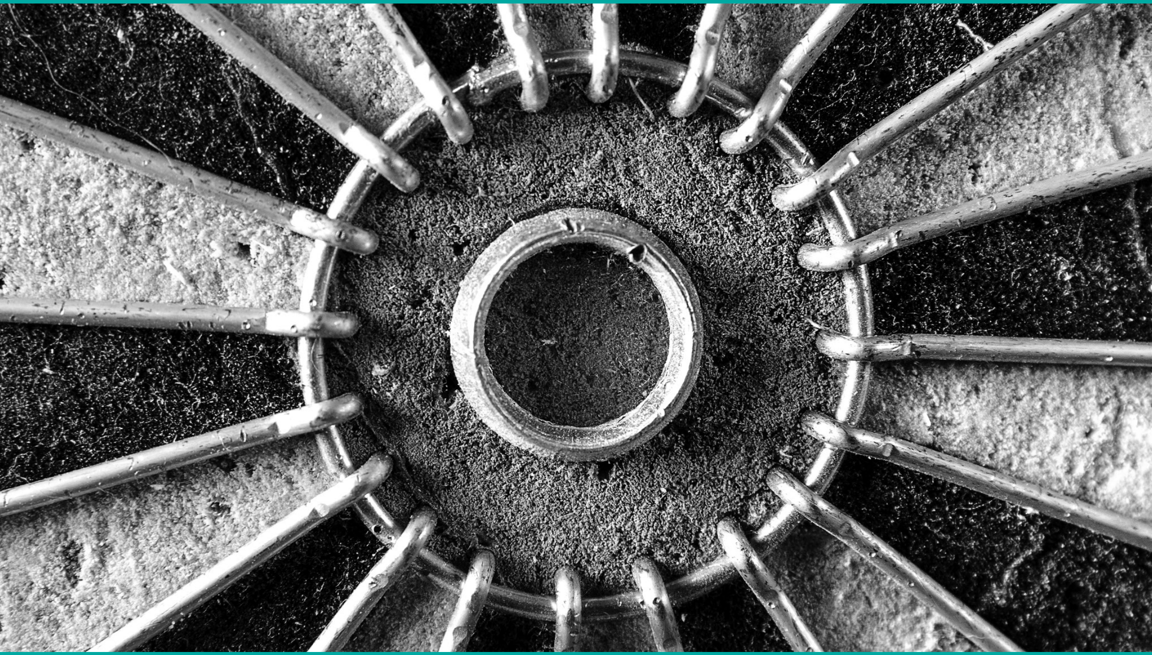
O'REILLY®

Effective Performance Engineering



Todd DeCapua & Shane Evans

Effective Performance Engineering

Todd DeCapua and Shane Evans

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Effective Performance Engineering
by Todd DeCapua and Shane Evans

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editor: Brian Anderson
Production Editor: Colleen Lobner
Copyeditor: Colleen Toporek
Proofreader: Rachel Monaghan
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

June 2016: First Edition

Revision History for the First Edition

2016-05-16:	First Release
2016-07-11:	Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491950869* for release details.

978-1-491-95086-9

[LSI]

Table of Contents

1. Getting Started. 1

What Is Effective Performance Engineering? 2

Why Is Effective Performance Engineering Necessary? 7

Focusing on Business Need 18

2. Overview of Performance Engineering. 21

Performance Engineering Throughout the Lifecycle 21

Stakeholders 50

Building in Performance 52

3. Proven Practices of Performance Engineering. 61

Requirements, Architecture, and Design 61

Proven Practices for DevTest 67

Proven Practices for Operations 74

4. Tying It All Together. 81

Metrics for Success 82

Automation 87

Market Solutions 91

Conclusion 103

CHAPTER 1

Getting Started

How do we get started with Effective Performance Engineering?

Let's start by defining it. When speaking with different individuals and organizations, we've found the definition of "Effective Performance Engineering" or "Performance Engineering" varies greatly, so we wanted to define it upfront.

Performance Engineering represents a cultural shift in the way organizations view their essential processes. It embraces practices and capabilities that build quality and performance throughout an organization. This enables organizations to increase revenue, customer attraction and retention, brand value, and competitive advantage—all while focusing on meeting and exceeding the expectations of their end users.

Let's go back to see where performance was first introduced in the modern computer era in technology history: when the Z1 was created by German Konrad Zuse in his parents' living room between 1936 and 1938. The Z1 is considered to be the first electro-mechanical binary programmable computer, and the first really functional modern computer. This marked the start of "Performance Engineering" for hardware and software related to the modern computer, some nearly 80 years ago.

Prior to the first functional modern computer, there are many examples of other performance-related topics associated with crops, livestock, medicine, mechanics, and plenty more. As with anything the challenge remains similar, but the practices and capabilities change. Many of these practices were handed down from mentor to mentee and through apprenticeship or learned through individual real-world experiences.

We will take a look at the Effective Performance Engineering of today, and how it relates to future technology and business capabilities (including computer and associated hardware and software) to serve the end user, and several other impacting factors and opportunities as we continue to accelerate.

What Is Effective Performance Engineering?

While Performance Engineering is often defined narrowly as ensuring that nonfunctional requirements are met (such as response times, resource utilization, and throughput), the trend has moved toward a much broader application of the term.

"Performance Engineering" doesn't refer only to a specific role. More generally, it refers to the set of skills and practices that are gradually being understood and adopted across organizations that focus on achieving higher levels of performance in technology, in the business, and for end users.

Performance Engineering embraces practices and capabilities that build in quality and performance throughout an organization, including functional requirements, security, usability, technology platform management, devices, third-party services, the cloud, and more.

Stakeholders of Performance Engineering run the gamut, including Business, Operations, Development, Testing/Quality Assurance, and End Users.

We'll explore several different facets of Performance Engineering in this book, providing a well-rounded overview of the practices and capabilities that make up Effective Performance Engineering.

Hardware

The traditional goal of Performance Engineering between the 1970s and the late 90s was to optimize the application hardware to suit the needs of the business or, more accurately, the IT organization that provided the services to the business. This activity was really more of a capacity planning function, and many teams charged with carrying the mantle of performance reported to operations or infrastructure teams. Some still do (and that's okay).

As hardware became more commoditized and the adoption of virtual infrastructure and "the cloud" more prevalent, this function took a backseat to development in an effort to deliver business applications and changes faster. It isn't uncommon now for teams to have multiple environments to support development, test, production, and failover. While certainly more cost-effective than ever, virtualization has given us the false sense that these environments are free.

The cloud allows service providers to charge a premium for computer power in exchange for the promise of higher uptime, higher availability, and virtually unlimited capacity. However, the cloud doesn't promise an optimal user experience. Applications need to be optimized for the cloud in order to maximize the potential return on investment.

Software

Over the last 30 years, software has transformed from monolithic to highly distributed, and even the concept of model-view-controller (MVC) has evolved to service-oriented (SOA) and micro-service architectures, all in an effort to reduce the number of points of change or failure, and improve the time-to-value when new functionality is implemented. Isolating components also allows developers to test their discrete behavior, which often leaves end-to-end integrated testing out of scope. The assumption here is that if every component behaves as it should, the entire system should perform well. In an isolated environment this may be true, but there are many factors introduced when you're building large-scale distributed systems that impact performance and the end-user experience—factors that may not be directly attributed to software, but should be considered nonetheless, such as network latency, individual component failures, and client-side behavior. It is important to build and test application components with all of these factors represented in order to optimize around them.

Culture

Every organization and group has a mission and vision. While they strive to attain these goals, performance becomes implied or implicit. But performance needs to be a part of all decisions around the steps taken to achieve a goal; it forms the basis of how an organization will embody Performance Engineering throughout their culture to achieve their mission and vision.

We need to treat performance as a design principle, similar to deciding whether to build applications using MVC or micro-services architectures, or asking why a new epic (or the relative size of a requirement, in Agile terminology) is important to the business, and how performance with the business/technology/end user will make a difference for all stakeholders. Performance needs to be an overarching requirement from the beginning, or we have already started on the wrong foot.

In order to build a culture that respects the performance requirements of the organization and our end users, there needs to be some incentive to do so. If it doesn't come from the top down, then we can take a grassroots approach, but first we need to quantify what performance means to our business, users, and team. We must understand the impact and cost of every transaction in the system, and seek to optimize that for improved business success.

Throughout this book are sidebars in which we look at five companies and examine how performance is built in to their culture. These organizations are from a variety of industries and verticals, showing the diversity of how a performance culture drives everything a business does, enabling it to deliver amazing results. Here are the five companies we will look at in more detail:

- Google
- Wegmans
- DreamWorks
- Salesforce
- Apple

The key takeaway here should be that performance is everyone's responsibility, not just the developers', the testers', or the operations team's. It needs to be part of our collective DNA. "Performance First" can be a mantra for every stakeholder.

Google: A Performance Culture Based on 10 Things

Within Google's "About Google" section is an area titled "What we believe," in which the "Ten things we know to be true" live. There are items written when the company was a few years old, which they continue to revisit to hold each other accountable.

There is a classic video from Google I/O 2014 where Paul Lewis and Lara Swanson talk about the performance culture at Google. Part of the abstract from this video states, "We can be deliberate about performance and mobile web, make smart use of performance monitoring tools, and cultivate a social atmosphere of collaboratively improving performance for our mobile users."

Tim Kadlec shared his notes on the video:

- 34% of U.S. adults use a smartphone as their primary means of Internet access.
- Mobile networks add a tremendous amount of latency.
- We are not our end users. The new devices and fast networks we use are not necessarily what our users are using.
- 40% of people abandon a site that takes longer than 2–3 seconds to load.
- Performance cops (developers or designers who enforce performance) is not sustainable. We need to build a performance culture.
- There is no "I" in performance. Performance culture is a team sport.
- The first step is to gather data. Look at your traffic stats, load stats and render stats to better understand the shape of your site and how visitors are using it.
- Conduct performance experiments on your site to see the impact of performance on user behavior.
- Test across devices to experience what your users are experiencing. Not testing on multiple devices can cost much more than the cost of building a device lab.
- Add performance into your build tools to automatically perform optimizations and build a dashboard of performance metrics over time. Etsy notifies developers whenever one of the metrics exceeds a performance goal.
- Surfacing your team's performance data throughout development will improve their work.

- Celebrating performance wins both internally and externally will make your team more eager to consider performance in their work.

Even the *New York Times* published an article, quoting Ben Waber, who has a Ph.D. from M.I.T., is the author of *People Analytics* (FT Press), and is, at 29, the median age of Google employees. His company, Sociometric Solutions in Boston, uses data to assess workplace interactions. "Google has really been out front in this field," he said. "They've looked at the data to see how people are collaborating. Physical space is the biggest lever to encourage collaboration. And the data are clear that the biggest driver of performance in complex industries like software is serendipitous interaction. For this to happen, you also need to shape a community. That means if you're stressed, there's someone to help, to take up the slack. If you're surrounded by friends, you're happier, you're more loyal, you're more productive. Google looks at this holistically. It's the antithesis of the old factory model, where people were just cogs in a machine."

The end-user experience should be at the forefront of thinking when it comes to performance. The satisfaction of your end users will ultimately drive business success (or failure), and can be quantified by a number of metrics in described in "Metrics for Success" on page 82. The point is that it shouldn't matter whether your servers can handle 1,000 hits/second or if CPU usage is below 80%. If the experience of the end user is slow or unreliable, the end result should be considered a failure.

Business

What does performance mean to your business? Aberdeen Group surveyed 160 companies with average annual revenue of over $1 billion, finding that a one-second delay in response time caused an 11% decrease in page views, a 7% decrease in "conversions," and a 16% decrease in customer satisfaction.

Google conducted experiments on two different page designs, one with 10 results per page and another with 30. The larger design page took a few hundred milliseconds longer to load, reducing search usage by 20%. Traffic at Google is correlated to click-through rates, and click-through rates are correlated with revenue, so the 20% reduction in traffic would have led to a 20% reduction in revenue.

Reducing Google Maps' 100-kilobyte page weight by almost a third increased traffic by over one-third.

The correlation between response time and revenue is not restricted to Google. A former employee of Amazon.com discovered that 100 milliseconds of delay reduced revenues by 1%. Whether you are selling goods online or providing access to healthcare registration for citizens, there is a direct correlation between the performance of your applications and the success of your business.

Why Is Effective Performance Engineering Necessary?

Over the years, we have all lived with the mantras "Do more with less," and "Faster, cheaper, and better." While some organizations have survived, many have not. Now we are faced with a different question: "How do we deliver highest quality, highest value, at the highest speed?"

In addition, organizations must focus on other key elements, like revenue, competitive advantage, customers, and brand value. Practices like Agile and DevOps have evolved and become more widely adopted. Regardless of the lifecycle, Effective Performance Engineering practices are enabling organizations to accomplish the preceding focus areas and goals for their end users. We will touch on each of these focus points next.

Revenue

Effective Performance Engineering enables organizations to increase revenue in several ways. In a recent survey, 68% of respondents expected Performance Engineering practices to increase revenues by "ensuring the system can process transactions within the requisite time frame." While all survey participants were generally in agreement regarding the tasks required, they had different expectations.

In addition to the increase in revenue, another result is reduced cost; 62% of the Performance Engineering–focused respondents felt that Performance Engineering practices should serve to avoid "unnecessary hardware acquisition costs."

By building in performance, organizations can start optimizing applications before the first piece of code is even written, or before that new capability lights up the hardware, thereby improving the end-user experience and proactively focusing on the business objectives. Cost reductions can be dramatic. As the result of reduced performance-related production incidents, organizations can often handle 30–50 percent more transactions with the same (or less) infrastructure.

Savings can quickly multiply—fewer machines means reduced capital expense as the business scales, including lower operational expenses related to power and cooling. Fewer resources are then needed to support the infrastructure. Savings also accumulate through the reduction in performance-related production incidents that need to be managed, which reduces the opportunity cost of putting your most valuable people to work on new features and functions for the end user and your business.

A catalog company, for example, might focus on total revenue from a specific product line or service, then track it after making specific changes to a promotional website to see if revenue increases. In another scenario, the value of the mobile application might be judged by the number of registered users and the frequency with which they access products or services. And, as the backend infrastructure (including web servers, middle tiers, and databases) takes on more roles inside the corporation, the metrics that track the performance of the larger organizational goals better reflect the quality of the supporting technology.

The rise in importance of Performance Engineering has been driven by practical concerns. At least 50% of respondents admitted that slowdowns and outages were discouraging customers and frustrating employees. Many characterized the problems as "repeated," and said they were often caused by large spikes in traffic that weren't anticipated when the applications were built.

The consequences are serious. The average firm that responded to the survey said that a major outage could cost between $100,000 and $500,000 in lost revenue per hour. Some of the larger companies with more than 10,000 employees said they could lose $5 million an hour from website or core system outages.

When organizations contemplated the scope and catastrophic range of these failures, they recognized that the traditional development process just wasn't ready to build a system with adequate provisions for surviving these kinds of issues. Transforming the organization and focusing individuals and teams on performance means empowering them with capabilities to anticipate problems and solve them before they occur. And when problems emerge after deployment, it means giving the team the ability to control failure and mitigate risk.

This is one of the reasons why a greater understanding of Performance Engineering as a cross-discipline, intra-business mindset is so essential. Revenue is often the first and foremost measurement of why something needs to change in any organization, but it is not always used as a measurement of Effective Performance Engineering—even though it's often seen as the key differentiator in gaining a competitive advantage related to delivering a product or service faster and better than anyone else in the market.

Salesforce.com: Entrepreneurial, Independent, and Results-Oriented

Salesforce provides quick insight to their culture on their website and states the following:

> Top talent across the globe comes to salesforce.com for the "change-the-world" mentality; the opportunity to excel in a fast-paced atmosphere; and the chance to be surrounded by peers and leaders that inspire, motivate, and innovate. Salesforce.com offers a unique career opportunity, regardless of what you do or where you do it.
>
> Our employees are entrepreneurial, independent, and results-oriented. If you like working hard in a place where hard work is rewarded, contributing to projects where contributions count, and growing in a company where growth knows no boundaries, salesforce.com is perfect place to do the best work of your career.
>
> The beneficiaries of our hard work extend into our communities through the Salesforce.com Foundation. Employees are encouraged to give back to the community and get four hours per month, or six full days per year, off with pay to spend on volunteer activities.

Looking to mainstream media, we quickly find Salesforce appears often in the "Fortune Top 100 Places to Work." When you look at the Employee Ratings, it is easy to see how the right culture can benefit an organization. Adopting Effective Performance Engineering capabilities supports many of these metrics.

Figure 1-1 shows Salesforce employee ratings, demonstrating that 93% of employees say their workplace is "great."

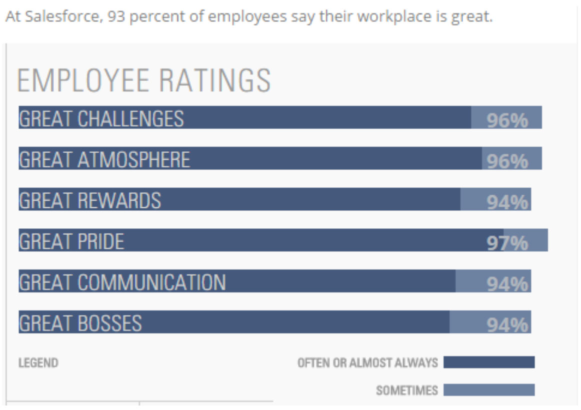


Figure 1-1. Salesforce employee ratings

On the Salesforce careers page, the company showcases a hashtag (#dreamjob) that says a lot for its culture. Going a bit further, they add, "A #dreamjob starts with passionate people who do work that matters, win as a team, and celebrate success together. It ends with knowing that together, we are the force innovating the future of business for customers. We are living our #dreamjob. Join us!"

Competitive Advantage

There's an obvious business reason to focus on the needs of end users. If they're your customers, they're consuming your products and/or services and possibly paying you for results. If you're a provider in a technology chain that defines a complex solution of some sort, you're still dependent on the satisfaction of users, however indirectly.

But it wasn't so long ago—say, 20 years—that users of computing systems were expected to live with high latencies, downtimes, and bug workarounds that were common in business systems. That's because users were employees and had to put up with the buggy, slow, or unpredictable systems their employers provided. Or they were users of a standalone application, such as WordPerfect for writing, or Lotus 1-2-3 for spreadsheets.

That's right, we're talking about the pre-Internet age, when very few users imagined doing actual business transactions online. But once e-commerce became a buzzword, and soon simply a word, users stopped being just users. They became *customers*, and it was obvious the best web-based experiences for customers would lead to repeat business.

Fast-forward to today's web-based and mobile business climate, where:

- User experience (UX) is a red-hot topic.
- Commoditization of virtually everything is a fact of life.
- Social media is the engine that can quickly sink an online retailer, transportation provider, and so on if the UX is poor—no matter the reason.

These days, performance is king, and your online or mobile customers can either be adoring subjects or a mob of thousands, with the social-media equivalent of torches and pitchforks, at your kingdom's gate. Or they'll just go on to the next provider, leaving you alone in the dark ages.

Thrill your end users (especially customers) by outperforming their expectations, and you'll get to keep doing more of whatever made them come to you in the first place, while making your company brand champions.

DreamWorks Animation: Inspiring Audiences to Dream and Laugh Together

Going to the DreamWorks website to learn about the company's culture was an experience in and of itself. We were surprised to see five areas of focus in the "Our Culture" section of the site:

1. Our Culture
2. Education
3. Campus Activities
4. Well-Being
5. Environment

Within the "Our Culture" area of focus it states, "At the heart of DreamWorks Animation is the desire to tell great stories that inspire audiences to dream and laugh together, pushing the boundaries of both creativity and technology. To do this effectively, DreamWorks is dedicated to providing the best work environment possible for the company's artists, engineers and everyone in between, so that they enjoy creating their work just as much as audiences enjoy watching it."

Going to the Indeed website revealed a bit more about the culture at DreamWorks, as nearly all reviews were 5 stars, nearly unheard of at other companies in this industry.

Another article from ChicagoCreativeSpace's Max Chopovsky concludes by stating, "we have found ways to combine these spaces and make them all onsite so that collaboration, creativity and efficiency are maximized."

Many companies recognize that it's not enough to call something a glitch, cross their virtual fingers, and hope it never happens again. They need to track their computer systems' online status; guarantee that they're responding to customers, partners, and employees; and measure whether they're delivering the information promptly so no one is clicking, tapping, pounding the return key, and wondering whether they should just go somewhere else.

As corporate leaders realize the importance of their online and mobile presence and start to measure just how much business comes in through all of their channels, they're reshaping organizations to keep information flowing quickly and accurately. This responsibility is gaining traction under the umbrella of Performance Engineering.

Proponents of this new vision believe that enterprises must build a performance-focused culture throughout their organizations—one that measures the experience of end users, both internal and external—and deliver a software and hardware experience that results in efficient performance for these users. Performance must be prioritized from the beginning of the process and be watched vigilantly after the code is deployed.

More efficient, faster systems leave employees less frustrated, something that almost certainly translates to customer satisfaction as well. The rest of the answers indicate a general awareness that the performance and throughput of computer systems are directly related to competitive advantage and the organization's ability to retain and attract customers.

Is Performance Engineering DevOps?

Putting an end to these academic distinctions, along with the day-to-day finger-pointing between Development and Operations teams (not to mention Technology versus Business teams), is part and parcel what the DevOps movement is all about. It involves an internal cultural shift toward the end user of the system(s) who wants an update, a bug fix, and some assurance the system has the full backing of the organization the customer is depending on.

There is a relationship between Effective Performance Engineering and DevOps. This relationship is one that DevOps will deliver higher value at higher quality and higher speed if the capabilities of Effective Performance Engineering are enabling the DevOps practices.

The competitive advantage—driven by faster time-to-market, with higher quality through Effective Performance Engineering—is a basis for how a business attracts and retains customers.

Customers: Acquisition and Retention

Acquiring and retaining customers should be the driving force in an organization, no matter the size. As we have mentioned, Performance Engineering plays a considerable role in enabling your organization to succeed in the marketplace, and success is defined in many ways for different organizations, and is not exclusive by industry or organization or product/service. But in general, success is defined by a few factors (and many times a combination), including: revenue, competitive advantage, brand value, and customers (acquisition and retention).

When your customers look across the market to discover where they can get a specific product or services, what do they see, and how do they evaluate it? Today, much of this process is accomplished in a number of near real-time and openly accessible formats, including app store reviews, feedback on websites, and even product- or service-specific feedback on your sites.

A myriad of feedback and comments are available to you regarding how your customers perceive your products and services, and what you need to focus on for your next release to deliver the features and functions they are seeking. The question becomes, "How are you capturing and what are you doing with this data today?" Using effective Performance Engineering practices, you can leverage a continuous feedback loop from production to market, provide the highest value to your customer, and increase your acquisition and retention.

Defining and knowing your customers is crucial. A recent infographic (cropped) posted by Rigor.com titled "Why Performance Matters to Your Bottom Line" illustrates how impatient online shoppers are, and the impact of a one-second delay (see Figure 1-2).

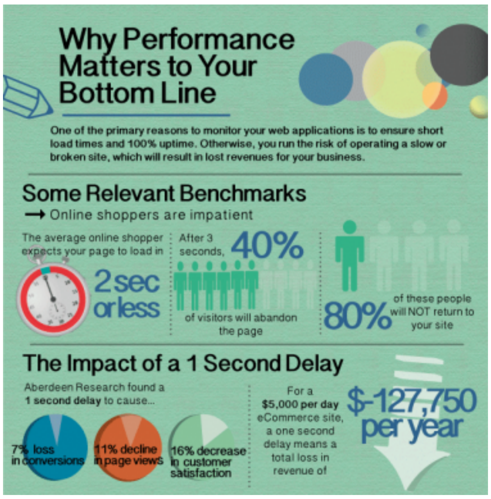


Figure 1-2. Why performance matters to your bottom line

The key point to highlight from this infographic is that online shoppers are impatient, and the cost of a one-second delay is substantial; understanding this, and doing something about it for your business, is why we are focusing on building a Performance Engineering culture and adopting practices to improve quickly.

Of course, this leads to another important consideration: metrics. Thinking about acquisition and retention of customers involves measuring many key elements, including the feedback in the marketplace for your products and/or services. For many companies, this becomes a great starting point and one they leverage to engineer into their practices.

As organizations consider and implement metrics, they're driven by the realization that computer system performance and throughput are directly related to competitive advantage and their ability to retain and attract customers. This focus on the customer is helpful both for acquiring new customers and retaining existing ones.

Wegmans Believes in Caring

On the Wegmans website, under "Our Values & Culture" is the following text:

> Because we're all part of the extended Wegmans family, we share a common set of values—we call them "Who We Are." And by living these values—handed down to Danny, Colleen and Nicole from Robert Wegman—we really have created something special: a great place to work where caring about and respecting our people is the priority.
>
> Wegmans believes in:
>
> *Caring*
> We care about the well-being and success of every person.
>
> *High Standards*
> High standards are a way of life. We pursue excellence in everything we do.
>
> *Making A Difference*
> We make a difference in every community we serve.
>
> *Respect*
> We respect and listen to our people.
>
> *Empowerment*
> We empower our people to make decisions that improve their work and benefit our customers and our company

In an article from the *Harvard Business Review* titled "Six Components of Culture," Wegmans is highlighted in component number three, stating:

> 3. Practices: Of course, values are of little importance unless they are enshrined in a company's practices. If an organization professes, "people are our greatest asset," it should also be ready to invest in people in visible ways. Wegmans, for example, heralds values like "caring" and "respect," promising prospects "a job [they'll] love." And it follows through in its company practices, ranked by *Fortune* as the fifth-best company to work for. Simi-

> larly, if an organization values "flat" hierarchy, it must encourage more junior team members to dissent in discussions without fear or negative repercussions. And whatever an organization's values, they must be reinforced in review criteria and promotion policies, and baked into the operating principles of daily life in the firm.

Forbes also published an article, "Focus on Your Company Culture, and Earnings Will Follow," in which Wegmans was also highlighted in the "100 best companies to work for in the U.S." The author writes, "Many organizations cling to 'what's always been done,' which constantly pushes against innovation; as a result, earnings and other key performance metrics begin to lag. Those dips can make managers do some interesting things in an effort to restore their companies to greatness. As executives become laser-focused on chasing earnings, they may lose sight of the bigger picture. They become focused on treating the surface-level symptoms, never diagnosing the deeper cultural dilemma."

Brand Value

In many cases, businesses continue to invest in both a capability and a culture as they work to build in the practices of Performance Engineering. These practices enable them to grow faster and become more stable. Done well, Performance Engineering avoids the large-scale catastrophes like the one that hit Best Buy in 2014 (see Fortune, CNBC, CNN Money) as well as the soft failures that come when slow services frustrate employees and turn away customers. The big failures may get all the media attention, but it's the gradual slowdowns that can be even more damaging, as they erode revenue and brand value. By the way, these are quick to be picked up in the media and amplified via social media channels, broadening and accelerating the damage to your brand.

Brand value is often something measured and tied to the stakeholders within your marketing organization(s). However, as we see the culture of performance and practices of Effective Performance Engineering, this is becoming relevant to all stakeholders (and for all the right reasons).

Keep your end users (especially customers) thrilled by outperforming their expectations and alternatives, and you'll get to keep doing more of whatever made them come to you in the first place, while making them brand champions.

All survey respondents viewed the downside of poor performance in much the same way. Together, 66% agreed that poor performance could hurt brand loyalty and brand value. As more users interact with an organization through online and mobile channels, it only makes sense that performance would reflect directly on the brand.

In conclusion, invest in Effective Performance Engineering, or risk costly failures. You'll be rewarded with smooth rollouts, lower overhead, and higher revenue. What if you don't invest? Expect damaged brands, lower revenue, and lower employee morale.

Apple Creates Wonder That Revolutionizes Entire Industries

Apple's website reads:

> The people here at Apple don't just create products—they create the kind of wonder that's revolutionized entire industries. It's the diversity of those people and their ideas that inspires the innovation that runs through everything we do, from amazing technology to industry-leading environmental efforts. Join Apple, and help us leave the world better than we found it.

Fast Company published an article titled "Tim Cook on Apple's Future: Everything Can Change Except Values." In this article, several key elements of Apple and their culture are investigated, showing how deeply important values were to Steve Jobs and are now to Tim Cook.

Tim Cook said of Steve Jobs, "It was his selection of people that helped propel the culture. You hear these stories of him walking down a hallway and going crazy over something he sees, and yeah, those things happened. But extending that story to imagine that he did everything at Apple is selling him way short. What he did more than anything was build a culture and pick a great team, that would then pick another great team that would then pick another team, and so on."

Tim Cook goes on to say, "We've turned up the volume on collaboration because it's so clear that in order for us to be incredibly successful we have to be the best collaborators in the world. The magic of Apple, from a product point of view, happens at this intersection of hardware, software, and services. It's that intersection. Without collaboration, you get a Windows product. There's a company that pumps out an operating system, another that does some hardware, and yet another that does something else. That's what's now happening in Android land. Put it all together and it doesn't score high on the user experience."

All of these elements—values, people, and collaboration—are at the core of Effective Performance Engineering.

The *Harvard Business Review* published an article titled "The Defining Elements of a Winning Culture," which shows how a company's culture can have a powerful impact on its performance.

HBR found a set of seven "performance attributes" that enable the best-performing companies. Here is their list:

1. Honest. There is high integrity in all interactions, with employees, customers, suppliers, and other stakeholders;
2. Performance-focused. Rewards, development, and other talent-management practices are in sync with the underlying drivers of performance;
3. Accountable and owner-like. Roles, responsibilities, and authority all reinforce ownership over work and results;
4. Collaborative. There's a recognition that the best ideas come from the exchange and sharing of ideas between individuals and teams;
5. Agile and adaptive. The organization is able to turn on a dime when necessary and adapt to changes in the external environment;
6. Innovative. Employees push the envelope in terms of new ways of thinking; and
7. Oriented toward winning. There is strong ambition focused on objective measures of success, either versus the competition or against some absolute standard of excellence.

The article mentions Apple and Steve Jobs specifically: "Steve Jobs builds a challenging culture at Apple —one where 'reality is suspended' and 'anything is possible'—and the company becomes the most valuable on the planet."

Focusing on Business Need

The business needs to ensure that revenue, competitive advantage, customer acquisition and retention, and brand goals are achieved. Doing so means expanding products and service offerings and/or businesses either through organic or acquisition approaches, all of which may depend on an existing or new platform(s), so end users can consume products and services without interruption when, where, and how they want.

As a result, businesses should always be seeking to adopt Effective Performance Engineering capabilities. How they choose to do so must be led by clear and visible objectives, key metrics and measurements, and communications. If they support and participate, they will see positive results from the previously defined goals and objectives—if not, negative results will be sure to follow in one or many of these areas.

CHAPTER 2
Overview of Performance Engineering

Now that we understand where to start with Performance Engineering, having defined both the term itself and also why it is important, let's dive into more specifics of how it is applied throughout the lifecycle. As we walk through each commonly defined step within a lifecycle, we will explore where and how companies leverage these capabilities to deliver the results we identified in the last chapter.

Performance Engineering Throughout the Lifecycle

As you start to incorporate Performance Engineering capabilities into your lifecycle, it is important to understand what some of these areas are, and put these into context with some typical flow nomenclature. In the following sections we define each of these key elements with specifics—what, why, and how—so you have a more complete understanding of how to add Performance Engineering throughout the lifecycle.

One of the challenges in building Effective Performance Engineering or a performance-first culture is defining who does what, when, and how. This kind of organizational alignment and agreement is as important as the daily scrum meeting of an Agile team. If everyone agrees that performance is important, but not on how to address it, then nothing is done about it.

First, we need to agree that while everyone is responsible for the performance of our business applications, someone needs to be accountable. One person, or team in a larger organization, needs to make sure everyone is playing along in order to meet our objectives. It could be the Scrum Master, Engineering Team Lead, QA Lead, or a separate role dedicated to performance.

Some organizations have even created a "Chief Performance Officer" role to bring visibility and accountability to the position, along with *information radiators* to show performance results as visual and accountable feedback throughout the process. Once that person or group is identified, it is important to include them in any standup meetings or architectural discussions, in order to raise any red flags early in design and avoid costly rework at later stages.

Culture needs to be built into an organization by design. There are several solid, cross-industry examples included in a Staples Tech Article; we'll examine these more closely and investigate how their culture is focused on Performance Engineering, and how they have built in these capabilities.

The following sections cover the what, why, and how of Effective Performance Engineering capabilities, so that as you look at this culture and the role(s) of adoption, you can start to understand more specifically how it might apply within your own organization.

Requirements

Features and functions, along with capabilities both for new applications and maintaining legacy, all fall into this section, where we will highlight some specific elements for consideration as you are adopting Effective Performance Engineering practices. Take a look at these items and understand the what, why, and how of each, so as you begin to transform, you can ensure consideration for each specific element is being considered and adopted.

Complete Stories

In defining the changes we are going to implement, complete and understood requirements or *stories* are a solid starting point. Mike Cohn, founder of Mountain Goat Software, and founding member of the Scrum Alliance and Agile Alliance, has created a *user story template*, shown in Table 2-1.

Table 2-1. Mike Cohn's user story template

As a/an	I want to...	So that...
moderator	create a new game by entering a name and an optional description	I can start inviting estimators
moderator	invite estimators by giving them a URL where they can access the game	we can start the game
estimator	join a game by entering my name on the page I received the URL for	I can participate
moderator	start a round by entering an item in a single multiline text field	we can estimate it
estimator	see the item we're estimating	I know what I'm giving an estimate for
estimator	see all items we will try to estimate this session	I have a feel for the sizes of the various items
moderator	see all items we try to estimate this session	I can answer questions about the current story such as "does this include ____?"
moderator	select an item to be estimated or re-estimated	the team sees that item and can estimate it

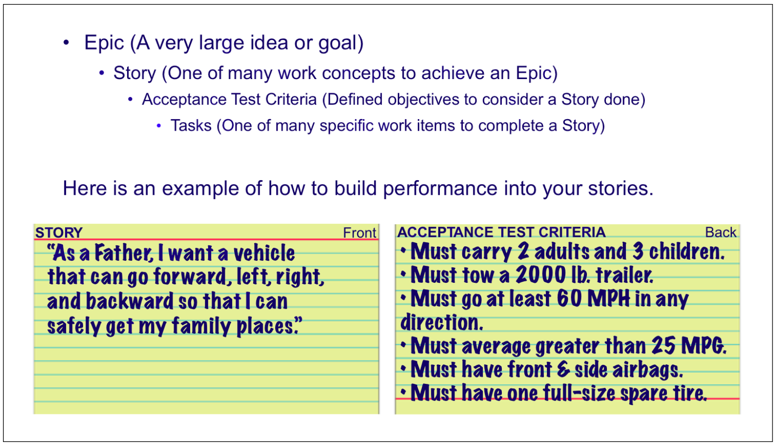
Using a thoughtful approach to stories has many benefits. With an incomplete definition of requirements and features, an individual or team is left to define what they believe the end user wants and needs. If Performance Engineering is not considered as part of a complete story, a technical component or architecture could vary considerably, resulting in underperforming or unutilized capabilities.

Organizations continue to evolve the way they create complete stories using models like Mike Cohn's user story template, and also by adopting prototyping or wireframe capabilities to accelerate the delivery of high-quality results to the end users.

Breakdown of Epic to Tasks with Acceptance Criteria

It's important to plan for the size, relationship, and priority of requirements and features, along with building in performance that shows the relationship to *epic* to *task*, in order to enable teams to collaborate and consider Performance Engineering needs and capabilities from the start.

Figure 2-1 shows the relationship and breakdown from epic to tasks, along with a story card with Story on front and Acceptance Test Criteria on the back.



- Epic (A very large idea or goal)
  - Story (One of many work concepts to achieve an Epic)
    - Acceptance Test Criteria (Defined objectives to consider a Story done)
      - Tasks (One of many specific work items to complete a Story)

Here is an example of how to build performance into your stories.

STORY Front

"As a Father, I want a vehicle that can go forward, left, right, and backward so that I can safely get my family places."

ACCEPTANCE TEST CRITERIA Back

- Must carry 2 adults and 3 children.
- Must tow a 2000 lb. trailer.
- Must go at least 60 MPH in any direction.
- Must average greater than 25 MPG.
- Must have front & side airbags.
- Must have one full-size spare tire.

Figure 2-1. Breakdown of Epic to Story, and example

In many cases, we observe a trend of more myopic or task-level views into stories. This practice limits the view and consideration across tasks, and limits the ability to build higher performing platforms, especially in the much-distributed and shared complex applications and systems we operate within today.

In Figure 2-1, you can see how the story is on the front of the card (typically only used portion) and on the back are the acceptance criteria, which is where you include Performance Engineering considerations. An example from a recent story about a login page is, "Perform with 180,000 people logging in per hour with 50% on varying mobile network conditions from 5 major US and 2 major International locations with the remaining from good WiFi and LAN connections with a maximum transaction response time of <5 seconds."

Doneness Criteria

A proven practice among organizations is defining a shared understanding and standard for what "done" means. Creating a "Feature Doneness" definition for all teams is critical, and Performance Engineering considerations need to be built in.

The standard for speed and quality must be a known and shared value across individuals, teams, business units, and organizations. Perhaps there are only 5–10 core criteria defined and agreed upon at a complete organization level, but this is the delivered standard.

This will enable a level of doneness in order to meet shared expectations and deliver within the agreed-upon time and quality.

A recent example of how a Dev/Test organization put together their doneness criteria is outlined below. Here, the organization had only eight criteria, of which the italicized one builds performance into their process:

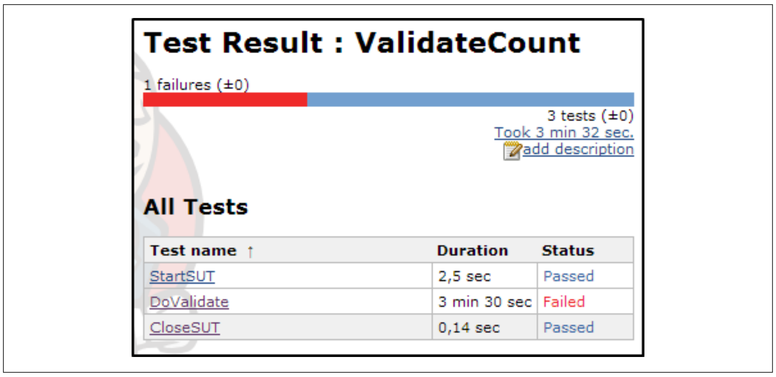
- The code is included in the proper branch in the source code control system.
- The code compiles from a clean checkout without errors using production branch [proposed: and is part of the AHP Build Life which is finally tested in QA].
- The code is appropriately covered with unit tests and all tests are passing using the production branch.
- The code has been peer reviewed by another developer.
- Database changes have been reviewed and approved by a DBA.
- *The code has passed integration, regression, stress, and load testing.*
- Application Support is aware of the backlog item and the system impacts.
- Deployment and rollback instructions are defined, tested, and documented.

Until these are all true, the feature is simply unfinished inventory.

Functional

Within the functional tests you run today, think about how to leverage these (typically single-user) tests to get performance results and share them with all stakeholders. This can take place at any stage throughout your lifecycle from unit to production, and the value of these results benefits the team throughout the lifecycle. A specific example could be the way performance is built in to your automated functional unit tests, as demonstrated in Figure 2-2, which illustrates how long a specific set of automated functional unit tests took to run and set a pass/fail threshold within Jenkins.

The reasons why this is important are numerous. One of the common areas we focus on is *speed with quality*. As you are increasing the number of builds per hour/day/week/month, these Performance Engineering practices enable early and frequent feedback on quality. The incremental value delivered with every build enables quick feedback loops and opportunities for DevTest teams to deliver higher quality faster by building in performance within their automated functional tests.


Test Result : ValidateCount

1 failures (±0)

3 tests (±0)
Took 3 min 32 sec.
add description

All Tests

Test name ↑	Duration	Status
StartSUT	2,5 sec	Passed
DoValidate	3 min 30 sec	Failed
CloseSUT	0,14 sec	Passed

Figure 2-2. Functional automated test results shown in Jenkins

The practice of gaining performance results from functional tests during the Performance Engineering adoption should be carried out throughout an organization. Just focusing on the automated functional tests immediately post-build, we can see this information enables a build-over-build continuous comparison, allowing the business to see trends over time. Adding *response time* along with *percentage of errors* variables is another way to quickly get a lot more value from functional tests you are already running with little to no effort, effectively driving performance results with immediate feedback.

Figure 2-3 shows performance trend results from JMeter that illustrate the history of performance from build to build.

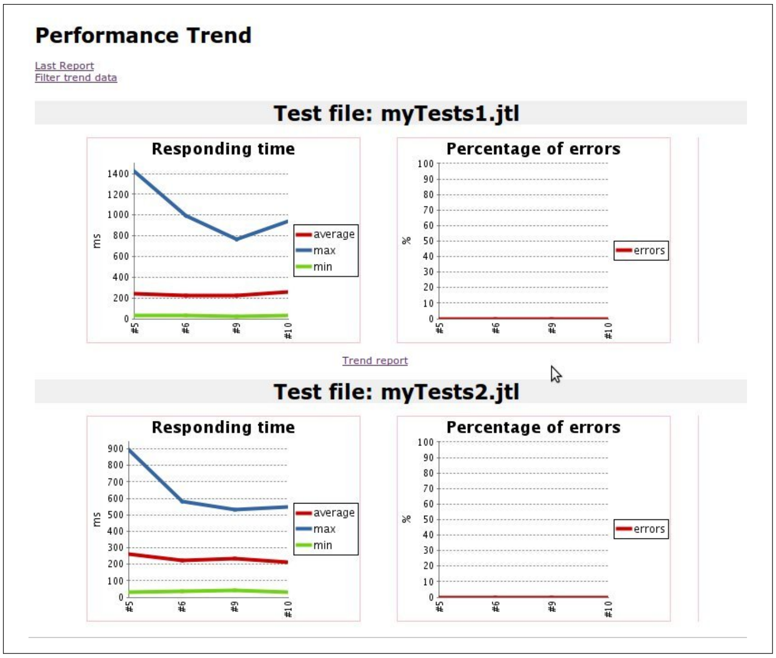


Figure 2-3. Performance trend results

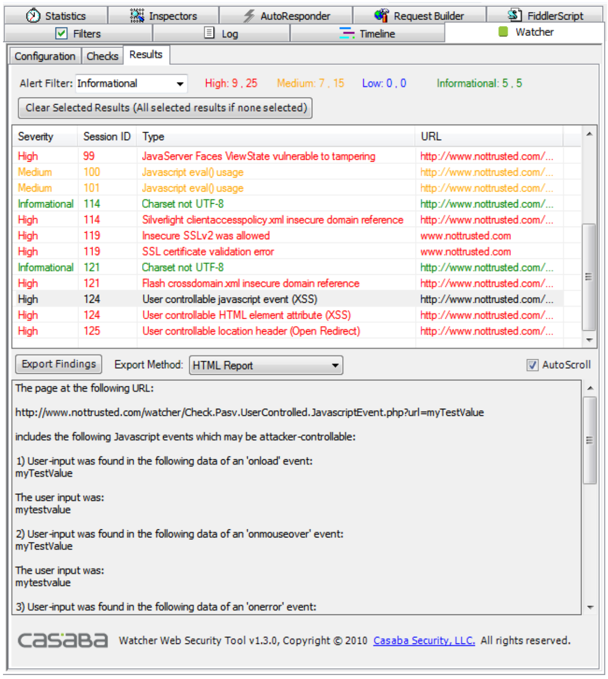
Security

Security is at the forefront of many organizations' priorities, and is addressed by a dedicated CISO (chief information security officer) and their team of professionals, often in an isolated group across the technology stack. Building performance into the security practices, and vice versa, becomes quite critical, enabling organizations not only to get actual results they may not have previously been able to achieve, but also to provide those results to a broader group of stakeholders earlier and more often in order to quickly mitigate vulnerabilities.

Fixing security vulnerabilities earlier in the lifecycle by adoptiong Performance Engineering practices reduces risk for the business and the end users. Providing the DevTest team with immediate, automated insight enables the organization to deliver more, faster, with higher quality. This advantage accrues when performance scenarios are run and security results come in from the tests, as well as when security has the opportunity to run their tests under more accurate, repeatable conditions.

There are two immediate ways that we can think about security and Performance Engineering practices. First is the ability to provide security results within the performance results throughout the life-cycle. Second is enabling performance conditions, for example, in network conditions (latency, bandwidth, packet loss, jitter), so when security tests are executed the results delivered are actual. This detail is often overlooked in security tests in two specific vulnerability areas: cross-site scripting and SQL injection. These have become popular due to mobile network conditions (needing to hold connections open longer due to higher latency conditions, and dynamic sessions not working).

You can capture security risks and vulnerabilities by providing automated and prioritized results, captured in a *flat file* and stored with the automated build results, so they can be remediated and results shown as a trend over time for the given code base/release candidate. Figure 2-4 shows how to get security results while you are running your automated tests using Watcher within Fiddler2.



Alert Filter: Informational   High: 9 , 25   Medium: 7 , 15   Low: 0 , 0   Informational: 5 , 5

Severity	Session ID	Type	URL
High	99	JavaServer Faces ViewState vulnerable to tampering	http://www.nottrusted.com/...
Medium	100	Javascript eval() usage	http://www.nottrusted.com/...
Medium	101	Javascript eval() usage	http://www.nottrusted.com/...
Informational	114	Charset not UTF-8	http://www.nottrusted.com/...
High	114	Silverlight clientaccesspolicy.xml insecure domain reference	http://www.nottrusted.com/...
High	119	Insecure SSLv2 was allowed	www.nottrusted.com
High	119	SSL certificate validation error	www.nottrusted.com
Informational	121	Charset not UTF-8	http://www.nottrusted.com/...
High	121	Flash crossdomain.xml insecure domain reference	http://www.nottrusted.com/...
High	124	User controllable javascript event (XSS)	http://www.nottrusted.com/...
High	124	User controllable HTML element attribute (XSS)	http://www.nottrusted.com/...
High	125	User controllable location header (Open Redirect)	http://www.nottrusted.com/...

Export Method: HTML Report

```
The page at the following URL:

http://www.nottrusted.com/watcher/Check.Pasv.UserControlled.JavascriptEvent.php?url=myTestValue

includes the following Javascript events which may be attacker-controllable:

1) User-input was found in the following data of an 'onload' event:
myTestValue

The user input was:
mytestvalue

2) User-input was found in the following data of an 'onmouseover' event:
myTestValue

The user input was:
mytestvalue

3) User-input was found in the following data of an 'onerror' event:
```

casaba Watcher Web Security Tool v1.3.0, Copyright © 2010 Casaba Security, LLC. All rights reserved.

Figure 2-4. Security test results using Watcher in Fiddler2

Performance

Too often we observe organizations treating performance as a checkbox. If this is the area where you need to adopt Performance Engineering practices, there are several ways to start. With Performance Engineering practices the performance team(s) no longer has to accept running the same scenario and watching the same results and reporting pass/fail, but instead can dive a lot deeper into what is going on along with how it is working in production, so they can provide more relevant and actual results to help optimize system performance.

Here are four key starting points in adopting Performance Engineering practices within performance, which we will identify now and define a bit more later:

Production incidents
: Enable you to see what is and is not working, along with creating some business value on the impact when something is not working; allow teams to see trends on where the trouble areas are, and show the value added as improvements are reflected in a reduced number of incidents.

Instrumenting
: Enables additional and broader insight into what is happening within an application and across a system, in order to visualize and simplify finding the "needle in the haystack"; helps with finding not just the first issue, but also a path of areas needing attention.

Virtualization
: Enables the re-creation of virtual users, services, networks, and data. Using virtualization technologies, you can very quickly re-create—with a high level of accuracy and low cost—the production environments anywhere in the world at any time for any period of time.

Monitoring
: Gives a visual performance dashboard of actual results in production, and when also used to observe the pre-production environments, provides teams assurance that they are building solutions that will scale and be resilient, especially as they are deployed to the end users.

Figure 2-5 shows how monitoring can be leveraged throughout all environments to build in performance and provide results to all stakeholders.

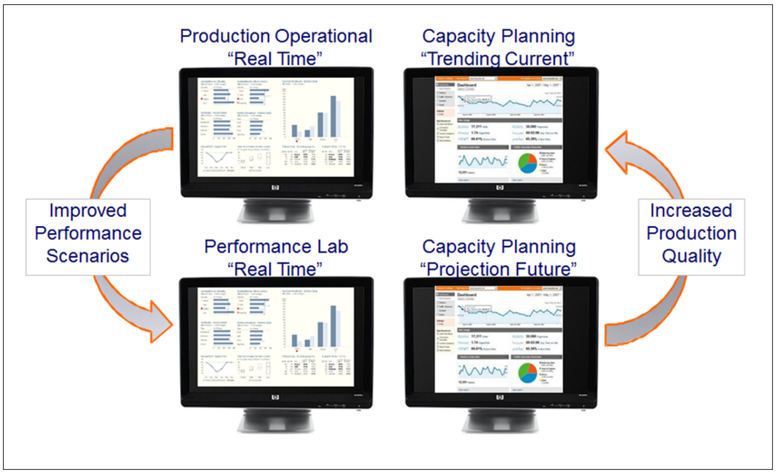


Figure 2-5. Using monitoring to increase performance

We could dive into any of these topics and quickly observe the value of adopting Performance Engineering practices in performance. Looking a little closer at monitoring, we start to see a number of different areas where these capabilities used both in production and pre-production environments can greatly enable the adoption of Performance Engineering practices within the performance team(s) and beyond:

- Synthetic monitoring provides the ability to simulate application performance many ways, ensuring you can deliver consistent and predictable performance to your end users.
- Real user monitoring enables real-time application performance of all users all the time, allowing you to automatically discover underlying infrastructure and classify user actions.
- Mobile app monitoring provides insight into the performance, stability, and resource usage of mobile apps. It gives you insight into what the user does, where they exited your app, and why.
- Deep-dive diagnostics allows you to drill into your backend performance to quickly isolate and diagnose bottlenecks to resolve issues.

- Transaction monitoring provides visualized process flows over the entire application and infrastructure environment so you can assess key business metrics.
- Transaction management enables you to track and confirm the path, step-by-step timing, and content or payload of each transaction, so you can understand the impact of critical transactions on business outcomes.

Figure 2-6 illustrates how Hewlett-Packard Enterprise Business Service Management (HPE BSM) can be used throughout your build life along with Service-Level Agreements (SLAs) to quickly show status across a variety of areas, leveraging these capabilities in both pre-production and production environments, and enabling the performance team(s) to adopt Performance Engineering practices.

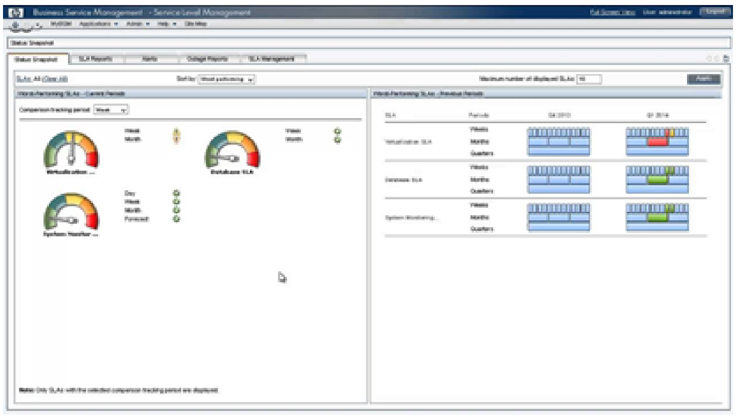


Figure 2-6. Using HPE BSM to monitor status

Usability

Usability for the purpose of this discussion is defined as the ease of use and learnability of a human-made object. Looking at the correlation of performance and facets of usability is a way to begin adopting Performance Engineering in this increasingly important and vested area.

It is important to assess your position in the market relative to your competition, and to measure usability as a trend across releases over time. Net Promoter Score (NPS) and User Sentiment Score (USS) are common elements used to measure usability. As we look at usability and Performance Engineering, it is important to note this practice should be applied throughout the lifecycle, not just at a single point late on the way to production deployment.

The NPS and USS Scoring Systems

The Net Promoter Score (NPS) system was developed by Fred Reichheld, Bain & Company, and Satmetrix. The NPS is a way of measuring user satisfaction. The methodology relies on the so-called "ultimate question," which measures overall satisfaction and, even more importantly, loyalty to the service or provider. A second question establishes the root causes for the given score.

The User Sentiment Score (USS), developed by HPE, is created by a *sentiment analysis service* that scans end-user comments on your applications, categorizes them, and provides a weighted score. You can go to *http://apppulse.io/#/* and type in "Target" to see example results.

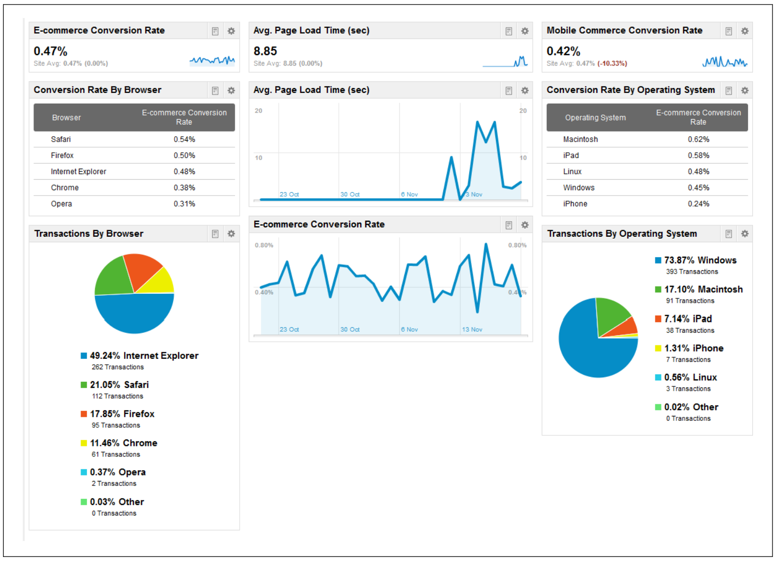
Correlation between performance and user satisfaction with usability is high. So, if you can make your interface and design easier to use, people will like it more—this seems obvious, but not everyone takes the time to implement this capability. There are several reasons interest in usability and performance has increased, including increases in competitive advantage and customer acquisition and retention. In addition, with the brand damage that can be inflicted through app store reviews and social media, getting usability right is now a mainstream challenge.

There are a variety of ways organizations can implement Performance Engineering within usability. A few key metrics to consider in this area include:

- The time it takes for an end user to complete a task. So, if an end user is seeking to make a purchase, what was the amount of time it took for that task end to end, and how long did each step take, and why?
- Workflow simplicity reflects the experience of the end user in being able to easily learn and navigate the interface to perform the task they need to complete.
- Success rate is a measurement of what percentage of end users were able to complete the task they set out to achieve.

- Subjective feedback tends to be a narrative of how users felt about the overall experience.
- Error rate is the percentage of time an end user received an error or failure while using the interface to perform a task.
- Heat map and eye tracking are another way to capture seemingly subjective feedback in an objective way, knowing where your end users are spending their time looking and clicking or tapping on the interface to perform a task.

Figure 2-7 is an example of a usability dashboard in Google Analytics, showing how to capture some of these key metrics both in preproduction and production in order to adopt Performance Engineering practices in usability.



E-commerce Conversion Rate
0.47%
Site Avg: 0.47% (0.00%)

Avg. Page Load Time (sec)
8.85
Site Avg: 8.85 (0.00%)

Mobile Commerce Conversion Rate
0.42%
Site Avg: 0.47% (-10.33%)

Conversion Rate By Browser

Browser	E-commerce Conversion Rate
Safari	0.54%
Firefox	0.50%
Internet Explorer	0.48%
Chrome	0.38%
Opera	0.31%

Conversion Rate By Operating System

Operating System	E-commerce Conversion Rate
Macintosh	0.62%
iPad	0.58%
Linux	0.48%
Windows	0.45%
iPhone	0.24%

Transactions By Browser
- 49.24% Internet Explorer, 262 Transactions
- 21.05% Safari, 112 Transactions
- 17.85% Firefox, 95 Transactions
- 11.46% Chrome, 61 Transactions
- 0.37% Opera, 2 Transactions
- 0.03% Other, 0 Transactions

Transactions By Operating System
- 73.87% Windows, 393 Transactions
- 17.10% Macintosh, 91 Transactions
- 7.14% iPad, 38 Transactions
- 1.31% iPhone, 7 Transactions
- 0.56% Linux, 3 Transactions
- 0.02% Other, 0 Transactions

Figure 2-7. Usability dashboard in Google Analytics

Design

As we look at the complexity of our systems and applications today, much of which we no longer control or house within our data centers, the need for Effective Performance Engineering increases considerably. Being able to adopt these capabilities throughout your lifecycle enables you to consider prototype options and build SLAs and performance budgets into each component you are designing, regardless of the stage of life of the component or system.

System design

A critical step in Performance Engineering is defining the many dependent pieces of a system, including data, user interfaces, internal and external dependencies, modules, components, architecture, and much more.

Without a design map and plan, teams are often left to what they know, and the results can be discarded or need significant rework. With an Effective Performance Engineering approach, performance considerations are built in, and collaboration across teams enables a more complete design from the start.

Often, an Enterprise Architecture team performs this step. However, as we can see, this is a critical element for which infrastructure architecture and application architecture must be designed, so involvement from the broader team, with knowledge of challenges and opportunities, is imperative.

Infrastructure architecture

Related to and often integrated with following system design is the need to define the required infrastructure in order to support the needs of the applications and forecasted business levels, and thus be able to deliver the required resources, often left to a few infrastructure people to create and verify.

A continuation of the overall system design is the infrastructure architecture. An Effective Performance Engineering approach provides for understanding and prototyping the various components and dependencies, along with configuration and sizing needs for the overall systems.

This is often delegated to R&D, product management, and product development to execute and deliver; however, to bring Performance Engineering practices into the picture, the team needs to be broader earlier and often include prototyping to find initial optimization.

Application architecture

The goal here is to design a composite architecture that will be scalable, available, manageable, and reliable. This involves many considerations and technological risks, along with software requirements and configurations, that must be integrated with those of the overall system design and infrastructure architecture.

Another extension of the overall system design is the application architecture. An Effective Performance Engineering approach provides for understanding and prototyping the various components and dependencies, along with configuration and sizing needs for the overall systems.

There may be some limitations around existing or defined standards, especially given the complexity and integrated nature of composite applications. Using the approach of prototyping enables quick build and running of a few quick performance scenarios to get results showing how these prototypes perform. Seeing how they perform is the first step. Knowing the underlying infrastructure and network components can also play a major role in being able best to architect a high performing solution. A collaboration between these functions must be taken to optimize architectures and overall system design for performance.

Deployment models

Deployment, or moving code and artifacts through environments to production, can range from manual to fully automated, and can be limited to only certain environments or all of them, and anything in between.

This step enables you to achieve fast feedback and increase stability across your environments, thereby eliminating many of the variables usually associated with deployment.

There are a number of open source and commercial capabilities available to help automate this process from check-in to build to deploy, and to enable the tracking of build quality through the life of the release.

Resiliency

When thinking about design and resiliency as they relate to Performance Engineering, we specifically are looking at the steps you take as you design your solution, so that when your product or service experiences a disruption it can continue to deliver value to your end user.

End users today expect to be able to access your products or services at any time, anywhere in the world. In order to enable this capability, we must consider how we design our technology to meet these expectations.

Asking "if this component is down or not available, how will it continue to deliver value to our end user?" is a great place to start the design process. Defining the top five revenue-generating workflows for your business, then designing how you can ensure they are resilient, is a good strategy.

Scalability

Scalability involves the considerations for how a system or application will be designed to handle a growing amount of work. Over the years, we have heard claims about how certain cloud environments enable exponential and linear growth; how your applications and systems are designed will enable you to see and understand the level of truth in such statements.

Increased demand for products and services by the end user of your business is a good thing, and if your technology can support and meet or exceed this demand, even better. However, too many times we've seen organizations make great assumptions that systems will scale as needed in a linear model. They provide more and more infrastructure capabilities to such a system, only to have it fail significantly because it was simply not designed to scale.

Starting with the share services and synchronous capabilities helps us to identify the two most common areas for scalability. Still, the complexities within your composite application architecture will likely also see middle-tier services and database constraints. Considering this during the design phase enables you to be ready to prototype early in the process.

Development

Practices of Effective Performance Engineering vary across teams and divisions as much as they can across industries and organizations. Following some basic principles and practices enables you to create and maintain sustainable, scalable, and high-performing applications on resilient infrastructure. With these guidelines, we will see how to do this, and deliver it quickly with quality.

Code, frameworks, and service reuse

When developing new applications and systems, it's important to leverage ways to accelerate delivery of capabilities. Some of these ways manifest themselves in code repositories, frameworks, platforms, libraries, and service components reuse. You can help DevTest teams maximize the focus on new and modified elements, and drive higher quality in a faster timeline, within the defined, shared standards.

As you move from a sole developer to development teams, then from many development teams to business units of development teams, and finally to many business units, challenges arise around standards and technology approaches and architectures. Many companies do not consider performance from an early perspective (if ever) and build it in.

Imagine having a code repository within a framework and service host that have already been built and optimized within the organizational standards, which you can leverage in building the new capabilities in front of you. Utilizing this approach saves you from spending time building and optimizing core capabilities and shared services.

Metadata repository

Determining how to quickly and easily leverage needed DevTest data continues to plague many teams and organizations. Attaining frontend data, data in motion, and backend data can be a significant challenge, and often one not easily solved. You must consider many facets in order to create and deliver to end users accurate, working applications and systems to consume that data.

Data continues to be one of the bigger challenges facing many today. Without accurate and reflective data, your confidence in application and system success is limited. In addition, to achieve speed and quality of automated lifecycle processes, it's critical to have quickly refreshing environments, especially those including data. Having a repository of maintained and ready data increases the quality and speed with which you can deliver optimized systems.

Making data a priority is a great way to start. Bringing in and assigning accountability to the business analysts and user acceptance testing teams to define and create data and data models is a parallel approach. Obfuscation of production data to be included, along with data dependencies, and storage must also be planned in order to ensure compliance with all regulatory and other required practices. In addition, providing your end users with early access and capturing the data and flows they use is another key way you can start to meet this challenge.

Automated tests

Testing throughout the lifecycle is an important step; however, determining specifically how to do it within DevTest is key, especially as continuous testing will be pushing more builds than before, and speed to verify and validate accurately is critical.

You must proceed with automated tests early, often, and throughout the lifecycle, especially given the need for repetitive and consistent results and measurements, in order to be able to see trends and optimization opportunities. If tests are not done in an automated way, you cannot meet goals of speed and quality, along with quality gates and build quality, automating more of the lifecycle.

Many open source and commercial capabilities are available to enable automated tests. In addition, using automation test frameworks will allow you to build tests that can be merged and versioned with code through the build and test process.

Lifecycle virtualization

Creating production-like environments within your pre-production environments is often expensive and labor-intensive. However, having a controlled and stable environment is important, and the ability to make it quickly available and refreshed in a timely manner is even more important. Virtualizing the users, services, network, and data gives you these capabilities, so you can deliver a better quality product faster.

Reducing costs, increasing speed, and improving quality are three reasons why utilizing lifecycle virtualization is key. Having production-like environments enables you to more accurately predict how things will perform once deployed. Given the complexity of our systems and applications today, this continues to be one of the bigger challenges, and lifecycle virtualization is a way to close the gap significantly.

Many open source and commercial products are available to provide you with these capabilities. *Virtual users*, for instance, are commonly found in functional and performance testing tools and enable you to create synthetic end users, performing transactions across a variety of workflows as they would in production, across all of your environments.

Then you can move to *service virtualization*, which enables you to capture and re-create services from production across all of your environments. *Virtual networks* enable you to capture network conditions in production (and all other locations) and re-create these in any environment so you can optimize your app or application to perform well, even over poorly performing network conditions. Last is *data virtualization*, which leverages some metadata repository capabilities to make it quick and easy to refresh data across your pre-production environments.

Quality gates

Quality gates are specific quality thresholds for each stage or environment. These thresholds are then used to automate the build/deploy/test steps and proceed to the next stage or environment. This allows for little to no manual intervention, and delivering high-quality assets in an automated and timely way. These thresholds enable you to set criteria on the build quality (Figure 2-8 shows a variety of "% Tests Pass" criteria) before proceeding on the path to production.

As you increase builds and automated tests and deployments, you will see things start to speed up. However, quality is a metric you also need to define, and measure at specific milestones throughout the lifecycle. Tracking quality enables you to observe a build as it moves through the lifecycle, and only intervene when it meets a certain quality threshold and you need someone to execute manual tests against the product.

There are several approaches for adding quality gates, many of which use open source or commercial products to deliver a framework that enables you to build in the performance, quality, and speed considerations. Some work will be needed to build and support this application lifestyle automation solution for your specific environment. Figure 2-8 shows how you can implement automated quality gates with virtualized and physical environments throughout the CI/CD process and where performance results are built in.

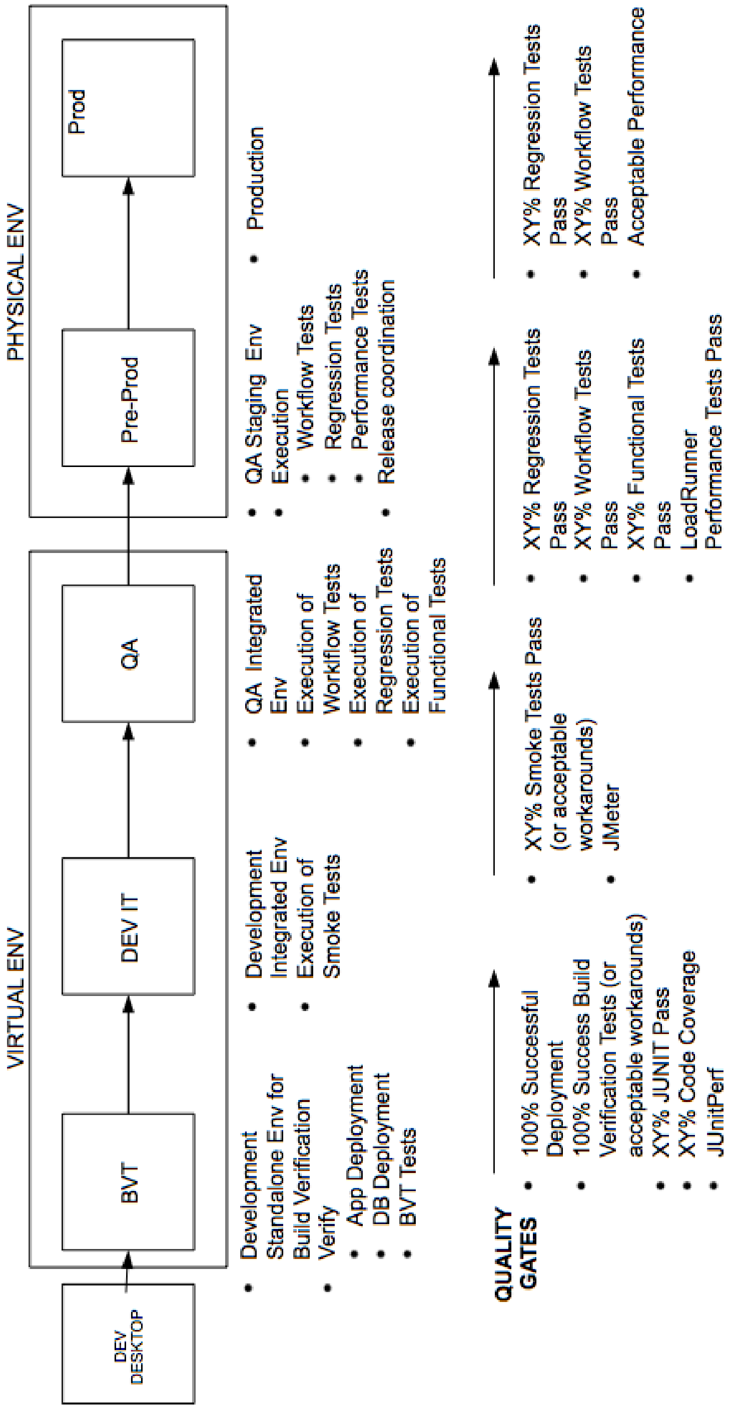

VIRTUAL ENV
PHYSICAL ENV
DEV DESKTOP
BVT
DEV IT
QA
Pre-Prod
Prod
• Development Standalone Env for Build Verification
• Verify
• App Deployment
• DB Deployment
• BVT Tests
• Development Integrated Env
• Execution of Smoke Tests
• QA Integrated Env
• Execution of Workflow Tests
• Execution of Regression Tests
• Execution of Functional Tests
• QA Staging Env
• Execution
• Workflow Tests
• Regression Tests
• Performance Tests
• Release coordination
• Production
QUALITY GATES
• 100% Successful Deployment
• 100% Success Build Verification Tests (or acceptable workarounds)
• XY% JUNIT Pass
• XY% Code Coverage
• JUnitPerf
• XY% Smoke Tests Pass (or acceptable workarounds)
• JMeter
• XY% Regression Tests Pass
• XY% Workflow Tests Pass
• XY% Functional Tests Pass
• LoadRunner Performance Tests Pass
• XY% Regression Tests Pass
• XY% Workflow Tests Pass
• Acceptable Performance

Figure 2-8. Quality gates

Test

Minimizing the risk to the business and your end users in production is paramount. Adopting Effective Performance Engineering practices ensures that you are most efficiently doing this early and throughout your lifecycle, and that you'll have many opportunities to verify, validate, provide feedback, optimize, and then continuously execute. The following sections cover specific, proven practices you can adopt to enable Performance Engineering capabilities.

Build results

After the commit/build/deploy/test process, the results are what matter, and how we capture these results and make them available to stakeholders is crucial, along with the summary performance metrics continue and live with the build life.

Building in this automated feedback throughout the lifecycle, starting immediately after a successful build, enables teams to respond early and often, which in turn accelerates delivery speed and increases quality.

There are a number of different ways to implement this capability. As a starting point, a basic automated delivery framework needs to be in place. Then, you can use automated unit, functional, or performance scripts to compare execution times at a minimum. Now you will be able to track detailed results from build to build, identifying any outliers, and potentially flag them from your trunk and/or main branch until they can be remediated for performance optimization.

Regression

As the continuous build/deploy/test cycle evolves into more mature builds and potential release candidates, capabilities and features grow, and you must run quick but complete automated scenarios to ensure no core functionality has been broken or degregated as a result of the new capabilities.

Speed and quality increase as more performance engineering capabilities are built into your automated processes. With the adoption of these capabilities comes the need to progress through a variety of environments on the release path. Building automated performance regression suites enables higher confidence and meets automated quality gates to enable the potential release candidates to move through the release cycle efficiently.

When you're starting down the path of implementing these capabilities, you must have an understanding of the key systems, applications, and transactions—those critical from both a regulatory and end-user perspective—so that you can implement these first within your automated performance regression suite. In the "lower level" environments (also known as BVT or build validation test), the results should be an automated performance regression suite that takes less than 15 minutes to execute, and provides results to ensure your most critical functions work as designed.

Automated service oriented

As the complexity of the composite application and system architecture grows, so will the dependencies on internal and external services. Automating the verification and validation of these services throughout the lifecycle, and especially testing, is key. You must also consider how to get initial results and flags for performance-related metrics, along with how to re-create these in a valuable and cost-efficient manner.

The explosive increase in composite application and system architectures has resulted in organizations with an exponential number of services. These services change only occasionally, compared to the frequently changing user interface, enabling a more stable and core technology to test with less maintenance.

You should start with a basic test harness that identifies the service and protocol mapping, then run through a barrage of positive and negative scenarios, delivering results for those that pass and fail. The objective of the team should be continuing to develop until all results pass.

Capability mapping and standardization

Organizations looking to scale Performance Engineering practices need to standardize core capabilities in order to enable a growing number of teams and individuals to leverage a shared model (mapping). Doing this integrates the new capabilities across IDEs, CI/CD systems, configuration, environments, and release management, just to mention a few key elements.

Having disparate and nonintegrated tools to do specific and individualized tasks is a mess at a small scale, and once this goes beyond a few teams, it becomes a challenge to manage from many different angles. Finding the capabilities that matter to your organization and standardizing them enables you to scale more efficiently. These standards also simplify the education of new team members and increase the stability of your integrated tools, so you can focus on delivering value to your end users.

Figure 2-9 shows how tool integration can be mapped to support processes to implement Effective Performance Engineering practices. As you continue down this path, it may be necessary to create an abstraction layer across the top, so all capabilities can be visualized and used as an information radiator for several stakeholders across your portfolio.

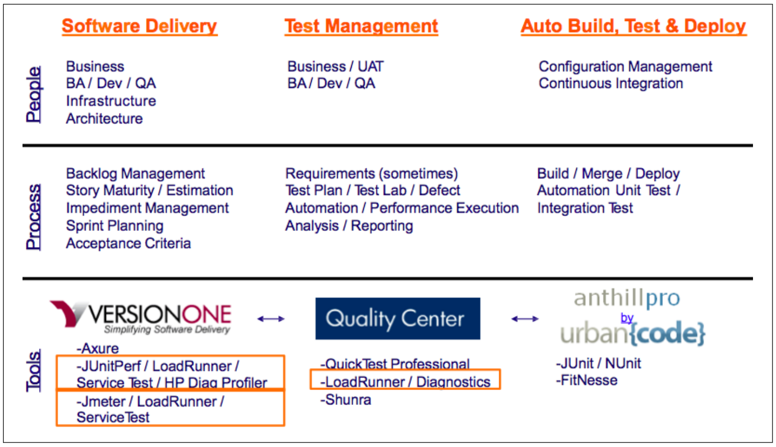


Figure 2-9. Mapping simple tool integration to support processes

Deployment

Releasing code and builds throughout a lifecycle is an art, and it's a key component to enabling Effective Performance Engineering. In this sectio we will highlight many of the critical ways this can be done, the required controls, and how to get continuous automated feedback throughout. The end objective is for deploying to production not to be an event, but simply another deployment in a series that have already been executed and delivered throughout the lifecycle in a rigorous yet fully automated process.

Automation throughout the lifecycle

Moving through the lifecycle with everything we've described previously is a huge success, but ensuring all that effort isn't for nothing is also important; we have seen several examples in which deployment to production is still is a manual process, which has its own drawbacks and risks.

Reducing the manual effort required increases accuracy and repeatability, and ensures deployment has been tested before the product deploys to production. You can automate deployments of builds and release candidates with commercial or open source products, but implementations vary a bit depending on your environment. The biggest challenge, however, is that of organizational and individual behaviors limiting the continued utilization of automated deployments to production.

Showing these organizations and individuals how deployment automation has been achieved and tested through pre-production environments, and how IP addresses and credentials can be secured, will go a long way toward utilizing automated deployments to production.

Live/live or blue/green

Live/live or blue/green is a deployment approach in which you have two production environments, enabling you to potentially take one offline, deploy to that environment, and introduce a small population of users to ensure it performs as expected, then either shutting that instance off or leave it on and deploy to the second environment so you are running two production environments again.

There are many reasons why you would want to leverage a *live/live or blue/green* approach. It provides resiliency and scalability, and increases performance. It also allows you to easily perform *canary tests* in a contained environment. All of these are key Effective Performance Engineering practices.

How this approach is implemented varies, depending on the organization. Figure 2-10 shows one way we have done it and opens a discussion of which specific Effective Performance Engineering practices are performed where. You will also see a *stand-in mode* added into this model, available in case the production environments (*n* and *n*+1) have issues (a *stand-in* providee end users with access to the products and services without interruption).

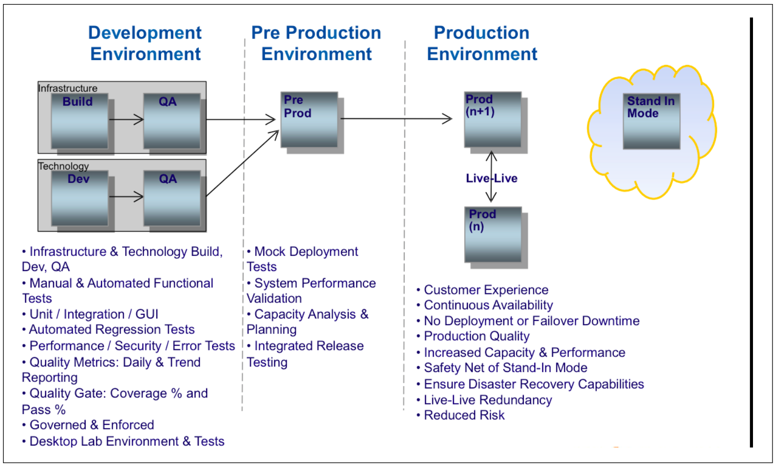

Development Environment
Pre Production Environment
Production Environment
Infrastructure
Build
QA
Technology
Dev
QA
Pre Prod
Prod (n+1)
Live-Live
Prod (n)
Stand In Mode
• Infrastructure & Technology Build, Dev, QA
• Manual & Automated Functional Tests
• Unit / Integration / GUI
• Automated Regression Tests
• Performance / Security / Error Tests
• Quality Metrics: Daily & Trend Reporting
• Quality Gate: Coverage % and Pass %
• Governed & Enforced
• Desktop Lab Environment & Tests
• Mock Deployment Tests
• System Performance Validation
• Capacity Analysis & Planning
• Integrated Release Testing
• Customer Experience
• Continuous Availability
• No Deployment or Failover Downtime
• Production Quality
• Increased Capacity & Performance
• Safety Net of Stand-In Mode
• Ensure Disaster Recovery Capabilities
• Live-Live Redundancy
• Reduced Risk

Figure 2-10. Live/live or blue/green deployment

Canary

In the canary deployment approach, you roll out a new capability to a very focused group of your population, then observe that group's performance and feedback, and finally decide whether to continue, stop, or pull back that deployment or release to a more general population.

Getting fast feedback and market-testing new capabilities is important to many organizations so they can test their theories and best guesses as to what the end user needs next. The canary test may also be a first time on a new infrastructure or application architecture, so they can benefit teams greatly to see how the architecture performs at a smaller scale in production. They can then apply their new knowledge to the next release, and deploy an incremental quality increase for the next round of canary testing, feedback, and measurements.

This kind of test is often limited to a single environment (perhaps a single cluster) and often utilizes a live/live or blue/green production environment, in order to further isolate the underlying technology and limit access to specific end users.

Fail over, fail back, and fail forward

The fail over, fail back, and fail forward approaches involve answering the question, "When we have problems, how do we plan to manage them?" The beauty of this process is it allows you to test your desired approach in your later-stage environments (pre-production or disaster recovery, perhaps). The *fail over/fail back* approach means you have a secondary production environment to fail over and back. The *fail forward* approach means your intent is to simply push a next release over the prior to resolve the issue.

In a production disaster recovery scenario, these processes are often defined in the *run book* and tested on how to fail over, but not always how to fail back. When you are failing back there are many challenges, along with many dependencies on databases and external services, that you need to consider. Not doing so, and as we have observed in several cases, can result in data loss and other more serious incidents, making for upset customers and regulatory issues.

To build these approaches into your tests, you must take the time to plan for them, and they will aid you greatly in ensuring you have a cross-functional team with shared responsibilities and goals. Finding ways to build in fail over, fail back, and fail forward into your overall deployment approach is a great way to start. Perhaps asking this question today will help you understand how you do or do not plan for problems, and imagining what might happen if the worst occurs, will get you to adopt this Effective Performance Engineering practice.

Monitoring

In many organizations, monitoring is a practice adopted only within production, and is often referred to as *application performance monitoring* (APM). However, reactive performance monitoring happens too late to have an impact on revenue, brand, competitive advantage, and customers. All those factors have already been determined once they're identified in production. In Effective Performance Engineering, we leverage several monitoring capabilities, explored in more detail in the following sections.

Continuous monitoring and feedback

Monitoring environments and components within your architecture and perhaps outside, both within production and pre-production systems, helps you learn about and improve capacity, resiliency, performance, and scale. In addition, continuous monitoring and feedback provides you with ways to observe and capture conditions in which production incidents occur, so you can re-create them in pre-production and validate a fix prior to pushing it back into production.

Without feedback throughout the lifecycle, you only have a best guess and a hope that a solution will work after it is deployed. We often see this in practice, and yet with Effective Performance Engineering practices, it does not need to be this way.

How this process is done varies by the organization and its maturity, but it starts with monitoring both in production and pre-production environments, then moving to measure the key performance indicators (KPIs) both from a technical and business perspective. Once these have been achieved, you can start by showing these results for the release candidate and then move to build-level results.

End-user feedback and analysis

End-user feedback can be measured in both objective and subjective results: objective from transactions completed, conversion rates, and response times, and subjective through reviews and interview and survey feedback.

Your end user is the person that matters the most; getting their feedback and analyzing the observations is a key piece of monitoring.

There is a vast array of ways to gather both objective and subjective feedback from your end users. These range from formal studies in which you invite participants to join you in a lab-like setting to view some new capabilities and features, asking them for feedback, to simply pushing a few new capabilities in production to your canary group, and observing the KPI metrics based on how they use the system. In some cases you might capture and measure heart rate, eye movement, screen tapping, and other behavioral and cognizant responses. This data will all be analyzed so it can be fed back to the teams, and who will use it to create the next iteration of the capability, before eventually deploying it to production.

Predictive: pre-production and production

You can use the data you've gathered and analyzed from production and pre-production environments to predict under what conditions degradation or system failure will occur, so you can operate proactively. This predictive data enables you to mitigate these issues throughout the lifecycle, so you can better deliver new feature function to your end users uninterrupted.

Getting your product wrong is a huge expense in many ways. So, having insight that provides predictive recommendations on what will go wrong and when is a huge advantage to your organization. This is used throughout pre-production and production environments.

When leveraged with correlation and big data analytics, predictive data provides us with insight into what could go wrong based on what has gone wrong historically. Results such as *Fundex* and *User Sentiment* indicate how your end users will and are responding to new capabilities.

Support

Effective Performance Engineering does not end at production; it is a continuous, iterative practice of integrating feedback, improving the entire lifecycle with every new piece of information, and automating this cycle. Advancements in big data and predictive analytics, combined with these practices, enable a more stable, high-performance experience for end users.

Threshold analysis and automated mitigation implementation

Threshold analysis starts with defining specific thresholds within your systems and applications for your application performance monitoring capabilities to alert and alarm, then building in rules-based *automated mitigation implementation*, which allows the product or service your end user is accessing to continue with minimal interruption. Utilizing this approach increases the resiliency of your architecture.

Many companies today have set up threshold analysis in the form of a network operations center (NOC), staffed by a few individuals around the clock, with dozens of monitors projecting thousands of alerts per day and sometimes hourly. Having this automated with built-in and tested mitigation implementation simplifies and ensures the business is protected, enabling the technology to grow and automatically scale with the company's needs.

While this area is still maturing from a solutions perspective, it can be done and delivered successfully when implemented by a team that considers it a collaborative goal.

Incident management, root-cause analysis, and reporting

Incident management is the practice of actively managing production (and in some cases pre-production) incidents. Its chief purpose is capturing the "current state" and enabling the re-creation of the incident in pre-production, while finding and fixing the root cause. This root-cause analysis, along with tracking and reporting trends, will support the technical debt and investment needed to ensure your architecture is performing well.

History does repeat itself. This is especially true for a technology-enabled business that follows a defined set of instructions the same way every time. Thus, we must build in learning cycles in order to understand why something happened and prevent it from recurring.

Many solutions exist off the shelf today. However, first you must assign accountability and responsibility for this activity. We have often seen an incident management team that owns the *process* of the production incident, but who owns the learning and feedback to prevent that incident from happening again is not always clear. In Effective Performance Engineering, each task must first have a defined process and owners, and only then be enabled with technology.

Re-create production incidents pre-production

All too often we see pre-production teams working on and attempting to fix production incidents. When production incident fixes are applied in production, what gets released are non-integrated updates that are not in any way validated or tested. Not only is this a risky approach for production, but now you have changes that are not being implemented into the existing pre-production assets for future deployments. This means these incidents will happen again. It is also helpful to ask, "Did these hotfixes really fix the problem, or just push them off to another day?"

Leveraging Effective Performance Engineering practices enables you to quickly capture a snapshot of what happened in production, then spin up the environments that pre-production deployed and re-create the incident, so you can follow your normal process to make quality fixes to your production release.

Stakeholders

With Effective Performance Engineering, the stakeholders are from all walks of life. Whereas with traditional Performance Testing, the sole responsibility for performance fell on a select group of individuals, in Performance Engineering, it is the entire team's responsibility to work in a broad and collaborative manner encapsulated by the organizational culture.

Once you drive incremental value and success within a group or team, this culture accelerates. Both as you start adopting these capabilities and as you continue to integrate them, it is critical that you understand your stakeholders, keep their best interests in mind, and communicate with them frequently. We will dissect what each stakeholder group looks like in the following subsections.

Development

Performance Engineering is often not a top priority for a typical development team, especially given all the unique and expanding challenges related to the complexity of composite architectures, distributed organizations, and end users. The reason for development's growing importance is to ensure they deliver the highest quality and performing product, and provide continuous feedback and optimization recommendations, so other teams can deliver quickly and in fully automated ways.

Testing and Quality Assurance

The responsibility of a testing and quality assurance team independent or integrated into teams is to ensure the delivery of a quality product. How this happens and where the focus is depends on who you are speaking with, and how they are aligned. For example, some organizations align by specialty (security, performance, functional, usability, and so on) and others have an integrated team. With effective Performance Engineering practices, a group of individuals is accountable for the delivery of a high-performing solution throughout the development process, and delivers that with clear measurements and communications throughout the lifecycle.

Operations

Operations teams are focused on ensuring the product is running and service is available. “Less is more” is this stakeholder’s perspective. In other words, the less impact a change has on the production environment, the better. This is why defining and validating what is moving into production is so important; the operations stakeholders will want to see how and why a capability is ready for the production environment. The practices of Effective Performance Engineering involve these individuals throughout, allowing them to contribute and be a part of the team steering a high-quality product through the environments and into production.

Business

The focus of this stakeholder is on ensuring that revenue, competitive advantage, customer, and brand goals are achieved. This includes expanding offerings and/or businesses either through organic or acquisition approaches, all of which may depend on existing or new platforms, so end users can consume products and services without interruption when, where, and how they want to.

End Users

End users ensure feedback is delivered, and value from product and/or service is realized. Users simply do not want security, performance, functional, or usability issues while interacting with your products or services. Making them a champion for your brand should be your goal; you do so by continuously meeting or exceeding their expectations, and asking them to share their positive experiences with others frequently.

Of course, you will encounter other stakeholders, but start by identifying which of these five existing roles fit into your organization. Begin thinking about how you can apply some of the Effective Performance Engineering practices in your organization today, aligning them with the interests of these stakeholders. Include your stakeholders in these conversations, ask them for feedback on what they get and what they need, and show them your desire to make them your biggest supporter and champion.

Building in Performance

Performance cannot be an afterthought; it needs to be at the forefront of your team's thinking from the very beginning and throughout. This often comes into conflict with the "release faster" mentality that drives many businesses today, but it doesn't have to. The speed/accuracy tradeoff that is often cited in Lean startup principles doesn't necessarily apply to the performance of the applications we build, but rather to the speed with which we deliver and the accuracy of our delivery compared to what the market needs. When it comes to Lean startups, it isn't necessary to hit the mark 100% right away. Fast feedback allows us to adjust our course and reiterate very quickly. But what happens if the products we deliver to the market perform poorly? There won't be a second chance to get things right if the customers' first impression is of a slow or unresponsive app.

It is our responsibility to bring these capabilities and practices to our business owners and CxOs so they can understand why they're important and how others are delivering compelling results to their end users, and will enable your business to do the same or more.

The List: 102 Questions to Ask

Performance Engineering is a complex discipline encompassing applications, infrastructure, security, and more. To truly optimize your performance, your organization needs to address a broad range of issues. To make informed decisions, individuals and organizations need to start asking some or all of the following 102 questions.

This list is intended to be inclusive but not exclusive, and apply across all DevTestOps approaches. No matter what your role, persona, and interest, this list should help you understand how your solution is engineered for performance, stability, and scalability.

Our goal is to help you, your team, and your stakeholders gain a common nomenclature for Performance Engineering so you can define your path and direction together. We intend the following as a checklist of actionable items, so you can ask informed questions as you start collaborating and adopting Effective Performance Engineering practices throughout your lifecycle.

Server sizing

- How many application servers are needed to support the customer base?
- What is the optimal ratio of users to web servers?
- What is the optimal web server–to–application server ratio?
- What is the maximum number of users per server?
- What is the maximum number of transactions per server?

Server tuning and optimization

- Which specific hardware configurations provide the best performance?
- How can vendor default configurations be tuned to suit this specific infrastructure and application?
- What system resources need tuning to give optimal performance?

Capacity planning

- What is the current production server capability?
- Is there room for growth?
- What hardware or software can be added to achieve the next level of performance or capacity?
- Is there excess capacity? Can a server be removed without compromising performance?

Third-party validation

- What is the current ISP and network capacity?
- Can the ISP deliver on the service-level agreement that was signed?

Security exposure

- Can system vulnerabilities be identified and minimized?

- What is the failover for firewalls?
- Are there new vulnerabilities when excess user load is added to the application?
- How susceptible is the system to DoS attacks?
- If a DoS attack occurs, how will the system respond?
- If the system goes down due to a hacker attack, how effective are the recovery procedures?

Infrastructure

The following are some questions you should ask regarding the infrastructure.

Browser/user profile issues

This subsystem is known as the *user community profile* and consists of business process definitions.

- What do the users do? (These are business-process definitions.)
- How fast do the users do it? What are the transaction rates of each business process?
- When do they do it? What time of day are most users using it?
- What major geographic locations are they doing it from?
- Is the application browser- or interface-dependent?
- Is modem, WAN, or LAN emulation necessary?
- Are there asynchronous communications between the browser/client and the backend servers?
- Are there any non-HTTP(s) communications between the browser/client and backend servers?

Internet issues

- What are the peering issues associated with the client's hosting/bandwidth provider?
- What is the hosting strategy?

Site web pipe issues

- How much bandwidth does the site have?
- Who is the client's bandwidth provider? (Peering issues)
- Are there multiple web pipes?

Border router issues

- What kind of load-balancing are the multiple pipes configured for?
- Does it use the same inbound pipe as outbound pipe?
- Is there equal distribution for outbound regardless of inbound pipe?
- Is there the same outbound pipe regardless of the inbound pipe?
- Are there multiple border routers?
- What is the failover configuration for multiple border routers?

Load-balance issues

- What type of load-balancing scheme is used? (Round robin, sticky IP, least connections, subnet based?)
- What is the timeout of LB table?
- Does it do any connection pooling?
- Is it doing any content filtering?
- Is it checking for HTTP response status?
- Are there application dependencies associated with the LB timeout settings?
- What failover strategies are employed?
- What is the connection persistence timeout?
- Are there application dependencies associated with the LB timeout settings?
- What are the timeouts for critical functions?

Peripheral systems issues

- Is the LAN/WAN system dedicated or shared with other applications?
- Are there any shared production resources?
- Are there any web pipes, ERP systems, mail servers, filesystems, DNS servers, and so on?
- Does it share databases with other applications?
- Does it share hardware with other applications?

External systems issues

- Are there any outside vendors that provide content distribution systems (CDS) for the architecture?

Distributed hosting issues

- Are these multiple mirrored sites?
- Is any site configured for failover operation?
- How is the traffic load-balanced across the sites?
- Are there architecture components on shared WAN connections?
- What is the failover and recovery behavior?

Firewall issues

- What is the throughput capacity?
- What is the connection capacity and rate?
- What is the DMZ operation?
- What are the throughput policies from a single IP?
- What are the connection policies from a single IP?

IDS: Intrusion detection systems

- Is there statistical content sampling?

- Is there an inverse relationship between throughput and security?
- How is content filtering achieved?

Application

Here are some of the questions you should be prepared to ask regarding the application.

Web server issues

- How many connections can the server handle?
- How many open file descriptors or handles is the server configured to handle?
- How many processes or threads is the server configured to handle?
- Does it release and renew threads and connections correctly?
- How large is the server's listen queue?
- What is the server's "page push" capacity?
- What type of caching is done?
- Is there any page construction done here?
- Is there dynamic browsing?
- What type of server-side scripting is done? (ASP, JSP, Perl, JavaScript, PHP, and so on)
- Are there any SSL acceleration devices in front of the web server?
- Are there any content caching devices in front of the web server?
- Can server extensions and their functions be validated? (ASP, JSP, PHP, Perl, CGI, servlets, ISAPI filter/app, and so on)
- Monitoring (Pools: threads, processes, connections, and so on; queues: ASP, sessions, and so on; general: CPU, memory, I/O, context switch rate, paging, and so on)

Application server issues

- Is there any page construction done here?
- How is session management done and what is the capacity?
- Are there any clustered configurations?
- Is there any load-balancing done?
- If there is software load-balancing, which one is the load-balancer?
- What is the page construction capacity?
- Do components have a specific interface to peripheral and external systems?

Database server issues

- Have both small and large data sets been tested?
- What is the connection pooling configuration?
- What are its upper limits?

Security

Here are some of the questions to ask when addressing security issues.

Firewalls and multiple DMZs

- Does the firewall do content filtering?
- Is it sensitive to inbound and/or outbound traffic?
- What is its upper connection limit?
- Are there policies associated with maximum connection or throughput per IP address?
- Are there multiple firewalls in the architecture (multiple DMZs)?
- If it has multiple DMZs, is it sensitive to data content?

IDS: Intrusion detection system

- Is there any content filtering?
- Is the system sensitive to inbound and/or outbound traffic?
- What are the alert thresholds?
- What are the acceptable security thresholds?

Naturally, these questions are only a starting point—you also need to come up with answers—and they don't cover every possible issue in Performance Engineering. How will you use these questions? What would you add to the list?

CHAPTER 3

Proven Practices of Performance Engineering

To explore the proven practices of Performance Engineering, we will start with the requirements, architecture, and design; hit some of the highlights of implementation; and walk you through a real-life scenario. The objective of this chapter is to present a complete case study for each practice so you can begin to understand what it means for you, and to provide you with a story you can use and share with your team or organization.

Requirements, Architecture, and Design

Here is a list of proven practices for requirements, architecture, and design:

- Identify components
- Set performance budgets
- Establish acceptance criteria
- Plan for outliers
- Build in performance culture
- Prototype (and test)

Introduction

One of the questions many people ask themselves while adopting Effective Performance Engineering practices is, “How do I engineer configuration and applications before starting development?”

The building in Figure 3-1 represents the requirements of software and architecture architecture and design.

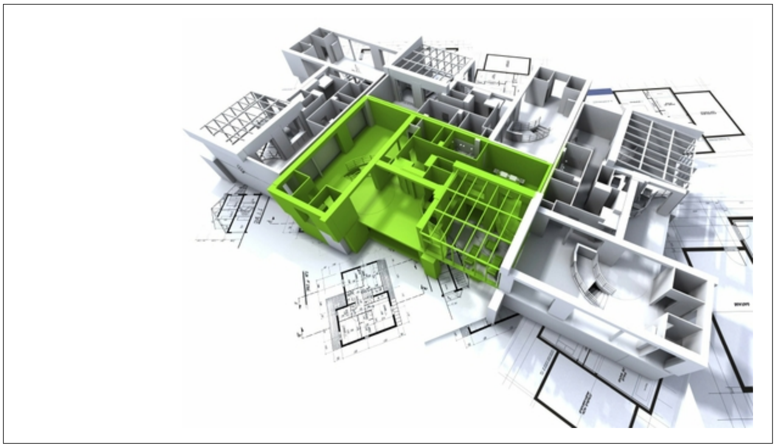


Figure 3-1. The requirements of architecture and design represented as a building

Today, teams architect and design within their own pillars, typically within a development or architecture team, and sometimes seen within a “Sprint 0” or other phase if the project or system is new.

The increasing complexity of composite applications, and the multitude of end users and ways of consuming products and services, has compounded the root-cause issues described in Chapter 2. Effective Performance Engineering provides a new way of thinking about software and hardware systems and how to architect and design them so your end user has a great experience, and you have a high-performing and resilient capability supporting your products and services.

In many cases these are complex and dependent systems—sometimes new, but often existing and integrated throughout both the frontend and backend. Knowing this, you may be focused initially on “big risk/big impact” systems like web and core systems.

Although the risk and reach is high, so is the value you can deliver to the business and your end users.

Scenario

Company ABC is a large financial services institution that has grown over the years to 9 million+ customers and continues to grow at a high rate both organically and through acquisition. They have been transforming to integrated and self-managed Agile teams for nearly a year. A new epic has been prioritized for a new capability that will be rolled out to all customers across all channels, and although it is a vendor commercial-off-the-shelf (COTS) product, it has not been previously deployed at this scale by the vendor or any other known enterprise organizations.

Challenges

As you might imagine, this task is not without its challenges—some known and some not. To help you spot these potential challenges, here are some additional details.

This is the largest Internet-based bank in the United States. The new epic has to do with a login security capability, in which end users will no longer be using their PIN but an image + phrase + login + password to gain access to their accounts. This is a very good brand, having earned the business of 9 million+ Americans and seeing low double-digit growth rates of new customers. 180,000 logins per hour across 7 total major geographic locations worldwide (5 of which are in the US). Of the 9 million+ customers, 40%+ most often access their accounts via mobile with a range of mobile conditions across 2.5G, 3G, and 4G connection types. When rolled out, 100% of all customers upon login will be requested to set up and complete the new login procedure prior to getting access to their accounts. This is planned as a software-only install, with no new hardware or other infrastructure upgrades needed, per the vendor.

You can imagine some of the complexities that might exist or quickly begin to surface with this scenario.

Option 1

Roll out the new login to all customers simultaneously, and provide additional infrastructure to support the initial spike in activity.

The pros of Option 1 include:

- Increased marketability of the new feature (enhanced security capability for end users)
- Shortest perceived time to first implementation
- Shows urgency in response to partnering technology with business

The cons of Option 1 include:

- Increased risk of failure
- Potentially high infrastructure cost
- Introduced elasticity requirement, forcing potential re-architecture of the application
- Dependency on potential unknown cloud service provider to scale

Option 2

Roll out the new login to a reduced segment of the population, using a staggered approach.

The pros of Option 2 are as follows:

- Perceived reduced risk of failure
- Ability to monitor the affected install base and make quick decisions to fix if needed
- Measurable incremental scaling of capacity monitored and observed with ramp-up

And the cons of Option 2 include:

- Duplicate infrastructure during the rollout to support dual authentication mechanisms
- Potential database and parallel user login profiles during transition and possibly higher risk remediation
- May take longer to roll out to large-scale customer base

Option 3

Allow customers to opt in to the new login method over a period of several days or weeks.

Here are the pros of Option 3:

- Lowest potential risk of impact from high adoption rates of new login credentials

And here are the cons of Option 3:

- No incentive for customers to switch
- High likelihood that customers will not switch, leading to longer time to support both login types
- Sample size of data through conversions potentially too small for any meaningful indication of future impact
- Probably will need to push a force to transition in the future, which could lead to massive customer demand
- Perceived risk in the market to customers and/or regulatory impact

Recommendation

Given these three options, we recommend Option 2. It provides a balanced approach by deploying the capability in a timely manner, while mitigating both the risk of overcapacity from demand as well as the business risk from brand and regulatory impact.

Summary

This scenario was based on a real-world situation in which the resulting impact was nearly catastrophic to the organization. At the time, there was no indication that a well-established commercial product would have such a detrimental impact.

Here we have the benefit of hindsight to tell us there were better ways to roll this change out, capture relevant usage and impact data from a subset of the production users, and make the necessary adjustments. It is important to ask, “What will be the impact if this fails?” and somehow mitigate that without adding weeks or months to the project. Architects rarely get to see their vision brought to life.

In the real-world scenario, what happened was a staggered approach to deployment. However, what was not known or predicted was the users' behavior: specifically, that nearly every user would scroll through all 100+ images to see which related most to them, and that each image was 700KB in size with 5 images per page displayed. As a result, even with only 1% of the population (90,000 customers) setting up their new login credentials, each pulled a huge amount of data (3.5MB per page, at 20+ pages per customer) through the data center network lines. The result was a massive production incident that taxed nearly all systems due to overcapacity issues. This was compounded by the fact that ~40% of users were on mobile connections, causing the sessions to remain open and ultimately run out of connections throughout all components.

How-To

In this scenario, you can see why the proven practices of Performance Engineering for requirements, architecture, and design should play a significant role within your organization. The first step is *identifying the components*, you should consider within the infrastructure and application architecture, knowing some will be internal and others external, and some private while others are shared. The next consideration is *setting performance budgets*, or allocating milliseconds per component to target, in order to deliver the desired end-user experience. Defining the acceptance criteria for each component, specifying the conditions and use case (e.g., network conditions and image sizes), is critical. You can see how *planning for outliers* might have led the bank to make a bit more capacity available, and a more conservative deployment approach. *Building in performance culture* would have helped a lot here, especially in considering vendors for commercial off-the-shelf (COTS) products and capabilities, and establishing performance criteria as part of the interrogation criteria prior to acquisition through the procurement process—let alone production deployment. This is where also *prototyping and testing* would have provided significant insight and information proactively, which the bank could have then applied and used as feedback in determining the best technology approach with the business for the end user.

Key Implementation Considerations

- Organizational
  - — Where do people sit (different teams versus integrated)?
  - — Who is accountable for performance?
- Cultural
  - — Place value on performance
  - — Hold people accountable for performance
  - — Tie compensation to performance
- Technical
  - — Build performance into the story
  - — Build performance into the architecture

Proven Practices for DevTest

Continuing our journey into the proven practices of Performance Engineering, next we describe a DevTest scenario and explore some of the highlights. In this section, we once again start with a list of proven practices, then walk through an introduction, followed by a scenario, summary, how-to, and key considerations.

Here are the proven practices of Performance Engineering for DevTest:

- Build performance into your UNIT tests
- Build performance into your build validation tests (BVT), getting results after every build
- Track the trend of results for systems and components
- Automate quality gates in the build in order to avoid perpetuating poorly performing components
- Consider a branching strategy that enables you to "keep out" pieces that will break the code and/or make it perform poorly

Introduction

Performance Engineering during application development consists of testing the application in as realistic an environment as is available, without impacting the velocity of the team, and getting relevant performance feedback into development in an automated way. Assuming we have the necessary goals defined from requirements, architecture, and design, this feedback should provide KPIs to support those goals. For example:

- The application needs to support 10,000 active users with subsecond responses for key transactions such as Login, Search, or Confirm Order.
- The application must support a peak volume of 1,000 transactions per second with processing time of no more than 500ms.

These KPIs should reflect the goals of the business, and provide a target for success and improvement such that every build and release of the application is measured consistently against these goals.

Moreover, the process by which these KPIs are measured should reflect the goals and delivery method best suited to support the business. This means an application that follows an Agile or hybrid approach to development and testing should not be held up by the activities of the team or individuals responsible for performance. This is critical, and should define the level of automation and measurement that can be achieved in order to provide the greatest level of feedback with the least amount of impact to the application delivery chain.

To support this statement, we will use an example scenario in which an application team is tasked with reaching certain business goals for performance, but is forced to make tradeoffs in order to maintain their release velocity.

Scenario

Company XYZ is about to launch a new version of their online shipping application, which currently services 2,000 businesses and 95,000 individuals via mobile and web clients in North America.

The new version is required to support their European launch, adding an estimated 150,000 new customers as well as 15 new regional shipping services and 5 new payment vendors across 28 countries.

The service-level objectives (SLO) of the business are to process all orders within 2 seconds, although key stakeholders do not have a good sense of peak user volumes. Estimates from business analysts in the Marketing team estimate an additional 45,000–65,000 individual users.

Challenges

For the Performance Engineering team to validate the SLO, they need an estimated three weeks to build a production-like environment, and another four weeks to complete testing and analysis. Even if a reasonable amount of overlap is achieved, this release delay is unacceptable to the business. The developers don't believe this work is needed, as they are confident in their architecture decisions and in the quality of their code. They also believe that any issues found in production can be fixed quickly enough to minimize the impact. In this example organization, all teams are independent and distributed groups, and Development carries enough influence that this line of thinking is gaining popularity.

Conversely, the media has run stories about the company's expansion into Europe, primarily led by competitor messaging that the company is likely to fail to meet the demands of such a broad and diverse region. Many in the company are aware of the risk of failure to launch, and support the need to validate performance. You can present one of three options in hopes of avoiding a major failure during this important launch.

Option 1

Test the end-to-end application in a completely integrated and scaled environment at the end of development.

The pros of Option 1 include:

- Complete picture of application performance before go-live date
- Ability to identify bottlenecks within and beyond the company's infrastructure

The cons of Option 1 are as follows:

- Massive delays to the project
- Difficult to resolve issues late in development
- Little room for additional delays or retesting

Option 2

Test the end-to-end application in a scaled-down and virtualized environment using service levels provided by third-party services to represent external dependencies.

Here are the pros of Option 2:

- Reasonable facsimile of production performance
- Ability to virtualize external dependencies reliably and capture "what if" scenarios
- Fewer delays to the project timeline, more time for re-testing

And here are the cons of Option 2:

- Assumption-based approach to capturing third-party dependency performance
- Still difficult to resolve issues late in development

Option 3

Test individual application components throughout development, with virtualized internal and external dependencies. The end-to-end performance will be tested at the end of each sprint/cycle.

The pros of Option 3 are:

- Fast feedback to the development team
- Little to no delays to the project
- Reasonable facsimile of production performance

The cons of Option 3 are:

- Assumption-based approach to capturing third-party dependency performance

Recommendation

The recommendation is Option 3. Application changes tend to focus on a few components that can be scaled close to production, while the surrounding dependencies are outside of our control. These may belong to other teams or organizations, or are cost-prohibitive to build to scale in a DevTest environment. Leveraging virtualization for these dependencies is key to isolating and identifying the impact to the components under our control.

Summary

In reality, the best-fit solution for testing any application involves some tradeoffs between the completeness of our picture of performance and the time in which we deliver. There is no one answer that fits every application, environment, and organization, but in most situations we should strive to deliver as much information as possible, as quickly as possible. In general, there are three factors you should consider when deciding how to test: cost, quality, and time. We can choose which of these is most important, but the closer we get to one, the more we sacrifice of the other two.

Figure 3-2 shows the triangle of three key factors that everyone wants; however, it is often said you can choose only two.

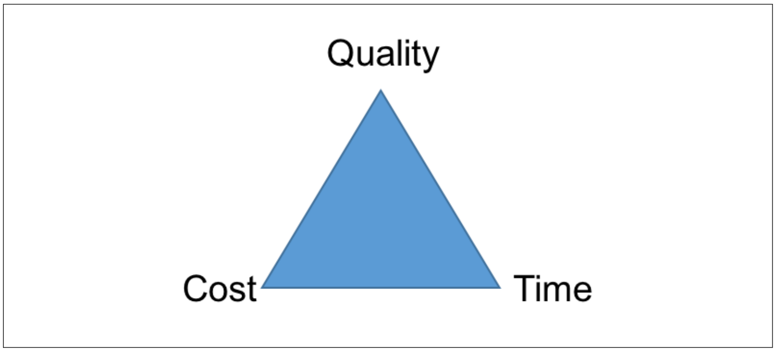


Figure 3-2. The three key factors

How-To

One approach to delivering critical performance information to the Development team without slowing or halting their progress is to automate the execution of performance testing during each and every build. The goal is to understand the performance delta between builds, with a focus on KPI trends rather than the accuracy of individual metrics.

In order to achieve rapid feedback in a continuous integration environment, you must reuse assets and virtualize dependencies captured during previous iterations. Automated execution and analysis are critical. Wherever possible, automation should extend to the analysis of tests as well. Most testing tools, such as Jenkins and Bamboo, extend continuous integration platforms to provide performance trending and feedback after each build is created. Figure 3-3 shows a typical lifecycle representation along with call-out boxes of how performance can be built in at specific stages, demonstrating a possible flow through development that provides quick time-to-value and will not significantly impede development velocity.

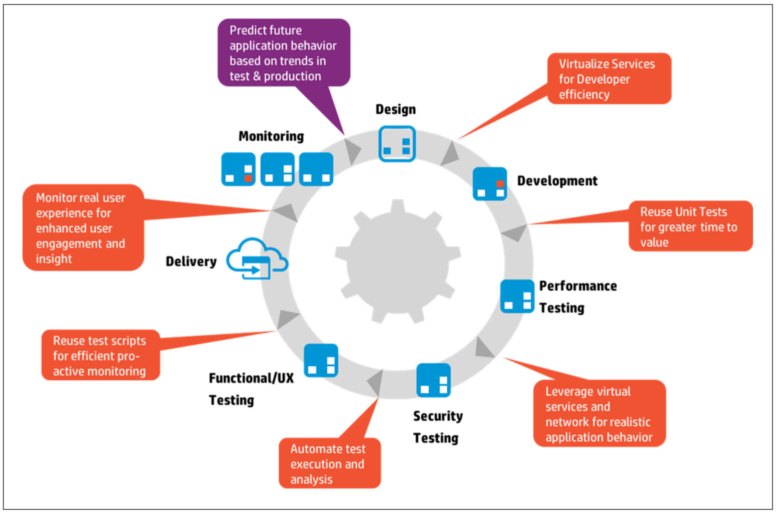

Predict future application behavior based on trends in test & production
Virtualize Services for Developer efficiency
Design
Monitoring
Development
Monitor real user experience for enhanced user engagement and insight
Reuse Unit Tests for greater time to value
Delivery
Performance Testing
Reuse test scripts for efficient pro-active monitoring
Functional/UX Testing
Security Testing
Leverage virtual services and network for realistic application behavior
Automate test execution and analysis

Figure 3-3. Typical lifecycle representation

The most significant feedback comes from performance testing, which should happen in three stages:

1. Execute the baseline test after each and every build.

2. Add/change incremental functionality to the test library as it becomes available.
3. Add new/changed features to the baseline at the end of each release.

In this context, the *baseline test* refers to the previously accepted release criteria, or some level of test that represents the most common usage and behavior of the application. BVT for performance may be contentious for some organizations that have not bought into the value of performance to the company. In these cases, it is important to understand:

- Who is responsible for the performance and stability of the build?
- What information can we deliver that will add value for that person or group?

Every team, technology stack, and application will be different, but we can strive to provide relevant, repeatable, and realistic data in a reasonable timeframe, to avoid blocking the development process.

Parallel to automated test execution and analysis, Performance Engineering teams should execute a series of focused tests to validate the scalability and resilience of the application. These are often run against a pre-production or dedicated performance environment (for the very fortunate), and account for the overarching performance requirements that are not easily captured in a single set of release requirements.

In addition to testing, there needs to be some sort of continuous feedback from production. In the following sections, we will discuss some of the ways that monitoring can be used to provide critical information to development and testing, and impact the design phase as well.

Key Implementation Considerations

- Organizational
  — Who is responsible for your builds?
  — Are they considering performance?
  — What does "pass" mean?

- Cultural
  - — What happens when you break a build?
  - — How is progress/status socialized?
- Technical
  - — How are you enabling automation across tools, teams, and roles?

Proven Practices for Operations

Lastly, in our exploration of proven practices of Performance Engineering, we will elaborate on Operations, highlighting, and illustrating important points through a real-life scenario. Again we start with a list of proven practices, walk through an introduction, and follow with a scenario, summary, how-to, and key considerations.

Here are the proven practices of Performance Engineering for Operations:

- Make sure disaster recovery, capacity planning, and resiliency are all known, tested, and observed.
- Provide continuous deployment and operations from preproduction environments through to production.
- Set up a production environment with: canary, live/live, or blue/green deployment approaches.
- Establish predictive, scaled growth, feature and configuration tracking, and continuous feedback.
- Maintain an inclusive monitoring strategy with complete and continuous feedback across all environments.
- Identify the most commonly used features and functionality, updated and used across the stack and lifecycle.
- Identify the "hot" areas: software, hardware, configurations, and so on.
- Establish a production incident review process focused on learning and long-term stability.

Introduction

Operations is often considered the most important part of the business, as this group is responsible for ensuring the Production team is able to provide the products and services that end users want and for maintaining very high uptime and an extremely low incident rate.

Several organizations (possible even a majority) have sought to reduce overall operational expenses by outsourcing operations tasks to a partner in a long-term contract (often 10+ years), in exchange for a guaranteed savings rate per year.

There is nothing wrong with this strategy. That said, many organizations have limited time and resources allotted for transitioning operations to the partner, and often that partner has not yet adopted Effective Performance Engineering practices.

That's a recipe for less-than-desirable results, and often becomes a point of contention wherein both parties spend much energy and effort negotiating the contract rather than focusing on the end user.

Let's take a look at such a scenario, and how you can mitigate these challenges with Effective Performance Engineering practices.

Scenario

Company 123 is a large international business with 9 reportable segments (Agriculture and Nutrition, Nylon Enterprise, Performance Coatings and Polymers, Pharmaceuticals, Pigments and Chemicals, Pioneer, Polyester Enterprise, Specialty Fibers, and Specialty Polymers) spanning more than 70 countries worldwide. They make 49% of consolidated sales to customers outside the US, employ a total of 93,000 people, and have a $68 billion market cap. In addition, they operate the largest Electronic Data Interchange (EDI) operations and are the largest SAP customer in the world.

They are seeking to optimize their growing operations and support systems, as well as reduce the associated costs, while still maintaining the service level for their end users. Their intent is to partner and outsource the global information systems and technology infrastructures in order to provide selected applications, software services, and information systems solutions designed to enhance manufacturing, marketing, distribution, and customer service.

The company is seeking to reach a 10-year agreement with a partner to transition all operations and support with a guaranteed savings of 10% per year for the 10 years of the initial contract term. Transition is scheduled to take place over a 3-month period and at least 70% of the existing company resources will transition to the new partner organization as part of this operations outsourcing agreement.

Challenges

If you have ever been a part of something like this, then you know: the sheer size of this maneuver is scary. Now add the fact you are outsourcing your "global information systems and technology" business for 10 years to a partner (where your talent is going), and it gets exponentially more interesting.

Needless to say, this transition was not without several challenges, and we will see how this organization became a cautionary tale for the importance of implementing Effective Performance Engineering capabilities that benefit all parties. Of course, let's not forget Company 123 is nearly 100 years old, comprises 9 segments in 70+ countries, serves 93,000 people, and has a $93 billion market cap.

Option 1

Single partner.

The pros of Option 1 are as follows:

- Single neck to wring if needed
- Little ambiguity or dependency on partner's ability to deliver on the contract to company
- Talent goes to a single-partner organization

The cons of Option 1 are as follows:

- Limited diversification, consolidated risk

Option 2

Two partners.

The pros of Option 2 are:

- Competitive environment

- Diversity of service, redundancy

The cons of Option 2 are:

- Multiple players
- Increased possibility for finger-pointing
- Talent dispersed over two partner organizations

Option 3

Hybrid (split: Company 123 resources and one or more partners).

Here are the pros of Option 3:

- Company can control speed of transition
- Competitive environment
- Iterative learning and feedback

And here are the cons of Option 3:

- May take longer to transition
- Reduction of operational expense delayed
- More company effort required

Recommendation

The recommendation is Option 3. Option 3 enables Company 123 to begin to realize the benefits and value of transitioning operations to a partner, and adapt and change their business practices over time, while observing the positive and negative effects on end users.

Summary

In reality, Company 123 moved forward with Option 2. As a result, Partner 1 contracted to operate a majority of Company 123's global information systems and technology infrastructure, and provide selected application and software services; and Partner 2 contracted to provide information systems solutions designed to enhance Company 123's manufacturing, marketing, distribution, and customer service.

The company reached a 10-year agreement with both partners to transition nearly all operations and support with a guaranteed savings of 10% per year for the 10 years of the initial contract term. Transition was scheduled to take place over a 3-month period with at least 80% of the existing company resources transitioning to the new partner organization as part of this operations outsourcing agreement.

Due to many of the identified "cons" of Option 2, this approach was riddled with challenges throughout the term of the contract, with the transition working out for one partner and not the other, and Company 123 ended up taking back much of the outsourced support and operations. The end users suffered the most, followed by Company 123's businesses; today, some 15 years later, the company has a $46 billion market cap—approximately 50% smaller than it had before the partnership.

How-To

In this scenario, you can see why the proven practices of Performance Engineering for operations should play a significant role within an organization. These practices start with *disaster recovery, capacity planning, and resiliency*, which should all be known, tested, and observed for critical capabilities and functions. *Continuous deployment and operations* spanning from the earliest pre-production environments through to production enables you to test the deployments, control and protect the deployment procedures, and measure the quality of the release candidate. A production environment enabling *canary*,*live/live*, or *blue/green*-type deployment approaches can mitigate the risks associated with traditional, single-data-center, big-bang deployments. *Predictive* models allow you the chance to gain insight into how a capability will scale under growth, track features and configuration, and get continuous feedback from pre-production through production and back, supported by an *inclusive monitoring strategy with complete and continuous feedback* across all environments. Other practices include identifying the *most commonly used features and functionality*, in order to show when updates happen and how they're used across and beyond the stack and lifecycle, and *identifying where the "hot" areas are* related to software, hardware, and configurations. Lastly, establish a solid cross-team *production incident review process* rooted in and focused on learning and the long-term stability of the team and all environments, in order to make products and services continuously available to the business and end users.

Figure 3-4 shows the intersection of workflow, toolchain, and artifacts, along with the automated flow from left to right of a common automated build/test/deploy cycle. It highlights the (often automated) check and release-to-production steps, as well as the integration of performance considerations and feedback throughout the cycle.

Key Implementation Considerations

- Organizational
  - Who does delivery through DevTest versus ProdOps?
  - When a delivery goes wrong, who is responsible?
- How are you measuring the quality of a release?
  - Stakeholders (business/technical/customer)
  - Are all parts considered?
  - Visualized and communicated
- Cultural
  - What happens when you have a failed delivery?
  - Is rollback tested every time prior to your release?
  - What are your performance metrics?
  - Ops = Chaos
  - Frequency of alarms, acceptable
  - Performance Engineering teams > time in production
- Technical
  - How are you mitigating quality and performance with frequency of release?
  - Are you using canary deployment approaches to minimize impact to production?
  - How are your blue/green releases?
  - Backup and recovery
  - Load-balance and CDN
  - End-to-end (client, server, network, and app layer)

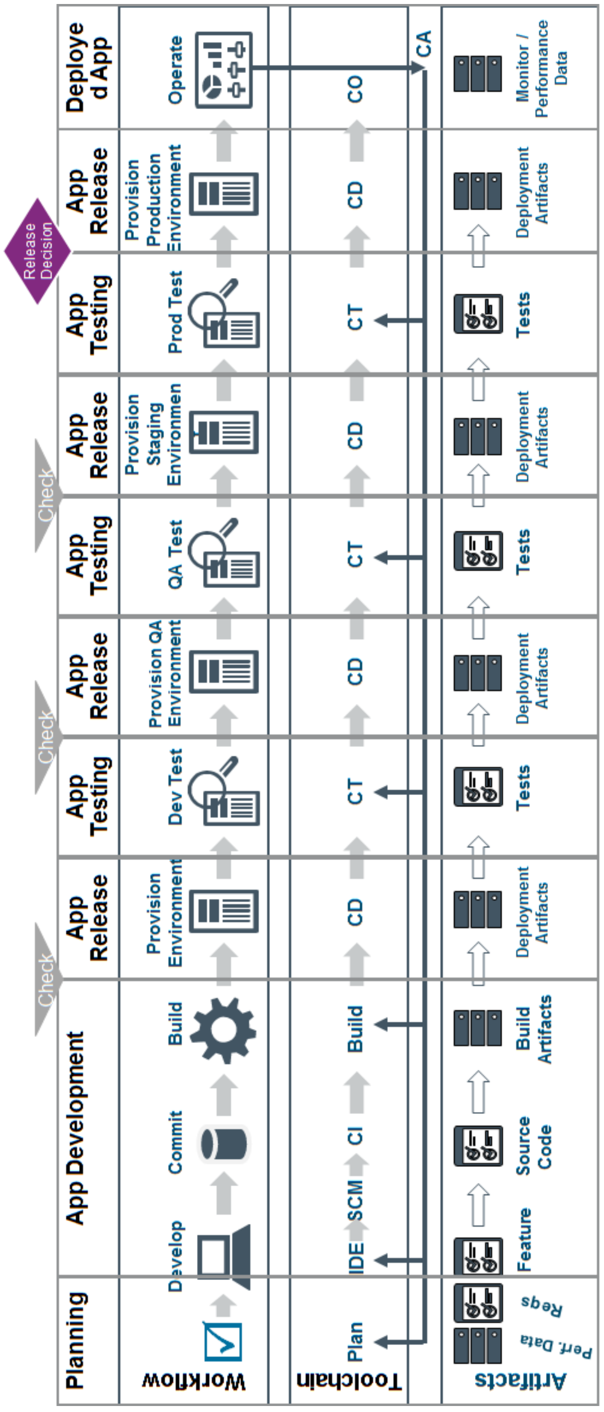


Figure 3-4. Intersection of workflow, toolchain, and artifacts in a build/test/deploy cycle

CHAPTER 4

Tying It All Together

As more businesses experience devastating production incidents, they are recognizing that they need to change, and are working to implement Effective Performance Engineering practices. They're restructuring their teams and redefining jobs such that some team members are focused on ensuring that the essential computer infrastructure and applications deliver good, stable performance at all times. They're embracing practices in Performance Engineering and treating them as critical, adopting an organizational culture supporting this transformation, and rewarding individuals for their contributions.

Keep in mind that Performance Engineering doesn't refer only to a specific job, such as a "performance engineer." More generally, it refers to the set of skills and practices that are gradually being understood across organizations that focus on achieving higher levels of performance in technology, in the business, and for end users.

Many naive observers often take the same attitude toward Performance Engineering: it's simply a matter of making sure the systems run fast. If possible, make them run really fast. When in doubt, just make them run really, really fast. And if that doesn't work right away, throw money at the problem by buying more hardware to make the systems go really fast.

But just as there's more to winning a track meet than being fast, there's more to building a constellation of quick, efficient web servers and databases than being fast. Just as athletes can't win without a sophisticated mixture of strategy, form, attitude, tactics, and speed, Performance Engineering requires a good collection of metrics and tools to deliver the desired business results. When they're combined correctly, the results are systems that satisfy both customers and employees, enabling everyone on the team to win.

Metrics for Success

One critical element of integrating a Performance Engineering culture within an organization is to determine what performance metrics you need to track and assess whether you can measure them with confidence.

How often do we hear development and testing organizations and even managers refer to lines of code written, scripts passed and executed, defects discovered, and test use cases as a measure of their commitment to software quality?

At the end of the day, these measurements are useless when it comes to delivering results that matter to your end users, that keep them coming back for more of your products and services. Think about it. Who cares how many defects you've found in pre-production? What does that measure?

We want to make a fairly bold statement: these old, standalone test metrics don't matter anymore.

When it comes to quality in development, testing, and overall operations, these are the questions you should be asking yourself:

- How many stories have we committed to?
- How many of these were delivered with high quality to the end user?
- How much time did it take to deliver from business or customer concept to production?

Finally, ask yourself this: "Are our end users consuming the capabilities they asked for?"

Activities Versus Results

The difference between this sort of focus and a purely technical focus is the difference between activities and results. If our team commits to 80 story points at the beginning of a 4-week sprint but we deliver only 60, then we're not meeting our commitment to ourselves, the business, or the customer. It also means our team is preventing another team from delivering on their commitments. The release will instead be put on hold and pushed to our next release. Ultimately, the business results are going to be less than what we promised.

Over the last several years, improvements in development and testing have provided an opportunity for organizations to apply new metrics that can lead to genuine transformation. The most common of these proven concepts is Agile development practices. When executed well, Agile methods can enable a team to quickly deliver high-quality software with a focus on the highest priority for the business and end user. As teams transform, having a few key measurements and producing results helps the organization evolve in an informed manner, with continuous feedback from the investments they're making.

Without these types of metrics, organizations will simply attempt their transformation blindly, with limited capacity to show results, including the business outcomes demanded of today's technology organizations.

Top Five Software Quality Metrics

Here are the top five quality metrics that really matter:

Committed stories versus delivered results meeting doneness criteria

Remember the last time someone committed to do something for you and either failed to deliver or didn't meet your standards? It caused delays and extra work, along with a lot of frustration. In software development, *stories* are the pieces of work that are committed to and, ideally, delivered on time and to a certain spec.

As you may know, stories represent the simple, high-level descriptions that form a use case, such as a user inserting a credit card into an airline kiosk. Each story needs to be delivered at a specific level of quality or "doneness" criteria. As teams continuously plan, elaborate, plan again, commit, and deliver, the ultimate goal should be to deliver these results in alignment with the broader team's doneness criteria. When that can be measured, the team can showcase its abilities to meet its commitments on schedule and with the highest standards.

Quality across the lifecycle

The demand for software delivery speed continues to increase along with the demand for reduced costs. But how can you achieve these goals when you don't have the time and resources to manually test every build? When you can't afford to wait and find those defects in your late-stage releases? The answer is to follow the build lifecycle from story to code on a developer desktop. Next, you should check, build, and unit test. Continue by using automation through the rest of the process, including automated functional, performance, security, and other modes of testing. This enables teams to show the quality of a build throughout the lifecycle with quality metrics and automated pass/fail gates.

Given the frequency of these builds and automated tests, build-life results can be created and measured in seconds, minutes, and hours. Now, your most frequent tests are fully automated, and you're only doing manual tests on the highest quality releases that make it through the automated lifecycle. This results in automated build-life quality metrics that cover the full lifecycle, enabling your team to deliver with speed and quality, while reducing costs through higher efficiency.

Production incidents over time and recurrence

Just as it's important to show the quality of the release over time, it's also important to minimize production incidents and their recurrence over subsequent releases. Table 4-1 illustrates a technique we've used to compare team performance over time. Imagine you are working with five teams over three completed releases; this shows how an *information radiator* can be used with simple and minimal key data to visually represent important results, such as "% Commit Done" and "# Prod Incidents," delivered across teams.

The target for this typical (though imaginary) organization is 95% of committed stories delivered and zero production incidents. Teams that didn't meet these goals are highlighted in bold red. Often, production incident numbers are found within an incident management process. Defining the root cause and implementing corrective measures enables continuous improvement and prevents recurrence of the same issue in subsequent releases. With these quality metrics in place, you can learn which teams meet specific goals. Finally, you can look across teams and discover why proven concepts work.

Table 4-1. Using an information radiator to visualize results

	Teams		Releases			
		Team Averages	2016-Jan	2016-Feb	2016-Mar	2016-Apr
% Commit Done	Alpha	97%	96%	98%	97%	
# Prod Incidents		**2**	**1**	0	**1**	
% Commit Done	Beta	**94.33%**	**92%**	95%	96%	
# Prod Incidents		**6**	**3**	**2**	**1**	
% Commit Done	Gamma	100%	100%	100%	100%	
# Prod Incidents		**1**	0	0	**1**	
% Commit Done	Delta	**93.33%**	100%	100%	**80%**	
# Prod Incidents		**2**	0	0	**2**	
% Commit Done	Epsilon	**92.33%**	**85%**	95%	97%	
# Prod Incidents		**1**	0	0	**1**	
% Commit Done	Totals	95.398%	94.6%	97.6%	94%	
# Prod Incidents		12	4	2	**6**	

User sentiment

Get to know your end users by measuring how they feel when interacting with an application or system. By capturing and dissecting the feedback they provide regarding new or improved capabilities, you can incorporate their needs into an upcoming sprint. At the very least, you can develop a plan to deliver something in response to those needs.

On a larger scale, your analysis and incorporation of user sentiment can expand to a more general market sentiment, which can broaden your impact and market presence. Several components of quality can be covered via this metric, including simplicity, stability, usability, and brand value.

Continuous improvement

Following retrospectives, allow time and effort to implement prioritized, continuous improvement stories. This enables the team to self-organize and be accountable for improving the quality of their process. When you allocate this time and make it visible to all, the team and stakeholders can show their immediate impact. They can demonstrate how one team, compared to others, has delivered results at increased speed, with higher quality and value to the end user. This allows team leads to ask and possibly answer these questions: are there certain practices that need to be shared? How do teams perform over time with certain changes injected? The continuous improvement metric can also justify recent or proposed investments in the team.

What Really Matters

It's amazing to see how many teams are still working the old-fashioned way. In fact, the empathy and sympathy poured out from others in the field is overwhelming. We hear and share the same stories we shared 20+ years ago. For example, have you heard this lately, "I have 3,896 test cases, and I'm 30% complete on test execution"? We should all ask, "So, what does that mean for time, quality, and cost, along with on-time delivery to the end user?" It's genuinely shocking when we hear from a VP about their mobile-testing process, only to learn that the company's mobile strategy is a "mobile guy" who does manual testing by putting the application on his phone and playing with it—maybe even wrapping it in aluminum foil and walking up and down some hills or taking the elevator to simulate real-world users and weak network conditions.

Let's start focusing on metrics that really matter. We need results that center on the value and quality we deliver to our end users. In the process, let's not forget how to deliver. We need teams to contribute creatively and improve the practices they have, while measuring quality via metrics they can use to evaluate, modify, and improve processes over time.

What happens when we insist on the old style of quality metrics?

Well, for one thing, it helps explain why so many CIOs hold their positions for less than two years or why a third of them lose their jobs after a failed project. We've seen this before: a new CIO or senior leader comes in, fires a few mid-level managers, reorganizes a couple of things, and brings in a new partner, and suddenly they're trying to measure results. Unfortunately, they don't have the right metrics in place to show how the team is delivering. Command and control fails again. Sadly, this fails the business, shareholders, passionate individuals, and ultimately the end user: the customer.

You do not want to fail your customer.

Other Performance Engineering Metrics

The top five quality metrics are a foundational and important starting point for Effective Performance Engineering. In addition, there are a variety of other Performance Engineering metrics that come into play:

- Release quality
- Throughput
- Workflow and transaction response time
- Automated performance regression success rate
- Forecasted release confidence and quality level
- Breaking point versus current production as a multiplier
- Defect density

This drive to explore new metrics and find better ways of understanding how software is succeeding (and failing) is going to continue and grow even more intense. Software engineers understand that it's not enough to simply focus on the narrow job of going fast. The challenge is capturing just how the software is helping the company, its employees, and its customers. If they succeed, then the software is a success.

There are big differences in the ways companies are approaching the challenge. They're mixing enterprise, commercial, and open source tools, and using a wide range of metrics to understand their results. We've seen key metrics that are accepted by all groups of stakeholders—metrics that all businesses can start using today. However, there's nothing like enabling the team to also measure what matters to them, because what matters to your team may matter deeply to your success.

Automation

Automation can mean different things to different people. In this section, we explore why performance testing is not enough, investigate the four key areas to focus on as a performance engineer, and discuss how to apply these practices in the real world. You will see how automation plays a critically important role in Performance Engineering.

Performance Testing Isn't Enough

Software, hardware, and the needs of application users have all changed radically in recent years, so why are the best practices many developers use to ensure software quality seemingly frozen in time? The world has evolved toward Performance Engineering, but too many developers still rely on performance testing alone. This can lead to disaster.

The initial failures of the Healthcare.gov website revealed how fragile underlying systems and integrated dependencies can be. Simple performance testing isn't enough. If you don't develop and test using a Performance Engineering approach, the results can be both costly and ineffective.

What went wrong with Healthcare.gov? A report in Forbes cited these eight reasons for the site's massive failure:

- Unrealistic requirements
- Technical complexity
- Integration responsibility
- Fragmented authority
- Loose metrics
- Inadequate testing
- Aggressive schedules
- Administrative blindness

President Obama, the CEO in this scenario, received widespread criticism over the troubled launch, which should have been a high point for his presidency. Instead, the site's poor performance tainted the public's perception of the program. When you embarrass your boss, you don't always get a second chance. In the case of Healthcare.gov, the Obama administration had to bring in new blood.

So, how do failures like this happen?

When developers and testers were working in a mainframe or client-server environment, the traditional performance testing practices were good enough. As the evolution of technology accelerated, however, teams have had to work with a mix of on-premises, third-party, and other cloud-based services, and components over which they often have little or no control. Meanwhile, users increasingly expect quick access anywhere, anytime, and on any device.

Four Key Areas of Focus

Performance Engineering practices help developers and testers solve these problems and mitigate risks by focusing on high performance and delivering valuable capabilities to the business.

The key is to start by focusing on four key areas:

- Building in continuous business feedback and improvement. You accomplish this by integrating a continuous feedback and improvement loop into the process right from the beginning.
- Developing a simple and lightweight process that enables automated, built-in performance. In this way, the application, system, and infrastructure are optimized throughout the process.
- Optimizing applications for business needs.
- Focusing on quality.

Applying the four key areas

Your team can head off unrealistic requirements by asking for and using feedback and improvement recommendations. To avoid technical complexity, your team must share a common goal to quickly define, overcome, and verify that all systems are engineered with resiliency and optimized for business and customer needs. Integration responsibility must be built into all environments, along with end-to-end automated performance validation. This should even include simulations for services and components that are not yet available.

The issue of fragmented authority won't come up if you create a collaborative and interactive team, and you can avoid the problem of loose metrics by using metrics that provide stakeholders the information they need to make informed business decisions. Inadequate testing will never be an issue if you build in automated testing, including functional testing for:

- Performance
- Security

- Usability
- Disaster recovery
- Capacity planning

Overly aggressive schedules are unlikely to occur if you provide automated quality results reports that highlight risks and offer optimization recommendations to support informed decision making. Finally, to prevent administrative blindness, focus on business outcomes, communicate with all stakeholders throughout the process, and build in accountability and responsibility for delivery.

It's your responsibility to ensure that your organization is moving from antiquated methodologies based on performance testing only to more comprehensive Performance Engineering practices. After all, no one wants to be the next Healthcare.gov.

Big Data for Performance

Performance Engineering has long been a practice adopted in the world of high-performance automotive. One of the results we often see in our "Performance Engineering" Google Alert is Lingenfelter Performance Engineering. When you go to the "About us" section of their website, it states:

> Lingenfelter Performance Engineering was founded over 43 years ago and is a globally recognized brand in the performance engineering industry. The company offers engine building, engine and chassis tuning components and installation for vehicle owners; component product development; services to manufacturers, aftermarket and original equipment suppliers; prototype and preparation of product development vehicles; late product life-cycle performance improvements; durability testing; and show and media event vehicles.

Looking at high-performance automotive organizations like Lingenfelter (and many others), it is easy to see a direct correlation between all of the components and engineered elements that make a high-performance automobile and these of our business systems (and between their drivers and our end users). The parallel that we want you to recognize is the now available "Big Data for Performance," which the high-performance automotive industry has been leveraging for many years, yet we as Performance Engineers are only starting to utilize. This big data and the accompanying predictive analytics, both of which leverage the capabilities of Performance

Engineering, will enable us to best support our businesses and end users through technology.

To finish out this analogy, do organizations like Lingenfelter only wait until final deployment to see how the automobile they are optimizing will perform? No, they have adopted practices for looking as a team at each component along the way, making decisions, and optimizing the components based on data to ensure they are high quality.

Performance as a Team Sport

Over the last few years, organizations have started to define and embrace the capabilities of Performance Engineering, recognizing that their systems are growing so complex that it's not enough to simply tell the computers or the individuals behind them to "run fast." This capability must be built into the organization's culture and behavior, and it must include activities for developers, database administrators, designers, and all stakeholders—each coordinating to orchestrate a system that works well, starting early in the lifecycle and building it in throughout. Each of the parts may be good enough on its own, but without the attention of good engineering practices, they won't work well enough together.

Market Solutions

As you look across the market, you will see there are a number of analysts, partners, and software tool vendors actively marketing their Performance Engineering capabilities.

To simplify the decision-making and implementation process for you, we've provided some Performance Engineering topics with links to key information at *http://www.effectiveperformanceengineering.com*.

In addition, we've included the results of a Performance Engineering survey that gives a lot more detail about what is going on in the market now.

Performance Engineering Survey Results

Hewlett Packard Enterprise has been working to support Performance Engineering in all organizations. In 2015, it contracted YouGov, an independent research organization, to survey 400 engineers and managers to understand how organizations are using tools and metrics to measure and evolve their Performance Engineering practices. The survey was conducted blind so that no one knew that Hewlett Packard Enterprise commissioned it.

The sample consisted of 50% performance engineers and performance testers, 25% application development managers, and 25% IT operations managers. All came from companies with at least 500 employees in the US. The results reveal a wide range of techniques and broad approaches to Performance Engineering and some of the practices through which organizations are using tools and metrics.

The survey asked, “When you look to the future of Performance Engineering, what types of tools do you and your stakeholders plan to acquire?” In response, 52% of large companies (those with 10,000+ employees) indicated “more enterprise and proven” tools; 37% of the larger companies said they expected “more open source and home-grown”; and the remaining 11% said they were planning “more hybrid of open source and enterprise.” The responses from companies of different sizes followed a similar pattern, but with a bit more balance (see Figure 4-1).

When the results were analyzed based on roles, the majority of respondents planned to acquire “more enterprise and proven” tools, with those identifying as “performance engineer/performance tester” (41%), application development manager (44%), and IT operations manager (51%), as shown in Figure 4-2.

When it comes to testing, an increasing number of companies are concentrating on *burst testing* to push their software closer to the breaking point. They’re spinning up a large number of virtual users and then pointing them at the systems under test in a large burst over a period of time. This simulates heavy traffic generated from sales, promotions, big events, or retail days like Black Friday or Cyber Monday, when a heavy load can wreak havoc on a system (Figure 4-3).

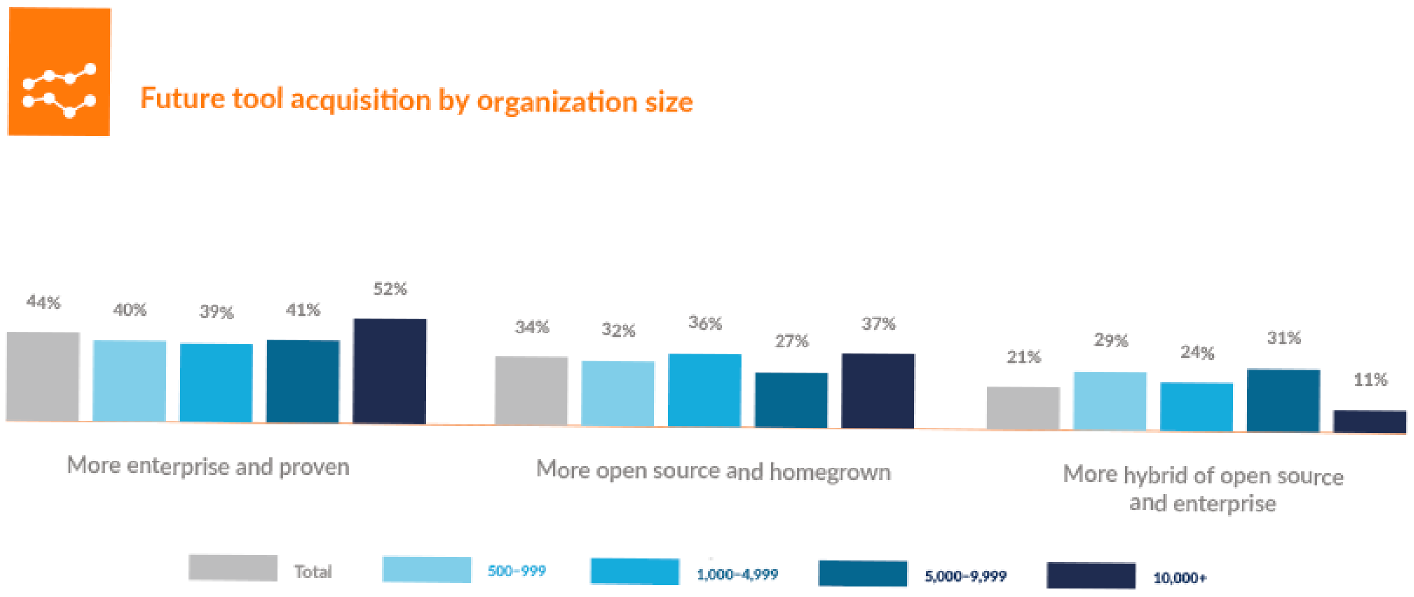


Figure 4-1. Future tool acquisition by organization size

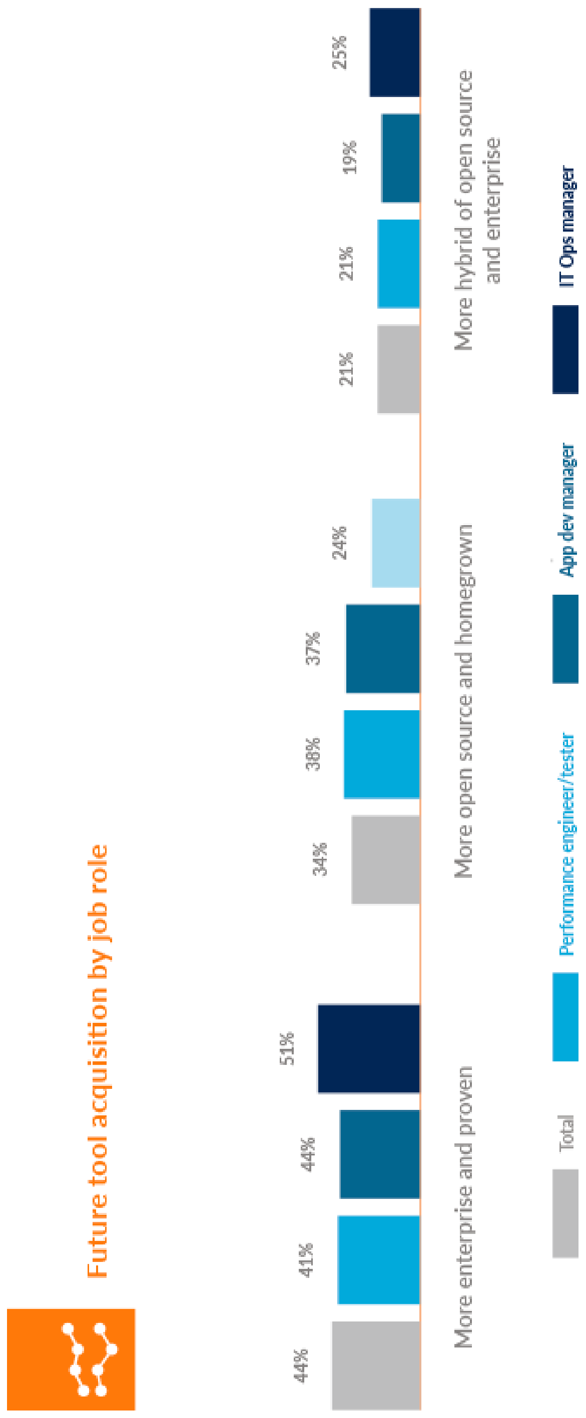


Figure 4-2. Future tool acquisition by job role

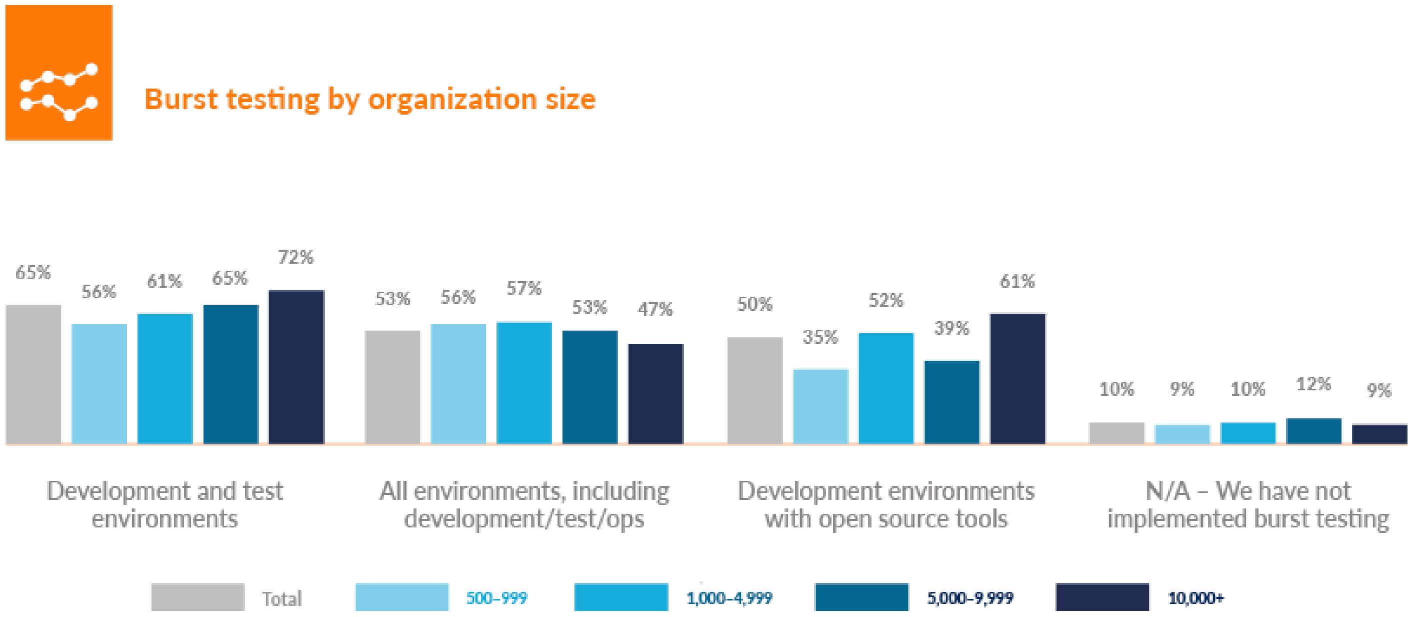


Figure 4-3. Burst testing by organization size

One of the most important options among tools like the ones just cited is the ability to deploy an army of machines to poke and prod at an organization's systems. The cloud is often the best source for these machines, because many modern cloud companies rent virtual machines by the minute. Those working on performance tests can start up a test for a short amount of time and pay only for the minutes they use.

The value of the cloud is obvious in the answers to the questions about the average size and duration of a load test. Only 3% of respondents reported testing with fewer than 100 simulated users. At least 80% of the respondents used 500 or more users, and 14% wanted to test their software with at least 10,000 users. They feel that this is the only way to be prepared for the number of real users coming their way when the software is deployed (Figure 4-4).

Growth in load testing points to the cloud.

This demand will almost certainly increase. When asked how big they expect their load tests to be in just two years, 27% of respondents said that they expect they'll need at least 10,000 simulated users. They mentioned much larger numbers, too; 8% predicted they'll be running tests with more than 100,000 simulated users, and 2% could foresee tests with 500,000 users or more.

While the number of simulated users is growing, duration isn't long enough to make a dedicated test facility economical. The tests are usually not very long; only 8% reported running tests that routinely lasted more than 24 hours. Most of the survey respondents (54%) said that their tests ran between 4 and 12 hours (Figure 4-5).

The largest companies are also the ones that are most likely to be using the cloud. Only 9% said that they don't use the cloud for testing, typically because their security policies didn't permit them to expose their data to the cloud (Figure 4-6).

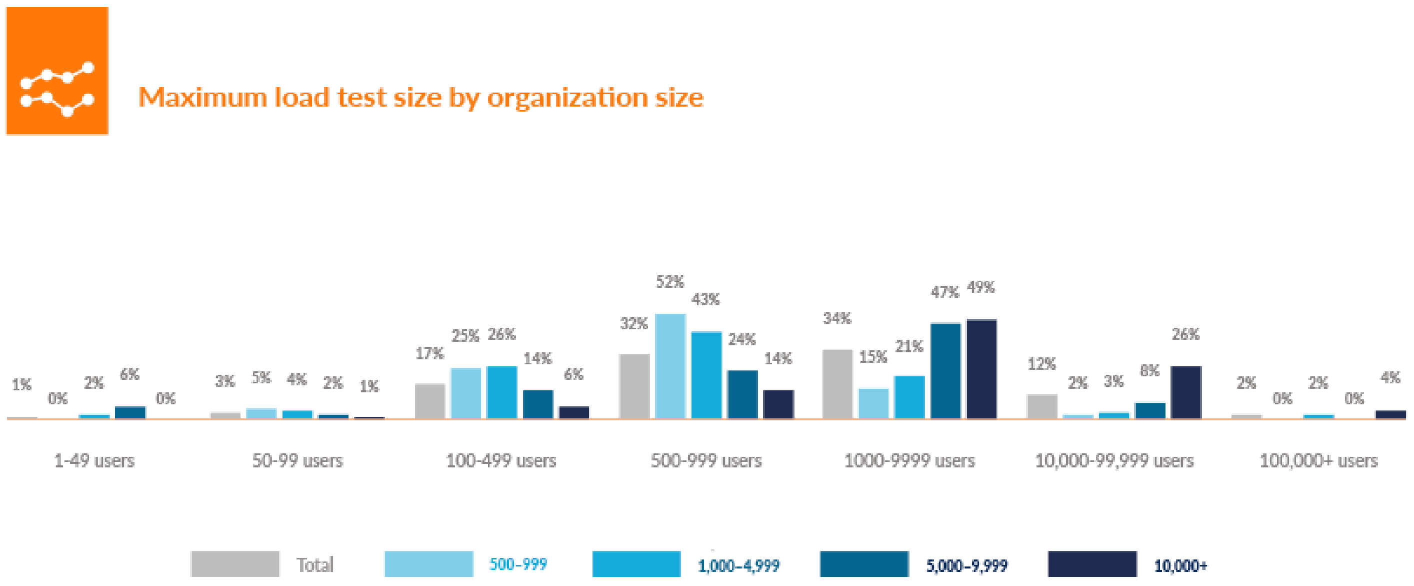


Figure 4-4. Maximum load test size by organization size

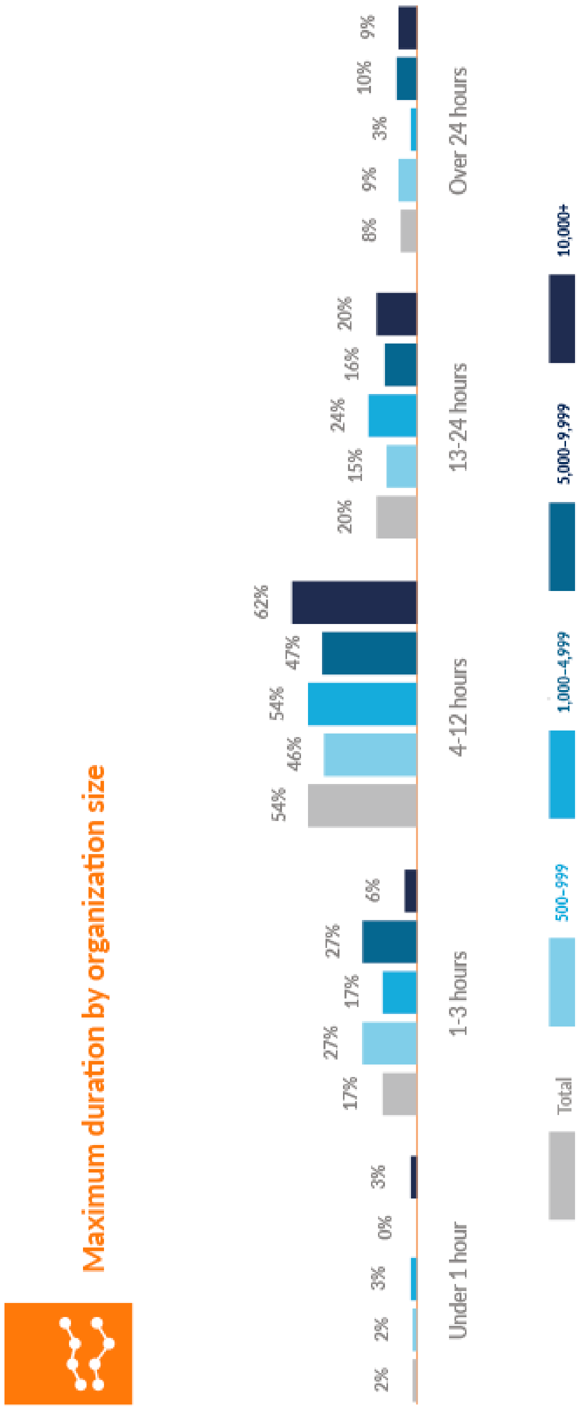


Figure 4-5. Maximum duration by organization size

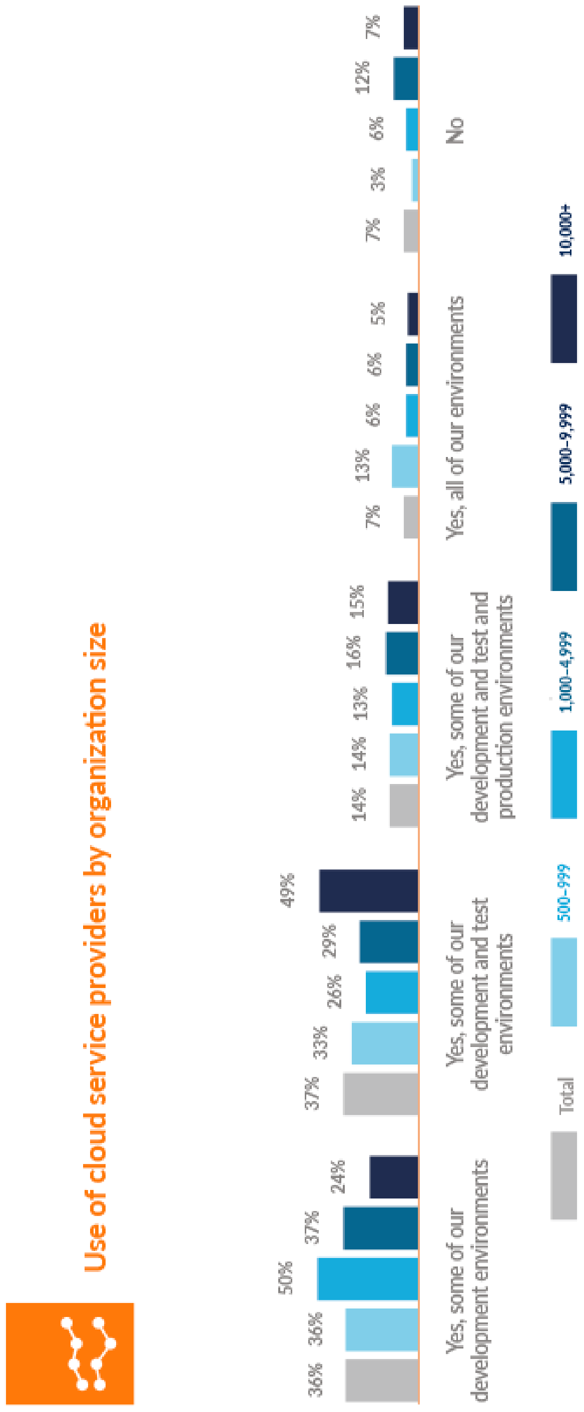


Figure 4-6. Use of cloud service providers by organization site

How to Choose a Solution

Now comes the time for you to start defining what a solution looks like for you. As you begin, we suggest you take a three-step approach: define your goals and objectives, define a timeline, and identify partners. In the following sections, we go through this process in a bit more detail, so you can get started on your path to choosing your Performance Engineering solution.

Define your goals and objectives

Transforming is a complex exercise and one that should have some thought behind it. When thinking about goals and objectives, begin with five key aspects of your teams and organization:

- Culture
- Technology
- Speed
- Quality
- Cost

Each of these considerations factors into the overall goals and objectives for Effective Performance Engineering, and decisions must be made now.

It is a journey and will take some time, and the path will not always be straight; however, getting started in a focused area with some support, adopting a few key practices, and sharing the results is the right approach.

Celebrate the success, examine the results, and then continue along the journey, never losing sight of the end user. With this collaboration and continued guidance and direction, you'll attain success and make forward progress, as you transform into an Effective Performance Engineering organization.

Define your timeline

Timelines are relative. By contrast, value to your end users and stakeholders is more objective and within your control, so you should focus on defining what is important there before setting your timelines.

From a purely leadership and budget/time perspective, it is important to define a timeline with clear goals and objectives within the given budgetary cycle. Doing so enables you to share and communicate results delivered in the prior period, activities being performed in the current period with their forecasted results, and commitments for the future period with their forecasted results.

A timeline should visually represent key milestones along with incremental measures indicating what should be achieved within these milestones. It should also map each task or activity to the value it will deliver to the end user and business.

Figure 4-7 illustrates what this could look like at a high level for you and your organization on your journey to Effective Performance Engineering.

Identify your partners

Partners and thought leaders are often a great resource to provide additional insight, experience, and practical advice from the market, in order to get you aligned with current trends and able to accelerate as desired.

Next we take a deeper look at some thought leaders, consulting partners, and analyst partners, with specific details and links to existing capabilities and assets, so you can quickly get a better and broader idea of some of the Effective Performance Engineering resources available to you.

Top thought leaders of today

Microsoft, Google, IBM, Apple, and Hewlett Packard Enterprise comprise the set of top five thought leaders and influencers around Performance Engineering and testing today.

This set of five is consistent by audience (performance engineers/testers, application development managers, and IT operations managers), as well as by organization size (Figure 4-8).

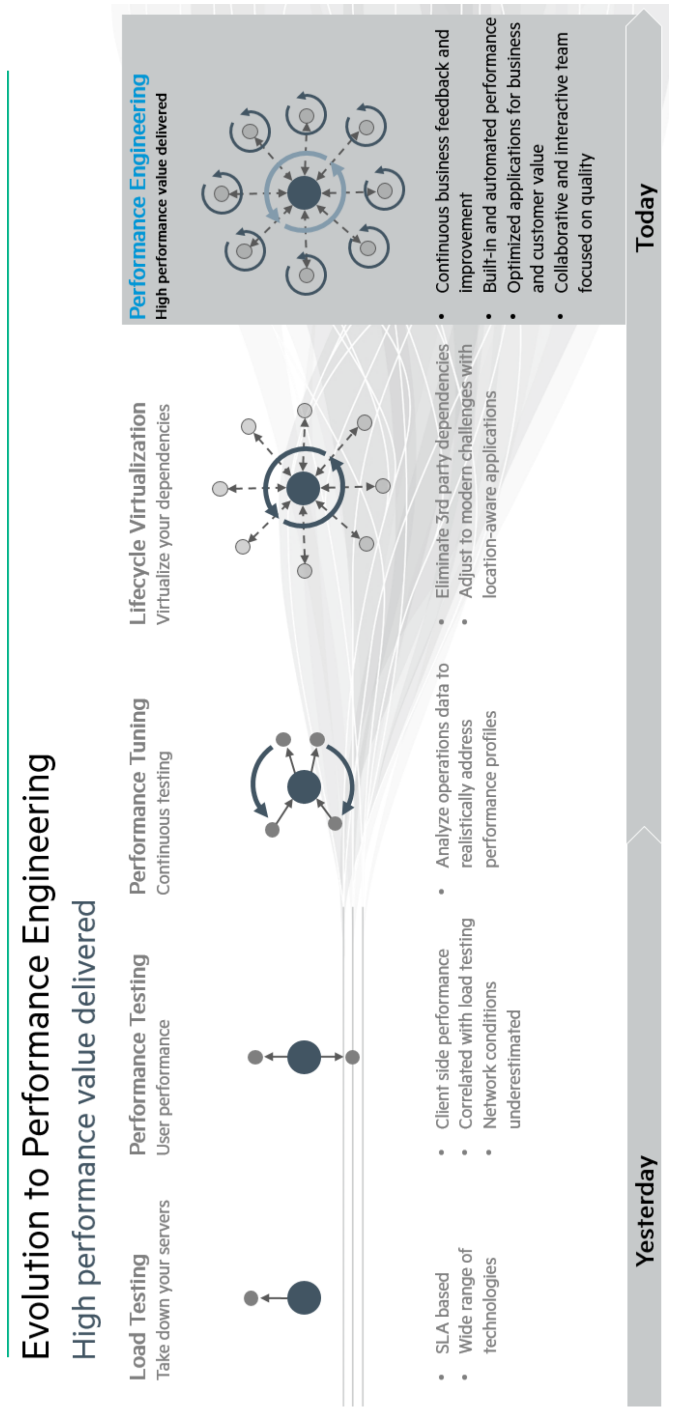

Evolution to Performance Engineering
High performance value delivered
Load Testing
Take down your servers
• SLA based
• Wide range of technologies
Performance Testing
User performance
• Client side performance
• Correlated with load testing
• Network conditions underestimated
Performance Tuning
Continuous testing
• Analyze operations data to realistically address performance profiles
Lifecycle Virtualization
Virtualize your dependencies
• Eliminate 3rd party dependencies
• Adjust to modern challenges with location-aware applications
Performance Engineering
High performance value delivered
• Continuous business feedback and improvement
• Built-in and automated performance
• Optimized applications for business and customer value
• Collaborative and interactive team focused on quality
Yesterday
Today

Figure 4-7. Evolution of Performance Engineering

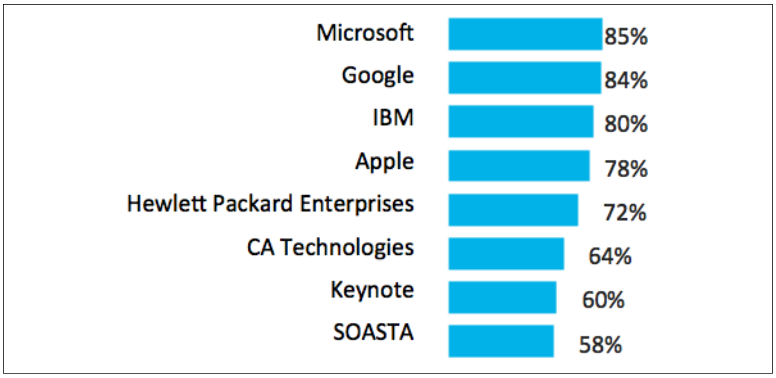


Figure 4-8. Top thought leaders

Preferred partners for Performance Engineering

Accenture, Infosys, Deloitte, HCL, and Tech Mahindra are the top five service providers most often chosen as Performance Engineering partners by the organizations surveyed.

Note that "None of the Above" only represented 7% of all other responses (Figure 4-9).

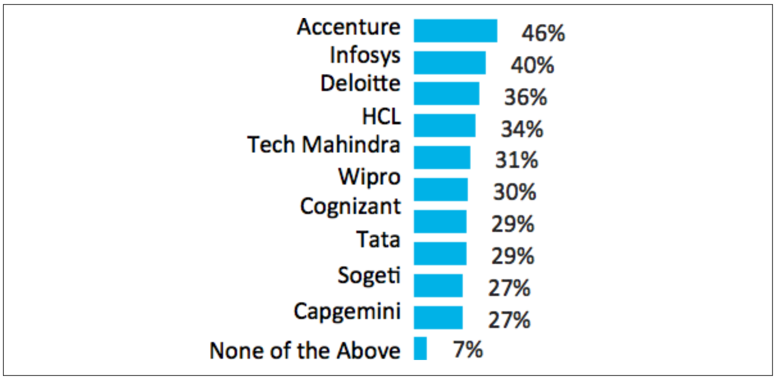


Figure 4-9. Preferred partners

Conclusion

Performance Engineering practices define a culture that enables teams to deliver fast, efficient, and responsive systems architected for large-scale populations of customers, employees, regulators, managers, and more. The careful application of these principles makes it possible for corporations to please customers, support employees, and boost revenues, all at the same time.

There is more to Performance Engineering than just testing. Done right, Performance Engineering means understanding how all the parts of the system fit together, and building in performance from the first design.

Making the journey from performance testing to Performance Engineering isn't easy. But the proven practices established over years of observation can help you on your way.

The Path to Performance Engineering

One of the first tasks that budding programmers are given is to write a program that produces the text "Hello world."

Next you start to play with the program and try to do more, to see how quickly it delivers data or answers queries, and try to optimize for the highest performance with the least amount of code. The requests come in, the responses go out, and you see results on a screen. Take this and add a long-time run script for performance testing, a script you run every time you push out your latest release. It's pretty easy when you're the author and the user.

Performance Engineering, though, is a broad set of processes, and it's also a culture. Performance Engineering is an art based on years of observation that have led to proven practices.

But moving from performance testing to Performance Engineering isn't an easy process. The team must be ready to move from simply running a checkbox performance test script and focusing on parts to studying the way that all parts of the system work together. These pieces encompass hardware, software, configuration, performance, security, usability, business value, and the customer. The process is about collaborating and iterating on the highest-value items, and delivering them quickly, at high quality, so you can exceed the expectations of your end user.

Here's a roadmap for making the trip from performance testing to Performance Engineering. Essentially, these are the steps to become a hero and change agent—and how you can enable your organization to deliver with proven Performance Engineering practices and the accompanying culture.

Define a culture

The success of a team depends heavily on the way leaders are nurturing the professional environment and enabling individuals to collaborate. Building this type of environment will inspire the formation of cross-functional teams and logical thinking.

Build a team

A Performance Engineering team means that technology, business, and user representatives work together. They focus on the performance nature of everything they're working on and figure out together how they can build in these capabilities. They need to know what specific area to focus on first, as well as how to measure along the way. They need to agree on the desired outcome. They must constantly remind themselves that the end goal of adopting Performance Engineering is to benefit the organization and end user.

Choose metrics

We often encourage teams to start with a manual metrics process, perhaps a whiteboard (we know, not really high tech for a technologist) and a few key metrics, then measure them over time and see why they matter (or don't). You'll quickly get a core set of metrics that matter for you and your organization, which have grown out of your cross-functional teams. Your people have the passion and understanding behind these, so trust them. They offer a good way to judge results against the desired outcome.

Once you have figured out enough of this manually, and individuals are starting to adopt and believe in them, take a look at your existing technology capabilities and see how you can get to automated reporting of these results fairly simply. These metrics will be key to your way of measuring what you do and the results you're able to deliver. Make sure you have a solid baseline, and take regular measurements.

Add technology

Performance Engineering requires a new way of thinking, related to your existing software and infrastructure, including the existing tools and capabilities. This is how you shape and form quick, automated results.

Define what your scope of effort is going to be and quickly learn what technology capabilities you already have available to you and your team. This will be an interesting experience, because you'll learn about the capabilities that other siloed teams have available to them. Now, with a shared vision of how you want to deliver Performance Engineering throughout the organization, you can leverage the technology to launch a single approach that aggregates these capabilities.

Perhaps there are a few technology areas you want to start thinking about from the lifecycle virtualization space, such as user virtualization, service virtualization, network virtualization, and data virtualization. These are the core capabilities that will enable your team to accelerate the transformation to Performance Engineering.

Build in telemetry

Now that you've started with culture, team, and technology, it's time to start integrating the telemetry and its data.

For example, how are you capturing the APM (application performance monitoring) data from production, and how about preproduction? Can you begin to examine these results and understand more about the behavior patterns of your users, systems, and transactions? From a cross-functional perspective, this will also pique the interest of the IT operations manager; so you'll continue to broaden your network, and you'll enable them to reduce the number of production incidents. This is just one example.

Think about other quick wins or simple integrations for your existing technology that will enable you to build more bridges. Correlate these types of results across your team so you can promote the culture and desired outcomes of Performance Engineering by building in telemetry.

Look for indirect metrics

There are hundreds of metrics available that you can use to estimate the success of a new capability or feature being released. As systems take on more roles inside a company, metrics that track performance become more readily available, and these enable you to begin partnering with your business peers to find out what metrics they watch and how they get these results.

Start looking at and asking about indirect metrics within the business that would show results related to revenue, customers (attraction and retention), competitive advantage, and brand value. These are important to measure as you make the transition to Performance Engineering.

Focus on stakeholders

Get to know your stakeholders. Who on your team has the most interest in delivering the highest-value items to the end user most quickly and with great results? Find these people and get to know them well. Remember, you're looking for your executive-level sponsors and peer champions, so you can transform the practices and culture of an organization to become a Performance Engineering delivery machine.

Start gathering information and sharing initial prototypes for the type of results, reports, and dashboards you want to show to your stakeholders on a regular basis. Typically, this would be a monthly show-and-tell exercise; however, as it matures it may become a set of automated results delivered with every build, consistently available if stakeholders want to review it. Also, you should consider regular, quarterly presentations to the executive board in which you share last quarter's results, talk about the current quarter, and seek funding for the next one.

Stay focused. Remember your objective. Find your champions. Deliver results.

Create stable environments

One of the earliest challenges will involve enabling teams with the capabilities they require. Some of this will come as you build these teams and the cross-functional tools, capabilities, and associated skills come together. But in the beginning, having a "like production" environment for Performance Engineering is key.

By leveraging the aforementioned lifecycle virtualization—including user virtualization, service virtualization, network virtualization, and data virtualization—you can quickly re-create production environments at a significant fraction of the cost, and you can duplicate them as many times as required. There are several other stable environment proven practices that have emerged along the way, which you can also learn and share through others.

Celebrate wins

Remember the old forming, storming, norming, and performing program developed by Bruce Tuckman? He believed these were the four phases necessary to building teams. If you're a leader or a team member, you'll see this in action.

It's important to remember why you're doing this, and know it's all part of the transformation. Stay focused on the business and end-user objectives, so you can measure your progress and keep your eye on the prize.

Just imagine what it will be like once you have delivered these capabilities to your end user. Conduct proper retrospectives, track your progress with your metrics, and celebrate the wins!

Add gamification

As you mature the capabilities just listed, think about how you can add *gamification* into the results. In other words, how do you make the results you're delivering fun and visual, and how do you make a positive impact on your end users and the organization in the process?

Rajat Paharia created the gamification industry in 2007. In his book *Loyalty 3.0* (McGraw-Hill) Rajat explains, "how to revolutionize customer and employee engagement with Big Data and gamification" and defines these "10 key mechanics of gamification":

1. Fast feedback
2. Transparency
3. Goals
4. Badges
5. Leveling up
6. Onboarding
7. Competition
8. Collaboration
9. Community
10. Points

Of course, you also want to ensure that you highlight the opportunities for improvement and show the wins and losses. You can also gamify Performance Engineering itself at a team level to encourage a little healthy competition within your group, and well beyond, then broadly share the results. This also enables you to leverage these results as information radiators for all stakeholders, showing how teams, systems, and applications are performing against defined baselines and goals.

Start small

When you first begin to incorporate Performance Engineering, you may be tackling a long-neglected maintenance list, or a new, up-and-coming hot project. Either can benefit from the focus of a Performance Engineering culture. Don't try to take on too much at first.

As you begin to elaborate on your requirements, stories, and features, it's important to remember that your whole team is working to define the what, why, and how of each item. As you continue down the Performance Engineering path, you will learn from each other's domain expertise," keeping in mind these learnings and results are from small experiments to show quick incremental value.

Start early

Performance Engineering works best when the team starts thinking about it from the beginning. The earlier the team begins addressing performance in the product lifecycle, the likelier it is that the final system will run quickly, smoothly, and efficiently. But if it can't be done from the very beginning, it's still possible to add the process to the redesign and reengineering work done to develop the next iteration or generation of a product.

About the Authors

Todd DeCapua is the Chief Technology Evangelist with Hewlett Packard Enterprise and cofounder of TechBeacon.com thought leadership site for IT Heros.

DeCapua is a seasoned software professional with 20+ years of experience in IT applications development, IT operations, technology integrations, channels operations, and business development in several domains, including Mobile, Agile, Cloud, and Performance.

Over the years Todd has transformed three organizations to Agile/DevOps, consulted with 100+ organizations worldwide, and amassed a variety of perspectives and practical experiences. He has earned an MBA in Finance and a BS; has been recognized with several industry certifications and awards; and is an industry-renowned leader, speaker, and author.

Shane Evans is an experienced IT Manager with over 12 years in the industry. His primary focus has been Performance Engineering and Performance Management, and he spent 7 years managing these for a major financial institution in Canada before joining Hewlett-Packard in 2009 as a Presales Solution Architect. After three years in the field helping ensure the success of customers across the country, he is now part of the Product Management team. Shane is an active member of the Performance Engineering community, and regularly contributes to the discussions on the HP Forums as well as Google Groups, Yahoo!, and LinkedIn.

Acknowledgments

We recognize Performance Engineering as both an art and a science. Thank you to those with whom we have been able to practice our art, and to those who continue to define the science with us.

This book is dedicated to our families, friends, and colleagues.

Notes

Page Number	Comment/Key Learning/Action

For more information, join us online:

- *http://www.EffectivePerformanceEngineering.com*
- @EffPerfEng on Twitter