TESTING IN DEVOPS: CONCEPTS, BEST PRACTICES & MORE



DevOps empowers smooth collaboration and communication between development and operations teams in today's competitive software development environments.

In DevOps, the two teams work together, sharing responsibilities towards achieving their primary goal: frequent and faster delivery of high-quality software that satisfies evolving customer needs. DevOps practices, along with relevant tools and technologies, drive organizations to accomplish tasks as efficiently as possible.

Some DevOps practices like Continuous Integration and Continuous Delivery (CI/CD) support frequent software releases. That means that testing plays an integral part in helping to maintain software quality at each step of the development lifecycle.

(This article is part of our DevOps Guide. Use the right-hand menu to go deeper into individual practices and concepts.)

How traditional testing worked

Software testing is not a new concept. But testing in traditional environment looks very different from testing in a DevOps environment.

In the days of the traditional waterfall methodology, software testing looked like this:

- Spanned only one phase of the life cycle
- Started after the software was developed completely
- Was a manual process that was highly error-prone and took a long time to complete

A huge difference is that software testers sat on a separate team, isolated from the development team. If any bug was detected at the testing stage, it was challenging and costly to go back and do that change. The reason for that was structural: the particular error scenarios were to be well-identified at the beginning.

Under these circumstances, it was difficult to maintain the required standard and quality within the expected timeline.

Features of testing in a DevOps environment

Software testing evolved considerably over the years after Agile started its ascent. Since then, faster and collaborative testing strategies, tools, and technologies have been introduced to the testing sphere.

This is what testing in a DevOps environment looks like:

- Testing is a continuous and automated process that enables continuous and faster delivery of software.
- Testing spans every stage of the software development lifecycle (SDLC).
- Each step of the SDLC involves different forms of testing. This minimizes backtracking in the case that you've detected an error.
- Testing is no longer the responsibility of one particular team. Shared testing responsibilities allow everyone to understand the impacts behind each change.

(Learn how to set up your own CI/CD pipeline.)



DevOps adopts Shift Left Testing Approach

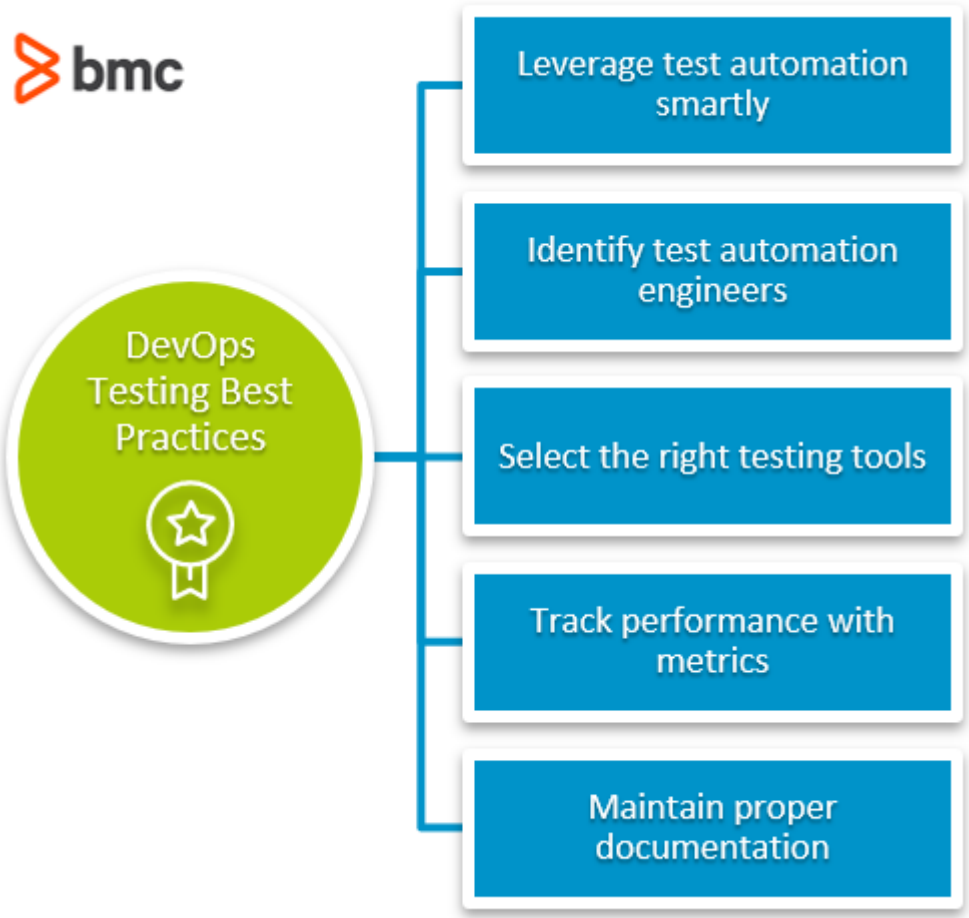
Importantly, DevOps culture adopts the Shift Left testing approach, which contrasts with traditional environments where testing happens at the end of the development.

Shift Left testing pushes the testing to the left, the earlier steps, of the software development process. There, testing is also started when the development is started. This approach helps to identify bugs as early as possible.

These inherent features of a DevOps testing environment contribute significantly towards improving software quality. Still, the success or failure of the testing strategy depends on how well organizations implement DevOps best practices for testing.

DevOps Testing Best Practices

In this section, we will go through some best practices for testing that organizations need to embrace in order to maximize your DevOps value.



Leverage test automation (smartly)

In a DevOps environment, developers frequently merge the code to a central repository. This means that the code updates continuously through continuous integration (CI). To prevent the risk of errors, you have to continuously test the code through different types of tests, including:

- Unit tests
- Functional tests
- Acceptance tests
- Integration tests

As a best practice, you can automate these tests to get faster and earlier feedback on continuously integrated code. More successful DevOps teams have a larger percentage of automated test cases, and they have often integrated automation suites.

As you progress in the SDLC, test automation spans further from code level into areas like:

- API testing
- Performance testing
- Load testing
- Endurance testing

This doesn't mean that everything in the testing scope should be automated. There should be some space for manual testing. This caution is important. If you automate a test process that isn't valuable, you're simply automating and introducing more waste into the process.

Recommendations for effective test automation include:

- Using good, quality test data
- Identifying test cases that are good candidates for automation
- Running test cases in parallel to improve automation speed
- Continually revising the testing plan with an up-to-date automated testing scope

Teams can often create and run more test cases to prevent bugs by utilizing test automation intelligently.



Identify test automation engineers

While testing is a shared responsibility among every team, organizations tend to use the specific

expertise of test automation engineers to increase the percentage of test automation coverage. For example, a dedicated team of such engineers would play a vital role in achieving automation targets.

A good test automation engineer might be:

- A former manual tester who subsequently learned to write automation scripts
- A software developer who trained as an automation engineer

Test automation engineers develop the overall automation strategy of the organization. They are responsible for:

- Identifying automatable test scenarios
- Creating automation scripts using a chosen test automation framework
- Finding the tool that best fits your team or organization's testing approach

Typically, during a software release, the automation engineer takes ownership in executing automated tests on the related environment and reporting any identified bugs. Then they work closely with the development teams to find the solutions to resolve those bugs. As part of their work, they clean up existing test cases according to the new changes introduced to the system.

You can see how test automation engineers contribute to leveraging the pros of test automation in a successful DevOps culture.

Select the right testing tools

To leverage the benefits of test automation, you need to integrate the right testing tools for your organization—not some "best of" testing tool.

When choosing a test automation tool, the first thing to evaluate is if your team has the required skills and expertise to use that tool. For example, some open-source testing tools require a decent level of programming skills to use them. Do your test engineers have this?

Then estimate the total cost of the tool, including training costs, updates, and maintenance, to see if it is within the testing budget. Always check if decent technical support is available for the tools. If not, are you prepared to service it yourself?

Automated testing tools should enable easy writing and execution of test cases without having to make complex configurations. Here is a list of both open-source and commercial software testing tools that are widely being used by many organizations.

Here are some of the most popular test automation tools:

- Selenium is an open-source framework that primarily automates web application testing. It provinces a suite of software for different testing needs. Selenium scripts can be written in multiple programming languages.
- Katalon Studio, ranked among the best automated testing software, has both free and paid versions. It can be used for automated web, API, desktop, and mobile testing. It supports many platforms.
- JMeter is a Java-based open-source software used for performance and load testing. It can be used to test many protocol types.
- SoapUI helps you test REST, SOAP, and GraphQL APIs. It is another open-source, cross-

platform automation tool that has a convenient graphical user interface (GUI).

(Learn more about top continuous testing tools in the software industry.)



Track performance with metrics

Using metrics to evaluate the success or failure of testing is another best practice. This practice enables management to get a clear picture of how the changes introduced to the software has impacted the organization.

You can track key metrics such as:

- Number of test cases passed vs failed
- Number of bugs identified
- Frequency of failing test cases
- Execution time of the automation suite

These metrics provide insights into areas that are highly vulnerable to failures, and continuous testing yields immediate values for them. Metrics also enable teams to foresee if the number of bugs will increase or decrease with more changes introduced to the system over time. Then, more innovative solutions can be applied to overcome failures.

Test execution time metrics help automation engineers to identify better ways of writing test cases that increase performance. As the execution of automation suits has become a significant step of a software release, test execution metrics helps to plan the release timeline effectively.

Maintain proper documentation

Maintaining proper documentation makes the testing process more organized and transparent to everyone in the organization. Successful DevOps teams frequently create testing-related documents, including:

- Quality Management Plans (QMP)
- Test summary reports
- Test case specifications
- Risk assessment reports
- Regression test reports

These are usually created either at the beginning of the testing phase or provided as the outcome of

test results. Documents like QMPs or Test Management Plans offer a comprehensive overview of the testing planned to carry out, covering every aspect of the features that are tested. Test summary reports provide an in-depth understanding of the test results.

Of course, another value for documentation is that it's great reference material for trainees. Proper documentation helps save organization costs.

As a best practice, keep all documentation in a safe place where they are accessible and visible to everyone in the organization. Standardize document formats and use templates to preserve the quality—hence the value.

DevOps testing is key to high-quality software delivery

For many organization, DevOps is the ideal solution to become, or remain, more competitive in your market. Continuous testing is an essential part of CI and CD pipelines that helps to deliver frequent, high-quality software.

The success of testing greatly depends on the best practices you adopt in your DevOps culture. Automation and its related tools are at the heart of DevOps testing strategies.

Related reading

- BMC DevOps Blog
- Quality Assurance (QA) in Software Testing: QA Views & Best Practices
- Automation In DevOps: Why & How To Automate DevOps Practices
- What's Testing as a Service? TaaS Explained
- SRS: Software Requirement Specifications Basics
- 15 Best Practices for Building a Microservices Architecture