User Experience, not Metrics

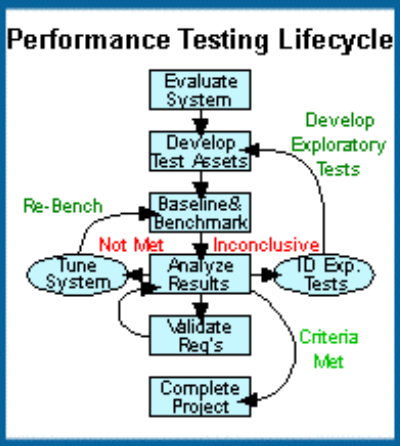
Effective Performance Testing

Part 3: Modeling Individual User Patterns

Web-based applications typically allow for many ways to accomplish a task, as well as many methods to navigate to the same place within a site. While users think this is a good thing (at least most of the time), we as testers find that this fact complicates matters, requiring that we look not only at what users do within an application but also at how they do it. This part of the "User Experience, Not Metrics" article series discusses how to use Rational TestStudio to script and design suites to accommodate the many possible navigation paths users can choose to take, so that the results of a testing effort accurately represent user experience.

(This item originally appeared on The Rational Developer Network, an online community for customers of IBM Rational Software. To find out more, including how you can get a free evaluation to the Rational Developer Network, please visit http://www.rational.com/services/rdn/find_out_more.jsp)

Stress-testing experts may argue that this level of detail in test script development is just extra work and doesn't add value to a project. I would agree with them . . . if our objective were to conduct a stress test. In Part 1 of this series, stress tests are defined as "any combination of scripts that are played back at a high user load excluding user delays . . . but are *not* valid for determining user experience." As you'll recall from Part 1, the focus of this series is on correlating customer satisfaction with performance as experienced by external users. The point is to model individual user patterns to determine user experience as part of a *load* test, not to conduct a stress test. (See "Website Stress-Testing" for a good explanation of the distinction between the two kinds of testing.) The next article will discuss how to apply these individual user patterns to groups or communities of users.



Part 2 of this series discussed how to model individual user delays. Like that article, the intent of this one is to discuss as well as demonstrate the use of TestStudio based on Noblestar's years of experience. The article is intended for all levels of TestStudio users but will be most useful to Intermediate tool users and above. I encourage you to prove the validity of the concepts presented in this article by applying them to test a site where you already know the user loads, patterns, and user experience measurements, and then comparing the user experience measurements to the actual measurements.

Determining User Patterns

To collect accurate user experience measurements, you need to apply the appropriate stress to the appropriate parts of the system at the appropriate time. It may sound complicated, but it isn't. You start by determining the user patterns of individuals. I'll suggest a number of different methods for doing this and will illustrate with the example of determining user patterns for an online bookstore.

By Scott Barber
www.perftestplus.com

Imagine that you manage this bookstore and want to determine what it's like for your users to buy books on the site from a performance perspective. The question you must answer is: Once a return user who wants to purchase the latest *New York Times* best-seller successfully logs into your online bookstore, what pages does that user navigate through to accomplish her goal? We can chart the "choose a fiction book" paths as shown in Figure 1. (The next article will detail how this graph is created.)

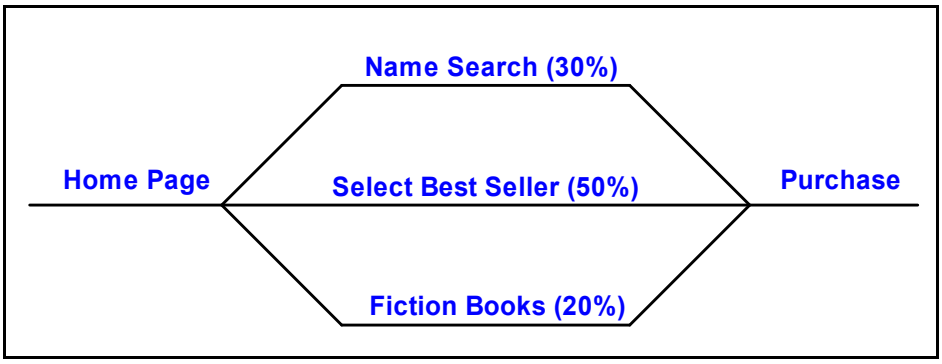


Figure 1: Main navigation paths to choose a fiction book

This graph depicts that all users in our example start at the bookstore home page and traverse the site in one of three different ways:

- They search for the book by name or author.
- They surf to the fiction section and scroll through all books in that section.
- They go straight to the little blinking link that says "*New York Times* best-sellers."

The percentage of users following each path is noted in the graph. All users in this example ultimately purchase the book. This is a simplistic example, but the concepts are just as easily applied to complex situations. You'll notice that the graph identifies activities by path but doesn't identify each step or page involved in completing the activity. Each horizontal line represents an activity that has no navigational decision points.

My approach in developing this graph, which may or may not be precise enough for your own purposes, was as follows. After choosing a Web site that sells books for this model, I determined that the most common individual user activity would be to purchase a book. Then I navigated the site for a while to figure out all the different ways I could actually find a book for purchase, assuming I had a particular book in mind. After figuring that out, I estimated how likely it was that I would follow each of those paths, based on my own experience and intuition as both a Web user and a Web tester. And then I drew a picture.

This method — using your intuition or best guess — is good for two things: (1) getting an intuitive sense of a Web site before taking a more scientific approach to identifying patterns, and (2) validating that the patterns identified by more scientific means still make sense. Since I didn't have access to the information required to be more scientific (such as log files, traffic-monitoring software, system design documents, system administrators, and such), I stopped there, but you might want to go farther.

If you're developing a real test in which accuracy of the model matters, you should determine user patterns with one of these methods:

- If you're testing a Web site already in production, you can determine the actual values and distribution by viewing statistics from traffic-monitoring software such as WebTrends or LiveStats. These software programs, when configured properly, will show you the percentage of users who do what on the site and what path they follow to do it, all on an easy-to-understand chart. In the absence of traffic-monitoring software, you can examine log files to determine distributions of user activity by page hits. Interpreting this information will be discussed in more detail in Part 4.
- When available, technical specifications for the Web site should tell you which activities a user could possibly carry out, which activities are most likely, and what navigation paths users are expected to follow to carry out the activity. Functional test scripts or test cases are sometimes useful in determining user activities and paths within the site. These resources don't tell you how users actually navigate the site, but they usually tell you how the system was designed to interact with users.
- If the site is new and has never been in production, you could try to gather information about user patterns from the administrators of similar sites. Of course, the sites you want information from will most likely be your competition, so the administrators may not be very cooperative.
- If none of the three previous alternatives are available, you can run simple in-house experiments using employees, customers, clients, friends, or family members to determine how different users navigate through the site. As I mentioned in Part 2, I find this to be a highly effective method of data collection for Web sites that have never been live, as well as an effective approach for validation of data collected using other methods.

Modeling User Patterns with TestStudio

TestStudio gives us numerous ways to create and model virtual testers. There are so many, in fact, that it's sometimes difficult to decide which method to use. The rest of this article will describe and demonstrate the three methods for modeling user patterns that I've personally found the easiest and most universally applicable, using the online bookstore as our example. The methods are as follows:

- **Entire path scripts method**: Record separate scripts for each navigation path and assign them the proper percentages using suites.
- **Path segments method**: Record separate scripts for each horizontal line segment on the graph and organize them properly using suites.
- **Smart script method**: Create smart scripts within your recorded scripts to handle minor exceptions to the major navigation paths.
-

These methods can be used in conjunction with one another when creating more complex user patterns. Each method has inherent strengths and weaknesses, which I'll discuss.

Entire Path Scripts Method

The easiest of the three methods to describe and execute is the entire path scripts method. This method involves recording separate end-to-end scripts for each possible navigation path and then creating a suite to execute those scripts in their proper distribution. Figure 2 shows what the scripts would include for the three possible paths to choosing a fiction book.

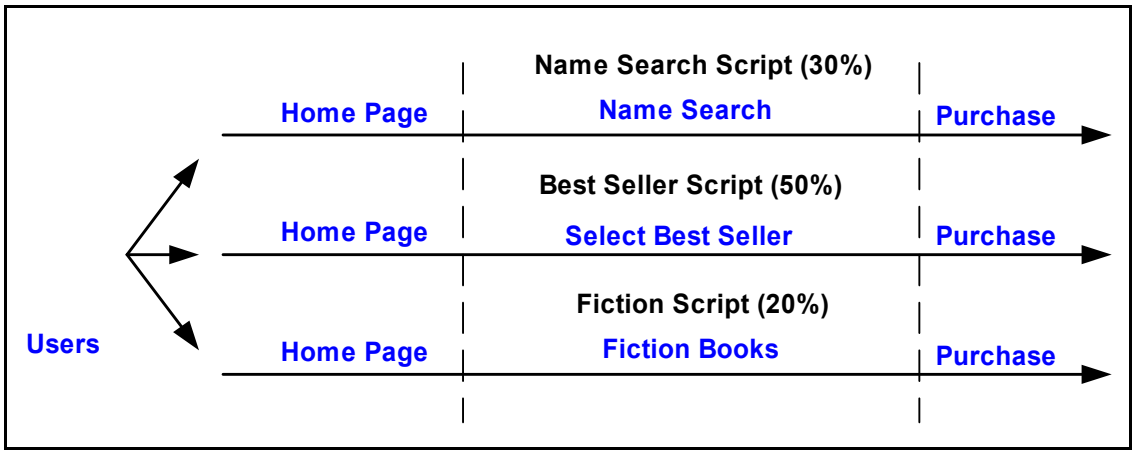


Figure 2: Using entire-path scripts to represent the "choose a fiction book" paths

I'm sure you can see that recording these three scripts would be fairly straightforward. Notice in the diagram that the Home Page and Purchase activities are duplicated in all three scripts, a fact that I'll mention again in my discussion of the other methods. Figure 3 shows the TestManager suite that would be used to execute the scripts in their proper ratios.

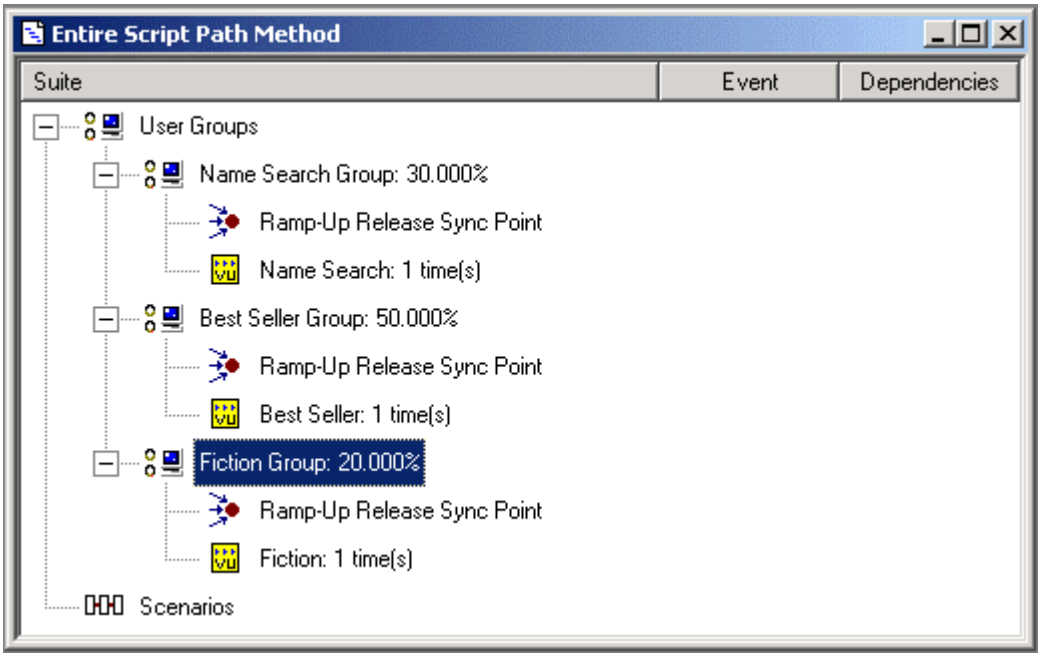


Figure 3: TestManager suite to execute the entire-path scripts

This method is a good way to model simple user patterns, where few scripts are required to cover all the possible user paths. Further, it doesn't require any manual modifications to the recorded scripts to model user patterns. Possibly the biggest advantage of this method is that because each user maps directly to a single script, it's easy to handle security issues when testing against sites with secure session IDs or cookies. (I'll say much more about handling secure session IDs in a later article.)

On the other hand, this method has some significant drawbacks. First, duplicated functionality across scripts means more overhead, so you can generate fewer virtual testers with your current test environment. If your test lab is like mine, you need to conserve all the resources you can on those big tests. You certainly don't want the test environment to be responsible for poor performance results because the testing hardware has become a bottleneck. Personally, I try to keep my total number of scripts to fewer than 12 per suite in most situations; when I have more than 12, I think twice and consider the options. The better the Master Station and Agents hardware you have, the more scripts you can have in a suite without affecting your test results.

The second, similar, drawback involves issues surrounding data that needs to be varied. Some of the redundant components in the scripts will probably involve data (such as login information) that needs to be varied by a virtual tester, likely using datapools. This can lead to having either separate or very similar datapools for each script, and/or having shared or persistent variables to coordinate the use of the data. These activities can yield huge amounts of overhead if not managed properly. In general, it's a good idea to eliminate persistent variables where possible and minimize the number and size of datapools used by varying only data that truly needs to be varied.

Finally, if something changes in either the Home Page or the Purchase section of the scripts, modifications will need to be made to all three scripts. If the changes are major, all three scripts will need to be rerecorded. This shouldn't be a problem if you're dealing with a production application that won't change during the testing effort; but if the application you're testing is still under development, this is a consideration.

Path Segments Method

The path segments method is also fairly straightforward. This method involves recording a separate script (also called a split script) for each segment of the path, as shown in Figure 4. These scripts are then organized into suites to be executed in their proper orders and distributions.

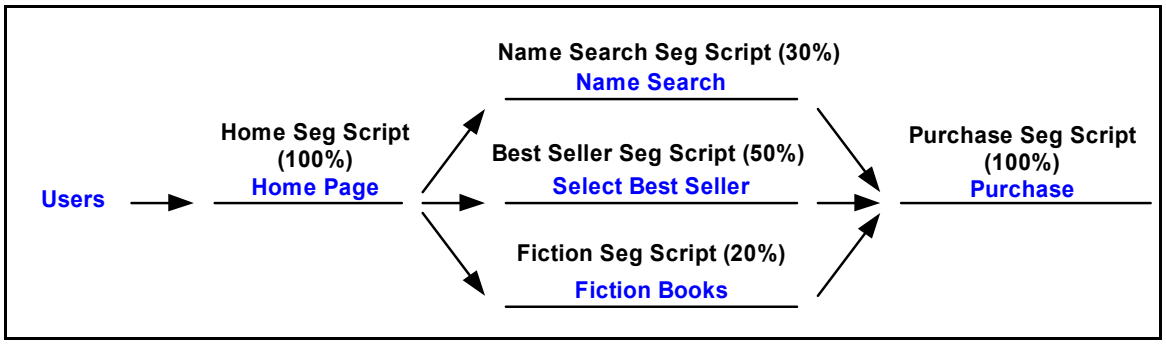


Figure 4: Using split scripts to represent the "choose a fiction book" paths

As you can see, this method involves two more scripts, but no activities are duplicated across scripts. Figure 5 shows one way these scripts can be organized in a TestManager suite to represent our bookstore navigation model.

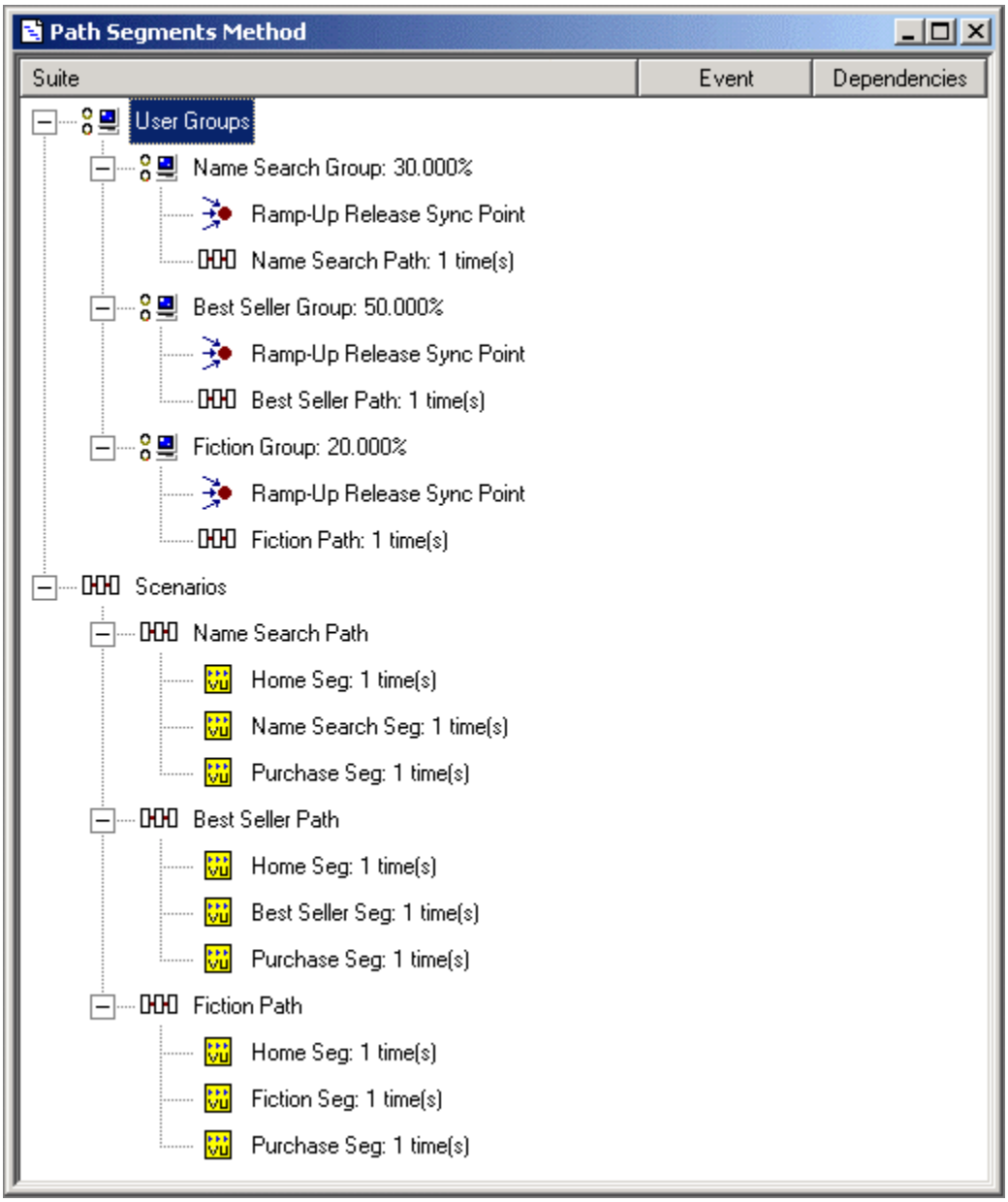


Figure 5: TestManager suite to execute the path-segment scripts

This method is perfect for simple user patterns requiring few scripts. Although the method uses more actual scripts, it doesn't require more overhead than the entire path scripts method. Five actual scripts are generated and processed for our online bookstore example, but they'll each generate less overhead than an entire-path script and will only be accessed in memory one at a time, so the total overhead is about the same. The real advantage of the path segments method is that using nonredundant script segments eliminates the possibility of multiple scripts accessing one datapool or the need for multiple similar or identical datapools.

This method also reduces the amount of rerecording required when something changes in one section of the application. This is a huge benefit if the application is being iteratively developed and tested, as with the Rational Unified Process. Once again, if you're testing a stable production application, this consideration doesn't apply.

There are two significant drawbacks to this method: (1) it may require the use of shared variables, and (2) if the scripts and scenarios don't work, debugging them may be complicated and lead to needing to use shared variables. If any information needs to be passed from one script to another for the scenario to execute correctly, that information will have to be passed using shared variables, a complicated topic I'll cover in a later article. When you record split scripts and put them together in a scenario that combines them in an order different from how they were recorded, it simply works or it doesn't. There's no surefire way to tell before recording if splitting the script will work or not. If it doesn't play back correctly, it's usually an issue related to information that would need to be passed using shared variables.

Smart Script Method

The smart script method allows you to model minor exceptions to the major navigation paths that users will follow. For example, in our online bookstore, some of the customers on each path may open informational pop-up windows, and not all customers will end up purchasing a book. We can model this as shown in Figure 6.

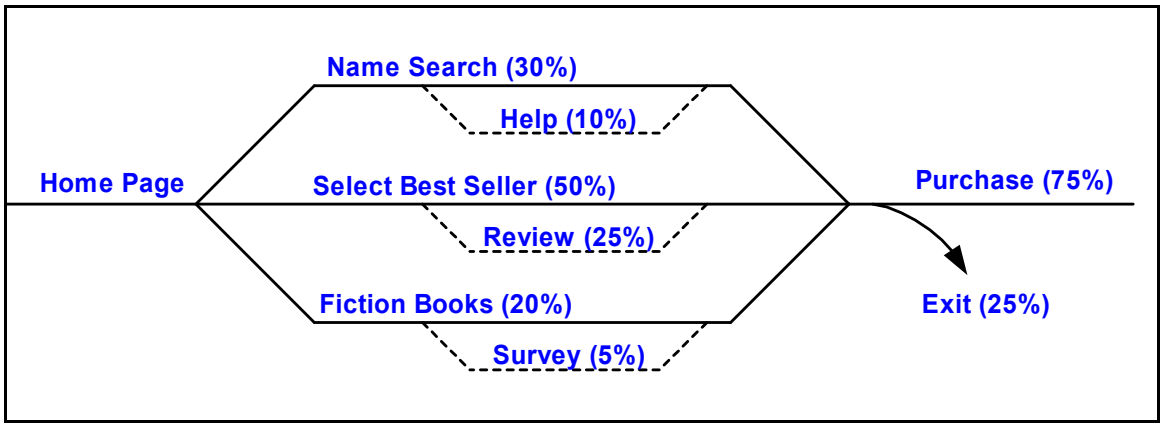


Figure 6: Main navigation paths to choose a fiction book, with minor exceptions added

The dotted lines indicate deviations from the main navigation path. Thus, the graph shows that 10% of those following the Name Search path will access the help screen, 25% of those following the Select Best Seller path will look at reader reviews, and 5% of those on the Fiction Books path will complete a survey. Furthermore, 25% of all customers will exit without purchasing a book.

With this method, we'll be able to keep our suite models but will have to rethink our scripts. Luckily, by handling the optional script deviations programmatically, we can avoid having to add any more entire-path scripts, split scripts, or user groups to the suite. We'll simply add some code to our split scripts to handle the deviations. This method does assume some comfort in writing basic code in C, but I'm going to walk you through this so you can cut and paste if you have no programming experience.

Let's look at how we would alter the Name Search Seg script to allow for the scenario where the customer clicks the Help button and reads and closes the pop-up window before clicking the Search button. Listing 1 shows a portion of this script (just timers and the header section) with code added to allow for this scenario.

```
/*
->-> Session File Information <-<-
*/
#include <VU.h>
{
push Http_control = HTTP_PARTIAL_OK | HTTP_CACHE_OK | HTTP_REDIRECT_OK;
push Timeout_scale = 200; /* Set timeouts to 200% of maximum response time */
push Think_def = "LR";
Min_tmout = 120000; /* Set minimum Timeout_val to 2 minutes */
push Timeout_val = Min_tmout;
DP1 = datapool_open("Name_Search_Seg");
datapool_fetch(DP1);
push Think_avg = 0;
start_time ["View Search Page"];
stop_time ["View Search Page"];
start_time ["View Help"];
stop_time ["View Help"];
start_time ["View Search Results"];
stop_time ["View Search Results"];
}
```

Listing 1: Portion of Name Search Seg script accessing the help screen, as recorded

So how do we simulate customers accessing the help screen only 10% of the time? Simple. We select a random number between 1 and 10 every time the script is called (for each virtual tester), and if the number is 1, we go to "View Help"; otherwise, we skip it. See Listing 2.

```
/*
->-> Session File Information <-<-
*/
#include <VU.h>
int percent; /* Declare the variable that will hold the random number */
{
push Http_control = HTTP_PARTIAL_OK | HTTP_CACHE_OK | HTTP_REDIRECT_OK;
push Timeout_scale = 200; /* Set timeouts to 200% of maximum response time */
push Think_def = "LR";
Min_tmout = 120000; /* Set minimum Timeout_val to 2 minutes */
push Timeout_val = Min_tmout;
DP1 = datapool_open("Name_Search_Seg");
datapool_fetch(DP1);
push Think_avg = 0;
start_time ["View Search Page"];
stop_time ["View Search Page"];
percent = uniform(1,10); /* Select a random number between 1 and 10 */
if (percent = 1) /* If the selected number is 1, view help */
{
start_time ["View Help"];
stop_time ["View Help"];
} /* If the selected number is not 1, script resumes here */
start_time ["View Search Results"];
stop_time ["View Search Results"];
}
```

Listing 2: Portion of Name Search Seg script, modified to access help screen 10% of the time

The other three optional paths would be coded the same way.

The smart script method is very useful for handling any activity that's not required to complete the overall script and ends at the same point that it starts. This makes pop-up windows with no data entry ideal candidates to be modeled this way. Other good candidates for this method are first-time user registration processes, which generally start and end at a login page but require a first-time user to enter personal data and/or select a username and password. Obviously, not everyone navigating to a site will be a first-time user, but some will, and that should be accounted for in your scripts.

Scripting this way is faster and takes up much less overhead than recording and executing additional entire-path scripts or split scripts. This method also minimizes the need for redundant recorded code. When you use it in concert with a combination of entire-path and split scripts, you can model just about any user pattern that you can navigate or draw.

The drawbacks to this method show up when you try to play your smart script back. If during your initial recording some sockets were opened outside your `if` block and referenced inside it, that will cause the script to fail when the contents of the `if` block aren't executed. The contents of this block could also generate a parameter that will be appended to the URL and will appear in the URLs after the block, whether you execute the commands inside the block or not. Several other similar things may happen when you play the script back. Always ensure that you play back any smart script with and without the code in the `if` block being executed; this way, the script will play back without errors and the log files will show the expected results at least once so you'll have something to evaluate potential errors against. If you experience one of the problems I just mentioned, you can debug it, but it's beyond the scope of this article to explain how. Generally, activities that occur in independent browser windows don't have these problems.

Which Method Should You Use?

For the most part, deciding which method to use to model user patterns is just a matter of personal preference. As I've outlined, each has both good and bad points, and you'll have to decide for each situation which method will work best. When testing a complex site, I often use a combination of all three scripting methods.

That said, here are some guidelines to follow when choosing a method.

Use the entire path scripts method for sites with

- limited redundancy among navigation paths
- limited datapool redundancy among navigation paths
- secure sessions
- data passed as parameters in the URL

Use the path segments method for sites with

- significant redundancy among navigation paths
- significant datapool redundancy among navigation paths
- limited security
- data not passed as parameters in the URL

Use the smart script method for sites that have

- optional navigation paths that start and end on the same page
- links that open a second browser window (pop-ups)

Now You Try It

To demonstrate the ease and usefulness of the three methods I just described, I'm going to walk you through an example of applying them to create models of user patterns for the Washington Metropolitan Area Transit Authority Web site (www.wmata.com). I chose this site because it has the necessary properties to demonstrate the methods, it doesn't require you to buy or register for anything, and it doesn't cause me to inadvertently provide free advertising or violate copyrights. Of course, you won't have any of these issues if you're coordinating your tests with the owners of a site/application. You can also try these methods out on your own favorite mostly static Web site with pop-ups.

I assume that you already know how to record and play back a VU script and how to insert timers while recording, and that you're at least familiar with the basic concepts of creating a suite using TestManager. I also assume that you read Part 2 of this article, and therefore I've left out any discussion of user delay times. The exercises that follow are designed for one virtual tester (or, in some cases, two). Don't execute large numbers of virtual testers against the WMATA site — if you did, it could be interpreted as a denial-of-service (DOS) attack, which is a federal offense, and I am *not* going to bail you out or take responsibility. (This has been a public service announcement.)

We'll begin by assuming that we've used our intuition or best guess to identify the user patterns shown in Figure 7.

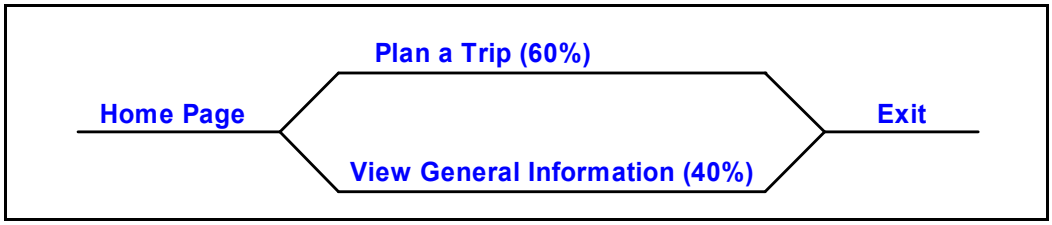


Figure 7: User paths through the WMATA Web site

Entire Path Scripts Method Exercise

Using the entire path scripts method to model our user paths, we'll need the two scripts graphically depicted in Figure 8.

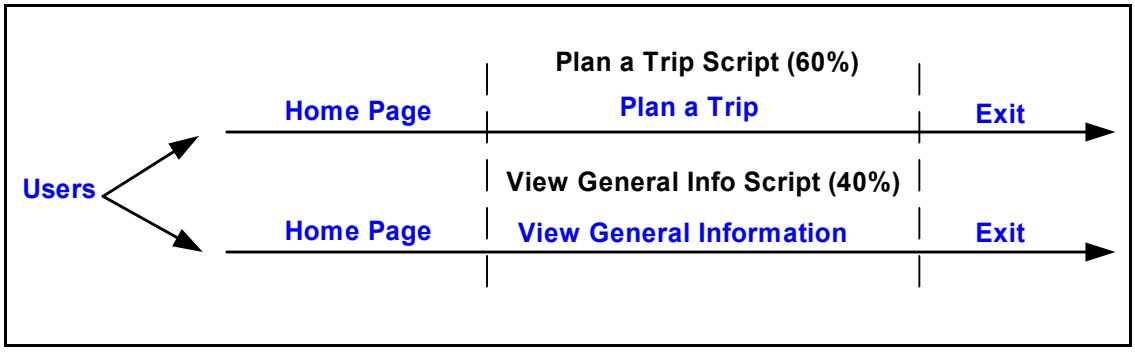


Figure 8: Entire-path scripts for representing WMATA user paths

We'll record these two navigation paths by clicking the appropriate links, and adding comments, timers, and delays, as discussed in Part 2.

For the Plan a Trip script, we'll plan a trip from the Vienna metro stop to the Smithsonian metro stop, taking us from the farthest point west on the metro rail and ending on the Mall in Washington, D.C., amid all the famous museums (which really do offer free admission). To record Plan a Trip, follow these steps:

- Start recording at the home page (www.wmata.com).
- Click “System map” in the “Riding Metro” section on the right side of the screen.
- Click the Vienna stop (last stop on the far left of your screen) on the Orange Line.
- Click “Stations” in the “Riding Metro” section on the right.
- Scroll down and click the link for the Smithsonian stop.
- Stop recording.

For the View General Information script, we’ll check out the work being awarded by the WMATA. To record View General Information, follow these steps:

- Start recording at the home page (www.wmata.com).
- Click “About Metro” in the “About Metro” section on the right side of the screen.
- Click the “Metro police” link.
- Click “Solicitations/awards” in the “Metro B2B” section on the right.
- Stop recording.

Now all we have to do is create a suite to execute this scenario. From TestManager, we choose Suites > New Suite > Blank Performance Testing Suite. Then we add two user groups with User Groups > Insert > User Group. We name the groups Plan a Trip Group and View General Info Group and assign them scalable percentages of 60% and 40%, respectively. Next we insert the scripts into these groups by right-clicking on the group, choosing Insert > Test Script, and selecting the appropriate script. We save the suite (Figure 9).

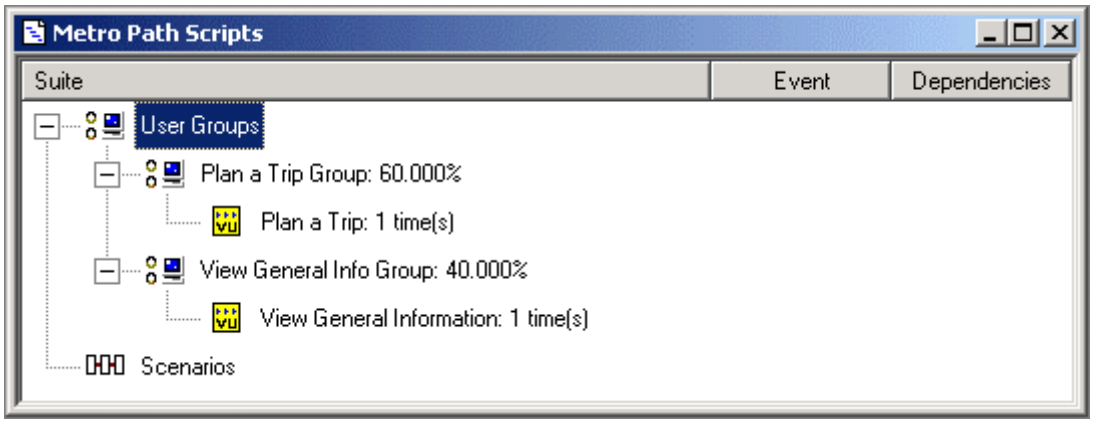


Figure 9: TestManager suite to execute the WMATA entire-path scripts

Simple enough? Good, let’s move on.

Path Segments Method Exercise

In this exercise, we'll record three split scripts, as depicted in Figure 10.

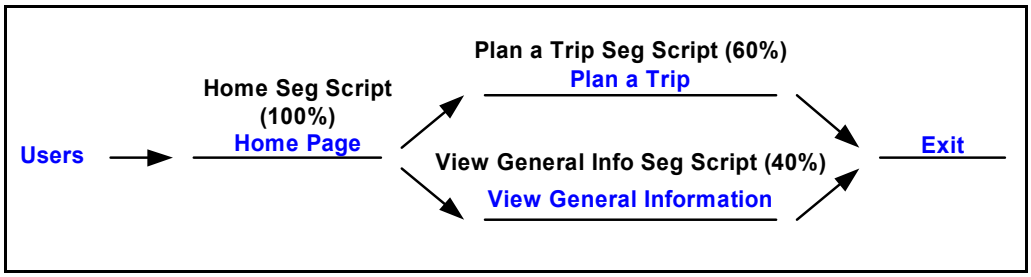


Figure 10: Split scripts for representing WMATA user paths

This method is also fairly simple. We'll start by recording the Home Seg script and the Plan a Trip Seg script in one recording session, and then we'll record the View General Info Seg script in another session. We'll add the same comments, timers, and delays in these scripts as in the last exercise.

First we open the home page (www.wmata.com) and split the script, either by choosing Record > Split Script on the Robot menu bar, or by clicking the Display Session Insert Toolbar button on the floating toolbar and then clicking the Split Script button on that toolbar. (If you're planning to use this method regularly, you can customize the floating toolbar to include the Split Script button.) We'll name the script we just finished recording Home Seg.

To record the next script segment, we proceed as follows:

- Click "System map."
- Click the Vienna stop.
- Click "Stations."
- Click the link for the Smithsonian stop.
- Stop recording and name this script Plan a Trip Seg.

There are two ways to record the View General Info Seg script based on whether we're recording using the Network or API recording method. (A discussion of these two recording methods is beyond the scope of this article. If you're unfamiliar with the differences between them, please refer to the documentation that came with your software.) With the API recording method, we begin recording, open the home page (www.wmata.com), split the script, and when prompted, name this script Throw Away or something equivalent, because we won't be using it.

To record the script segment:

1) Click "About Metro."
2) Click the "Metro police" link.
3) Click "Solicitations/awards."
4) Stop recording and name the script View General Info Seg.

If we're using the Network recording method, we can navigate to the home page before starting our recording, thus eliminating the Throw Away script. We open the home page (www.wmata.com), start to record, and follow the steps above.

To create a suite to execute these scripts (see Figure 11), we simply follow the same basic instructions as in the previous exercise. We could also put the scripts into scenarios and add the scenarios to the user groups. In this case, the methods are interchangeable.

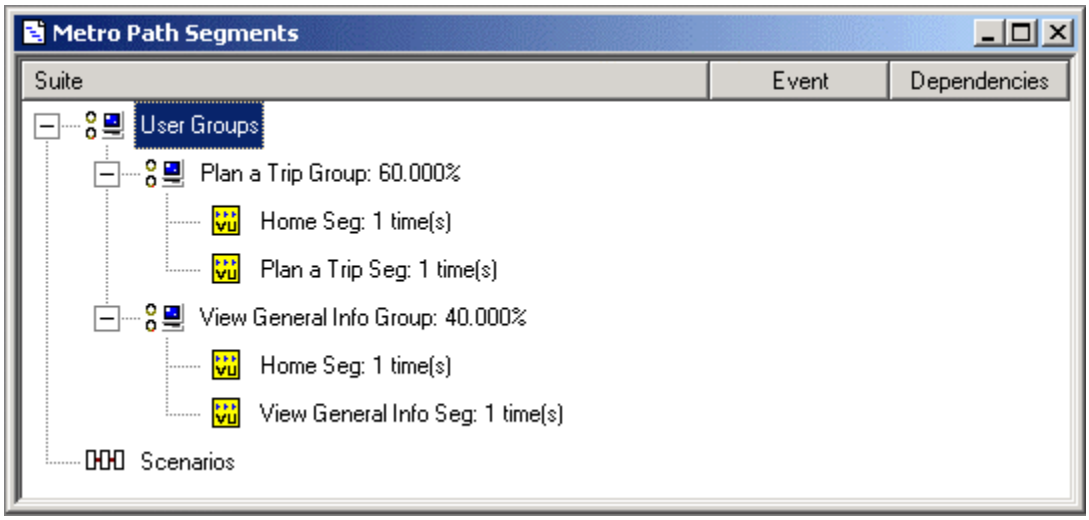


Figure 11: TestManager suite to execute the WMATA split scripts

Now on to the fun stuff!

Smart Script Method Exercise

If you've been doing the exercises, you may have noticed on the WMATA Web site that on the last page of our Plan a Trip path there's a link to determine how much it costs and how long it takes to travel between metro stops. You may even have clicked this link out of curiosity. Let's add this optional path to our Plan a Trip script, as shown in Figure 12. I've guessed that about half of the users planning a trip will want to know the trip cost and times.

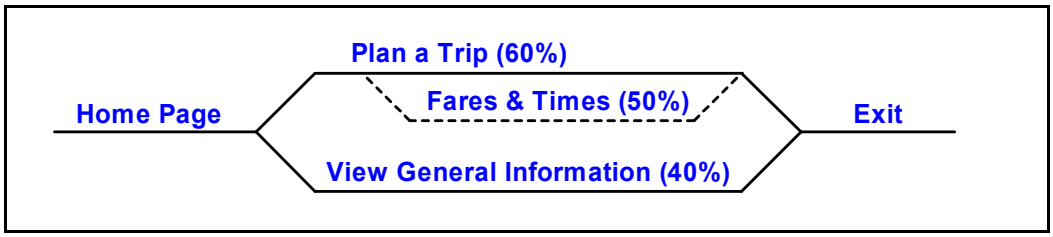


Figure 12: WMATA user paths with optional path added

For purposes of this exercise we'll focus on the Plan a Trip section of this model. First we'll record a script (using the entire path script method, though we could also use the path segment method) including the Fares and Times section.

For the first part of the script, we'll follow the same steps as earlier. We'll start at the home page (www.wmata.com), click "System map," click the Vienna stop, click "Stations," and scroll down and click the link for the Smithsonian stop. Next we'll record the new part of the script. We'll click the "Fares & travel times between stations" link (notice that this brings up a new browser window, making this the perfect place to use the smart script method), choose "Vienna" from the drop-down list, click the Get Fare Data button, click the Close button after viewing the results, and then end recording. (If you want, you can view the entire script as recorded by me.)

Now we'll add the custom scripting. After modifying the delays as described in Part 2, we'll add code to the script to allow for the scenario where the user seeks fare information. Listing 3 shows a fragment of the script as modified. (If you want, you can also view the entire script as modified.)

```
#include <VU.h>
int percent; /* Declare the variable that will hold the random number */
{
push Http_control = HTTP_PARTIAL_OK | HTTP_CACHE_OK | HTTP_REDIRECT_OK;
push Timeout_scale = 200; /* Set timeouts to 200% of maximum response time */
push Think_def = "LR";
Min_tmout = 120000; /* Set minimum Timeout_val to 2 minutes */
push Timeout_val = Min_tmout;
push Think_avg = 0;
start_time ["Home Page"];
www_wmata_com = http_request ["Metro_T~001"] "www.wmata.com:80",
. . .
http_disconnect(www_wmata_com_1);
stop_time ["Home Page"];
delay(uniform(6000,14000));
start_time ["System Map"];
//set Think_avg = 17525;
www_wmata_com_2 = http_request ["Metro_T~014"] "www.wmata.com:80",
. . .
http_nrecv ["Metro_T~057"] 100 %% ; /* 8052 bytes */
stop_time ["System Map"];
delay(uniform(7000,12000));
start_time ["Vienna"];
//set Think_avg = 6650;
set Server_connection = www_wmata_com_4;
. . .
http_nrecv ["Metro_T~078"] 100 %% ; /* 15581 bytes */
stop_time ["Vienna"];
delay(uniform(9000,13000));
start_time ["Stations"];
//set Think_avg = 23514;
set Server_connection = www_wmata_com_5;
. . .
http_disconnect(www_wmata_com_4);
stop_time ["Stations"];
delay(uniform(9000,13000));
start_time ["Smithsonian"];
//set Think_avg = 13199;
set Server_connection = www_wmata_com_5;
. . .
http_disconnect(www_wmata_com_7);
stop_time ["Smithsonian"];
delay(uniform(9000,13000));
percent = uniform(1,10); /* Select a random number between 1 and 10 */
if (percent < 6) /* If the selected number is greater than 6, view fares */
{
start_time ["Fares and Times"];
//set Think_avg = 13199;
smartbenefits_wmata_com = http_request ["Metro_T~113"]
"smartbenefits.wmata.com:443",
. . .
http_disconnect(smartbenefits_wmata_com);
stop_time ["Fares and Times"];
delay(uniform(12000,15000));
start_time ["Get Fare Data"];
//set Think_avg = 19268;
smartbenefits_wmata_com_1 = http_request ["Metro_T~119"]
"smartbenefits.wmata.com:443",
. . .
http_disconnect(smartbenefits_wmata_com_1);
stop_time ["Get Fare Data"];
}
pop [Think_def, Think_avg, Timeout_val, Timeout_scale];
```

Listing 3: Portion of Plan a Trip script with code added to access fare info

As in our bookstore example earlier in the article, the method simply involves declaring a variable, assigning it to a random number value, and executing the section of the script based on the random value selected.

Note to advanced users: The smart script method is as easy as it sounds, but only when the navigation inside the `if` block is in a separate browser window. This scripting method can be used for any optional navigation that ends in the same place as it begins, but individual sockets (`set Server_ connection` and `http_disconnect` commands) will need to be managed manually. It would double the length of this article to discuss how to do that in any detail; suffice it to say that in my experience, unless a very strong case can be made to do this, it's usually less painful to just record an additional script to cover that portion of the user model.

Summing It Up

This article discussed three methods to model individual user patterns that can be used in concert with the power of suites in Rational TestManager. You can use the entire path scripts method, the path segments method, and the smart scripts method either alone or in combination to model just about any individual user patterns effectively. The next article in the series will detail how to use TestManager to create suites to accurately model entire communities of users, rather than just individual users.

References

- "Website Stress-Testing" by Serdar Yegulalp (ExtremeTech Web site)

About the Author

Scott Barber is a System Test Engineer and Quality Assurance Manager for AuthenTec, Inc. and a member of the Technical Advisory Board for Stanley-Reid Consulting, Inc. With a background in network architecture, systems design, database design and administration, programming, and management, Scott has become a recognized thought leader in the context-driven school of the software testing industry. Before joining AuthenTec, he was a consultant specializing in Performance Testing/Analysis, a Company Commander in the United States Army, a DBA and a Government Contractor in the transportation industry.

Scott is a co-founder of WOPR (the Workshop on Performance and Reliability), a semi-annual gathering of performance testing experts from around the world, a member of the Context-Driven School of Software Testing and a signatory of the Agile Manifesto. He is a Discussion Facilitator in the Rational Developer Network public forums and a moderator for the Performance Testing and Rational TestStudio related forums on QAForums.com. Scott's Web site complements this series. Please visit it to find more detail on some topics and view slides from various presentations he's given recently. You can address questions/comments to him on either forum or contact him directly via e-mail.

AuthenTec, Inc. is a leading semiconductor company providing advanced biometric fingerprint sensors to the PC, wireless, PDA, access control and automotive markets. AuthenTec's FingerLoc and EntréPad product families utilize the Company's patented TruePrint™ technology, the first technology capable of imaging everyone under virtually any condition.

Stanley Reid Consulting, Inc. is a small, niche consulting company that focuses on IT organizational improvement and recruiting and staffing of technical experts. Stanley Reid Consulting helps your team achieve consistent, successful delivery of IT projects through the following services: IT Organizational Improvement Consulting, Expert Supplemental Staffing, Technical Recruiting & Placement.