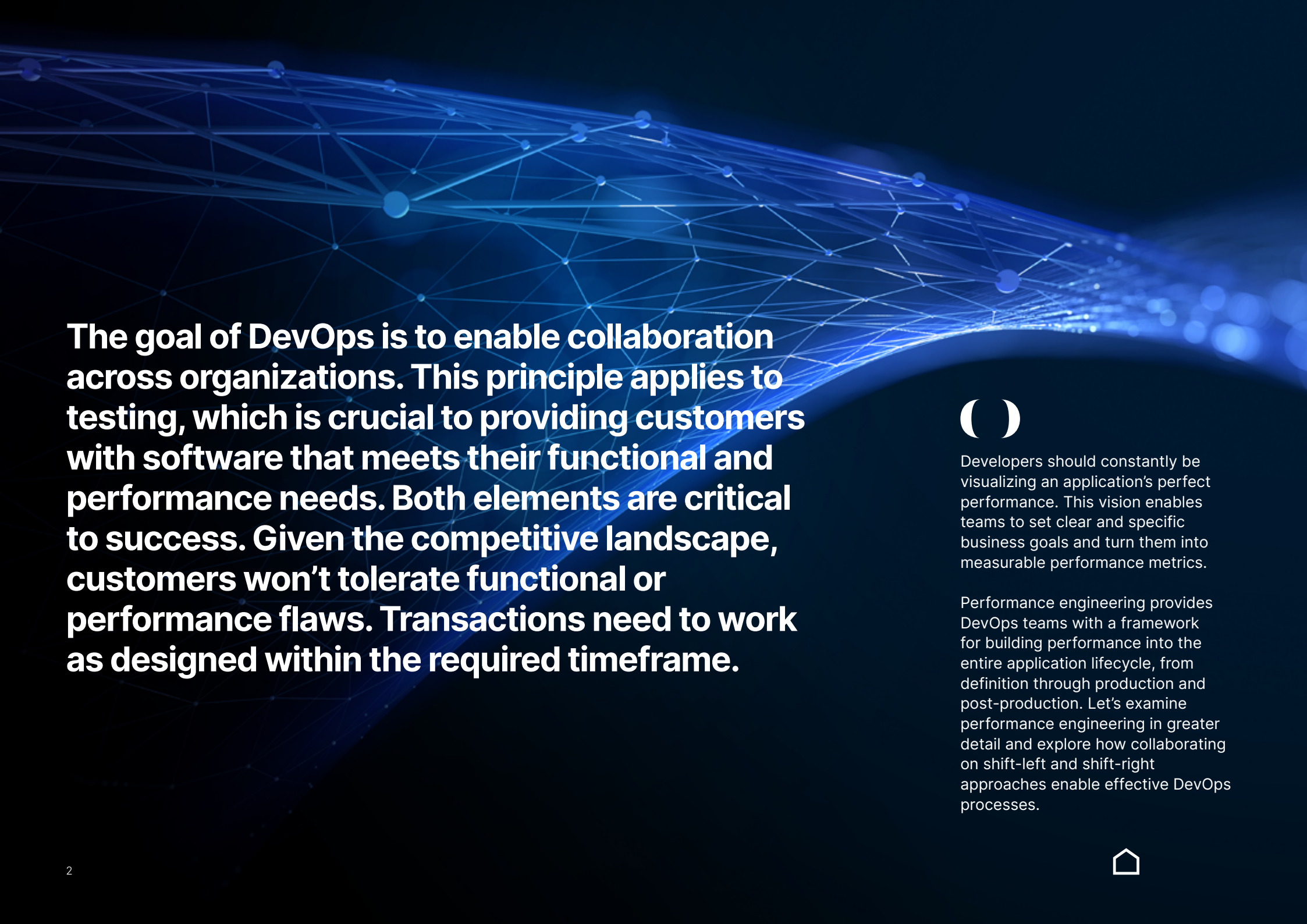


opentext™



# ADM Market Insight: How Performance Engineering Enables DevOps



**The goal of DevOps is to enable collaboration across organizations. This principle applies to testing, which is crucial to providing customers with software that meets their functional and performance needs. Both elements are critical to success. Given the competitive landscape, customers won't tolerate functional or performance flaws. Transactions need to work as designed within the required timeframe.**



Developers should constantly be visualizing an application's perfect performance. This vision enables teams to set clear and specific business goals and turn them into measurable performance metrics.

Performance engineering provides DevOps teams with a framework for building performance into the entire application lifecycle, from definition through production and post-production. Let's examine performance engineering in greater detail and explore how collaborating on shift-left and shift-right approaches enable effective DevOps processes.



## What is performance engineering?

**Performance engineering** is the process of ensuring that non-functional elements meet or exceed standards. Performance engineering seeks to deliver bug-free applications as fast as possible through collaboration among all sections of a DevOps team. It operates, from the beginning, with established performance metrics in mind. This approach also enables engineers to detect potential issues as early as possible.

The user experience plays one of the largest roles during short development cycles, frequent releases, and changing market needs. In response to this trend, you need to take a consumer-focused approach for quality standards in every phase of the SDLC. When done right, performance engineering enables QA engineers or testers and developers to build all the necessary performance metrics from the initial design.


Implementing a continuous testing environment provides metrics and focus for all parts of the DevOps process. Yet, a continuous testing environment is only one element of performance engineering. Performance engineering provides a proactive, continuous testing discipline built around four major components:

**Ensuring performance is a priority for everyone**—developers, testers, and performance engineers.

**Increasing the span** of performance engineering by bringing it into the CI/CD process.

**Monitoring performance** throughout the software lifecycle (from build to production and post-production).

**Using performance data** to collaborate across teams to improve performance.



**Performance engineering seeks to deliver bug-free applications as fast as possible through collaboration among all sections of a DevOps team.**



## Who's responsible for performance engineering?

Performance engineering is everyone's responsibility, not just the performance engineer. It should be designed into the entire DevOps process, not relegated to the end of the cycle when your app launches. You must incorporate user experience and your application's managed performance throughout the app's lifecycle.

Performance engineering is data-driven. It needs tooling integrated into DevOps processes to provide this data, rather than being viewed as just an add-on. That way, teams can access the performance metrics they need to ensure they meet business goals. Although some performance engineering is automated, it still takes people with a variety of expertise to work together to implement it successfully.



## Shift-left testing: bringing performance engineering into the development cycle

DevOps is about developing and deploying applications to production as rapidly as possible. The DevOps process has emphasized ensuring the application meets the customer's functional expectations. The non-functional elements have traditionally taken a back seat to functional needs in DevOps methodologies.

But how an application behaves in a production environment is critical to this task. Shift-left testing moves the exploration of non-functional aspects earlier into the development cycle.

The IBM Systems Sciences Institute estimated the cost to fix bugs found during testing could be 15 times more than the cost of repairing those found during design. Shift-left testing occurs when the application code is more straightforward, making errors easier to detect and remediate. Also, because team members discuss and fix errors collaboratively, they can engineer solutions for future development and avoid, rather than remediate, defects. The result is reduced testing time and cost with improved software quality.

Integrating shift-left testing into the CI/CD pipeline enables unit testing and test script automation. Rather than waiting until near the production deadline, shift-left testing provides

the information developers need to modify the code before it moves downstream. Code that is being developed can have unexpected interactions with code that is already committed. So, you should test interactions between code modules as early in the development cycle as possible. For example, if the module passes or receives information from a committed module, ensure the information passes in the expected form.

**The cost to fix bugs found during testing could be 15 times more than the cost of repairing those found during design.**

You can integrate shift-left testing into your Agile DevOps cycle at any point. For example, when your team adds new functionality to the CI/CD pipeline, it's beneficial to unit test performance before and after commitment to determine its effect. Similarly, if you're expanding the types of production environments, it's helpful to understand performance before and after the addition.

This piece of the performance engineering process corrects errors earlier in the lifecycle and enables the DevOps team to learn from the mistakes and avoid them in the future.



## Shift-right testing: monitoring the customer experience

Shift-right testing is the process of continually testing applications or software after deployment in its actual production environment. This testing discovers defects that teams missed in the development environment. Additionally, data pulled from production helps make tests more realistic and reduces risk and assumptions.

Organizations deploy software to various environments, from multicloud to mobile. Each of these environments may introduce multiple operating methods, security risks, and third-party applications into the mix. These environments are in almost constant change because of technological advances, security patches, and other factors. In a cloud environment, organizations no longer have complete control of the production environment, as they did in their on-premises architecture. So, the production environment is in a constant state of change that may affect performance. Shift-right testing is designed to uncover new issues before the customer does.



The goal of shift-right is to ensure software performance. Shift-right testing happens in production, so it shows how the software behaves in a real-world setting. It also is designed to future-proof the application. Having spent considerable resources rolling out the functionality, you don't want some of your customers to lose their tempers because of changes to the production environment.

For DevOps, shift-right testing uncovers issues that nobody anticipated in earlier parts of the development cycle. Rather than just letting operations fix the problem, this performance engineering approach enables the DevOps team to understand the issue's root cause and adjust processes to avoid it in the future.



## Collaborating to implement engineering in DevOps

Shift-left and shift-right testing provide the data that feeds performance engineering. [Performance engineering](#) is about analyzing this data and working collaboratively across DevOps to build performance into software continuously throughout its lifecycle.



**In the world of software, from development to deployment and customer use, the environment is constantly changing. The changes may come from deploying additional customer functions or applying security patches and other updates to the host environment. Implementing a performance engineering program fed by continuous testing and analytics is a priority in coping with these changes and minimizing their impact on the customer.**

While performance engineering is everyone's concern, you still want to minimize its impact on other internal job functions as well, so the entire DevOps team can be as productive if possible. Automation, reuse, and integration are three pillars that help.

One of testing's more time-consuming elements is developing a test script. Scripts simulate how users will interact with the application that you are testing. Creating a testing script takes time away from development. You want to be able to minimize the script development time, as well as make the scripts a reusable asset.

Having an online editor that records your navigation as you go through a business process is one way to help speed test development. Once it captures the basics, you can edit and adapt these actions to create various scripts that represent users' actions. You can replay them in volume to simulate performance in multiple environments and gather

and analyze test performance data. This test development functionality needs to be available across your integrated development environment (IDE) of choice and sharable across testing platforms.

Teams use different tools across their development environment. You want to have a single performance engineering toolset that works across all IDEs, protocols, and CI/CD environments. Here, integrating your performance engineering toolset across all combinations of environments is vital in saving time and effort. To do this, you want to choose tools that provide a wide variety of integrations, including being fully compatible with open source. If your environment has multiple CI/CD platforms (for example, Jenkins, Azure DevOps, and TeamCity), you can still use a single toolset. Having a single tool that supports testing and analytics across all your environments breaks down tools silos and removes roadblocks to collaboration.

Automating test and execution analysis is another way to make performance engineering more efficient. One factor in accomplishing this is to provide a toolset that centrally automates test execution and analyzes the results, reducing duplication of effort across teams. Teams can improve performance using a single result set to determine any problem's root cause and avoid passing defects downstream.



## Enabling continuous testing and feedback

Change is constant in software development, production, and post-production. For testing to be most effective, it needs to monitor the effects of changes on performance as close to real-time as feasible. Shift-left testing does this for the development teams, while shift-right testing enables you to monitor real-time engagement with customers. Together they provide a framework for continuous testing and gathering information.

To move from continuous testing to performance engineering, you need an infrastructure that seamlessly takes in testing data and provides the analytics to diagnose performance defects. With the analysis in hand, teams need to work collaboratively to determine the defect's root cause. Feedback loops are essential to performance engineering because even an error detected in production may have originated during code development. This analysis is necessary for correcting the current issue and engineering changes to the process that protect against repeating the problem.

Collaboration requires various people with differing responsibilities and toolsets to work together, analyzing the data and all phases of testing, from script development to implementation automation. This scenario requires a toolset integrated across tools and job responsibilities. A performance engineering tool needs unified functionality with various interfaces and integrations to accommodate multiple roles.



## Summary

Properly tooled and implemented continuous testing can lead to performance engineering. It requires tooling on the performance testing side that integrates into various job functions, tools, environments, and production environments to provide a consolidated view of the analytics.

The LoadRunner suite of tools enables your teams to work collaboratively and ensure performance across your entire software lifecycle. For more information, [explore OpenText' integrated performance engineering.](#)

Learn more



**opentext™**

