Beyond Performance Testing

by:

R. Scott Barber

BPT Part 3: How Fast is Fast Enough?

"You thought that was fast? I thought it was fast. Well, was it?"

– Jodie Foster as Annabelle in the movie Maverick (1994)

As a moderator of performance-related forums on QAForums.com, I've seen questions like this one posed numerous times:

"I'm desperately trying to find out what the industry standard response times are. What are reasonable benchmarks that are acceptable for Web sites at the moment? Is 1.5 seconds a reasonable target????"

My answer to questions like this one always starts with "It depends on . . . " My friend Joe Strazzere addressed this question particularly well, as follows:

"There are no industry standards.

You must analyze your site in terms of who the customers are, what their needs are, where they are located, what their equipment and connection speed might be, etc., etc.

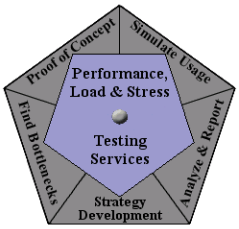
I suspect 1.5 seconds would be a rather short interval for many situations. Do you really require that quick of a response?"

The bottom line is that what seems fast is different in different situations. So how do you determine how fast is fast enough for your application, and how do you convert that information into explicit, testable requirements? Those are the topics this article addresses.

This is the third article in the "Beyond Performance Testing" series. Here's what the series has covered so far:

Part 1: Introduction
Part 2: A Performance Engineering Strategy

This article is intended for all levels of performance testers/engineers and will be particularly useful to managers and business analysts involved in determining the performance requirements of a system. It expands on concepts mentioned in "User Experience, Not Metrics, Part 5: Using Timers," so you should have read that article before tackling this one.



Considerations Affecting Performance Expectations

Let's start by discussing the leading factors that contribute to what we think fast is. I believe these considerations can be divided into three broad categories:

- user psychology
- system considerations
- usage considerations

None of these categories is any more or less important than the others. What's critical is to balance these considerations, which we'll explore individually here, when determining performance requirements.

User Psychology

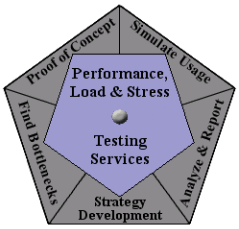
Of the three categories, user psychology is the one most often overlooked – or maybe a better way to say this is that user psychology is often overridden by system and usage considerations. I submit that this is a mistake. User psychology plays an important role in perceived performance, which, as we discussed in detail throughout the "User Experience, Not Metrics" series, is the most critical part of evaluating performance.

Consider this example. I recently filled out my tax return online. It's a pretty simple process: you navigate through a Web application that asks you questions to determine which pages are presented for you to enter data into. As I made a preliminary pass through my return, I was happy with the performance (response time) of the application. When I later went back to complete my return, I timed the page loads (because I almost always think about performance when I use the Internet). Most of the pages returned in less than 5 seconds, but some of the section summary pages took almost a minute!

Why didn't I notice the first time through that some pages were this slow? Why didn't I get frustrated with this seemingly poor performance? I usually notice performance as being poor at between 5 and 8 seconds, and at about 15 seconds I'll abandon a site or at least get frustrated. There's no science behind those numbers; they're just my personal tolerance levels. So what made me wait a minute for some pages without even noticing that it was slow? The answer is that when I requested a section summary page, an intermediate page came up that said:

"The information you have requested is being processed. This may take several minutes depending on the information you have provided. Please be patient."

When I received that message, I went on to do something else for a minute. I went to get a drink, or checked one of my e-mail accounts, or any of a million other things, and when I came back the page was there waiting for me. I was satisfied. If that message hadn't been presented and I had found myself just sitting and waiting for the page to display, I would have become annoyed and eventually assumed that my request wasn't submitted properly, that the server had gone down, or maybe that my Internet connection had been dropped.



So, getting back to the initial question of how fast is fast enough, from a user psychology perspective the answer is still "it depends." It depends on several key factors that determine what is and isn't acceptable performance.

The first factor is the response time that users have become accustomed to based on previous experience. This is most directly tied to the speed of their Internet connection. My mother, for example, has never experienced the Internet over anything other than a fuzzy phone line with a 56.6-kilobytes-per-second modem. I'm used to surfing via high-speed connections, so when I sign on at my mother's house I'm frustrated. My mother thinks I'm silly: "Scott, I think you're spoiled. That's as fast as we can get here, and a lot faster than we used to get! You never *were* very patient!" She's right – I'm not very patient, so I have a low tolerance for poor Web site performance, whereas she has a high tolerance.

Another factor is activity type. All users understand that it takes time to download an MP3 or MPEG video, and therefore have more tolerance if they're aware that that's the activity they're performing. However, if users don't know that they're performing an activity like downloading a file and are just waiting for the next page to load, they're likely to become frustrated before they realize that the performance is actually acceptable for the activity they're performing.

This leads us to the factor of how user expectations have been set. If users know what to expect, as they do with the tax preparation system I use, they're likely to be more tolerant of response times they might otherwise think of as slow. If you tell users that the system will be fast and then it isn't, they won't be happy. If you show users how fast it will be and then follow through with that level of performance, they'll generally be pretty happy.

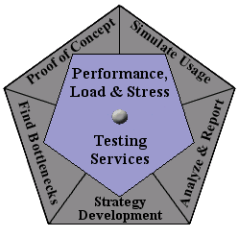
The last factor we should discuss here is what I call surfing intent. When users want to accomplish a specific task, they have less tolerance for poor performance than when they're casually looking for information or doing research. For example, when I log on to the site I use to pay bills, I expect good performance. When I'm taking a break from work and searching for the newest technology gadgets, I have a lot of tolerance for poor performance.

So with all of these variables you can see why, as Joe Strazzere said, "There are no industry standards." But if there are no industry standards, how do we know where to start or what to compare against? I'll describe some rules of thumb later, when we get to the topic of collecting information about performance requirements.

System Considerations

System considerations are more commonly thought about than user psychology when we're determining how fast is fast enough. Stakeholders need to decide what kind of performance the system can handle within the given parameters. "Fast-enough" decisions are often based purely on the cost to achieve performance. While cost and feasibility are important, if they're considered in a vacuum, you'll be doomed to fielding a system with poor performance.

Performance costs. The cost difference between building a system with "typical" performance and building a system with "fast" performance is sometimes prohibitive. Only by balancing the need for performance against the cost can stakeholders decide how much time and/or money they're willing to invest to improve performance.



System considerations include the following:

- system hardware
- network and/or Internet bandwidth of the system
- geographical replication
- software architecture

Entire books are dedicated to each of these considerations and many more. This is a well-documented and well-understood aspect of performance engineering, so I won't spend more time on it here.

Usage Considerations

Usage considerations are related to but separate from user psychology. The usage considerations I'm referring to have to do with the way the Web site or Web application will be used. For example, is the application a shopping application? An interactive financial planning application? A site containing current news? An internal human resources data entry application? "Fast" means something different for each of these different applications. An application that's used primarily by employees to enter large volumes of data needs to be faster for users to be satisfied than a Web shopping site. A news site can be fairly slow, as long as the text appears before the graphics. Interactive sites need to be faster than mostly static sites. Sites that people use for work-related activities need to be faster than sites that are used primarily for recreational purposes.

These considerations are very specific to the site and the organization. There really isn't a lot of documentation available about these types of considerations because they're so individual, depending on the specific application and associated user base. What's important is to think about how your site will be used and to determine the performance tolerance of expected users as compared to overall user psychology and system considerations. I'll say more about this in the next section.

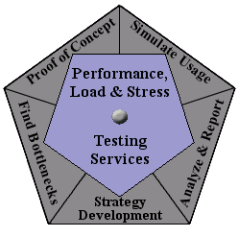
Collecting Information About Performance Requirements

So how do you translate the considerations described above into performance requirements? My approach is to first come up with descriptions of explicit and implied performance requirements in these three areas:

- user expectations
- resource limitations
- stakeholder expectations

In general, user and stakeholder expectations are complementary and don't require balancing between the two. For this reason, I start by determining these requirements. Once I've done this, I try to balance those with the system/financial resources that are available. Truth be told, I generally don't get to do the balancing. I usually collect the data and identify the conflicts so that stakeholders can make decisions about how to balance expectations and resources to determine actual requirements.

Determining the actual requirements in the areas of speed, scalability, and stability and consolidating these into composite requirements is the final step. I'll describe that process in detail later on, but first



let's look at each of the three areas where you'll be collecting information about requirements.

User Expectations

A user's expectations when it comes to performance are all about end-to-end response time, as we touched on earlier in our look at user psychology. Individual users don't know or care how many users can be on the system at a time, how the system is designed to recover in case of disaster, or what the cost of building and maintaining the system has been.

When a new system is replacing an old one, it's critical from the user's perspective for the requirements of the new system to be at least as stringent as the actual performance of the existing system. Users won't be pleased with a new system if their perception is that its performance is worse than the system it's replacing – regardless of whether the previous system was a Web-based application, client/server, or some other configuration.

Aside from this situation, there's no way to predict user expectations. Only users can tell you what they expect, so be sure you take the time to poll users and find out what their expectations are before the requirements are set. Talk to users and observe them using a similar type of system, maybe even a prototype of the system to be built. Remember that most users don't think in terms of seconds, so to quantify their expectations you'll have to find a way to observe what they think is fast, typical, or slow.

During my tenure as a performance engineer, I've done a lot of research in the area of user expectations. I believed at first in the "8-second rule" that became popular in the mid-1990s, simply stating that most Web surfers consider 8 seconds to be a reasonable download time for a page. But since then I've found no reliable research backing this rule of thumb, nor have I found any correlation between this rule of thumb and actual user psychology. I'm going to share with you what I *have* found, not to suggest these findings as standards but to give you a reasonable place to start as you poll your own users.

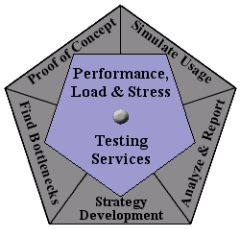
I've found that most users have the following expectations for normal page loads when surfing on high-speed connections:

- no delay or fast – under 3 seconds
- typical – 3 to 5 seconds
- slow – 5 to 8 seconds
- frustrating – 8 to 15 seconds
- unacceptable – more than 15 seconds

In my experience, if your site is highly interactive or primarily used for data entry you should probably strive for page load speeds about 20% faster than those listed. For mostly static or recreational sites, performance that's about 25% slower than the response times listed may still be acceptable.

For any kind of file download (MP3s, MPEGs, and such):

- If the link for the file includes a file size and the download has a progress bar, users expect performance commensurate with their connection speed.
- If users are unaware they're downloading a file, the guidelines for normal pages apply.



For other activity:

- Unless user expectations are set otherwise, the guidelines for normal pages apply.
- If users are presented with a notice that this may take a while, they'll wait significantly longer than without a notice, but the actual amount of time they'll wait varies drastically by individual.

When users are made aware of their connection speed, their expectations about performance shift accordingly. Part of my experiments with users was to create pages with the response times in each of the categories above over a high-speed connection, then to throttle back the connection speed and ask the users the same questions about performance. As long as I told them the connection rate I was simulating, users rated the pages in the same categories, even though the actual response times were very different.

One final note: There's a common industry perception that pages that users find "typical" or "slow" on high-speed connections will be "frustrating" or "unacceptable" to users on slower connections. My research doesn't support this theory. It does show that people who are used to high-speed connections at work but have slower dial-up connections at home are often frustrated at home and try to do Internet tasks at work instead. But people who are used to slower connections rate the same pages as fast, typical, slow, and unacceptable over their typical connection as people who are used to high-speed connections and try to load these pages over their typical connection.

Resource Limitations

Limitations on resources such as time, money, people, hardware, networks, and software affect our performance requirements, even though we really wish they didn't. For example, "You can't have any more hardware" is a resource limitation, and whether we like it or not, it will likely contribute to determining our requirements.

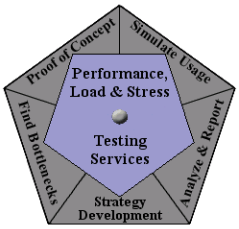
Anecdotally, there's a lot to say about the effects of resource limitations on performance requirements, but practically all it really comes down to is this: Determine before you set your performance requirements what your available resources are, so that when you're setting the requirements, you can do so realistically.

Stakeholder Expectations

Unlike user expectations, stakeholder expectations are easy to obtain. Just ask any stakeholder what he or she expects.

"This system needs to be fast, it needs to support ten times the current user base, it needs to be up 100% of the time and recover 100% of the data in case of down time, and it must be easy to use, make us lots of money, have hot coffee on my desk when I arrive in the morning, and provide a cure for AIDS."

OK, that's not something an actual stakeholder would say, but that's what it feels like they often say when asked the question. It's our job to translate these lofty goals into something quantifiable and achievable, and that's not always an easy task. Usually stakeholders want "industry standards as written by market experts" to base their expectations on. As we've already discussed, there are no standards. In



the absence of standards, stakeholders generally want systems so fast and scalable that performance becomes a non-issue . . . until they find out how much that costs.

In short, stakeholders want the best possible system for the least possible money. This is as it should be. When it comes to stakeholders, it's our job to help them determine, quantify, and manage system performance expectations. Of the three determinants of performance requirements that we've been discussing, the stakeholders have both the most information and the most flexibility. User expectations very rarely change, and resource limitations are generally fairly static throughout a performance testing/engineering effort. Stakeholder expectations, however, are likely to change when decisions have to be made about trade-offs.

Consider this. Recently I've been involved with several projects replacing client/server applications with Web-based applications. In each case, the systems were primarily data entry systems. Initially, stakeholders wanted the performance of the new application to match the performance of the previous client/server application. While this is in line with what I just said about user expectations, it's not a reasonable expectation given system limitations. Web-based systems simply don't perform that fast in general. And I've found that even users who are accustomed to a sub-second response time on a client/server system are happy with a 3-second response time from a Web-based application. So I've had stakeholders sit next to users on the prototypes (that were responding in 3 seconds or less) and had those users tell the stakeholders how they felt about performance. When stakeholders realize that users are satisfied with a "3-second application," they're willing to change the requirement to "under 3 seconds."

Speed, of course, isn't the only performance requirement. Stakeholders need to either inform you of what the other requirements are or be the final authority for making decisions about those requirements. It's our job to ensure that all of the potential performance requirements are considered – even if we determine that they're outside the scope of the particular project.

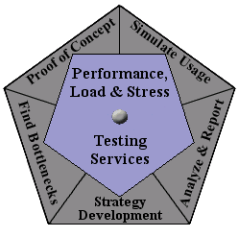
Determining and Documenting Performance Requirements

Once you've collected as much information as possible about user and stakeholder expectations as well as resource limitations, you need to consolidate all of that information into meaningful, quantifiable, and testable requirements. This isn't always easy and should be an iterative process. Sending your interpretation of the requirements for comment back to the people you gathered information from will allow you to finalize the requirements with buy-in from everyone.

As I've mentioned before, I like to think of performance in three categories:

- speed
- scalability
- stability

Each of these categories has its own kind of requirements. We've been discussing speed and scalability extensively in both the "User Experience, Not Metrics" series and the "Beyond Performance Testing" series. Stability is a slightly different type of performance that we won't be discussing much in either series. However, I think the topic of collecting stability requirements is important enough to include here. In the sections that follow, we'll discuss how to extract requirements from expectations and



limitations, then consolidate those requirements into composite requirements – or what some people would refer to as performance test cases.

Speed Requirements

If you've gone through the exercise of collecting expectations and limitations, I'm sure that you have lots of information about speed. Remember that we want to focus on end user response time. There may be individual exceptions in this category for things like scheduled batch processes that must complete in a certain window, but generally, don't get trapped into breaking speed requirements down into subcomponents or tiers.

I like to start by summarizing the speed-related information I've collected verbally – for example:

- normal pages – typical to fast
- reports – under a minute
- exception activities (list) – fast to very fast
- query execution – under 30 seconds
- nightly backup batch process – under an hour

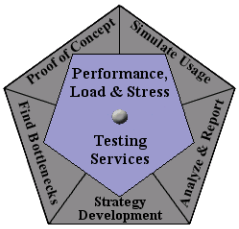
You'll see that some of that information is fairly specific, while some isn't. For this step, what's important is to ensure that all activities fall into one of the categories you specified. You don't want every page or activity to have a different speed requirement, but you do want the ability to have some exceptions to "typical" performance.

Now we must assign values to the verbal descriptions we have and extrapolate the difference between goals and requirements. You may recall from the Performance Engineering Strategy document that performance requirements are those criteria that must be met for the application to "go live" and become a production system, while performance goals are desired but not essential criteria for the application. Table 1 shows the speed requirements and goals derived from the descriptions above.

Activity type	Requirement	Goal
Normal pages	5 sec	3 sec
Reports	60 sec	30 sec
Exception activities (listed elsewhere)	3 sec	2 sec
Query execution	30 sec	15 sec
Nightly backup	1 hour	45 min

Table 1: Speed requirements and goals example

Of course, speed alone doesn't tell the whole story. Even getting agreement on a table like this doesn't provide any context for these numbers. To get that context, you also need scalability and stability requirements.



Scalability Requirements

Scalability requirements are the "how much" and "how many" questions that go with the "how fast" of the speed requirements. These might also be thought of as capacity requirements. *Scalability* and *capacity* are used interchangeably by some people. Much of the information that we need in order to extract specific scalability requirements is contained in "User Experience, Not Metrics, Part 4: Modeling Groups of Users." Please refer to that article for more detail.

Here's an example of scalability requirements that go with the speed requirements above:

- peak expected hourly usage – 500 users
- peak expected sustained usage – 300 users
- maximum percentage of users expected to execute reports in any one hour – 75%
- maximum percentage of users expected to execute queries in any one hour – 75%
- maximum number of rows to be replicated during nightly backup – 150,000

As you can see, we now have some context. We can now interpret that the system should be able to support 300 users with about a 3-second typical response time, and 500 with an under-5-second typical response time. I'm sure you'll agree this is much different t from single users achieving those results.

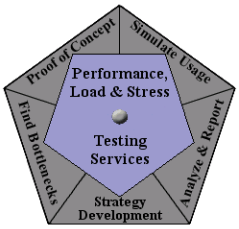
Another topic related to scalability is user abandonment. We'll discuss user abandonment in detail in Part 4 of this series; for now, suffice it to say that a general requirement should be to minimize user abandonment due to performance.

Stability Requirements

Stability covers a broad range of topics that are usually expressed in terms of "What will the system do if . . . ?" These are really exception cases; for instance, "What is the system required to do if it experiences a peak load of double the expected peak?" Another, broader term for these types of requirements is *robustness requirements*. Ross Collard, a well-respected consultant, lecturer, and member of the quality assurance community, defines *robustness* as "the degree of tolerance of a component or a system to invalid inputs, improper use, stress and hostile environments; . . . its ability to recover from problems; its resilience, dependability or survivability." While robustness includes the kind of usage stability we're focusing on, it also includes overall system stability. For our purposes we'll focus on usage stability and not on topics such as data recovery, fail-over, or disaster recovery from the system stability side.

Some examples of stability requirements are as follows:

- System returns to expected performance within five minutes after the occurrence of an extreme usage condition, with no human interaction.
- System displays a message to users informing them of unexpected high traffic volume and requests they return at a later time.
- System automatically recovers with no human interaction after a reboot/power down.
- System limits the total number of users to a number less than that expected to cause significant



performance degradation.

Now, let's put these together into some real requirements.

Composite Requirements

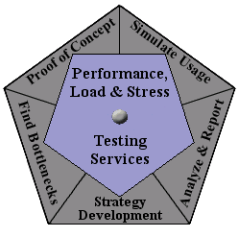
The types of requirements discussed above are very important, but most of them aren't really testable independently, and even if they are, the combinations and permutations of tests that would need to be performed to validate them individually are unreasonable. What we need to do now is to consolidate those individual requirements into what I term composite requirements. You may know them as performance test cases. The reason I shy away from calling these performance test cases is that my experience has shown that most people believe that once all the test cases pass, the testing effort is complete. I don't believe that's always the case in performance engineering, though it may be for performance testing.

Meeting the composite requirements simply means the application is minimally production-ready from a performance standpoint. Meeting the composite goals means that the application is fully production-ready from a performance standpoint *according to today's assumptions*. Once these composites are met, a new phase begins that's beyond the scope of this series – capacity planning, which also makes use of these composite requirements but has no direct use for test cases.

Let's look at how the individual requirements we came up with map into composite requirements and goals.

Composite Requirements

1. The system exhibits not more than a 5-second response time for normal pages and meets all exception requirements, via intranet, 95% of the time under an extended 300-hourly-user load (in accordance with the user community model) with less than 5% user abandonment.
2. The system exhibits not more than a 5-second response time for normal pages and meets all exception requirements, via intranet, 90% of the time under a 500-hourly-user load (in accordance with the user community model) with less than 10% user abandonment.
3. All exception pages exhibit not more than a 3-second response time 95% of the time, with no user abandonment, under the conditions in items 1 and 2 above.
4. All reports exhibit not more than a 60-second response time 95% of the time, with no user abandonment, under the conditions in items 1 and 2 above.
5. All reports exhibit not more than a 60-second response time 90% of the time, with less than 5% user abandonment, under the 75% report load condition identified in our scalability requirements.
6. All queries exhibit not more than a 30-second response time 95% of the time, with no user abandonment, under the conditions in items 1 and 2 above.
7. All queries exhibit not more than a 30-second response time 90% of the time, with less than 5% user abandonment, under the 75% report load condition identified in our scalability requirements.



8. Nightly batch backup completes in under 1 hour for up to 150,000 rows of data.
9. The system fully recovers within 5 minutes of the conclusion of a spike load.
10. The system displays a message to users starting with the 501st hourly user informing them that traffic volume is unexpectedly high and requesting that they return at a later time.
11. The system limits the total number of users to a number less than that expected to cause significant performance degradation (TBD – estimated 650 hourly users).
12. The system automatically recovers to meet all performance requirements within 5 minutes of a reboot/power down with no human interaction.

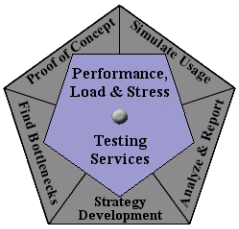
Composite Goals

1. The system exhibits not more than a 3-second response time for normal pages and meets all exception requirements, via intranet, 95% of the time under a 500-hourly-user load (in accordance with the user community model) with less than 5% user abandonment.
2. All exception pages exhibit not more than a 2-second response time 95% of the time, with no user abandonment, under the conditions in item 1 above.
3. All reports exhibit not more than a 60-second response time 95% of the time, with no user abandonment, under the conditions in items 1 and 2 above.
4. All reports exhibit not more than a 30-second response time 95% of the time, with no user abandonment, under the conditions in item 1 above.
5. All reports exhibit not more than a 30-second response time 90% of the time, with less than 5% user abandonment, under the 75% report load condition identified in our scalability requirements.
6. All queries exhibit not more than a 15-second response time 95% of the time, with no user abandonment, under the conditions in item 1 above.
7. All queries exhibit not more than a 15-second response time 90% of the time, with less than 5% user abandonment, under the 75% report load condition identified in our scalability requirements.

These requirements may be more detailed than you're used to, but I hope you can see the value of insisting upon explicit composite requirements such as these.

Summing It Up

No matter how many people ask for one, there's no industry standard for Web application performance. In the absence of such a standard we must depend on our own best judgment to determine just how fast is fast enough for our application. This article has discussed how to determine how fast is fast enough and how to convert that information into explicit, testable requirements. While explicit requirements based on reasonable performance expectations don't ensure project success, they do ensure the ability to evaluate the performance status of a system throughout the development life-cycle, and that alone can be invaluable.



Acknowledgments

- The original version of this article was written on commission for IBM Rational and can be found on the IBM DeveloperWorks web site

About the Author

Scott Barber is the CTO of PerfTestPlus (www.PerfTestPlus.com) and Co-Founder of the Workshop on Performance and Reliability (WOPR – www.performance-workshop.org). Scott's particular specialties are testing and analyzing performance for complex systems, developing customized testing methodologies, testing embedded systems, testing biometric identification and security systems, group facilitation and authoring instructional or educational materials. In recognition of his standing as a thought leading performance tester, Scott was invited to be a monthly columnist for Software Test and Performance Magazine in addition to his regular contributions to this and other top software testing print and on-line publications, is regularly invited to participate in industry advancing professional workshops and to present at a wide variety of software development and testing venues. His presentations are well received by industry and academic conferences, college classes, local user groups and individual corporations. Scott is active in his personal mission of improving the state of performance testing across the industry by collaborating with other industry authors, thought leaders and expert practitioners as well as volunteering his time to establish and grow industry organizations. His tireless dedication to the advancement of software testing in general and specifically performance testing is often referred to as a hobby in addition to a job due to the enjoyment he gains from his efforts.

About PerfTestPlus

PerfTestPlus was founded on the concept of making software testing industry expertise and thought-leadership available to organizations, large and small, who want to push their testing beyond "state-of-the-practice" to "state-of-the-art." Our founders are dedicated to delivering expert level software-testing-related services in a manner that is both ethical and cost-effective. PerfTestPlus enables individual experts to deliver expert-level services to clients who value true expertise. Rather than trying to find individuals to fit some pre-determined expertise or service offering, PerfTestPlus builds its services around the expertise of its employees. What this means to you is that when you hire an analyst, trainer, mentor or consultant through PerfTestPlus, what you get is someone who is passionate about what you have hired them to do, someone who considers that task to be their specialty, someone who is willing to stake their personal reputation on the quality of their work - not just the reputation of a distant and "faceless" company.