# 3 AREAS WHERE TRADITIONAL APMS LEAVE YOU EXPOSED

ZEROTURNAROUND

# Contents

# **Introduction:**
# Your Worst Performance Management Nightmare

If you're responsible for creating or managing a customer-facing application for your organization, you have a long list of things to worry about. A scenario like this may actually be at the top of the list: you've recently launched a new version of your application to the world, and customers start finding serious issues in production. Excessive latency in the application is destroying the UX. While the APM you're using is picking up on some of these issues, it is catching them too late. Your customers are complaining directly to the company, and voicing their displeasure on social media, and your management team is asking, "How did this happen?"

This nightmare scenario is the kind of thing that even the best companies in the world can experience. Google, for example, found that traffic dropped by 20% with just an extra half second in search page generation time. Amazon discovered that each additional 100ms of latency resulted in 1% fewer sales [1]. If even these giants can fall victim to application issues in production, it can happen to anyone.

1: Linden, Greg. "Marissa Mayer at Web 2.0." Geeking with Greg, 9 Nov. 2006, glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html.

In this e-book, we'll explore why relying solely on traditional APMs for performance testing can be a risky endeavor. We'll explore the following three major areas where traditional APMs may be leaving you open to risk:

- Finding Performance Issues
- Diagnosing the Source of Performance Issues
- Fixing Performance Issues

# 01 Finding Performance Issues

One of the biggest questions for those managing application performance is whether they are finding issues as early as possible. The answer for most organizations is no. In fact, 75% of application issues make it to production (see following page for breakout of where issues are found) [2].

Most APM solutions were designed to work in production only. That means organizations relying solely on traditional APMs are exposed to risk. This section explains why traditional APM tools don't always find issues early enough.

2: Maple, Simon, and Oleg Shelajev. Developer Productivity Report 2015: Java Performance Survey Results. ZeroTurnaround, 2015,
Developer Productivity Report 2015: Java Performance Survey Results, zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/.

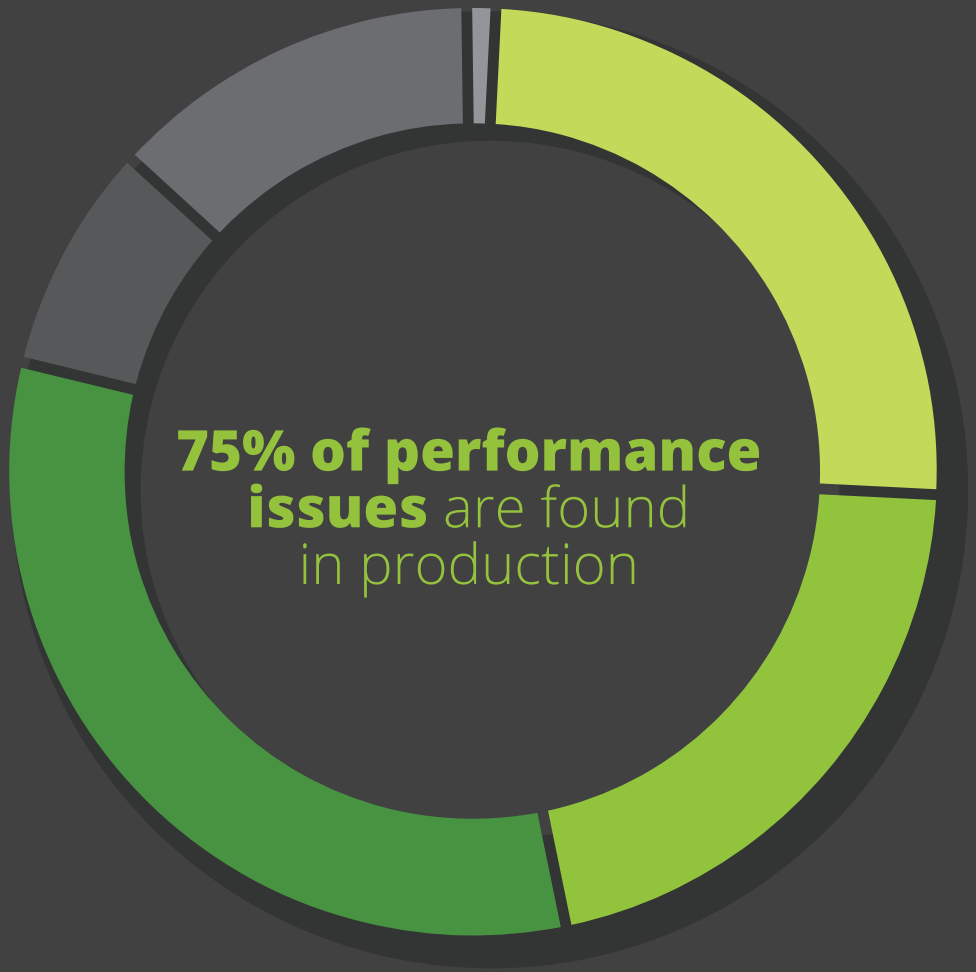
# **How do you most commonly find** performance issues in your application?

REBELLABS by ZEROTURNAROUND

**13.00 %** 3rd party profilers

**8.3%** Home grown software

**31.9%** User reports & feedback




**1.1%** Other

**25.1%** Performance monitoring tools

**20.6%** Systems faults/ crashes

Maple, Simon, and Oleg Shelajev. Developer Productivity Report 2015: Java Performance Survey Results. ZeroTurnaround, 2015,
Developer Productivity Report 2015: Java Performance Survey Results, zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/.

# Traditional APMs aren't built **for the testing phase**

While traditional APMs are generally built to focus on production environments, some organizations try to use them for performance testing in the earlier stages of development and test. What they often find is that the metrics and reporting aren't effective for these stages. A production-focused APM will provide a statistical analysis of your application performance that is essentially an aggregated result of thousands of transactions. This can help point to major issues that may be affecting performance, but because there isn't any transaction detail, it can be a very vague indicator of the problem. Bottom line: traditional APMs are indicators of trends but those trends aren't always real problems.

# **Developers are disconnected from** how their code changes affect overall performance.

In many companies, we still have a situation where developers aren't tied directly to the performance of the applications they build. They build their applications and throw them over the wall to an operations team in production, and when that team finds issues, they are thrown back to the development team to fix. The DevOps movement has urged companies to try to get away from this by creating one big virtual team and to "shift left" some of the functions from operations to development.

But even in DevOps environments, we still see the majority of testing happening in production, and the majority of APM tools geared to operations. Because of this, developers don't always feel they are responsible for providing performant code, as long as they are meeting functional requirements. This has created a bit of a divide between development and operations that still makes it difficult to find issues.

In order to bridge across these two teams, developers should have more of an ability to influence the performance of the applications they're building. Today's production-focused APMs don't give them the ability to do that.

# 02 Diagnosing the Source of Performance Issues

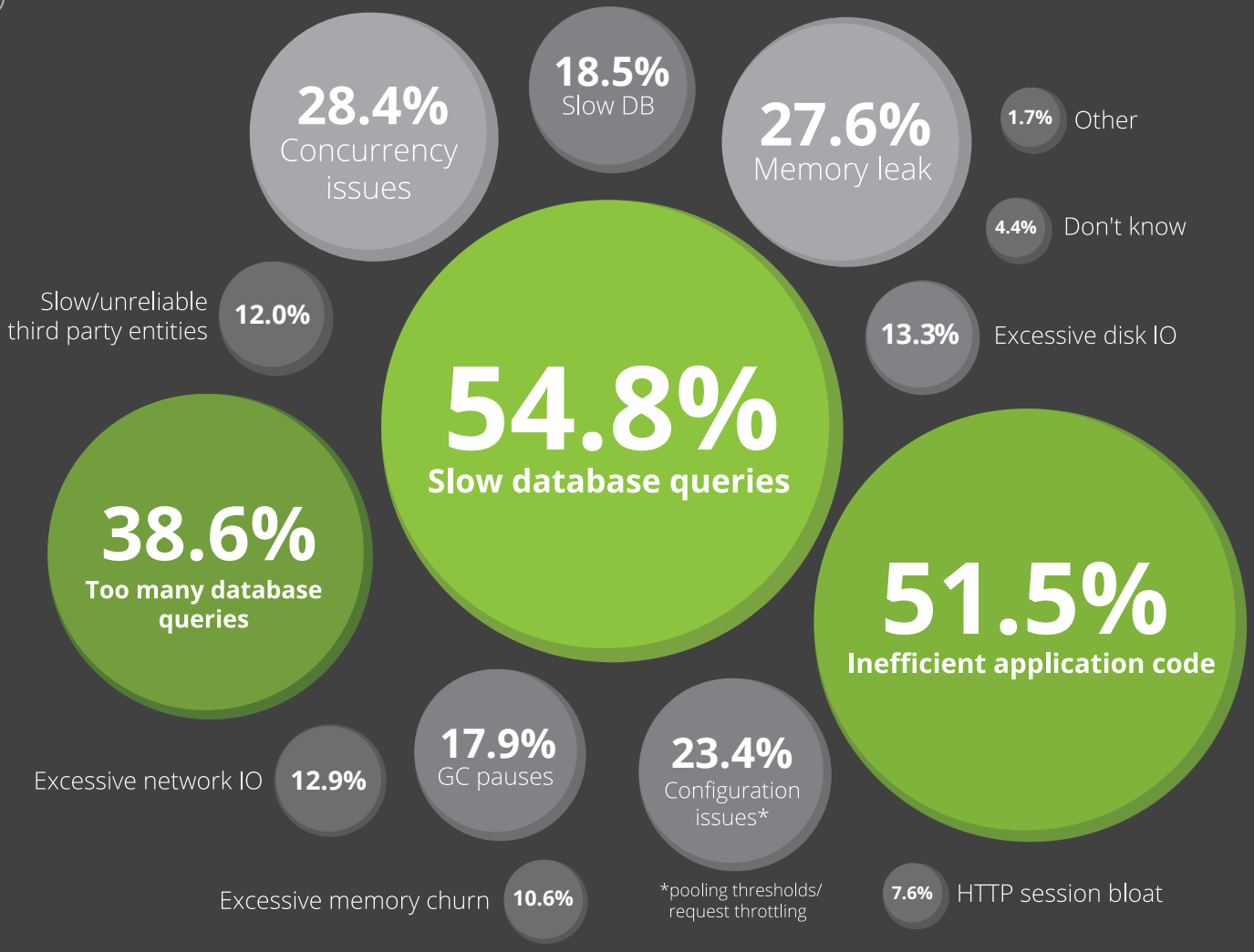
Once you've found an application issue, you have the difficult task of diagnosing the source of the issue. This is a task that becomes more and more difficult as you move away from the development process into production. Teams that test too late are forced to reverse-engineer performance issues that are happening in complex infrastructures. In reality, 86% [2] of root causes are application-level issues that will manifest in development environments, and scale with the environment (see following page). It makes sense therefore to try to catch these application-level issues early when it's easier to find the root cause. This section examines why it's difficult to diagnose the source of application issues if you rely solely on traditional APMs.

2: Maple, Simon, and Oleg Shelajev. Developer Productivity Report 2015: Java Performance Survey Results. ZeroTurnaround, 2015, Developer Productivity Report 2015: Java Performance Survey Results, zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/.

# What are the typical root causes you most often experience [2]

REBELLABS by ZEROTURNAROUND

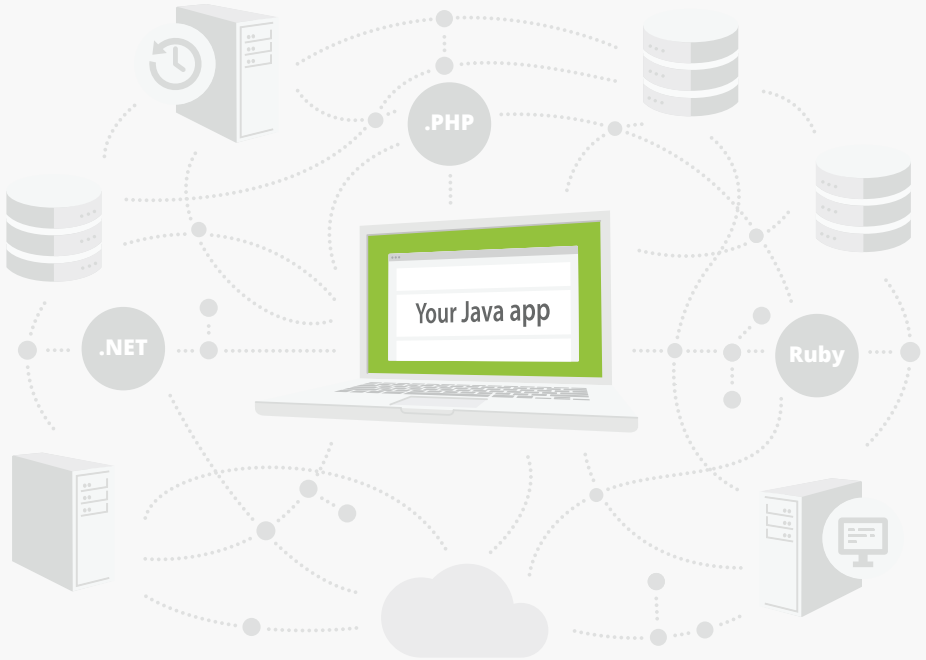
** Answers were multiple choice, so the numbers don't add up to 100%. Deal with it :)*




**86% of issues** have a root cause in the application

# Overly complex scenarios

Once an application makes it to production, it is part of a system with millions of pieces. It is no longer just about whether the application works, but is about all of the technologies that surround the app, from the network infrastructure to distributed systems. A study found that on average, a single transaction uses 82 different types of technology [3]. This makes trying to diagnose the source of a performance issue in production like finding a needle in a haystack.




Because this complexity makes it difficult to accurately diagnose the source of the issue, most problems aren't actually solved, they're simply patched. Worse yet, hastily delivered fixes often break something else, and with every day that passes, the problem gets worse and more convoluted.



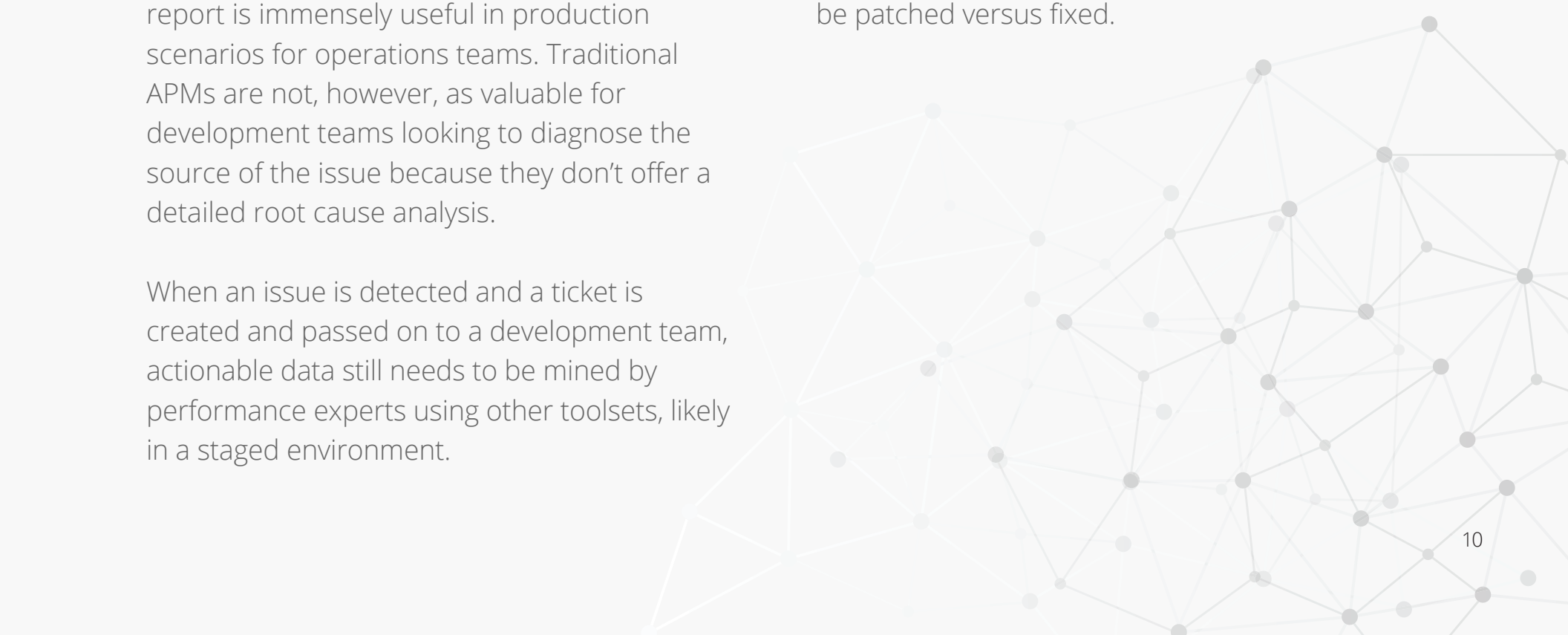
3: Enforce Best Practices. Dynatrace, www.dynatrace.com/resources/ebooks/javabook/enforcing-development-best-practices/.

# **No Root Cause** Analysis

As we already covered, traditional APMs are high-level enough to tell you that a problem exists and and point to the general area that is affected. They're built to monitor incredibly complex infrastructures, so a general health report is immensely useful in production scenarios for operations teams. Traditional APMs are not, however, as valuable for development teams looking to diagnose the source of the issue because they don't offer a detailed root cause analysis.

When an issue is detected and a ticket is created and passed on to a development team, actionable data still needs to be mined by performance experts using other toolsets, likely in a staged environment.

The issue may be conditional and hard to reproduce, delaying the diagnosis even further, especially if you don't have any affected customers volunteering to be guinea pigs. All of this again leads to situations where an issue may be patched versus fixed.

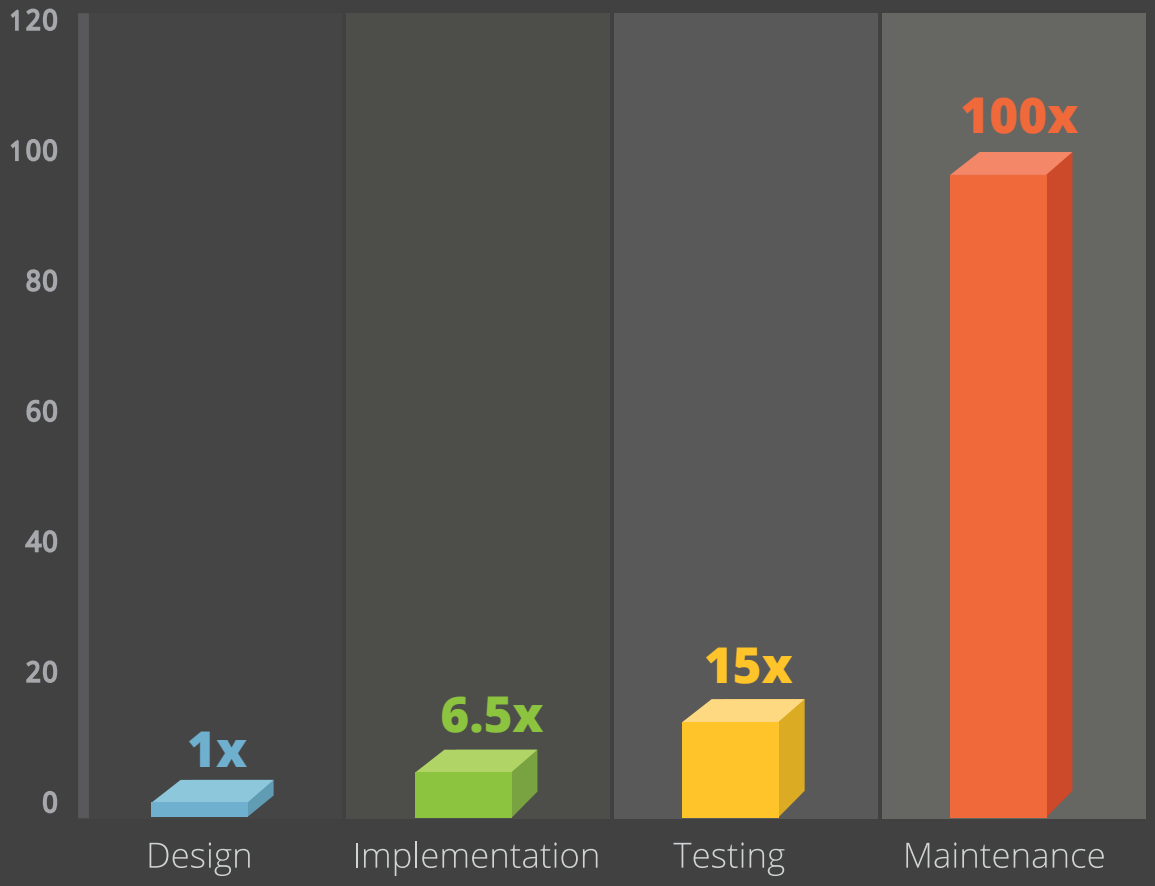


# 03 Fixing Performance Issues

This is the area left most exposed by traditional APMs, as issues are ultimately fixed in development. Production-focused APMs don't line up with the workflow of a developer's day-to-day, so adoption and usage among development teams is a challenge; developers are already dealing with tight deadlines and product pressures, so the complexity of traditional APMs simply does not make it worth their time to figure out how to get actionable data.

On top of that, traditional APMs are seen as absolute overkill in a development environment. After all, they're built for operations, not development, and have many features that developers don't need. They alert you of an issue and point you in a general direction, but they don't provide low-level data presentations that cater to the needs of developers fixing the issues. Because of that, companies run into the following problems when trying to fix issues with traditional APMs.

## IBM System Science Institute Relative Cost of Fixing Defects




The findings from an IBM Systems Sciences Institute research paper show that the time spent on fixing bugs grows exponentially the closer you get to production.

Dawson, Maurice & Burrell, Darrell & Rahim, Emad & Brewster, Stephen. (2010). Integrating Software Assurance into the Software Development Life Cycle (SDLC). Journal of Information Systems Technology and Planning. 3. 49-53.

# **No Fix Validation** Available

Setting up and configuring a traditional APM on a development machine is a large task for potentially little return, as they don't provide features that aid in isolating, fixing and testing an issue in a development environment. Traditional APMs are unable to provide developers with immediate feedback so they can see how code changes are impacting the performance of the application they're working on.

In order to verify a bug fix, development teams have to wait until it's been deployed to production. The fix-test iterations are incredibly costly in time and business-impact if the bug is live.

## **Long feedback loop** between the owner of the code and manifestation of issues in production

The process for fixing problematic code often involves going to the author of the code with the assumption that he/she can easily pick up where they left off. However, because it can often take months for code to be released into production from when it's developed, the developers aren't seeing this problematic code until long after they created it. At this point, the code may be unfamiliar, even to the developer who created it, and others may have built on top of the problematic code making it part of a big spaghetti codebase. In the time it takes to research, replicate and develop a fix for an issue, hundreds and thousands of customers can be affected.

# Takeaways

**The way that most companies currently handle performance management is broken.** When you wait until production to catch issues with your application, your customers will find them before you do. And when you take issues that are found in production and send them back to development teams to fix, it will take longer and cost more than if you had fixed them in the development or test phases to begin with. Every true DevOps team should take a close look at how they can improve the speed with which they find, diagnose and fix performance issues.

**If you're not testing early, your customers are your testers.** If you're subjecting real users to production code that hasn't been thoroughly performance tested, this is a great recipe for losing your customers.

**If you're testing early with production APMs, you're not using the right tools.** Traditional APMs are built for operations, and are essential to production, but are not built for developers in testing and development.

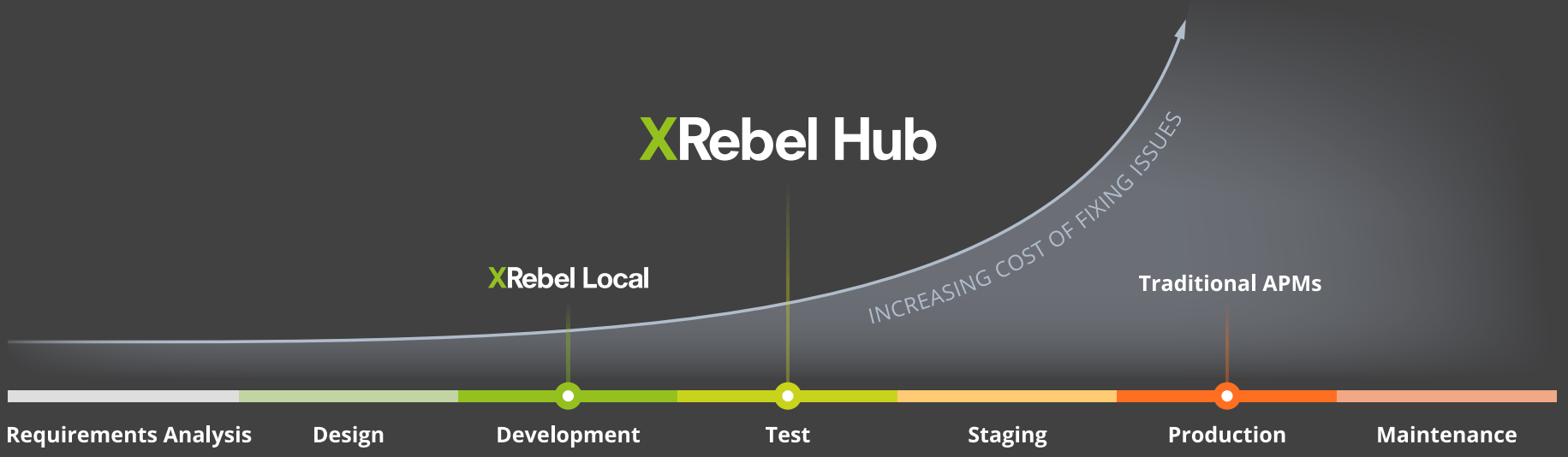
There is now a new kind of APM that is built specifically for development and test. Organizations that want to shift left to catch performance issues earlier, need to also shift their toolset towards development-focused solutions.

For more information, go to xrebel-hub.com

# XRebel Hub

## The Only APM for Development & Test

XRebel Hub is a new APM product from ZeroTurnaround built to catch performance issues in the test phase. Combined with XRebel Local, this product fulfills the promise of a complete APM solution for development and testing.




For more information, go to xrebel-hub.com


ZEROTURNAROUND