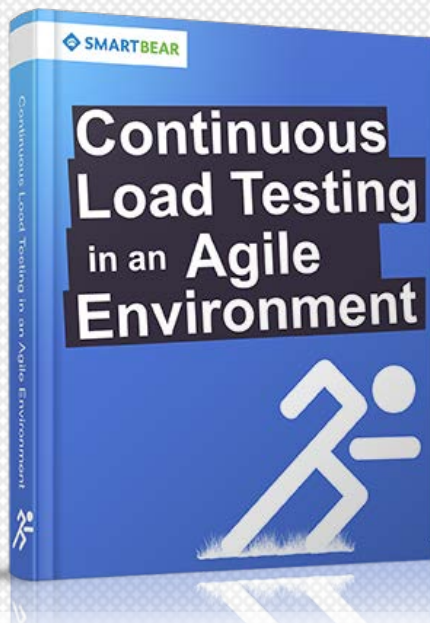




LoadComplete

by SMARTBEAR



Consider how much time you spend finding and fixing problems in production. Even with the most comprehensive functional testing process, you still discover elusive bugs under simultaneous load of thousands of users after going live. Early testing and rapid feedback, as well as effective communication, are cornerstones of the Agile approach to software development. Best practices and techniques for early testing can also be applied to non Agile projects, so even if you haven't jumped on the Agile bandwagon, you and your team can benefit from this ebook.

Four Best Practices for Agile Load Testing	4
Test incrementally	4
Establish performance goals	5
Reuse functional tests	5
Include performance tests in the build process	5
Considerations / Obstacles on the path to Agile load testing	6
Is your Test (Staging) Environment an Accurate Representation of Production?	6
Are you testing with “real” data, or will you create dummy data for your testing?	7
Who needs to be involved during the testing and in what capacity?	8
What criteria will be used to determine success of the test?	8
Typical Pitfalls	9
Overestimating the number of virtual users	9
Agile Load Testing Solutions	10
Define your workload	11
Plan: Start Small	12
Plan: Small does not refer only to virtual users	12
Plan: Inside or outside the firewall	12
Plan: Some Examples	14
Flexibility	14
Data preparation	14
Test	15
Test: The Importance of Parameterization	15
Analyze	16
Analyze generated test reports	16
Approach analysis from a multi-disciplinary perspective	16
Determine bottlenecks or areas of concern	16
Networks	17
Third-party calls	18
Fix	18
Definitions of Terms Used in This E-book	19
Load Testing	19
Continuous Testing	19
Agile Development Environment	19
About SmartBear’s LoadComplete	20
Affordable Cloud Load Testing Tool	20
Load Testing Rich Internet Apps	20
Creating Realistic Load Tests	20
Running Distributed Load Tests	20
Server Performance Monitoring with LoadComplete	20

In many applications, performance is a critical factor for success. This is certainly the case with pretty much any serious Web or Cloud application. The need to accelerate delivery and improve user experience also leads many development teams to embrace Agile as the development process of choice.

While performance is known to be crucial for application success, what we often see in the industry is that in absence of additional resources during development, performance testing gets pushed aside until the very end of the project. Historically, and certainly in waterfall types of setups, performance would be scrutinized at the end, with the expectation that only minor tweaks and tune-ups will be needed to meet the performance requirements. Last minute changes, bugs that prevent test execution, infrastructure limitations, or lack of dedicated testing resources all contribute to these flawed practices.

In the meantime, application delivery took a quantum leap towards flexible, redundant production systems – often hosted in the Cloud with full support of virtualization. This additional flexibility comes with added complexity, and performance issues detected late are likely not to be small tune-ups, but may lead to significant contract re-negotiations, if not to re-architecture of the application.

This Ebook Provides:

- ◆ The intersection of fast agile sprints and performance testing
- ◆ The adoption of effective and valuable performance testing
- ◆ An overview of a performance solution that can help you improve your agile development process

[Click here for more in-depth definitions](#)

Four Best Practices for Agile Load Testing

One of the main goals of Agile is to allow you to effectively manage change in both external and software functionality, and to accelerate the development of quality software. Part of the challenge with managing change is providing quick feedback on when the change in software happens, and the impact of the change.

One of the important organizational points of Agile is the concept of a team producing a usable version. This approach should put Dev, QA, and tech writers on the same page and in the case of performance testing it should designate a resource who will be responsible for the testing. The other aspect is that all stakeholders, including those on the business side need to be tightly involved.

The concept of performance testing in Agile is different from “test early and often” because that concept still refers to waterfall. In the waterfall terminology “early” means “before the software is transitioned to QA”. With Agile, you test all the time. Functional testers define functional tests to match the sprint functionality. Functional testing is an opportunity to introduce performance testing scenarios. Here are four handy suggestions for Agile Load Testing:

Test incrementally

Incremental testing starts at the lowest functional levels. Test a small function, and then build from there. If all of the individual components function and perform correctly, chances are they will work well together and this knowledge will help you analyze and debug performance bottlenecks when you test on the system level later on.

Look at performance testing in agile as testing of different performance layers that can be approached in an onion-like fashion in the rhythm of the sprints. In order to achieve flexibility you also need to plan your performance testing efforts and break a bigger challenge into smaller tasks. As a comparison, it is difficult and expensive to solve a 12 man-year problem at once, but

one can solve 12 one man-year problems in sequence. The same approach can be applied to signing off the performance of an application.

Later in the document you can learn some concrete methods to plan for incremental performance testing.

Establish performance goals

Create performance tests to work alongside your functional tests, and you also want to incorporate the performance test into your build process. Start by establishing performance goals when you write your user stories.

For example, let's say you're creating an online store. You want a user be able to add an item to their cart, and want this process to take no more than half a second. The first thing you should do is create your performance test alongside your functional test to allow you to measure performance and continuously test it.

Incorporate the performance test into your build process

Reuse functional tests

Always check if you can reuse existing or future functional tests as performance tests. It may be possible to incorporate timers into an automated test to give you a feel for how the application is responding. Review your functional tests with a designated performance expert and make the decision.

This can be one of the key anchor points for Agile performance testing. This also drives development of work scenarios. These can later be used as building blocks for the final load tests.

Include performance tests in the build process

Finally, you should include your performance tests in the build process so that each build goes through some level of performance testing. By doing this, it's going to be easier to determine what caused the slow down in your application's performance. Also, if you need to roll back to an earlier build,

you know how well that particular build performs.

Considerations / Obstacles on the path to Agile load testing

Agile performance testing, is a strong value proposition but, there are some significant obstacles on the way. Here are a couple of suggestions to keep in mind when considering agile performance testing.

Is your Test (Staging) Environment an Accurate Representation of Production?

There are some things that are hard to do until later in the delivery cycle. Stress and high-capacity load testing is an example. If you are performing a test at the very high end where you're finding out how many users can use the application, how quickly they can ramp up on the system and how many errors are generated when you are at your peak, that is something that you want to save until the end.

The staging environment may not support that level of capacity. And you will need to take that into consideration when you are performing your tests. You may need to perform some of these tests in your production environment.

One of the things that is important to consider early on is whether it is feasible to test the functionality of all or some of the application components before you perform an integrated system test.

Often, your staging environment and production environment differ significantly in the amount of resources that are they are allocated. This will force you to perform those tests in production environment. Also, if you're performing tests in production, you may need to test from outside your firewall,

coming in from the cloud. To allow for higher velocity of your performance testing efforts identify tests that can be done inside your firewall in order to eliminate all of the other noise that might come into play during your test.

If you do not have the available resources to make your staging environment an exact replica of Production, at a minimum, have staging represent a scaled-down version of Production.

Are you testing with “real” data, or will you create dummy data for your testing?

Another important question to consider early on in order to correctly scope the effort is: Are you going to use real data? Are you going to create dummy data and if yes, do you need to create it? In a lot of environments out in the industry, especially in the medical world, using live user data for testing may be difficult. This factor may also determine how data can be accessed and if you can generate load from outside of the corporate network.

If you do have access to real data, don't pass this opportunity. After tests are done, make sure you can set your database back to its pre-test condition. Since your database is going to cache a lot of information you want to use data that's as realistic as possible.

Furthermore, you need to consider altering data between test cases. If you run a load test that is logging-in as the same user every time and it's filling in a form field, a set of form fields the same way every time and doing the same inserts into a database, you're not really testing your system at all. There may be some data that will not be caught or will cause an error. And you might not catch that if you just use homogenized data.

One of the situations you may need to work around is when you have a test that's going to run for a long period of time and it also uses large numbers of virtual users and your data set is limited. The best scenario, of course, would be to have a very large set of data and not have anything ever be repeated.

But depending on the test that you're running, maybe having data you need to each virtual user and then have it repeated at some point might be a possibility. Especially if you're using live data, you may not have the ability to scale it up as large as you might like.

Who needs to be involved during the testing and in what capacity?

Clearly, if you have a dedicated performance engineer, then this resource should definitely be involved from the beginning and be an integral part of the agile team. When tests are run, the performance testing results should be available to all of your team members – the database administrator (DBA), the network specialist, the developers – they should all have appropriate access to the latest results. Different people on the team may provide different perspective on the observed performance problems or any errors that come up.

A DBA is always going to see a performance problem as being related to the database. A network person might tend to think that it's bandwidth that's the problem. And your developers might think that it was the way that something was structured in the code and that something might need to be refactored. This becomes even more important when you run your load tests at the end of the whole application, because sometimes it takes a number of different sets of eyes to provide the clarity that you need to find the actual problems.

What criteria will be used to determine success of the test?

From the very beginning, when you want to run a load test, you need to determine what your criteria for success are going to be. Regardless of whether you are looking for an absolute response time, or a relative change from a previous test you need to define the criteria and have a clear expectation before you start testing.

Typical Pitfalls

Overestimating the number of virtual users

One thing that you can do, if you're running your larger tests, look at your analytics and determine historical performance. If it's a completely new application, then the rule of thumb should be to start small and build up from there.

“I can think of a number of load tests that have fallen apart pretty quickly under load. And if some smaller tests had been done initially on various portions of the application, these bottlenecks would have been known and corrected long before the large test.”

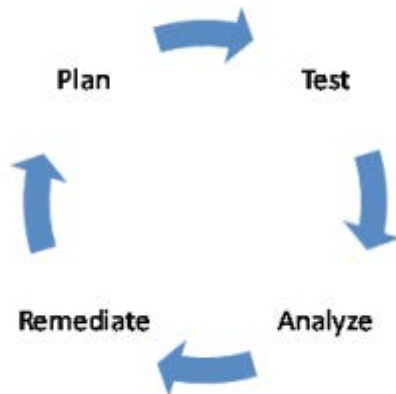
– Mike Punsky

If you have tools that can generate load of the large number of users like SmartBear's LoadComplete it may be tempting to start with a large number of users, but it is important to remember that you can get a lot of good results at very low levels of load. By starting small, you're going to be able to correct a lot of problems before you do your heavy testing. And this is going to ensure better results when you do your heavy testing.

Not following this best practice and falling into the trap of overestimating the number of virtual users early on will likely force you to repeat that large test numerous times in order to get the results you're looking for.

Plan

Performance Testing is an Iterative Process



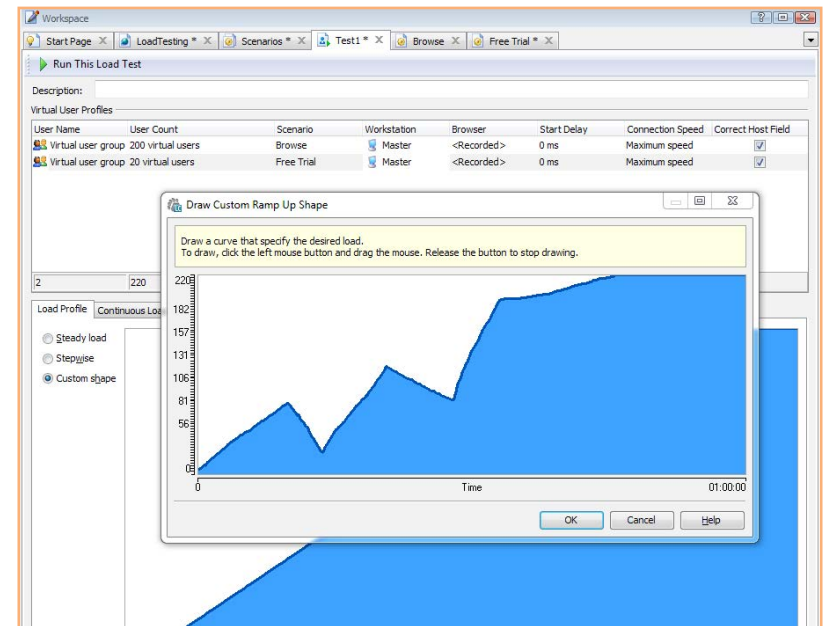
- ◆ Put performance tasks on a SCRUM board
- ◆ Make fixing functional bugs that block performance test a high priority
- ◆ Do not delay fixing performance issues identified by tests
- ◆ Retest after fixes are applied to know if they worked
- ◆ You cannot guess where the next bottleneck is
- ◆ Don't pre-optimize before testing
- ◆ Understand the production environment
- ◆ Analyze logs

- ◆ Know when performance is good enough
- ◆ Apply low level load to the system under test during manual testing
- ◆ Use load test scripts to find functional and stability bugs, memory leaks, reproduce intermittent problems

Define your workload

Start out doing small, incremental tests. Later, combine these into a more realistic whole. For example, we have a test set up for an e-commerce site. And here, you'll see that this is coming in from the cloud. And also, it has two scenarios running at the same time.

One is a group of users that is only browsing your site. The other group of users are actually purchasing something and checking out. This is a great way to have a realistic load test of your system. However, this is something to save for the very end when you are hitting production.



Initially, consider looking at one virtual user executing one scenario, running from your local workstation so that you're able to get your numbers. This way, you're able to tweak and tune. Additionally you know that your component is working to its best capabilities, but not putting a lot of mud into your test.

You can get many good results at very low levels of load. By starting small, you can correct a lot of problems before you do your heavy testing, ensuring better results.

Plan: Start Small

There is a tendency to start out throwing a large number of virtual users against the site being tested. Since actionable results can be obtained with a moderate number of VU's, it makes sense to begin with running many moderate tests.

Plan: Small does not refer only to virtual users

In addition to moderating the number of virtual users, it also helps to run single scenarios (one script as opposed to multiple scripts) during the iterative testing.

From there, build your way up to multiple scenarios. The smaller your test, the greater your insight into where your errors and bottlenecks are coming from.

Plan: Inside or outside the firewall

Some load tests are generated inside the firewall with software, and others are generated outside the firewall by a service. It seems that most people champion one over the other. There are valid reasons for both types of testing. Internally-run software is best suited for iterative testing.

Load origin is key here. There's been a lot of people band-standing about having load come from the cloud versus having load come from real browsers. Either way, there is a need and a use case for all types of load testing.

For small, iterative load testing, the best tool to use is software that you can run inside your firewall. You're paying one licensing fee, and you can run as many tests as you want for no additional charge.

Run as many tests as possible to get the best results possible. Tweak and tune at every step. By running the tests from within the firewall, you won't

get any of the noise or X-factors of additional users coming in from the outside or any of the other pieces such as your providers and external influences.

Once your application is tuned as well as it can be, run something from outside the firewall, generally from the cloud. You can have users coming from different cloud

regions. You can scale really high. And, you could find out how your application performs under real-world conditions.

Test in real browsers as well as staged tests to get very accurate results. How does the page appear? How is the JavaScript loading? How is the CSS looking under this load condition?

The key here is to use the right tool for the task at hand. If you're doing unit testing, application server tuning, smoke testing, and bottleneck analysis, these are all good candidates for inside the firewall.

You can do some bottleneck analysis from outside the firewall, as well as heavy load testing, stress testing, and capacity testing all from outside the firewall.


Run as many tests as possible to get the best results possible. Tweak and tune at every step.

Test in real browsers as well as staged tests to get very accurate results. How does the page appear? How is the JavaScript loading? How is the CSS looking under this load condition?

Plan: Some Examples



Task	Inside The Firewall	Outside The Firewall
Unit Testing	✓	
Application Server Tuning	✓	
Smoke Testing	✓	
Performance Analysis	✓	✓
Load Testing		✓
Stress Testing		✓
Capacity Testing		✓
Environment Tested	Staging	Production



Flexibility

To be flexible is to rapidly respond to change. In an Agile environment, things can change quickly as new requirements come in or are modified. As this happens, performance can be left by the wayside. However, if you're testing continuously, you will be more likely to catch problems sooner and make the needed corrections.

Data preparation

It's probably easier to prepare relevant data gradually as the functional scenarios become available. Smaller scale, single-user performance tests are

key. These are not scaled up, but provide some metrics that are. If you test at the smallest increments and you use a small amount of load, you still get valuable information as you perform your iterations.

Let's say you have five users, and your performance is half a second. If the same five users produce a response that is double that after some changes, you've got a problem. With Agile testing however, you can always see the delta between where you were and where you're going to be.

If the success criteria for a sprint includes delivery of a working software version, performance and capacity should be a sign-off metric. If there's bootstrapping of functionality in multiple sprints, then you need to, at least, look at these items as challenges. The main goal of Agile was to set up an approach to more effectively manage change in both external and software functionality.

Let's say you have five users, and your performance is half a second. If the same five users produce a response that is double that after some changes, you've got a problem.

Test

It is important to have key personnel present during testing (whether in-person, remote or on-call). Design and follow a process when performing iterative tests, and compare results of various iterations.

Test: The Importance of Parameterization

When defining workload for a test, the importance of using a broad spectrum of data cannot be overstated. Many factors can only be properly tested by using either actual data, or at least a very good representation of it.

Another factor here is parameterization. Many factors can only be properly tested if you're using actual or, at least, realistic data. And some examples

of things that you may want to parameterize are user IDs and passwords, different search terms.

It's possible certain search terms will take longer to return than others. You want to test a wide variety of those search terms and in different quantities. By testing this way, you may uncover some bugs in the application as well. Additionally, any time a form is going to be inserting data, test varied information on the form to maximize your results.

Examples of Things to Parameterize

- ◆ Users
- ◆ Search terms
- ◆ Quantities
- ◆ Form values

Analyze

Analyze generated test reports

A particular art in testing is the ability to analyze your generated test reports. You want to be able to look at your test reports, understand everything being presented. Come at it from different angles as well, and have the business owners look at your reports. Also, always be asking network people, DBAs, developers, and anyone else involved to provide their own perspectives.

Approach analysis from a multi-disciplinary perspective

- ◆ Developer
- ◆ DBA
- ◆ Network Admin
- ◆ Business Owner

Determine bottlenecks or areas of concern

Missing or non-optimal indexes can really slow down any application. With non-optimal code loops, it's very easy for the developers to write code that does things in a brute force sort of way. The paradigm that's being used for

a particular loop or a section of code might not be as optimized as possible. Finding these things and correcting them can help to reduce bottlenecks.

Another thing that is very key in your applications is caching. There are so many different layers of caching that can be involved in your application. Part of your application can be 100 times faster if it's cached than if it's not

cached, depending on what it is and what level of caching is implemented.

You can cache pages of your site. You can cache queries. You can cache your results to RAM or to disk.

And then there's the question of what's the duration that you're going to be caching for. Is this information

that can be cached for a day? Is it information that can be cached for five minutes?

There are cases where you need pages and queries to be as current as possible. However, if there is a two-minute delay where it can be cached, you could actually gain a lot of performance.

Hardware is usually seen as one of the first things people want to look at when it comes time for performance. Ease your mind – it's usually not the hardware. If you get everything else working fine, your current hardware might be completely adequate for the task. Sometimes, hardware can be the problem. Testing frequently and analyzing results can help establish where problems lie.

Networks

If while running a test from outside your firewall, you notice load is building as you add virtual users, and see it flat line, then chances are you've have

“Part of your application can be 100 times faster if it's cached than if it's not cached, depending on what it is and what level of caching is implemented.”

something limiting your amount of bandwidth. This could be a piece of hardware acting up, or your provider. Again, testing frequently is the best way to catch these issues.

Third-party calls

These days, third-party calls and content are all over people's web pages. Whether it's social media, advertising, or analytics, your company's personal content might be 20 % of your home page. 80 % of that could likely be coming from other sources.

Unfortunately for you, a customer coming to your web site thinks of everything that's on your site as being you. So if one of your third-parties is not performing well, it reflects directly on you.

Fix

When remediating these issues, you'll be adding or rebuilding your indexes, and adding or optimizing your cache. You'll be upgrading hardware, adding new hardware, optimizing network components and settings. You'll work with your third-parties to optimize any of those calls. You're going to need to optimize code loops, refactor code.

“By utilizing continuous testing, there will be fewer costly surprises at inopportune times.”

All of these things take time to remediate. To cut down on that time, test constantly and provide your corrections diligently. The sooner you begin your remediation, the better. By utilizing continuous testing, there will be fewer costly surprises at inopportune times.

One thing that you see all the time now is a development group that puts together an application, and releases it without load testing. Much to their dismay, they find out the amount of traffic on that site is going to totally take it down once it's up. Include performance testing continuously and there will be no surprises at the end.

Definitions of Terms Used in This E-book

Load Testing

Load testing is the process of putting demand on a system and seeing the results. With load testing, you want to be able to measure the response of your system, how that changes over the course of time.

Load testing is performed to determine your system's behavior. And this can be under both normal and anticipated peak load conditions. And as you'll see later on, we're not always talking about a very heavy load. You can get a lot of results at lower levels of system load.

Continuous Testing

Often closely related to the Agile development environment, the goal of continuous testing is to make testing a regular part of your development process regardless of whether it is an Agile process, or not. As you may expect, you need to test continuously throughout the entire development process, and performance testing should be plugged in from the beginning.

Agile Development Environment

Performance testing and Agile fit well together. Agile development environment employs iterative and incremental development. It also includes iterative testing, incremental testing as well as rapid and flexible response to change. These are important characteristics of Agile, and are also very important in load testing.

When performing load testing, do it in an iterative way. Therefore you test, find out where you're at, make improvements, test again, and then you set a new benchmark. It's a never-ending process to ensure you don't go backwards on performance.

About SmartBear's LoadComplete

Affordable Cloud Load Testing Tool

Create load tests with pre-configured SmartBear machine images available in the Amazon cloud. Let LoadComplete launch, start and stop cloud computers for you in order to minimize cost of cloud services.

Load Testing Rich Internet Apps

Need RIA load testing? We've got you covered. LoadComplete can test many different types of applications including Rich Internet Application (RIA) technologies like ASP.NET, Ajax, Flex and Silverlight applications.

Creating Realistic Load Tests

For realistic load testing, you need a program that can simulate users and activities as close to real life conditions as possible. We've got that. LoadComplete provides various techniques that will simulate unique users, perform tests with different workload levels and get session handling support.

Running Distributed Load Tests

Share the load among several workstations to create tests simulating a very high workload. The Remote Agent software is included in each LoadComplete license, so you can distribute virtual users among multiple computers on your network.

Server Performance Monitoring with LoadComplete

Get real-time server performance monitoring, so you can track the behavior of the tested application and quickly identify failures and bottlenecks that affect functionality.



About SmartBear Software

More than one million developers, testers and operations professionals use SmartBear tools to ensure the quality and performance of their APIs, desktop, mobile, Web and cloud-based applications. SmartBear products are easy to use and deploy, are affordable and available for trial at the website. Learn more about SmartBear, the company's award-winning tools or join the active user community at <http://www.smartbear.com>, on [Facebook](#) or follow us on Twitter [@smartbear](#) and [Google+](#).