Performance engineering: much more than performance testing

EY
Building a better working world

We've all heard about dramatic website or program performance failures, from crashing government portals, to Black Friday e-commerce disasters, to unplayable game updates. The biggest performance failures may even have destructive effects on lives and livelihoods. They often end up splashed all over the news, are met with complaints from users and chagrin from owners — and organizations often take a big reputation hit to go along with it. But even smaller performance failures, such as long wait times, impact user experience and can lead to customers leaving in droves for greener (and faster) pastures.

When we know the long-lasting effects performance failures can have, why do they still happen, and how can we avoid them? The answer is to consider the risks a project faces from failures, then implement a performance engineering process that is proportional to those risks, beginning with the planning process.

Content

Performance = speed and scale

Most of us think of performance as how fast something happens – for instance, how fast a website loads or how fast we get search results. But another key aspect of performance is scale: how well the program works when you increase the load. What happens when you go from a single user to a hundred users? Thousands of users? How about millions? Often, companies delay evaluating scalability until the last minute; but when problems inevitably arise, there is little or no time left to fix them before launch. It's important to remember that scalability and speed are equally important aspects of performance and are much easier to manage when testing starts early.

Implementing the four dimensions of performance engineering early and thoroughly will result in a better quality product, a better user experience and reduced or eliminated delays in going live.

The four dimensions of performance engineering

The terms "performance engineering" and "performance testing" are often used interchangeably, but performance testing is just one part of the discipline of performance engineering.

Performance engineering is a broad, holistic view of performance across the entire lifecycle of an application or program and consists of four parts:

1. Architecture and modeling
2. Performance diagnostics and profiling
3. Performance testing
4. Monitoring and capacity

Architecture and modeling
Architecture and modeling should begin before development. This stage includes nonfunctional requirement gathering and scenario development to answer fundamental questions about the project: How many users do we expect? How many transactions and how much traffic do we expect? What performance bottlenecks are there from other integrated systems?

Starting here enables the performance architect to understand the overall design and architecture of the project from the very beginning, so they can spot and surface issues right away, before they get to development. This also gives the developer a clear picture of how the program needs to perform, so they can make the right design choices at the beginning, saving them from going back and re-architecting later in the process. For example, if a developer knows that a million records a minute need to be processed, rather than a thousand records a minute, they will make different design choices.

This phase doesn't need to capture everything perfectly, but developing a good fundamental understanding of the project will make a big difference later.

Performance diagnostics and profiling
Performance diagnostics and profiling are intended to identify performance weak points as early as possible in the development process and occur throughout development, including within agile sprints. Diagnostic tools allow you to dive deeply into performance at the code and function levels. For instance, they may record how long it takes for results to be returned every time someone clicks on a search button. Then, the tools can not only indicate when there is a problem but also pinpoint where the problem lies. Over time, diagnostic tools will help develop a baseline, which allows for anomaly detection. For instance, if your baseline for a search is normally 30 seconds, if it suddenly takes 60 seconds, that could indicate an issue that needs to be addressed.

When embedded into the development process, performance diagnostics are also the first opportunity to understand and diagnose performance issues for a single user, before you begin the performance testing stage for multiple users.

Performance testing
Performance testing looks at what happens when the program scales to multiple users. Sometimes referred to as load testing, it uses automated scripts to simulate users in increasing numbers and flag any issues. It also validates the capacity of the systems and scalability of the applications. At this point, if performance diagnostics have been completed thoroughly, many issues should have already been resolved, so problems will be solely because of load or capacity. This allows developers to more easily and quickly pinpoint and fix the source of the problem.

Types of performance testing

- Analysis of individual processes at a light load
- Identify performance issues earlier in development lifecycles
- Leverage lower-level environments
- Extended time to diagnose and remediate performance issues

- Load testing of individual applications (e.g., "four-wall" testing)
- Batch test under anticipated volumes
- Evaluate performance under anticipated and higher-than-anticipated loads

- Validate performance in end-to-end integrated transaction flows
- Validate key elements of batch schedule
- Scope to include baseline load from level 2, with integrations and limited profiling added

Level 1
Unit performance profiling

- Does the transaction meet performance expectations at the single user level?
- Does the call flow execute as designed?
- Can the feature/function be tuned now?

Feature/function
Examples:
- Online banking login and account overview
- Returning results from a search
- Save and submit actions
- API methods

Level 2
Process performance testing

- Do processes within an application perform well under an individual load?
- What is the performance limit of the application?
- Is the application stable over time?

Processes
Examples:
- Make 500 trades
- Create 50 policies
- Generate a report with 10,000 records

Level 3
Integrated performance testing

- Does the whole system perform well under production-like workloads?
- At what points does performance degrade, and what are the bottlenecks?
- Does infrastructure scale?

End-to-end testing
Examples:
- Simulate a peak hour for consumer banking and include all transactions
- Apply for a mortgage loan, process the paperwork and close the transaction

Monitoring and capacity

Once launched, monitoring and capacity are implemented to watch for issues in the real world. Tracking user and system trends helps to proactively manage performance and capacity over the long term and make plans around potential issues. The goal is to predict a bad user experience before it happens and shorten the time to solve the problem.

Monitoring and capacity can also offer feedback into previous processes. For example, initial testing may have been completed with 10,000 users, but during monitoring, you see that there are now 20,000 users. You can now revise testing based on what's actually happening. There may also be different ways that people are interacting than were expected. Those can now be included in testing as well.

In a cloud or virtual environment, monitoring and capacity planning can enable notifications, automated scaling and adding capacity in production in real time (e.g., adding more servers or using backup sites to load balance).

Conclusion: implement performance engineering early and often

Traditionally, performance testing is last on the long list of things to do before launch, and performance engineering as a whole doesn't make the list at all. In a typical development cycle of in sprint/out of sprint/release, performance testing often begins only a month or so before the planned go-live date. Inevitably, issues appear, including some in fundamental design and architecture that are difficult to fix late in the game. The go-live may be postponed or, in the worst-case scenario, goes forward and there is a dramatic crash, complete with negative news coverage. So, while you may be tempted to skip performance engineering due to time and budget constraints, when something goes wrong at the end, you'll actually spend more time and money fixing it than it would have cost had you implemented it early.

If the four dimensions of performance engineering are started early and done well, you'll have a better quality product, as well as a better user experience, and will go live faster and most likely without any delays.

How to get started with performance engineering

1. Start now and mature over time. You don't have to implement everything discussed here on day one. Start with low-hanging fruit, such as the timing of key transactions during manual testing, adding performance requirements to acceptance criteria or starting load testing earlier with fewer transactions and/or less volume.
2. Develop an approach based on your unique risks. The considerations of a big retailer will be very different than a local government.
3. Evaluate tools to make performance engineering easier: application performance monitoring (APM) tools, automated load testing tools, log aggregators, cloud-native toolkits and similar tools.

EY | Building a better working world

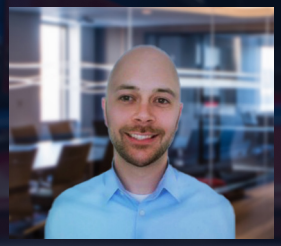
EY exists to build a better working world, helping to create long-term value for clients, people and society and build trust in the capital markets.

Enabled by data and technology, diverse EY teams in over 150 countries provide trust through assurance and help clients grow, transform and operate.

Working across assurance, consulting, law, strategy, tax and transactions, EY teams ask better questions to find new answers for the complex issues facing our world today.

US SCORE no. 13636-211US_2
2104-3758146
ED None

ey.com



Justin Hanke
Senior Manager
Ernst & Young LLP
justin.hanke@ey.com
+ 1 614 547 2580

Justin Sebastian
Senior Manager
Ernst & Young LLP
justin.sebastian@ey.com
+ 1 704 331 0382