conflicts with each other. For example, Continuous Testing is a longevity test anti-pattern. The pattern of quick test cycles prescribed by continuous testing appears to be a contrary anti-pattern to the fundamental concept of long duration "longevity" testing. Fortunately, DevOps infrastructures implemented according to best practices have a variety of complementary continuous longevity testing patterns to choose from.[14]

14. Continuous Test Management

All the same requirements of test management for traditional QA testing apply to DevOps testing also. Test planning, test guidelines, test code version management, and version management of test results are some examples. A key difference with DevOps testing is the higher rate of changes. To keep up with the rapid change cadence required for DevOps, all aspects of test management need to be handled by tools that are well integrated into the continuous delivery pipeline and any test framework being used for DevOps test automation. In addition, the test environment must be highly stable because a failed test environment will break the continuous delivery pipeline.[15]

Design technical and business metrics to monitor the success of DevOps testing for technical continuous improvement and to justify business cases needed to implement those improvements.

15. Advanced Continuous Test Topics

Cognitive automation, machine learning, self-remediation, and predictive analysis are emerging techniques for test automation. These techniques increase the ability to meet increased time-to-market pressure with the right levels of quality to secure business outcomes. Adopting smarter automation solutions will be essential to cope with the emergence of smarter applications and smarter products that demand an integrated, intelligent, and automated approach to testing this continuously changing products in rapidly changing business environments.[15]

How Do I Get Started with DevOps Testing?

The primary key to success with DevOps is DevOps test engineering. As this article has explained, DevOps testing is much more than automating tests or integrating a tool into the continuous delivery tool chain. A methodical approach to realize effective and efficient DevOps testing is recommended.

1. Skill yourself and your team on DevOps testing. This likely involves formal training.
2. Engage a DevOps test consultant and appoint a DevOps test architect to conduct an assessment of your current testing practices and define a recommended solution roadmap.
3. The solution roadmap should define a test strategy for the product. The strategy will define the test coverage strategy, how tests will be created, which tests will be conducted at each stage of the continuous delivery pipeline, how test results will be handled and metrics to monitor performance for testing. Select a pilot application.
4. Implement test infrastructure, test frameworks and tools according to the solution roadmap.
5. Train cross functional team members on test creation methodologies and tools.
6. After a pilot project with a one product application and team, expand the implementation to more teams the team and applications.
7. Monitor progress and put in place continuous improvement procedures.

The DevOps Institute course DevOps Test Engineering (DTE)® provides a comprehensive learning curriculum for DevOps testing as well as DevOps Test Engineer certification. Anyone involved in DevOps leadership or DevOps practitioner in Dev, QA, Ops, InfoSec or project management can benefit from the DTE course. Refer to the following link to learn more about the course or find an accredited training provider:

www.devopsinstitute.com

References

1. State of DevOps Report by Puppet Labs, IT Revolution, DORA | June 2017
2. The Phoenix Project by Gene Kim, Kevin Behr, George Spafford | January 2013
3. The DevOps Handbook by Gene Kim, Patrick Debois, Jez Humble, John Willis | October 2016
4. Out of the crisis, W. Edwards Deming| 1986
5. Software Reliability – Principles and Practices, Glenford J. Myers| 1976
6. The Art of Software Testing, Glenford J. Myers| 1979
7. From QA to Continuous Testing, Marc Hornbeek| DevOps.com | February 2015
8. DevOps – it's all about Continuous Testing, Marc Hornbeek| DevOps.com | February 2015
9. Seven Best Practices for Accelerating Continuous Testing, Marc Hornbeek| DOES 2015 |November 2015
10. DevOps Impact on Testing and Culture, Shirly Ronen-Harel| August 2015
11. Continuous Testing Accelerated, Marc Hornbeek| DevOps.com | March 2015
12. Test Results Analysis at the Speed of DevOps, Marc Hornbeek| DevOps.com | March 2015
13. DevOps Makes Security Assurance Affordable, Marc Hornbeek| DevOps.com | April 2015
14. Isn't Continuous Testing a Longevity Test Anti-pattern?, Marc Hornbeek| DevOps.com | March 2015
15. Isn't Continuous Testing a Longevity Test Anti-pattern?, Marc Hornbeek| DevOps.com | March 2015
16. Continuous Testing Stability, Marc Hornbeek| DevOps.com | April 2015
17. World Quality Report, 9th Edition, | 2017-2018

About the Author

Marc Hornbeek, also known as “DevOps_The_Gray”, is a DevOps consultant advisor, teacher, mentor and author. Author of the Continuous Delivery Architect course and the DevOps Test Engineering course for the www.DevOpsInstitute.com. Blogger on www.DevOps.com. Specialist / expert in DevOps maturity assessments, DevOps transformation roadmaps, Continuous Integration (CI), Continuous Test automation (CT), Continuous Delivery and Deployment (CD), Lab-as-a-Service (LaaS), and DevOps cloud-native infrastructures. Awarded IEEE outstanding engineer, Western USA. Senior IEEE member, 42 years. To learn more of Marc’s publications and services refer to https://www.linkedin.com/in/marchornbeek/ and www.ConfinityConsulting.com.

DevOps Test Engineering (DTE)®

Continuous Testing Skills for All IT Professionals

DevOps INSTITUTE
THE CONTINUOUS LEARNING COMMUNITY

COURSE OBJECTIVES

The learning objectives for DTE include a practical understanding of:

- The purpose, benefits, concepts and vocabulary of DevOps testing
- How DevOps testing differs from other types of testing
- DevOps testing strategies, test management, and results analysis
- Strategies for selecting test tools and implementing test automation
- Integration of DevOps testing into Continuous Integration and Continuous Delivery workflows
- How DevOps testers fit with a DevOps culture, organization, and roles

Find an Education Partner Today

DevOpsInstitute.com