CONTINUOUS PERFORMANCE IMPROVEMENT IN AN AGILE WORLD:
A Guide to Implementing Continuous Performance Testing and Monitoring
Best in Class Synthetic Monitoring for APIs, Mobile and Web Applications
LoadComplete is a desktop tool for load, stress, scalability testing of websites and web applications. Designed from the ground-up to help you focus on the tasks at hand, it does not require advanced programming skills.
Contents

Defining Application Performance

Overcoming Performance Testing and Monitoring Roadblocks

An Introduction to the CPI model

Using CPI to Transform User Experience

Selecting the Right Performance Tools
Modern organizations have moved from serial development of monolithic applications — assembling and releasing once or twice a year — to an agile, DevOps, and CI-CD motion, where applications are developed and deployed more often and in smaller blocks.

We’ve all heard about the internal and external advantages of rapid software delivery. However, while pursuing the goal of rapid delivery in all points of the lifecycle, performance checks often take a back seat.

Where do performance testing and monitoring fit into the Agile and DevOps development process?

No matter how seasoned the development team is, less than 30% of software projects using the more traditional software development processes actually end on time.

Rapid delivery processes can certainly help, but only when continuous performance improvement is planned and implemented at each iteration. Implementing a process for improving the performance of your applications requires the right tools to help you do it. These tools go beyond the responsibilities of your development team to ensure that applications are tested in pre-deployment and monitored after your application is live in production.

For some teams, implementing such a strategy is often easier said than done.

Like any other process change, continuous performance improvement can come with challenges.

The common challenges related to budget or bandwidth are often viewed as blockers:

- “We don’t have time allotted for this.”
- “We don’t have infrastructure.”
- “There is no budget to hire skilled people.”

But once internal buy-in is achieved and continuous improvement is made a priority, these can be resolved with simple tactical steps.
This eBook will provide the tactical advice you need to implement a strategy that works for your organization.

We’ll discuss:

- How to define performance
- Overcoming performance testing and monitoring roadblocks
- An introduction to the continuous performance improvement model
- Using continuous performance improvement to transform user experience
- Choosing the right performance tools

We’ll also take a look at how you can test and monitor for performance issues with tools like LoadComplete and AlertSite.
Defining Application Performance

Quality and performance are both terms that can be a bit tricky to define.

How you view performance as an application consumer can be vastly different from how you view performance if you’re responsible for building and maintaining an application. Then there are differences within your own teams — for example, those who work as developers or in operations will each have their own perspective on performance.

Performance is typically defined by a number of key factors:

- Uptime
- Availability
- Functional Correctness
- Stability
- Security
- Reliability

But even these factors need to be examined beyond face value. For example, if you have two applications, both showing 99% availability you may say, “Don’t these both have similar stability?” But that might not always be the case.

For one application, the 1% downtown could be attributed to just one outage. But if the application is running perfectly otherwise, it could be considered to be highly stable. Compare that to another application with the same downtime, but that downtime is spread out over many days, and is constantly up and down. They may have the same availability, but one application is far more stable than the other one.

Another important factor in performance is functional correctness. Even if the application is up and available, if it’s not functionally correct it’s as good as unavailable for the consumer.

For example, let’s say you’re using a Weather app to look at weather in Boston, Massachusetts and it’s showing weather from Mumbai, India. If the application is available but functionally incorrect, it’s as good as unavailable. For a lot of people, availability simply means: is that URL available? Is it up? Can a consumer really interact with that application?
Not unless they log in or put something in the shopping cart. That’s why we look to functional correctness now as another term to help define if the application available. Is it performing? Can it be consumed as it was designed to be consumed?

**Bottom line:** Application performance is not just one thing; it’s a collection of all the different factors that go into making sure your application correctly, regardless of device, location, or number of end users.

**Why do we continue to see performance failures?**

Major retailers like Target, Neiman Marcus, and Best Buy all faced downtime during recent holiday seasons. Social networks like Twitter have also experienced outages, which impact users but also cause issues for applications that rely on a social network’s API. And more recently, the breakout Pokemon Go game experienced multiple outages because its servers weren’t prepared to handle the volume of load that resulted from its explosion in popularity.

All of these companies have ample resources. They have people. They have tools. They have money to ensure that their applications are completely guarded from all the different directions, but then there are still unforeseen issues that happen in production.

Why haven’t we still mastered the art of application performance? Let’s take a look at some of the reasons teams fail to implement some of the key safeguards to protect from performance problems.
Overcoming Performance Testing and Monitoring Roadblocks

We’ve already outlined some of the pain points teams face when implementing a continuous performance improvement strategy. But when you take a closer look at these challenges, there are solutions that many teams often overlook.

By the time any software development project nears completion, it likely will have gone through numerous tests, particularly in an Agile testing environment where testing and development happen concurrently. But no matter how many tests you’ve run, once your application is nearly complete, there’s really only one way to know whether or not your software can handle the actual demands your army of end users will soon be placing on it. This is where load testing comes into play.

Load Testing is a type of performance testing. (Other two types being stress testing and capacity testing). Load testing is used to verify your application’s behavior under normal and peak load conditions. This allows you to verify that your application can meet the desired performance objectives; which are often specified in a service level agreement (SLA). Load testing also enables you to measure response times, throughput rates, resource utilization levels, and to identify your application’s breaking point, assuming that breaking point occurs below the peak load condition. Load testing helps you check your web server’s performance under a massive load, determine its robustness, and estimate its scalability.

But while most teams understand the importance of testing for performance, a large percentage still fail to make performance checks a top priority for their applications. Here are some of the common reasons we hear from SmartBear customers:

1. Lack of Skills

One of the most common challenges teams face when implementing load testing into their software delivery process is understanding the skills needed for effective load testing.

We often hear:

- *We tried a few tools, they require coding skills*
- *We don’t have expertise to analyze the results*
But while there are certainly some level of expertise needed to do effective load testing, today there are tools that can simplify the process and provide the features you need to be successful. A tool like LoadComplete, makes load testing simple and convenient. You can simulate users and create realistic load tests without writing a single line of code. LoadComplete allows you to easily generate traffic using the cloud, virtual machines, or even through on-premise computers.

2. Lack of Time:

You’re already doing a lot of tests to ensure the quality, performance, and functionality of your applications. Do you really need to invest additional time into running performance tests?

Sound familiar? Many teams choose to not make load testing a priority because they’re testing resources are already spread too thin and they don’t think they have the time to do testing effectively.

We often hear:

- We are already doing so many other tests
- Performance responsibility is fragmented among teams

While timing and bandwidth will always be considerations when deciding how to prioritize testing responsibilities, it’s important to understand the cost of poor performance that can result from not doing performance tests. It takes a lot longer to have to identify and resolve performance issues after there’s a problem. This is time you could have saved by doing testing upfront when there’s time to remediate issues.

3. Not a Part of the Process

The biggest hurdles teams face when implementing performance testing is that it’s not part of their established process. We’ll go into more detail on how to implement a process that fits into your organization’s workflow later in the eBook.

We typically hear that:

- There is no performance department
- We don’t have specific target/KPI setup, no performance requirement

One thing to bear in mind is that a lot of times when you find a problem through performance testing, those problems can take a long time to remediate. Some of them are quick fixes. It might be turning on
It might be changing some settings. It might be changing some queries that are reaching out to the database to bring back information.

There are other times when the problem might be that you have to refactor a large portion of code, or that additional hardware is necessary. We all know that acquiring hardware and then configuring it and then putting it into the mix can take a while. It’s better to do these things early, and make changes over time rather than being hit at the last minute with all these changes that you have implement all at once.

4. No Test Environment

There are a lot of companies that do not invest in a staging environment because budget doesn’t exist. For these teams, the cost of ramping up enough load to get a valid test seems to be more than it would be worth.

We often hear:

- *We lack necessary budget to do effective load testing*
- *We don’t have the hardware to generate tests*

Even without a staging environment, you can still run tests on a live environment for your application. To do this effectively, it’s important to run tests on off hours to limit the impact on actual end users. With the right strategy, you can get a lot of information from the actual environment and do it at a time that will impact your customers the least.

Performance monitoring lets you find problems before they impact your end users. If the right techniques are employed, monitoring can provide a view from the same perspective as end users, to tell you what the performance is – and whether customers will be satisfied with the experience.

A robust performance monitoring solution will enable you to set up monitors to watch for performance issues in real time. These tools should provide the scalability you need to monitor different devices, from different locations around the world. It should also provide intelligent reporting, so whether there’s a delay in response time from a third party API, or your site crashes during a user transaction, you’ll be reported so you can act quickly to resolve the problem.

When we talk to customers who are getting started with our synthetic monitoring tool, Alertsite, we often hear that the decision to invest in a monitoring solution came after addressing some common roadblocks.
These include:

1. **Uptime/availability monitoring is sufficient**

Often times people inherent performance monitoring as a responsibility, and feel like they need to learn a legacy monitoring tool that requires coding experience.

As a result, we typically hear:

- *We don’t have the skills or budget for advanced monitoring*
- *Too much confidence in their application*

Modern performance monitoring solutions do not require experience with writing code. In fact, these tools are designed to make it quick and easy to setup monitors on your application with limited setup time or cost. If you’re using a tool like AlertSite for example, you can use our DejaClick solution to perform and record an action on your site and then setup a monitor on the different steps of that action.

If your culture is indifferent to performance or don’t believe that it can go down, you’re not going to invest the tools you need to ensure performance. So start by making performance a top priority.

2. **Our service provider takes care of monitoring**

It’s a common practice to believe that a software vendor, CDN, or another third party that you’re integrated with will take care of performance issues. After all, if they are doing monitoring on their end, why should you be concerned with monitoring the backend of your applications?

We sometimes hear:

- *Blindly trust on third party service providers, such as CDNs or platform provider*
- *No way to measure SLAs*

But if your application depends on third party APIs, then their performance is directly tied to how your users view your application. Even if third party partners may provide regular reports, it’s your responsibility to hold them accountable and make sure they are meeting the SLAs you’ve agreed on. Setting up these monitors will also benefit you when it comes time to negotiate with third parties.
3. Lack of understanding of the need for performance monitoring

In many cases teams rely on customers to report performance problems. If customers are going to notify you when a site is slow or an application crashes, do you really need to invest in a monitoring solution? Similarly, we hear that teams who conduct performance tests don’t believe monitoring is necessary.

- Use customers as the front line monitoring tool
- Rely on performance testing

It’s easy to think that relying solely on the feedback of your users is enough to catch and resolve performance problems. But what if that performance problem is something that requires a major fix and results in a lengthy outage for your application? What if that issue is tied to a transaction within your application that results in a bigger issue than just a slow response from your application?

Consumers expect your applications to be functional and responsive at all times, regardless of their location or the device or operating system they are using. Failing to monitor your application’s performance can put your organization at a major disadvantage.

4. Don’t have a culture of performance

Believe it or not, there are teams where performance isn’t viewed as a necessary area of focus. As discussed above, these teams don’t make performance checks or monitors part of their process and don’t have established metrics they use to measure performance.

Again, we hear:

- Not a part of the process
- We don’t have performance KPIs in place

But while performance itself may not be a major focus for your team, software quality likely is. No one wants to push out a buggy feature or spend hours of their day dealing with issues reported by users.
An Introduction to Continuous Performance Improvement

We’ve seen a number of examples of organizations, across a variety of organizations implement performance strategies within their organizations.

Implementing a performance strategy requires the right tools, but also starts with adopting a culture of performance. Everyone has a role to play in the whole performance picture from a different perspective from different parts of the software development lifecycle, but then it’s a shared responsibility across the board. This is the main reason why it evolved from the Waterfall software development method, and we moved to a more iterative, more agile software development methodology.

Agile in itself is all about continuous testing. Applications are broken down into smaller parts. You do frequent iterations and multiple iterations and integrations, which give the developers, testers, as well as everyone involved in the Agile or DevOps process. It gives multiple chances to validate and review the application performance. It also gives multiple chances of finding and fixing application issues as the come.

Even with the Agile and DevOps, we still see performance issues. There are a number of reasons for this but some of the primary roadblocks include:

- **Just throw a tool at the problem**: Selecting the right tools is important but the team should be aligned about how it fits in with your performance strategy.

- **Pressure to implement a solution**: Implementing a solution is important but you should take the time to select a tool that can be customized to fit your team. Don’t rush into a solution because of internal pressures. Evaluate the tools that are evaluate and take steps to customize the solution as needed.

- **Lack of alignment**: Alignment should go beyond your development, QA, and operations teams. Marketing and business leadership should also understand that performance is a shared responsibility and alignment is critical when scaling your operations.

- **Performance takes a backseat**: Even with the right tools, if performance checks are pushed off until the end of a project, you will continue to push performance to the backseat.
What is continuous performance improvement?

Continuous performance improvement (CPI) normally takes the following structure, which is iterative and fits well with both Agile and DevOps software development methodologies.

1. **Define**
   - Identify SLOs and set SLAs to control the performance driven.
   - Tools, teams, processes to measure the performance.
   - Experts and tools working in conjunction to derive data-driven insights.

2. **Control**
   - Improve on identified hot spots and test the improved app.
   - Uptime, availability, breaking point, robustness, scalability etc.

3. **Analyze**
   - Analyze and improve based on the defined and controlled performance.

4. **Improve**
   - Improve on identified hot spots and test the improved app.
   - Uptime, availability, breaking point, robustness, scalability etc.

This iterative process ensures continuous improvement in performance.
Define
Define what performance means to you. We’ve discussed how that can be subjective, based on the applications you deliver and the audience you serve. Take the time to establish a shared definition of what performance looks like on your team. What are the performance thresholds your application needs to meet to be deemed “high performing?” On the other side, how do you define a performance issue and when should your team be alerted?

Measure
Next step is setting up the right matrix to show and measure performance. Here the goal is to set up realistic metrics to analyze the performance data. After that, once you get the data from all the different directions — it could be from your tools, your applications, or a third party — the next step is to establish some data-driven, actionable insights from it.

Analyze
Most of the time we see organizations spend a lot of time and resources on testing and monitoring tools, and not so much time analyzing the data that they’re generating to improve on performance. It’s all about finding the balance between data gathering and making sense of the data. At this stage the goal is to find tweaks that you can implement in your application and do those changes. Is it just simple things such as changing the settings, adding more cache, or is it adding more hardware? All of these activities you can do at this stage.

Improve
Measurement, analysis, and improvement go hand in hand at this particular stage. We all know that what gets measured, gets improved, and what gets improved sticks with the customers. It’s very important to measure everything from the business side as well as from the technology and application side what matters to your application, and then analyze that data and improve upon it.

Control
Control means identifying the key criteria which control the overall performance of your application. It could be your CDN. It could be a third-party API, or it could just be a plugin you use to add payment processing activity to your card. Identify these control elements and then set up some service level objectives, or SLA, if it’s your third-party provider, and control those. Changing these levers allows you to make sure that performance is stellar across all the different stages. This the overall structure of this continuous performance improvement cycle.
There’s no end to the tweaks and adjustments that you can make to improve performance. You can customize them based on your particular needs, but all of those stages are absolutely important. It’s all about finding the balance.

How do you actually implement a strategy, following these steps? The same cycle could be even further simplified into four building blocks:

- **Analyze back-end metrics to get the whole picture**
- **Analyze front-end metrics to get a realistic understanding of your end user’s experience**
- **Tie performance with revenue-driven or cost-driven transactions**
- **Identify application “hot-spots” and focus on those**
1. **Analyzing the backend matrix** such as your CPU, your server capacity, and everything that goes on in your datacenter or in the cloud, but supports your application.

2. **Analyze the frontend performance matrix** to get a truer understanding of your end users’ experience. Agile and DevOps methodology is all in an attempt to improve your end users’ experience with your application, with your tool, with your service that you’re providing online.

3. **Tie in the application’s performance** with the revenues generating and cost generating transactions. This is where you have to expand your performance team and involve more business owners, maybe even your VP of marketing. Then get their perspective on either defining the revenue generation as well as cost generating transactions.

4. **Identify the hot spots in your application**, and then you improve on those particular parts. You put in some extra effort to make sure that they are performing well when the application is in production.
Using Continuous Performance Improvement to Transform User Experience

Agile is about ensuring that end users have excellent performance all of the time, irrespective of their location, ISP, or device that they’re using.

Implementing a continuous performance improvement strategy enables you to achieve these performance goals by utilizing the right tools to test and monitor your application’s performance. As you implement your strategy, there are a few important considerations you’ll want to keep in mind.

Establish a Global and Local Perspective

Your application doesn’t exist in a silo, and your performance strategy shouldn’t be limited to how you want your application in a controlled situation. Consider how different users will be using your applications across a variety of locations.

Even if the majority of your users are located in a specific region today, it’s likely that you’ll be expanding to different geographies as you scale your application.

Talk to business stakeholders to understand where they want to go with the company and get there ahead of time rather than finding out after the fact the application’s slow in another state or country.

Also keep in mind that your application may already depend on servers that are located across different geographies. If the images that are being served up in your application are hosted in Europe but your users are located in North America and Australia, it’s likely users in different geographies will have different experiences with your application. For example, if users in the US are experiencing slower load times, you may determine that images need to be hosted locally rather than on an external server. Without testing based on specific geographies, you’ll never understand the global experiences people are having with your application.

Don’t Forget About Internal Applications

Organizations run on internal applications, whether that’s a CRM solution, ERP solution, or intranet solution. For these types of internal application, it’s absolutely important to ensure the performance internally, from within your network as well.
Unfortunately, in a lot of cases, it will be just the one copy of the application that resides in one geography, and a lot of the other satellite offices will complain about how long it takes to bring up the application and use it, whereas the customers in the home office, are getting it very quickly and don’t understand what everybody’s complaining about.

Most APIs today are internal facing only. Companies rely on those APIs to be successful. Unfortunately, since they’re not front and center they often get forgotten. We’re seeing more and more customers not only monitoring the application, but also the APIs carrying that application.

A synthetic monitoring tool, like AlertSite, gives you visibility to those internal facing APIs. Are they available, performing, and functionally correct?

**Don’t rely solely on end users**

Instead of relying on end users to report performance problems, you can use virtual and synthetic users to test and monitor your application’s performance.

Virtual and synthetic users can help:

- Test and monitor diverse use cases
- Test and monitor applications on real browsers and real devices, without actually impacting end users

In load testing, use virtual users to represent actual users. Think about all the different ways real users use and interact with your application. The key is to be able to model what the various use cases are.

You’re going to have people that are running reports almost exclusively. You’re going to have people that are coming in and making a purchase and actually checking out. You’re going to have others that come in to look at your site and decide to go somewhere else.

For e-commerce companies for example, 95% of customers come in, take a look, and leave, and only 5% will make a purchase. You want to be able to represent those various types of users by creating virtual users in your scripts and scenarios to represents those types of users. Then mix them up in such a way that the model, the actual traffic, to your site.
It wouldn’t make sense, for example, to have a 100% of your virtual users going to checkout. It makes a lot more sense to have 5% go to checkout and have the other 95% just do a search, or maybe put something in the cart, and then abandon it, which is a lot more realistic. In that way, you’re getting a clearer picture of what’s going to happen when those members or users are on your site.

When the functional tester hands off the application, you know at that it’s working, but we monitor it to understand if something’s working at a later date. A later date is five minutes from now, 10 minutes from now. It’s a random point in the future. *Always be monitoring.*
Selecting the Right Performance Tools

Implementing the strategies outlined above, means selecting the right tools for your organization. As you begin your search for a solution, there are a few things you’ll want to keep in mind:

**Don’t sacrifice sophistication for the sake of simplicity:**
There are so many tools in the market which are very simple but that comes with a price. These tools are not sophisticated enough to get those data driven insights and make changes into the application. Having the simple tool is always good, but don’t sacrifice the sophistication for the sake of simplicity.

**Allows you to test/monitor where users are:**
Your users are located across the globe and are using your application with so many types of use cases. Sometimes they’re internal. Most of the time they’re external. It’s absolutely essential for you to test or monitor your application from where your users are.

**Ability to simulate realistic use cases:**
Users will browse a bunch of different pages before going on to your revenue generating page, or they will drop off and log in again, and do their activity. It’s absolutely essential that the tool lets you design all of these actual end user transactions and help you set test cases or monitors based on these transactions.

**Data and reports that could be shared across the whole performance team:**
This helps you get the actionable insights to get beyond just monitoring and gathering data. You have to complete it with actually analyzing the data and improving your application.
Best in Class Synthetic Monitoring for APIs, Mobile and Web Applications

AlertSite

TRY IT FOR FREE
LoadComplete allows you to generate load from virtual machines, on-premise computers, or even the cloud. Customized reports in LoadComplete help you get end-to-end visibility and find problems at both client and server side.

TRY IT FOR FREE
Over 4 million software professionals and 25,000 organizations across 194 countries use SmartBear tool.

- 4M+ users
- 25K+ organizations
- 194 countries

See Some Successful Customers >>

**API READINESS**
- Functional testing through performance monitoring

**TESTING**
- Functional testing, performance testing and test management

**PERFORMANCE MONITORING**
- Synthetic monitoring for API, web, mobile, SaaS, and Infrastructure

**CODE COLLABORATION**
- Peer code and documentation review

See API READINESS Products
See Testing Products
See Monitoring Products
See Collaboration Products