**User Experience, Not Metrics**

by:

R. Scott Barber


# Part 10: Creating a Degradation Curve

The previous four articles in this series have each dealt, at least in part, with the topic of performance-related results reporting. This article will conclude our look at this topic by discussing the single most powerful performance graph at our disposal, the degradation curve.

This is the tenth article in the "User Experience, Not Metrics" series, which focuses on correlating customer satisfaction with your Web site application's performance as experienced by users. This article is intended for both Rational TestStudio users and managers with some Microsoft Excel experience. All of the information will be provided to create degradation curve charts, but I'm assuming that you've read Parts 6, 7, and 9 and are comfortable with the Excel walkthroughs included in those articles.

## What's a Response Time Degradation Curve?

"Although Internet bandwidth and Web server capacity have improved in recent years, Web site performance problems continue to challenge developers and testers. The combination of complex Web-based applications and the dynamic characteristics of Internet traffic can cause significant degradation in Web site performance," write Steven Splaine and Stefan P. Jaskiel in *The Web Testing Handbook*.

I recently had the opportunity to attend a presentation given by Steve Splaine. Toward the middle of his presentation, he showed a slide with a chart that he simply called "the performance graph." I was both pleased and surprised to see that this was the same chart that I call a response time degradation curve. Regardless of the name you give this chart, by the end of this article I'm certain you'll agree that this is the most powerful chart in a performance tester's arsenal for presenting information to stakeholders. The value of this chart lies in the fact that it answers the questions that begin with "How many . . . " and "How fast . . . ."

Figure 1 shows a relatively basic example of a response time degradation curve. This chart plots user experience against user load. The end-to-end response time in seconds is plotted vertically on the left side and the total number of users accessing the system is plotted horizontally across the bottom. This particular version of the chart also includes the data table. You'll notice that the user experience time increases, or degrades, as more users are introduced to the system, exactly as we would expect.

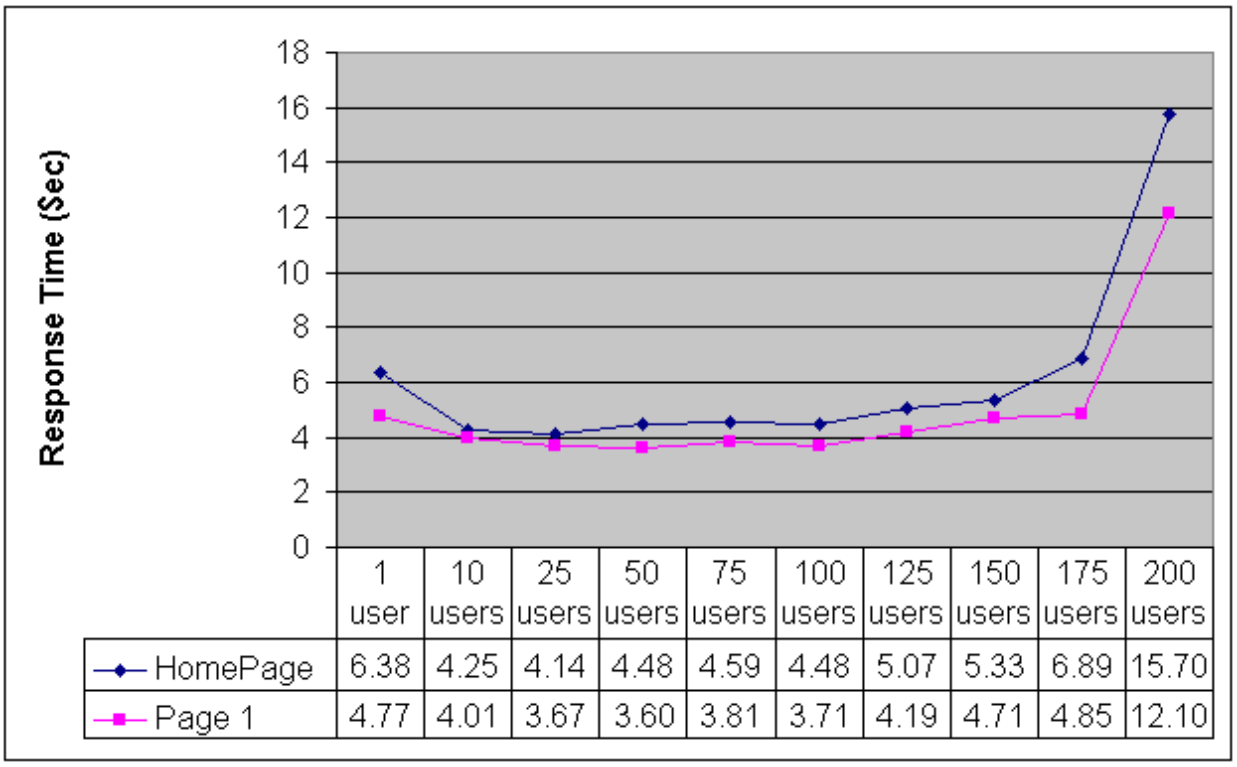



**Figure 1: A basic response time degradation curve chart**

I chose this particular chart because the data yielded the most common shape for a response time degradation curve. The shape of the curve in Figure 1 is what you'll see more than 95% of the time when you create this chart. I would go so far as to say that if you don't see this shape, one of the following conditions is almost certainly true:

- The user community model isn't accurate.
- The scripts aren't representative of the user community model.
- The system under test can't handle multiple users at all.
- The test didn't actually stress the system.

## Regions of the Curve

The shape of a typical response time degradation curve can be broken down into four regions:

- the single-user region
- the performance plateau
- the stress region
- the knee in performance

Each of these regions contains a significant amount of useful information about the system under test.

The following sections discuss these regions in detail. All of this is what makes this chart so valuable. Knowing what the shape of the curve will be before test execution allows you to make accurate preliminary assessments based on the location of these different areas of the chart without additional analysis.

## *The Single-User Region*

The first region of the curve, as we move from left to right, is the single-user region, as highlighted in Figure 2.

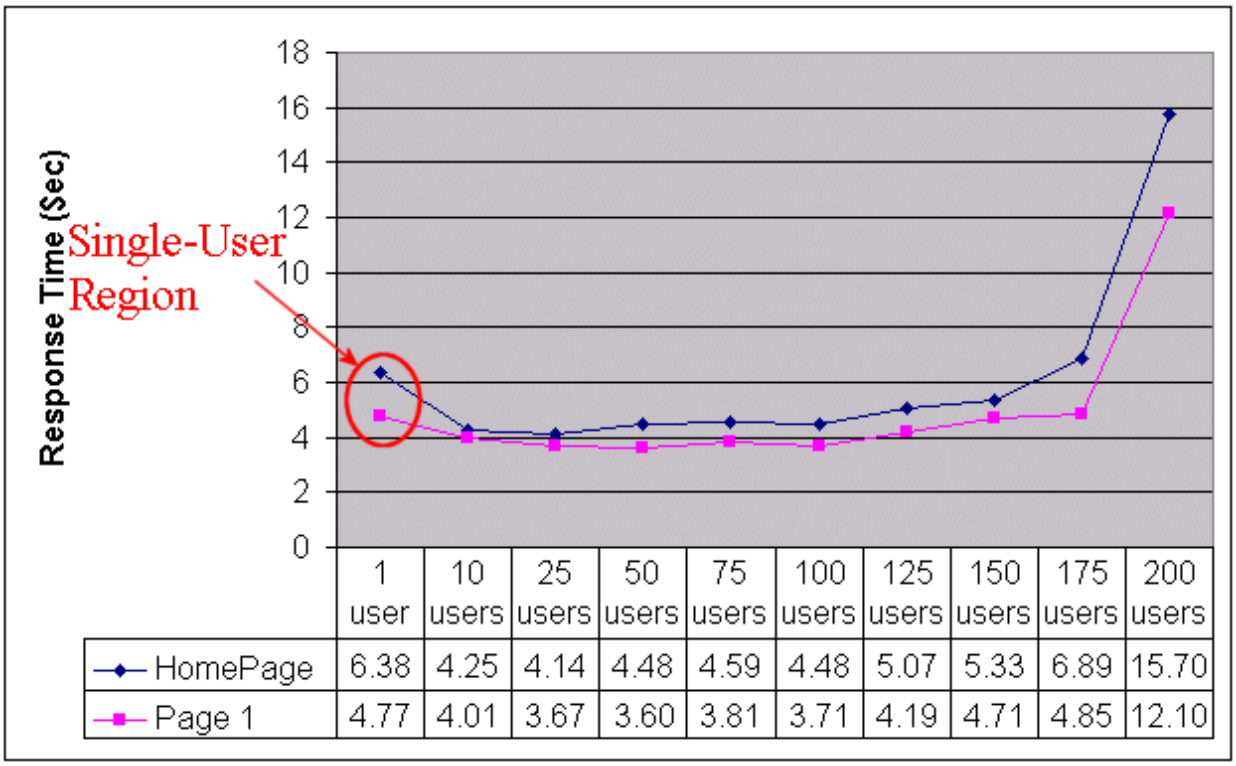



**Figure 2: The single-user region of a degradation curve**

Viewing this region, we see that the performance for a single user is actually slower than for several levels of multiple users. This is normal. The response time for single users (as generated by Rational tools or any other load-generation tool) will generally be slightly slower than the best performance seen on a site, for reasons that have to do with how load-generation tools work (that is, by threading), caching, "sleeping" hard drives, and such. You can think of hitting the site with a single user as being like driving your car on a cold day. If you don't let your car warm up a little before taking it on the freeway, its performance will probably be a little sluggish until it does warm up. While this isn't a perfect analogy, it does give you an idea of what's going on. This is why I don't recommend using single-user loads as a basis of comparison for future load tests.

## The Performance Plateau

I like to call the region to the right of the single-user region the "performance plateau," as noted in Figure 3.

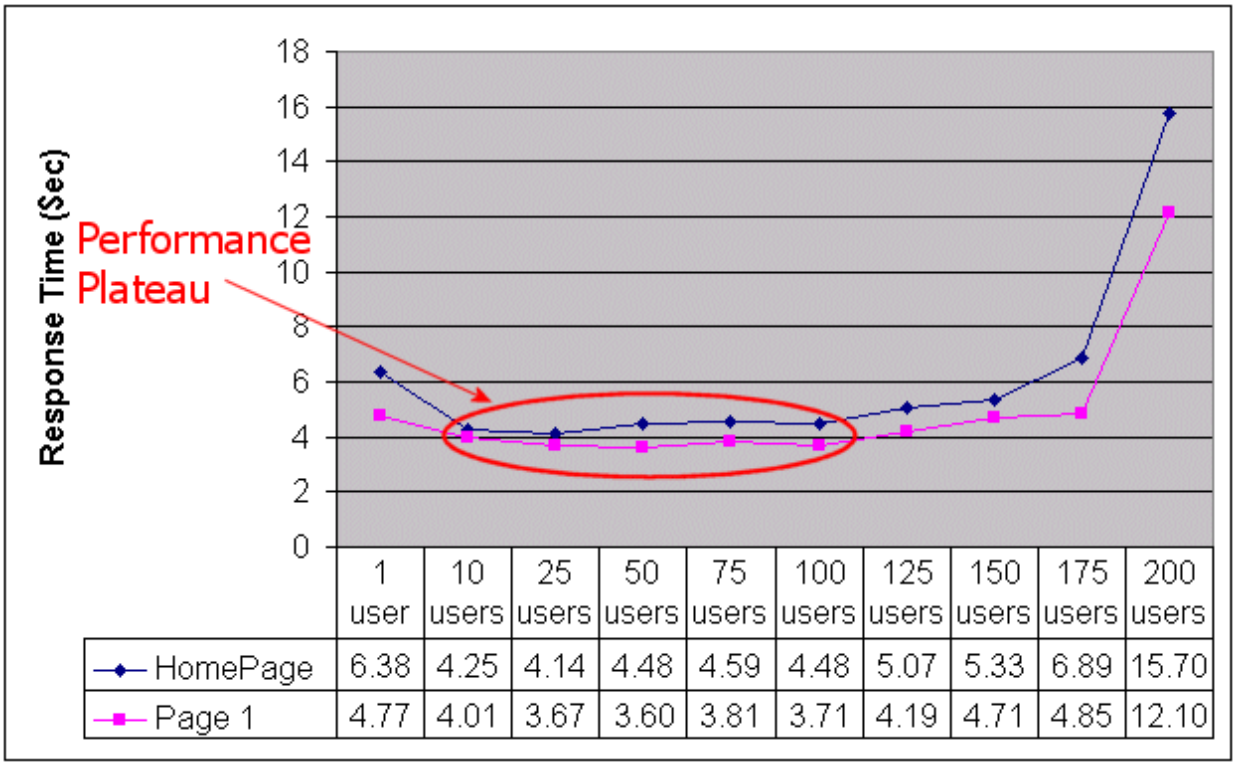



**Figure 3: The performance plateau of a degradation curve**

In this region (in this case, in the range from about 10 to about 100 simulated users) we see that the performance gets better and stays pretty consistent for a while. Whatever the performance of the system is in this region, it's the best performance you can ever expect from your system without conducting further tuning (assuming your tests are modeled properly). Results from any test residing in the performance plateau are good candidates for baselines or benchmarks to be used as a basis of comparison for future load tests. I generally recommend that benchmarks be 15% of the greatest user load before the knee in performance (described below).

## The Stress Region

In the region between the 100-user and 175-user loads in our example curve, we see that response times start getting longer as the load increases. This is known as the stress region, as pointed out in Figure 4.

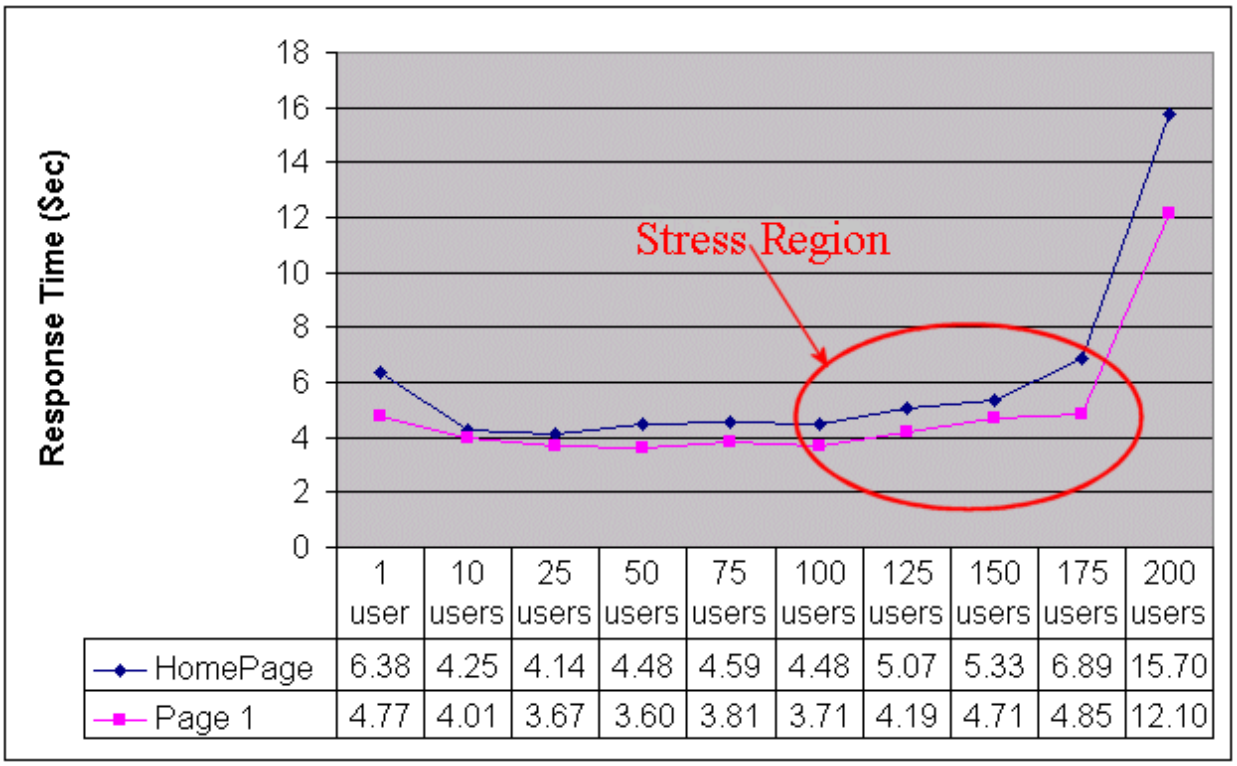



**Figure 4: The stress region of a degradation curve**

Technically, this region is the degradation part of the curve, where it's evident that the system is being stressed but is basically handling the load. The stress region begins when the response time starts to increase slowly and ends at the knee (discussed below). This is the area where the application/system is said to "degrade gracefully." What this means is that as the load increases in this region, the response time also increases, but not excessively. When the application is fully tuned, the maximum recommended user load should be the load at the beginning of the stress region, but the system continues to perform relatively well above this load for a period of time. This gives the stakeholder a level of confidence that if more users than expected are accessing the system, the system will remain stable and functional.

## *The Knee in Performance*

Above the 175-user load, we see response time start to climb very quickly in our example curve. The system is no longer handling the load gracefully; it's likely not functioning properly and may even be unstable. This change in response time often happens very quickly and without warning, causing a sharp change in the slope, or direction, of the curve that we call the knee in performance, as emphasized in Figure 5.

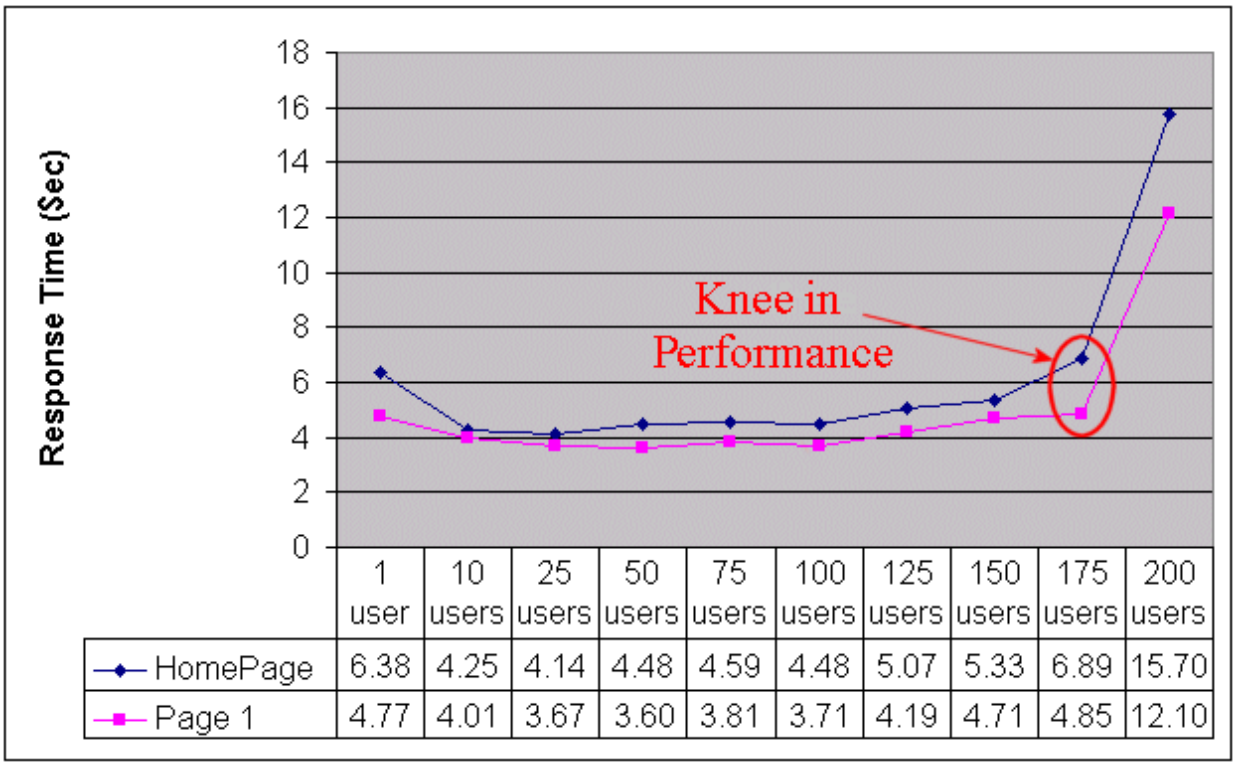



**Figure 5: The knee of a degradation curve**

Where the knee occurs is the absolute maximum load you ever want your application/system to encounter. If you're still testing, this is the load that has exploited your critical bottleneck and should be researched in detail and corrected if at all possible.

There will always be a knee in performance. If your chart doesn't show one, it's probably because you haven't stressed your system enough to find it. I recommend, whenever possible, that you continue testing until the knee is found. That's the only way to make conclusive estimates about the extent of the application's scalability and to begin conducting capacity planning exercises.

## What's the Confidence Interval of the Curve?

Level of confidence can be an extremely statistically complex topic. According to the StatSoft Inc. glossary, "the confidence intervals for specific statistics (e.g., means, or regression lines) give us a range of values around the statistic where the 'true' (population) statistic can be expected to be located (with a given level of certainty)." For example, Figure 6 shows a 90% confidence interval for the regression line.

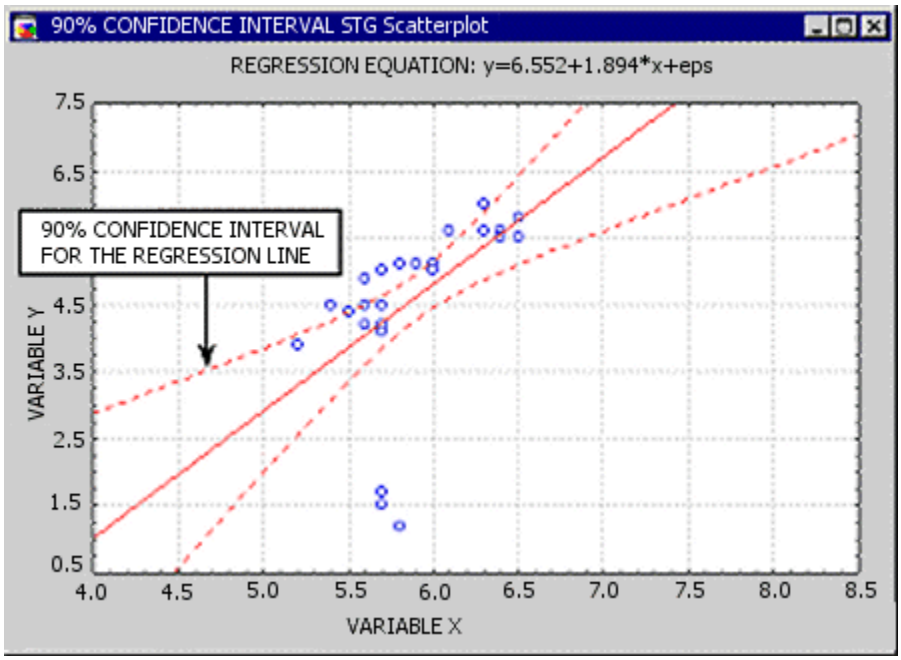



**Figure 6: An example of a confidence interval from the StatSoft glossary**

Personally, I have no desire to develop a complex mathematical equation to determine the "range of values around the statistic where the 'true' statistic can be expected to be located." Instead, let's explore another commonsense approach.

As you may recall from previous articles, I recommend reporting on the 90th percentile response. We've previously interpreted this measurement to mean that 90% of all users will experience a page load time of not more than the reported time. The question that follows is, How confident are we in those results? My answer is that I'm exactly as confident in those results as I am in the accuracy of the model we tested. Of course, that isn't a very useful answer. This is where judgment comes into play.

I happen to know through experience and testing that if I model my workload distribution correctly and develop scripts to match that distribution, the results from my tests are, at worst, statistically equivalent to results collected from the production application. What that means is that I have 100% confidence in the accuracy of the results but not of the tests themselves — and not of the model and/or my scripts. If we *did* have 100% confidence in both our model and our scripts, we could say that based on the tests, we're 100% confident that 90% of the users will experience a page load time of not more than the reported time. I don't know about you, but I would never make that statement.

So how do we answer the question, How certain are we that the user community model and the scripts are correct? Unless you're lucky enough to be testing a site that's already in production and run direct comparisons, there's no way to know the answer to this question. However, there *is* a way to demonstrate a confidence interval in the results.

We discussed using multiple user community models in Part 4: Modeling Groups of Users. This discussion was based on the difference between expected-case, best-case, and worst-case usage of the application. If we were to design three user community models based on these criteria, then execute and graph them, we would get the results shown in Figure 7.

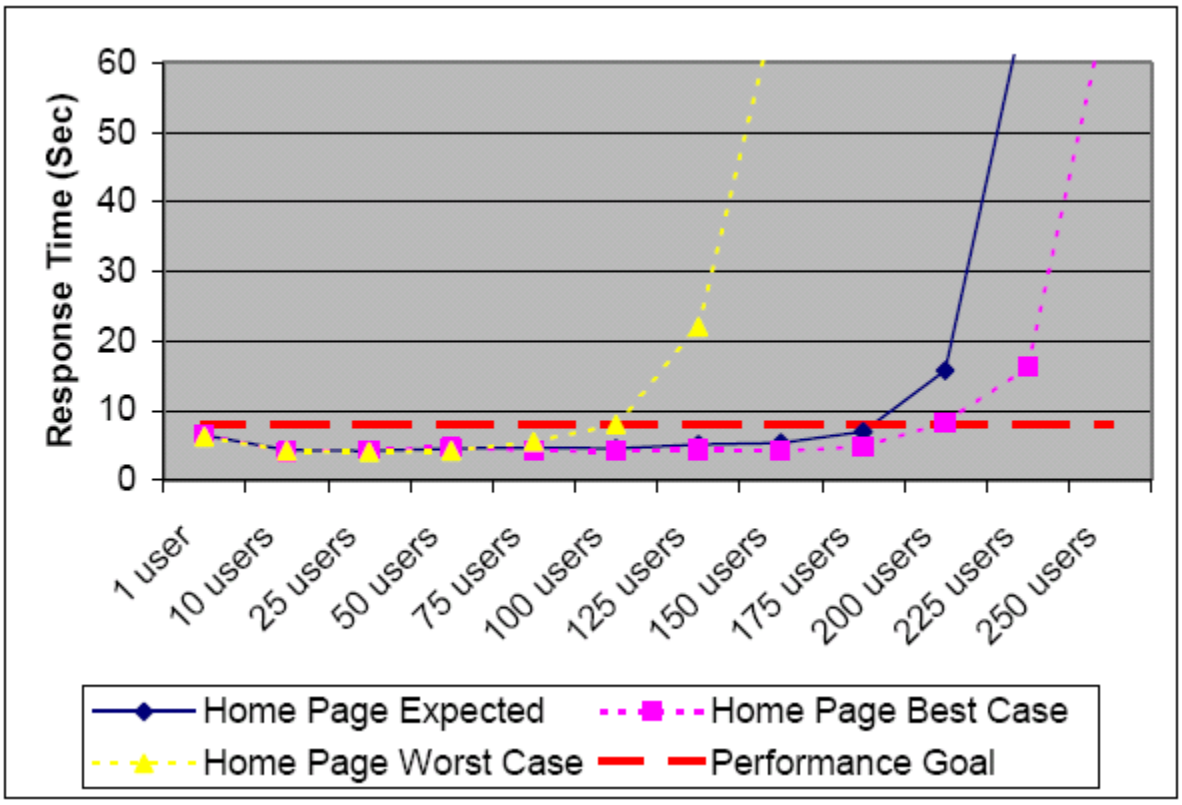


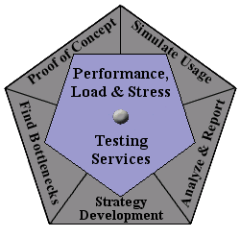

**Figure 7: A margin-of-error degradation curve**

A red dashed line has been added to represent the performance goal, and there are two additional lines on the chart, representing the results from the series of tests using the worst-case (most performance-intensive) user community model and the best-case (least performance-intensive) user community model. As you might expect, the blue curve, from the expected user community model, falls in between the best- and worst-case lines. By looking at where those curves cross the red line, we can see how many users can be accessing the system in each case while still meeting the stated performance goal. If we're 95% confident (by our own estimation) that the best- and worst-case user community models are truly best- and worst-case, we can read the results from this chart as follows: "The tests show, with 95% confidence, that between 100 and 200 users can access the system while experiencing acceptable performance."

You may be thinking that an acceptable-performance range of between 100 and 200 users is a pretty large range, and it is in this case. When I reported these results, the stakeholders of the system also thought this range was too large for comfort. This graph led to further discussion of the question, How good is good-enough performance?

## Determining Good-Enough Performance

Defining what constitutes good-enough performance is one of the most difficult tasks of the performance testing/engineering team. There are no industry standard rules governing performance, so




it's up to the stakeholders to determine what performance they'll deem acceptable, or good enough. Answering these key questions can help the stakeholders decide what they'll consider good-enough performance:

- What's the maximum user load representing 90% of the expected use of the site?
- What are the performance expectations of the users of the system?
- What abandonment rate are the stakeholders willing to accept?
- How critical is it that the performance be improved immediately?
- How long will it be until the maximum user load on the system will be realized (that is, what's the rollout schedule of the application)?

In the case of our example, the stakeholders answered these questions and determined that the performance represented by the charts above was acceptable for the phase 1 release but needed to be improved before phase 2. This gave the developers and testers three extra months to improve the performance of the application. In those three months, they improved the performance so it would be acceptable at a load of between 425 and 500 users.

## Creating the Degradation Curve

Now that we've discussed how to interpret and use the degradation curve, it's time to describe how to create one. There are three parts to the curve creation process:

- determining which tests need to be executed, and/or included in the curve
- determining which page or pages are to be included in the curve
- physically creating the curve

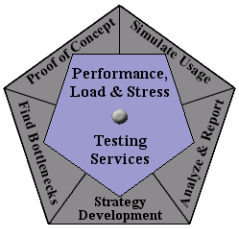
If these three activities aren't each completed properly, the curve will likely be inaccurate and cause the stakeholders to draw incorrect conclusions.

You'll see that this can be an iterative process. For instance, you may determine which tests are to be executed and which page load times will be included, then create the degradation curve only to find out that you didn't find the knee in performance. In this case, more tests will need to be executed with increasingly higher loads and the degradation curve recreated until the knee is found.

### *Which Tests Are Required?*

Determining the required tests is the easiest of the three steps. No matter what the maximum expected user load of the application is, there are at least seven tests that need to be executed. They are as follows:

- a single-user test of every page to be included in the chart
- a benchmark test of every page to be included in the chart (normally 10–15% of the maximum expected user load, using the agreed-upon user community model)
- a 25%-of-maximum-expected-load test of every page to be included in the chart, using the agreed-




upon user community model

- a 50%-of-maximum-expected-load test of every page to be included in the chart, using the agreed-upon user community model
- a 75%-of-maximum-expected-load test of every page to be included in the chart, using the agreed-upon user community model
- a 100%-of-maximum-expected-load test of every page to be included in the chart, using the agreed upon user community model.
- A 125%-of-maximum-expected-load test of every page to be included in the chart, using the agreed-upon user community model

The exact percentage distribution of the tests may vary, but remember that to see the proper shape of the curve, at least seven points are required. Four of the points must represent the single-user load, the benchmark load, the maximum expected load, and a load greater than the maximum expected load. The other three points should be between the benchmark and maximum expected user loads.

If these seven points don't result in a degradation curve that clearly shows the four regions described earlier, you should add more tests at the proper loads to ensure the regions are clearly identifiable. This isn't an exact science, but rather requires a combination of trial-and-error and experience.

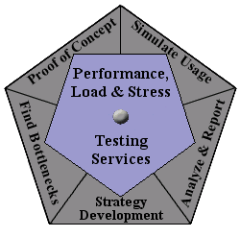
Remember that if you're going to create a margin-of-error degradation curve, you'll need to execute each of these tests using your three user community models — best case, expected case, and worst case.

## *Which Pages Get Reported?*

Determining which pages get reported is a little trickier. Once again, this isn't an exact science. In examining which pages deserve to be included, you may find that several degradation curves are needed to display all of the desired pages. I use the following rules to determine which pages to report:

1. Always report performance of the home page. It's the first page your users will see and if it performs poorly, they're unlikely to continue to use your site.
2. Always report on the most server-intensive page, often a search or submit page. This page is often easy to find by manual screening of the site. Simply pick the one that feels slowest.
3. Always consider pages that generate reports or generate lists on the fly. For example, "my payment history" or "view today's most popular items" are good pages to consider.
4. Consider including the simplest static page — often the "FAQ" or "About us" page.
5. Consider any other page that may be suspected of poor performance for some reason.

Keep in mind that although this entire series of articles has focused on Web-based applications, all of the concepts apply equally well to other types of applications such as client-server. For the purposes of designing a user experience test, it may be helpful to think of the Web as an application presentation method rather than a software architecture.




## How Is the Degradation Curve Created?

Once you've executed the tests required and determined which pages to include, it's time to create the curve. Once again, we'll be copying our data from TestManager into Excel. But before we copy the data, we need to create the table to hold the data. This table won't need any special formatting like some of the tables in previous articles. Simply list the pages to be included in the curve down the left side of the table and the user loads across the top of the table, as shown in Figure 8.

| | 1 user | 10 users | 25 users | 50 users | 75 users | 100 users | 125 users | 150 users | 175 users | 200 users |
|---|---|---|---|---|---|---|---|---|---|---|
| HomePage | | | | | | | | | | |
| Page 1 | | | | | | | | | | |

**Figure 8: Unpopulated degradation curve table**

Now that we've created the table, it needs to be populated. The process of populating this table can be a little tedious, but easy. Simply enter the 90th percentile (or whichever percentile measurement your organization prefers) for each page and user load into the empty cells. The entered data for our example is shown in Figure 9.

| | 1 user | 10 users | 25 users | 50 users | 75 users | 100 users | 125 users | 150 users | 175 users | 200 users |
|---|---|---|---|---|---|---|---|---|---|---|
| HomePage | 6.38 | 4.25 | 4.14 | 4.48 | 4.59 | 4.48 | 5.07 | 5.33 | 6.89 | 15.74 |
| Page 1 | 4.77 | 4.01 | 3.67 | 3.60 | 3.81 | 3.71 | 4.19 | 4.71 | 4.85 | 12.12 |

**Figure 9: Populated degradation curve table**

All that's left is to create the chart. We'll do this in the same basic way that we've created charts in our previous articles. We highlight all of the information in the degradation curve table, then choose Insert > Chart from the menu bar and select the Line chart type. You can choose from the remaining options based on personal preference. I like to leave the data table on this chart as shown in Figure 1, but you may choose not to. If you choose not to display the data table, ensure that you do display the legend. You can also add the performance goal line as seen in Figure 7 by following the instructions in the next section.

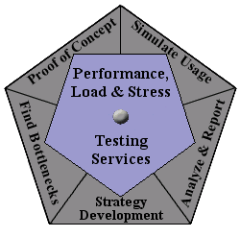
## How Is the Margin-of-Error Degradation Curve Created?

To create the margin-of-error degradation curve, we must first create a new table to populate that resembles the one in Figure 10.

| | 1 user | 10 users | 25 users | 50 users | 75 users | 100 users | 125 users | 150 users | 175 users | 200 users | 225 users | 250 users |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HomePage Expected | | | | | | | | | | | | |
| HomePage Best Case | | | | | | | | | | | | |
| HomePage Worst Case | | | | | | | | | | | | |
| Performance Goal | | | | | | | | | | | | |

**Figure 10: Unpopulated margin-of-error degradation curve table**

We need to populate the expected, best-case, and worst-case rows with the 90th (or other) percentile measurement, much like we did for the degradation table. In the Performance Goal row, we need to enter the stated page load time goal for the home page in every column. This will create the




performance goal line in the chart. See Figure 11 for a populated version of this table.

| | 1 user | 10 users | 25 users | 50 users | 75 users | 100 users | 125 users | 150 users | 175 users | 200 users | 225 users | 250 users |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HomePage Expected | 6.38 | 4.25 | 4.14 | 4.48 | 4.59 | 4.48 | 5.07 | 5.33 | 6.89 | 15.74 | 70.00 | 130.00 |
| HomePage Best Case | 6.45 | 4.02 | 4.28 | 4.69 | 4.15 | 4.07 | 4.33 | 4.12 | 4.78 | 8.23 | 16.28 | 70.00 |
| HomePage Worst Case | 6.15 | 4.15 | 3.98 | 4.13 | 5.48 | 7.93 | 21.99 | 70.58 | 130.00 | 130.00 | 130.00 | 130.00 |
| Performance Goal | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 | 8.00 |

**Figure 11: Populated margin-of-error degradation curve table**

To create the margin-of-error degradation chart, we highlight all of the information in the margin-of-error degradation curve table, then choose Insert > Chart from the menu bar and select the Line chart type. For this chart I prefer to not show the data table. I also prefer to use dotted or dashed lines to represent all of the lines other than the expected case. These aren't requirements, though. I encourage you to experiment with formatting the chart to meet your individual needs.

## Now You Try It

This article has described the considerations and steps for creating a degradation curve. The best way to internalize this knowledge is to use it on your own projects. Unfortunately, I can't realistically give you an entire Web site and set of stakeholders to work with to practice every step of this process. What I can do is give you a set of data exported from TestManager into Excel for you to practice with. I encourage you to use this data to evaluate which pages to include in your charts, and then actually create both the degradation curve and the margin-of-error degradation curve from this data.

## Summing It Up

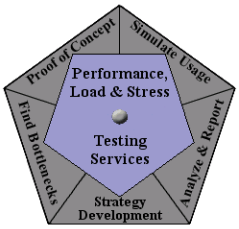
The degradation curve chart is the most powerful single chart in the performance tester's / engineer's arsenal. With this one chart, all stakeholders of the system/application can see exactly how the system/application is performing at different user loads as compared to the stated performance goals, all at a quick glance. Adding this single chart to your reports and summaries will greatly improve communications about and understanding of current performance of your system/application.

## 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.