Beyond Performance Testing

by:
R. Scott Barber

Part 4: Accounting for User Abandonment

Have you ever gotten tired of waiting for a Web page to load and exited the site before completing the task you had in mind? No doubt you have, just like every other Web user. Although user abandonment is a routine occurrence, it isn't commonly discussed in the context of developing and analyzing performance tests.

Here in the fourth article of the "Beyond Performance Testing" series, we'll explore performance-testing issues related to user abandonment and how to account for these issues using the Rational Suite® TestStudio® system testing tool.

So far, this is what we've covered in this series:

Part 1: Introduction

Part 2: A Performance Engineering Strategy

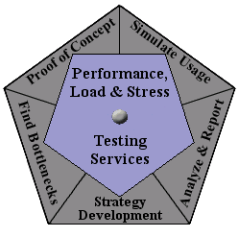
Part 3: How Fast Is Fast Enough?

This article is intended for all levels of Rational TestStudio users and will be particularly useful to advanced VU scripters. It discusses why users abandon a site, explains why abandonment must be accounted for in performance testing, describes in detail how to create an abandonment model for your Web site, and outlines how to adapt VU scripts to correctly handle abandonment using the model you've created.

Before we get started, I want to acknowledge the work of Alberto Savoia, a true thought leader in the field of performance testing and engineering. Before reading his article "Trade Secrets from a Web Testing Expert," I hadn't been considering user abandonment in my testing. After reading his article, I expanded on what I read and applied it in my VU scripting. What you're about to read paraphrases and extends Savoia's ideas about user abandonment. The application to Rational VU scripts is my own work.

Why Do Users Abandon a Site?

User abandonment occurs when a user gets frustrated with a site, usually due to performance, and discontinues using that site either temporarily or permanently. As we discussed in the previous article in this series, "Part 3: How Fast Is Fast Enough?," different users have different tolerance levels for performance. The reasons users abandon involve some of the same factors that we discussed when we were talking about determining performance requirements — namely, user psychology, usage considerations, and user expectations. When we get to the section about how to create an abandonment model, I'll show you a method for quantifying each of these factors to account for user abandonment.



User Psychology

Abandonment is all about user psychology. And when it comes to abandonment, user psychology is more than just "Why *do* we abandon sites?" That answer is simple: "Because we get tired of waiting." The answer to the follow-up question — "*Why* do we get tired of waiting?" — is not so simple. Why do *I* get tired of waiting and subsequently decide to abandon? Here are a few of my common reasons:

- The site is just painfully slow.
- I lose interest while waiting for a page to download.
- I get distracted during download.
- I figure I can find what I need somewhere else faster.
- I just plain get bored.
- While waiting I check my e-mail and forget to go back.

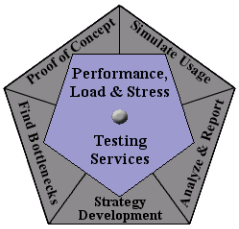
While all of those reasons can be summarized as "the site is too slow," what "too slow" means to me can vary depending on the particular reason. For instance, after about 8 or 10 seconds I start getting bored and may move to another browser window, even though 8 to10 seconds isn't painfully slow. As another example, it seems that almost every night I manage to be on the computer when dinner's ready. If I'm waiting for a page, I get right up and proceed to the dining room. If I'm reading a page, I finish what I'm reading. If what I'm reading continues on to the next page, I click the link. If it comes up very quickly, I read that one too, but if it hesitates, I go to dinner. Once I go to dinner, I almost never come back to what I was reading. While I'm sure my wife appreciates those slow sites because I show up for dinner sooner, those sites are definitely losing me as a viewer based on performance — and not excessively poor performance at that.

Usage Considerations

Usage considerations are even more relevant to user abandonment than they are to determining requirements. In Part 3, I mentioned that in filling out my federal income tax return online I was more tolerant of slowness because my expectations had been set properly. While this expectation setting may have kept me from getting annoyed as quickly, the truth is that I wasn't going to abandon that site no matter how slow it got. The thought of losing my return and having to start over or of having an incomplete return submitted that might lead to my being audited would have kept me from abandoning. In contrast, when I'm just catching up on the news I'll abandon one site and check another long before I'll wait for a slow site.

When thinking about usage considerations, we're really examining how much longer than a user's typical tolerance she's willing to wait before abandoning a task she considers very important. The perceived importance of the activity being performed online is the key factor here. An activity's importance is usually perceived as higher in cases that involve the following:

- real or perceived loss of money as a result of not completing the transaction
- concern that inaccurate information will be submitted if the transaction isn't completed
- lack of an alternative means to accomplish the task at hand



- existence of a real or perceived deadline for completing the transaction

Later we'll discuss how to determine and apply this "importance factor" to our abandonment model.

User Expectations

I mentioned expectations above. Abandonment is unquestionably affected by user expectations, as demonstrated by my experience in preparing my tax return. Users abandon sites when their expectations aren't met, or when their expectations haven't been set properly in advance. Our concern in performance testing isn't to manage user expectations but rather to ensure that their expectations drive our requirements. We probably don't have the ability to poll the potential users of the system with questions like these:

- What speed connection do you use?
- How fast do you expect this Web site to be?
- How long will you wait before abandoning your session?

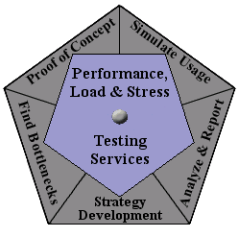
If polling a subset of potential users is possible, we should probably do it. But if we can't, how do we account for user expectations? Our best bet is to evaluate what potential users are employing now to accomplish the same activity and assume that they'll become frustrated if our solution is noticeably (let's say 20% or more) slower than that. This isn't a hard science, but doing this will give us a place to start while deriving our abandonment model.

Ideally, this thought process was part of your requirements gathering and contributed to your overall performance requirements. If you collected requirements well, I submit that your users' expectations should be (statistically speaking) equivalent to the performance requirement associated with each page, meaning that if a requested page downloads within the time specified in the performance requirements, users won't abandon the site for performance reasons.

Why Account for User Abandonment?

Before we create a user abandonment model based on the factors I just discussed, let's take a step back and talk about why we should be concerned about accounting for user abandonment in performance testing. Alberto Savoia says this in "Trade Secrets from a Web Testing Expert": "You should simulate user abandonment as realistically as possible. If you don't, you'll be creating a type of load that will never occur in real life — and creating bottlenecks that might never happen with real users. At the same time, you will be ignoring one of the most important load testing results: the number of users that might abandon your Web site due to poor performance. In other words, your test might be quite useless."

One of the great things about most Web sites is that if the load gets too big for the system/application to handle, the site slows down, causing people to abandon it, thus decreasing the load until the system speeds back up to acceptable rates. Imagine what would happen if once the site got slow, it stayed slow until someone "fixed the server." Luckily, abandonment relieves us of that situation, at least most of the time. Assuming that the site performs well enough with a "reasonable" load, performance is generally self-policing, even if at a cost of some lost customers/users. So one reason to correctly account for user abandonment is to see how your system/application behaves as the load grows and to



avoid simulating bottlenecks that might never really happen.

A second reason to correctly account for abandonment is that it provides some extremely valuable information to all of the stakeholders of the system. You may recall that we specified acceptable abandonment rates in our composite requirements in Part 3. Correctly accounting for abandonment allows you to collect data to evaluate those requirements.

Another way to think about why correctly accounting for user abandonment is important is to ask what happens if we don't account for abandonment at all or if we mis-account for abandonment.

Not Accounting for Abandonment

If we don't account for abandonment at all, the script will wait forever to receive the page or object it requested. When it eventually receives that object, it will move on to the next object like nothing ever happened. If the object is never received, the script never ends. I can think of no value this adds to the performance-testing effort, unless you have a need to show some stakeholder, "Under the conditions you specified, the average page-load time was roughly 2.5 hours." Unfortunately, we do occasionally have to make a point by generating ridiculous and meaningless numbers, but that's not a performance test, and it doesn't get us any closer to delivering a quality, well-performing application.

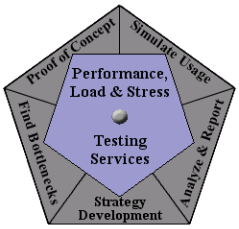
On the other hand, if you don't account for abandonment and virtually all of your pages load faster than your requirements, then your test was perfectly accurate (in terms of abandonment simulation) . . . by accident. Don't settle for being correct by accident. Take the extra few minutes to include abandonment in your performance tests and have the confidence that your results are honestly accurate as opposed to accidentally representative.

It should be pointed out that there *are* some cases where not accounting for abandonment is OK. Many Web-based applications that my coworkers and I have tested are exclusively for internal audiences that have no choice but to wait. For example, my current client has a policy that all employees (and consultants) must enter the hours they worked that week between noon and 5 p.m. every Friday — unless they aren't working that day, in which case they're required to notify their managers of their hours so the managers can enter this information into the system. With roughly 3500 employees and consultants accessing this system during a five-hour period on top of the typical traffic, the system gets very slow. Under other circumstances, users might abandon the site and try later or go somewhere else. In this case, they have no choice but to wait, so abandonment isn't an issue.

Mis-accounting for Abandonment

Mis-accounting for abandonment is what load-generation tools do by default. They assume that all users abandon at a predetermined time (in TestStudio, that default time works out to be 240 seconds) yet still continue on, requesting the following page like nothing happened. Of course, you can change settings to improve that by changing the time limit or having the virtual user actually exit, but that's still not context specific by page. If you were really motivated, you could change the parameters before every request based on how long you thought a user would wait for that particular object, but that still wouldn't account for the page as a whole.

Improper accounting for abandonment can cause results that are even more misleading than if abandonment weren't accounted for at all. To support this statement, let's consider a couple of



examples and their side-effects.

"At 240 seconds, stop trying to get this object, log a message, and move on to the next object" (TestStudio default) — If you have objects taking more than 240 seconds to load, this may cause unexpected errors in situations where subsequent objects need the former objects to have loaded successfully, because the tool will now be "forcing" the application to serve pages that a real user couldn't reach. This will also skew page and object load times, because you don't actually know how long the object would have taken to load, yet that 240 seconds is calculated as if the download were successful. Worst of all, the subsequent errors normally mask the initial cause, making it appear as if the script is flawed. This is all not to mention that the additional load applied after the time-out (that a real user wouldn't be applying) may skew your results. (Note that TestStudio's default action, which is to ignore failures and continue playback, can be modified with the `Timeout_act` variable. The default string setting is `"IGNORE"` but this can be changed to `"FATAL"` to cause virtual users to bail when an error takes place.)

"Just log when people would have abandoned for analysis but don't actually exit the virtual user" — While this may be useful during early testing (which I'll say more about in a minute), it paints a very inaccurate picture of the actual abandonment rate for a laundry list of reasons, including these:

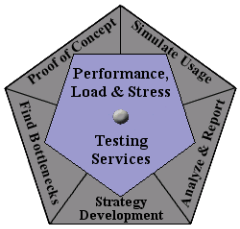
- Once a VU gets one page slow enough to abandon, they usually get more if not exited, resulting in statistics showing an artificially high abandonment rate.
- Allowing a VU to continue that would have abandoned keeps the current load on the system rather than reducing the total load for others, which is going to cause other VUs to experience response times in the abandonment range, once again resulting in statistics showing an artificially high abandonment rate.
- If the abandonment level response time was actually due to an error, subsequent page requests may also produce errors, making the actual problem (one slow page) much more difficult to detect.

Please note that we're talking here about grossly misrepresenting real abandonment. Mis-modeling the abandonment range by a few seconds isn't going to cause this kind of problem. Your abandonment model needs to be reasonable, not perfect.

I mentioned a minute ago that just logging potential abandonment and not exiting the VU may be useful during early testing. This is true for several reasons. For example, suppose you have an abandonment model that says all users will abandon if they encounter a page-load time of 30 seconds, but while your site is under development it's taking an average of 45 seconds to return a page, even at very low user loads. You'll still want your scripts to run all the way through to gather information and create system logs to help track down the reason the times are so slow. In this situation, abandoning all of the VUs when they hit the home page gives you no information to help tune the system. Use your best judgment early in testing about whether to just log or actually exit users when they reach the abandonment response time, but always exit users when you're executing a test intended to predict the experience of real users in production.

Building a User Abandonment Model

Now I'll show you my approach to building representative user abandonment models for a particular Web site. For each type of page on the site, you'll want to establish these four parameters:



- the abandonment distribution
- the minimum abandonment time
- the maximum abandonment time
- the importance level

These parameters relate to the factors discussed earlier that affect user abandonment. You'll initially organize these parameters in a table similar to Table 1. This sample table doesn't model any site in particular but contains values for five page types that have different abandonment parameters. Later on you'll adjust the performance requirement parameter and the absolute abandonment time parameter based on each page's importance level and your appraisal of the context in order to arrive at the model you'll use in your VU scripts.

Page Name	Abandonment Distribution	Abandonment Min Time	Abandonment Max Time	Importance Level
Home Page	Normal	5 sec	120 sec	Low
Pay Bill	Uniform	5 sec	900 sec	High
Search Web	Negexp	8 sec	120 sec	Low
Submit Taxes	Inverse Negexp	15 sec	900 sec	Very High
Validate Field	Normal	3 sec	120 sec	Medium

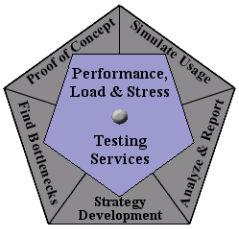
Table 1: Sample abandonment parameters for five different page types

The Abandonment Distribution Parameter

Earlier I listed some reasons that users might get tired of waiting for a page to load. These reasons can be extremely variable, not only between different users but also between visits by the same user. A user's tolerance for waiting might change dramatically from session to session. While this makes it hard to predict when an individual might abandon, it also means that abandonment is likely to follow standard distribution models such as normal or negative exponential distributions. Our task is to determine for each page type in our table which distribution model best represents the likely abandonment behavior of users.

You might want to review my discussion of distribution models in "User Experience, Not Metrics, Part 2: Modeling Individual User Delays" at this point, because I'm not going to go into much detail about distribution models here. Suffice it to say that either a normal (bell curve) or a uniform (linear) distribution will be most accurate in the majority of cases.

If you've ever taken a statistics or psychology course you know that almost everything that real human beings do (over a large enough sample) can be represented by a bell curve. You may also recall that the key to an accurate bell curve is the standard deviation. We know two things about standard deviations when it comes to Web usage: (1) they're exceptionally large (statistically) in comparison to the range of values, and (2) they're almost impossible for non-mathematicians to calculate accurately. What that means is that in most cases we actually end up with a very flat bell curve that, in effect, approaches a linear distribution. Statistics aside, if you don't have a strong reason to do otherwise, choose between either a normal or a uniform distribution based on your best judgment.



The negative exponential or negexp (logarithmic or one-tailed) distribution is much less common. It applies in cases where most users will behave one way but a few will behave in an opposite manner. I'll give two examples.

- As I've mentioned, I was willing to wait as long as it took to prepare my taxes, but if the performance had gotten bad enough, eventually the system would have ceased responding and I would have effectively abandoned. While it's likely that I would actually have waited until the system timed out, I might have abandoned sooner if I'd believed the system wasn't responding. This situation is represented by a one-tailed distribution where a few users may abandon in a short period of time but most users hang in there as long as possible.
- On some Web sites, fields in a form are automatically populated when the user starts entering data and, for example, chooses a value from a list box. Usually we don't even expect this to happen, but as long as it happens quickly (and the values that appear are correct) we don't mind — or at least, I don't. But if it doesn't happen quickly all we know is that our page is frozen and we can't enter data in the next field (or worse, we can enter data, but it gets erased when the screen finally does refresh). That situation will cause users to abandon a site faster than any other situation I can think of. Thus, I represent user abandonment of field population with a logarithmic distribution that has most people abandoning quickly and only a few people hanging in there.

If you really don't know which distribution is best, use a random distribution. As Savoia put it, "It's important to realize that even the most primitive abandonment model is a giant leap in realism when compared to the commonly used 60- or 120-second timeouts."

The Minimum Abandonment Time Parameter

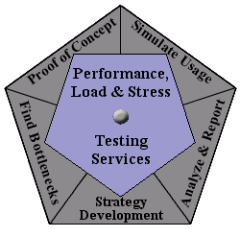
The minimum abandonment time parameter is our estimate of the amount of time that users expect to wait for a page to load. This is the minimum amount of time we think they'll wait before abandoning a site. Recall that I said above that users' expectations should be (statistically speaking) equivalent to the performance requirement associated with a page. To supply this parameter, simply copy the performance requirement from your Performance Test Strategy document. If you have multiple requirements based on user connection speed, you'll want to follow this process and ultimately create an abandonment model for each connection speed.

The Maximum Abandonment Time Parameter

No matter how patient a user may be, sometimes a Web site simply fails to respond due to circumstances like these:

- secure session time-out (requiring users to either abandon or start over)
- browser "page cannot be displayed" time-out errors
- temporary (or permanent) Internet connectivity interruptions on either end

While this doesn't technically count as user abandonment (since the Web site abandoned the user and not the other way around), it does provide the upper bound for how long users could potentially wait before ceasing their Web-surfing session. Most load-generation tools (including Rational TestStudio) assume this category is the only time abandonment occurs, but as you've seen, this is highly misleading



when trying to predict production-level performance accurately.

The maximum abandonment time parameter is the most scientific of the parameters. This is simply the time after which either your browser stops waiting for a response (in our example case, 120 seconds) or your secure session expires and you have to start your session over (in our example case, 900 seconds or 15 minutes). To determine these numbers, simply ask the architect/developer responsible for the presentation tier (or Web server) to provide you with the information.

The Importance Level Parameter

The importance level parameter is simply a commonsense assessment of the perceived importance of a particular activity to the typical user. You can certainly poll users and stakeholders to obtain this information, but I wouldn't spend the time to get more scientific than the four-tier rating system (low, medium, high, and very high) used here. What we're going to do with these importance levels is to assign them values that we'll use to adjust the abandonment min and max times. Remember, users are likely to wait longer to abandon a page when the perceived importance of completing the task on that page is high.

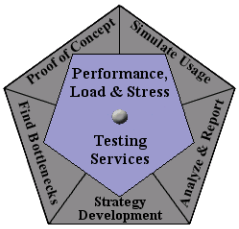
Adjusting the Parameters to Create the Abandonment Model

Now you'll make the necessary adjustments to the parameters table to create the abandonment model you can use in your VU scripts. Table 2 shows the values I suggest using to adjust the minimum and maximum abandonment times in your table for each level of importance. Allow me to stress that these are guidelines; be sure to apply a healthy dose of common sense when applying these factors and take the context of your particular situation into account.

Importance Level	Min Time Factor	Max Time Factor
Low	1	~ 1/4
Medium	~ 1.5	~ 1/2
High	~ 2	~ 3/4
Very High	~ 4	1

Table 2: Factors for adjusting minimum and maximum times for importance

There are two notes I want to make about these guidelines. First, you'll notice that the min time factor for low importance and the max time factor for very high importance are both 1. This is by definition in our model. If you find yourself wanting to change those factors, consider reassessing your *parameters* rather than the importance factor. Second, you'll notice that for small ranges, applying these factors blindly could result in the minimum time being larger than the maximum time. If this happens, simply revise the importance factors and recalculate until your minimum is once again smaller than your maximum.



Applying those factors to our example parameters, we come up with the abandonment model shown in Table 3. Take a moment to review the table and form your own opinions about the times as they're listed.

Page Name	Abandonment Distribution	Abandonment Min Time	Abandonment Max Time
Home Page	Normal	5 sec	30 sec
Pay Bill	Uniform	10 sec	450 sec
Search Web	Negexp	8 sec	30 sec
Submit Taxes	Inverse Negexp	30 sec	900 sec
Validate Field	Normal	5.5 sec	60 sec

Table 3: Preliminary abandonment model

As I review Table 3, I'm quite comfortable with those values . . . except two. I don't believe that anyone is going to wait 450 seconds (7.5 minutes) for confirmation of a bill payment. This is probably because the session keep-alive is configured for longer than it needs to be (15 minutes versus maybe 5 minutes), but assuming that configuration is nonnegotiable, I'm simply going to arbitrarily change the abandonment max time value for the Pay Bill page to 240 seconds. I also don't believe anyone is going to wait 60 seconds for some fields to dynamically populate, so I'm going to change that value to 20 seconds.

Does that mean that I disagree with my own guidelines? I don't think so. I think it just means that I'm taking a step back to look at my model in the context of the unique aspects of the system I'm modeling. (I strongly recommend that you take the time to do that throughout your projects.)

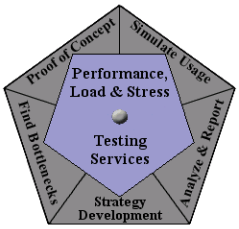
Finally, Table 4 shows the model that we'll apply to our example script.

Page Name	Abandonment Distribution	Abandonment Min Time	Abandonment Max Time
Home Page	Normal	5 sec	30 sec
Pay Bill	Uniform	10 sec	240 sec
Search Web	Negexp	8 sec	30 sec
Submit Taxes	Inverse Negexp	30 sec	900 sec
Validate Field	Normal	5.5 sec	20 sec

Table 4: Final abandonment model

Adapting VU Scripts to Correctly Handle Abandonment

With our abandonment model in hand, we're ready to accurately account for user abandonment using Rational TestStudio. I suggest you either record a short script with blocks or timers, or pick an existing script, and follow along. Any script that's currently working on an active site will do.



Adding the Abandonment Procedure

In "User Experience, Not Metrics, Part 2: Modeling Individual User Delays" I introduced the `normdist` function to create bell curve distributions. The first thing you'll need to do is add that function, shown in Listing 1, to the top of your script as we did in that article, since it's used in the abandonment procedure we're about to discuss.

```
#include <VU.h>

int func normdist(min, max, stdev); /*Specifies input values for normdist
function.*/
int min, max, stdev; /* min:Min value; max:Max value; stdev:degree of deviation */
{
/*Declare range, iterate, and result as integers. */
  int range, iterate, result;
  range = max – min; /*Range is the difference between the max and min values.*/
  iterate = range/stdev; /*Number of iterations ensures proper curve shape.*/
  result = 0; /*Integers are not automatically initialized to 0 upon declaration.*/
/*Compensation for integer vs. floating-point math.*/
   stdev += 1;
   for (c = iterate; c != 0; c--) /*Loop through iterations.*/
   result += (uniform (1, 100) * stdev) / 100; /*Calculate and tally result.*/
   return result + min; /*Send final result back.*/
}
```

Listing 1: The normdist function

Immediately after the `normdist` function but before anything else in your script, insert the abandonment procedure shown in Listing 2. If you're not familiar with C, this procedure does nothing more than take in the parameters you've created in your model, determine if the abandonment threshold has been met, and if it has, print a message to the log file and exit the virtual user. For cases like we discussed above where you want to just log the abandonment and not exit the user, simply comment out the `user_exit` line by preceding it with a double slash (`//`).

```
int int_tempstart, int_tmpend, int_tmptime; /*Used to calculate page load times*/
proc abandon(int_tmptime, int_min, int_max, str_distro) /*Specifies input values.*/

/*int_tmptime: last pageload time; int_min, int_max, str_distro: abandonment
parameters*/
int int_tmptime, int_min, int_max;
string str_distro;
{
   int int_abandon = 0;
/*if block determines the abandonment threshold based on passed parameters.*/
   if(str_distro=="norm")int_abandon=normdist(int_min,int_max,(int_max-int_min)/3);
   else if (str_distro == "unifom") int_abandon = uniform(int_min, int_max);
   else if (str_distro == "negexp") {
      int_abandon = negexp(int_min) + int_min;
      if (int_abandon > int_max) int_abandon = int_max;
      }
   else if (str_distro == "invneg") {
      int_abandon = int_max - negexp(int_min);
```

```
      if (int_abandon < int_min) int_abandon = int_max;
      }
   else int_abandon = = ((rand() / 32767) * (max - min)) + min;

/*If the threshold has been met, write a message to the log.*/
   if (int_tmptime > int_abandon){
      testcase ["Abandon"] 0, "", "Virtual User " + itoa(_uid) + " Abandoned after

   Command_id " + _cmd_id + "     after " + itoa(int_tmptime) +"milliseconds";
      user_exit(0, "User Abandoned " + itoa(int_tmptime));
   }
}
```

Listing 2: Basic abandonment procedure

You may choose to add the following line to this procedure between the `testcase` and `user_exit` lines to create a separate output file that just contains messages about abandonment:

```
printf( "Virtual User " + _UID + " Abandoned after Command_id " + _cmd_id + "
```

after

```
"  + itoa(int_tmptime) +"milliseconds");
```

Once you've completed this (and recompiled your scripts to make sure you didn't copy in anything extra — like I did when I was reviewing this), you're ready to take the next step.

You may also choose to save these into a separate `.h` file and include that file instead. If you want to do that, copy and paste the code above into a text editor (such as Notepad) and save the file as `common_functions.h` (for example) into the include folder below the `TMS_Scripts` folder. Then, at the top of the script you want to apply the abandonment procedure to, simply add the line

```
#include <common_functions.h>
```

immediately following

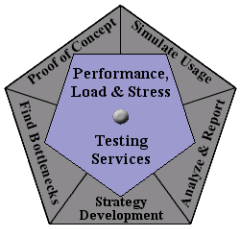
```
#include <VU.h>.
```

This works exactly the same way and saves you from having to copy and paste that code to the top of all your scripts.

Adapting Existing Scripts

Adapting the rest of the script is actually very simple. Let's say your script looks like the one in Listing 3.

```
start_time ["tmr_home"];
hq2unx144_fhlmc_com_1 = http_request ["AP_top_~1.016"] "www.yahoo.com",
   HTTP_CONN_DIRECT,
```




```
. . .

http_nrecv ["AP_top_~1.018"] 100 %% ; /* 3119 bytes */
http_disconnect(hq2unx144_fhlmc_com_1);
stop_time ["tmr_home"];
```

Listing 3: Unadapted script

All you need to do is capture the actual start and stop times, calculate the difference between those times, and then pass that difference and the rest of your abandonment model parameters to the abandonment procedure. See the adapted script segment in Listing 4.

```
int_tmpstart = start_time ["tmr_home"];
hq2unx144_fhlmc_com_1 = http_request ["AP_top_~1.016"] "www.yahoo.com",
   HTTP_CONN_DIRECT,

. . .

http_nrecv ["AP_top_~1.018"] 100 %% ; /* 3119 bytes */
http_disconnect(hq2unx144_fhlmc_com_1);
int_tmpend = stop_time ["tmr_home"];
int_tmptime = int_tmpend-int_tmpstart;
abandon(int_tmptime, 5000, 30000, "norm");
```

Listing 4: Adapted script

Remember when entering the parameters into the abandonment procedure call to convert seconds to milliseconds and to put the name of the distribution in quotes. As written, the abandonment procedure accepts the following distribution parameters:

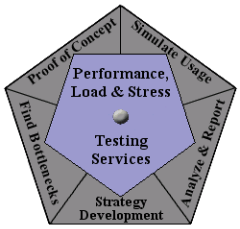
- “norm” for normal or bell curve distributions (with a standard deviation of one third the range)
- “uniform” for uniform or linear distributions
- “negexp” for negative exponential or logarithmic distributions weighted toward the minimum value
- “inv_negexp” for negative exponential or logarithmic distributions weighted toward the maximum value.
- any other entry in quotes for a random distribution between the minimum and maximum values

Making a couple of these replacements by hand isn’t too bad, but for long scripts I copy the scripts into a text editor with good search/replace functionality to modify the scripts for me. Then I just go back and adjust the abandonment parameters according to the model.

Interpreting Results

Most of the rest of this series is about interpreting results, so I won’t spend a lot of time talking about it here. But here are a few things I wanted to point out:

- Check your abandonment rate before you evaluate your response times. If your abandonment rate for a particular page is less than about 5%, look for and handle outliers. If your abandonment rate for a particular page is more than about 5%, you probably have a problem worth researching further



on that page.

- Check your abandonment rate before drawing conclusions about load. Remember, every user who abandons is not applying load. Your response time statistics may look good, but if you have 25% abandonment, your load may be 25% lighter than you were expecting.
- If your abandonment rate is more than about 20%, consider commenting out the `user_exit` line and re-executing the test to help gain information about what's causing the problem.

For Advanced Users

If you're thinking "That's great . . . but once a user closes his browser (abandons) it's not going to request the remaining objects, where this procedure will," keep reading.

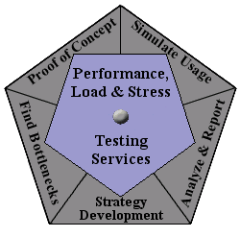
To take the next step and model abandonment down to the individual request/receive pair, you'll need to replace the abandonment procedure I gave you above with the one in Listing 5.

```
proc abandon(cmd_id, int_tmptime, int_min, int_max, str_distro,
   int_wholepage) ) /*Specifies input values for abandonment procedure.*/

/*cmd_id,  read-only  variable  identifying  the  commands;  int_min,  int_max  and

  str_distro: abandonment model parameters; int_wholepage identifies if the
  evaluated abandonment is for the whole page or a single request/receive pair*/
  int int_tmptime, int_min, int_max, int_wholepage;
  string cmd_id, str_distro;
  {
    int int_abandon, total_bytes = 0, nrecv_total = 1;
    /*if block determines the abandonment threshold based on passed parameters.*/
    if (str_distro == "norm") int_abandon = normdist(int_min, int_max,
      (int_max - int_min) / 3);
    else if (str_distro == "uniform") int_abandon = uniform(int_min,
      int_max);
    else if (str_distro == "negexp") {
      int_abandon = negexp(int_min) + int_min;
      if (int_abandon > int_max) int_abandon = int_max;
      }
    else if (str_distro == "invneg") {
      int_abandon = int_max - negexp(int_min);
      if (int_abandon < int_min) int_abandon = int_max;
      }
    else int_abandon = ((rand() / 32767) * (int_max - int_min)) + int_min;

   /*If this isn't a whole page - receive the requested data and time it.*/
   if (int_wholepage != 1) {
      /*Get how much data should be transmitted.*/
         total_bytes = atoi(http_header_info("Content-Length"));
         /*Loop until the bail time is reached or everything is
           downloaded.*/
         while ((time() - _lr_ts < int_abandon) && (nrecv_total >=
            total_bytes)) {
         if (n = sock_isinput()) http_nrecv [cmd_id] n;
         nrecv_total += _nrecv;
```




```
        }
      }
   /*If everything wasn't downloaded in time or the page took too long,
     abandon.*/
   if ((int_tmptime >= int_abandon) || (nrecv_total < total_bytes)) {
      testcase ["Abandon"] 0, "", "Virtual User " + itoa(_uid) +
         " Abandoned during Command_id " + cmd_id + " after "
         + itoa(int_abandon) +" milliseconds";
      /*And exit the user (may be commented out when appropriate).*/
      user_exit(0, "User Abandoned " + itoa(int_abandon));
   }
}
```

Listing 5: Advanced abandonment procedure

Without explaining the code, I'll say that the significant difference between these two procedures is the input parameters. Look at how the new procedures are called in Listing 6, a modified version of the script from Listing 3, and then I'll explain the new parameters.

```
int_tmpstart = start_time ["tmr_home"];
hq2unx144_fhlmc_com_1 = http_request ["AP_top_~1.016"] "www.yahoo.com",
HTTP_CONN_DIRECT,

. . .

//http_nrecv ["AP_top_~1.018"] 100 %% ; /* 3119 bytes */
abandon(_cmd_id, 0, 5000, 30000, "norm", 0);
http_disconnect(hq2unx144_fhlmc_com_1);
int_tmpend = stop_time ["tmr_home"];
int_tmptime = int_tmpend-int_tmpstart;
abandon(_cmd_id, int_tmptime, 5000, 30000, "norm", 1);
```

Listing 6: Adapted advanced abandonment script

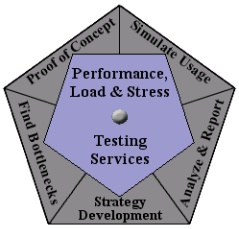
The first thing we see is that we're now replacing every `http_nrecv` command with a call to the abandonment procedure. The syntax is

```
abandon(_cmd_id, 0, [min], [max], [distribution], 0);
```

where `[min]`, `[max]`, and `[distribution]` are the same parameters we've already discussed, `_cmd_id` is the command ID read-only variable to identify which specific command we're referring to, the first zero is a hard-coded `int_tmptime` that's not used in this instance, and the final zero is used as a flag to show that this isn't an entire page-load abandonment call.

After our stop time, the abandonment procedure has the same general parameters but with different default values. The syntax is

```
abandon(_cmd_id, int_tmptime, [min], [max], [distribution], 1);
```



where `_cmd_id`, `int_tmptime`, `[min]`, `[max]`, and `[distribution]` are the same parameters we've already discussed and the final parameter serves as a flag set to 1 to indicate that this is a full-page abandonment call.

Summing It Up

In this article, we've explored the premise of user abandonment, determined how to model it, and demonstrated how to implement that model in our VU scripts.

Alberto Savoia concluded his 2001 article with this:

"When you adopt concurrent users as a load testing input parameter and fail to account for user abandonment you run the risk of creating loads that are highly unrealistic and improbable. As a result, you may be confronted with bottlenecks that might never occur under real circumstances."

It's now 2003 and user abandonment is still a relatively unheard-of topic. I hope this article will help increase awareness and encourage people to avoid the mistake of not taking abandonment into account during performance testing.

References

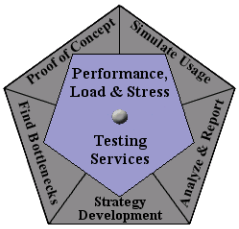
- "Trade Secrets from a Web Testing Expert" by Alberto Savoia (STQE magazine, May/June 2001)

Acknowledgments

- Special thanks to Alberto Savoia for taking the time to review this article.
- Thanks to Chris Walters for developing and providing the code for the `normdist` function and developing the additional code for the advanced abandonment procedure.
- 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.