# Realism in Testing

# Lessons Re-learned about Modeling for Trustworthy Conclusions

**By Scott Barber**

I have spent most of the past year conducting a performance testing engagement for my first large Commercial Off The Shelf (COTS) application (PeopleSoft 8.x). During the course of this year, I have learned many lessons about the quirks and uniqueness of testing both COTS applications in general, and PeopleSoft 8.x in particular. This paper, however, focuses on one lesson that I had to re-learn in a new context. I have chosen to share this experience because it was particularly poignant to me, and because I hope that it will keep you from falling into the same traps I found myself in.

## Initial Lesson

Before I share my experience with you about my lesson re-learned, please allow me to briefly share with you the initial lesson that I didn't heed. About 4 years ago, I was testing a financial planning application. Being fairly new to Performance Testing, and thinking that many of the models I had seen others use to supposedly represent actual production workload were woefully inadequate, I decided that I was going to model that application "correctly". By the time I was done, I had developed the precursor to what I now refer to as the User Community Modeling Language, had pages and pages of models that then took me roughly 4 months to script accurately, and had slew of reusable functions for my load generation tool of choice (Rational).

Unfortunately, when I finally started executing the tests, I ended up having to cut out more than half of the user paths due to resource limitations of the load generation environment at high loads. In an attempt to validate the results of this "reduced model" I executed the reduced model at lower loads – only to find out that the results were statistically identical to the previous tests. I quickly realized that although I had created a lot of reusable artifacts, I had spend roughly 3 months modeling and scripting activities that added no noticeable value to my tests.

The lesson was simple – don't waste time over-modeling, you'll quickly reach a point of diminishing returns.

Over the next four years, I've come to teach that lesson using what I call the "20/80 plus, plus, plus rule" which states:

"Target the usage model of your system to model the **20%** of the functionality that is exercised **80%** of the time:

**PLUS** mission critical activities

**PLUS** known performance intensive activities, or areas of known poor performance

**PLUS** high profile activities"

This is obviously not meant to be inflexible or limiting, but rather an easy to remember "rule of thumb" to think about while developing the appropriate model for your specific context. Now let's look at how I violated this lesson.

## Context of the Lesson

When I arrived on the client site, I had a contract for 6 months to do performance testing for a project called "PeopleSoft Financials 8.4 Upgrade". After several weeks of meetings, I came to realize what the project really looked like was seven completely separate projects rolled into one. Each PeopleSoft module to be upgraded had a separate Project Manager, Development Team, and Analyst Team. Five of the modules would share hardware, one had completely independent hardware, was a completely different version of the application (8.3) and only interacted with the first five modules through an overnight batch process that was determined to be out of scope. The seventh module was determined to be completely out of scope because it was actually a Lotus Notes plugin that added insignificant amounts of data to the PeopleSoft database during non-peak times.

In an attempt to organize this matrix of organization, I started meeting with each module Project Team independently. I was continually told that the modules were completely independent of one another – all the way back to the database. I collected requirements separately (99% of users only have access to a single module) and modeled activities separately.

As I went forward with my modeling effort, I built models for each module separately with my modeling considerations in mind. I have included the graphical depiction of one of my models from roughly three months into the engagement below.

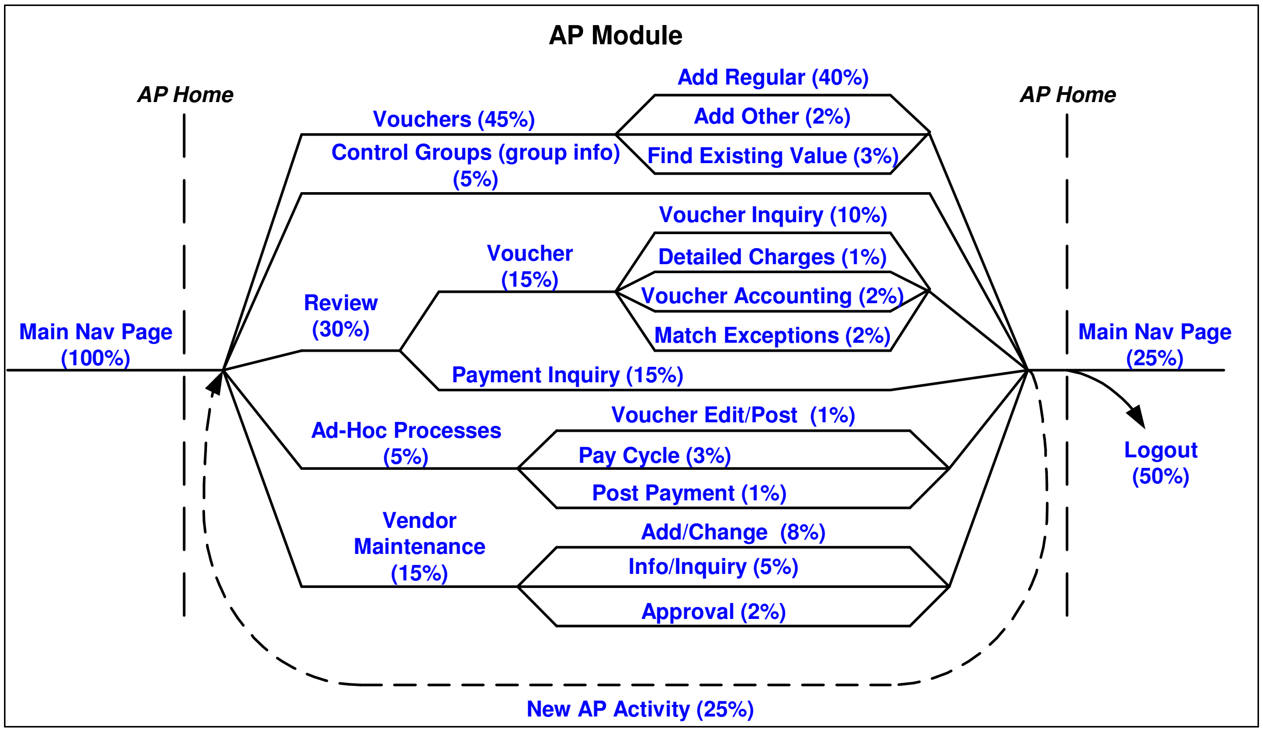


**Figure 1 – AP Module Initial Model**

Figure 1 is representative of all six models I built – not the simplest, not the most complex. A point worth noting is that the expected peak usage of this module is under 50 hourly users. If you are not familiar with either PeopleSoft or the User Community Modeling Language, this diagram simply represents user activities and the frequency they are exercised. Each listed activity has an associated navigation path, data, etc and average roughly 10 steps (in this case pages or tabs) each.

Due to a string of issues around test data and environments, these models ended up testing a very limited set of data. Both the business analysts and developers assured me this data was representative, but I continued to list it as a risk.

I spent the next 2.5 months scripting these models (*note* there were a lot of technical issues that slowed down this effort, but this was still an excessive percentage of the overall project). At the end of that time, I benchmarked each module separately, then I put the five modules together for a collective load test, which showed the same things as the modules had shown independently, which was that some Searches, Queries, Reports and Authentication were slow.

## Lesson Left Behind

It was at this point that I began to realize that I had made some mistakes. Luckily for me, it was at about this point that the project was put "on-hold" for about 4 months. During that time I stayed with the same client and tested other applications – mostly other modules of PeopleSoft Human Resources that were being implemented in a different environment all together. During these engagements, the mistakes became very clear to me. I categorized them into four specific mistakes.

Mistake #1 was that I had allowed myself to get sucked into the corporate hierarchy and started looking at the modules as separate systems. In fact there were two separate systems, but not six.

Even in my rule I state…

"Target the usage model of your ***system***…"

This is actually a lead contributor to Mistake #2 which was that I allowed the module teams to over emphasize the "pluses" in the model. Looking back over the models, I realized that with fewer than 50 hourly users, during the course of an hour long test any activity at 2% or less may or may not even be executed if I allowed the percentages to stand. What actually happened, however, was that I "forced" those activities to occur during each test execution to ensure I got at least some measurements. When I looked at the model as a whole, I realized that if I eliminating all of the activities that occur 2% or less, the total number of activities in the model shown in figure 1 would be cut in half. In summary, Mistake #2 was that more than half of my model represented less than 10% of the expected usage, there was no clear indication that ANY of those activities counted in a PLUS category and I was forcing them to occur more often than the model predicted, just to get more measurements, thus taking virtual users away from more important activities.

My third mistake became clear to me when I finally took the models for the five different modules and put them into a single picture. I immediately saw that each of the five modules had Searches, Queries, Reports and Authentication activities modeled. Naturally, the Authentication part needed to stay associated with every user but, mistake #3 was that the Searches, Queries and Reports were the same activities over and over just with different navigation paths to get to them and different data, thus a whole lot of extra, redundant scripts to yield exactly the same performance data.

Mistake #4 was that I didn't push the issue of insufficient test data. Developing the test data properly would have both made the tests more accurate and highlighted Mistake #3. While there always seemed to be valid reasons to put the test data risk on the back burner "for a little while longer", I fell into the trap of not realizing that "a little while longer" had become "forever."

## Lesson Remembered

When the project restarted recently, we started by revalidating the old scripts since several new patches and bundles had been applied to the application, and several significant defects had been resolved. Even though I thought our old models were over complicated and misrepresentative from a whole system perspective, I thought this was a reasonable first step to see what had changed during the "on-hold" period. Once this revalidation was completed, however, the project manager wanted to "revalidate the models and conduct a production simulation." As it turned out, there had been a LOT of development and rework that had occurred during the "on-hold" period, so as the module teams started drawing big red X's through the old models and drawing new ones, I called a halt and brought the entire team together to discuss a "more efficient approach" to achieving a production simulation.

Luckily, during the "on-hold" period, an overall application architect had been hired, so when I presented my ideas about our models being too broad (too many user paths) and too shallow (not enough data variance) – and proposed a new model that treated all of PeopleSoft Financials as a single system, he was there to validate my assumptions that most of the low percentage user paths were not really a value add in terms of types and volumes of load from an application perspective, and that the Searches, Queries, and Reports really all belonged in one bucket as the only difference in the model was a few static HTML pages between the Home Page and the actual Search, Query or Report and the actual test data. (We had previously tested Load Balancing, Authentication, and Web Server components separately and extensively and shown that they were not causing bottlenecks.) He was also able to help me put renewed focus on creating appropriate test data.

Figure 2 shows the new model that is being used for testing currently (The testing will continue through Sept 17, but this paper was due Aug 22. I will update the paper upon conclusion of the testing). The activities with the red stars (*) are activities that include search functions.

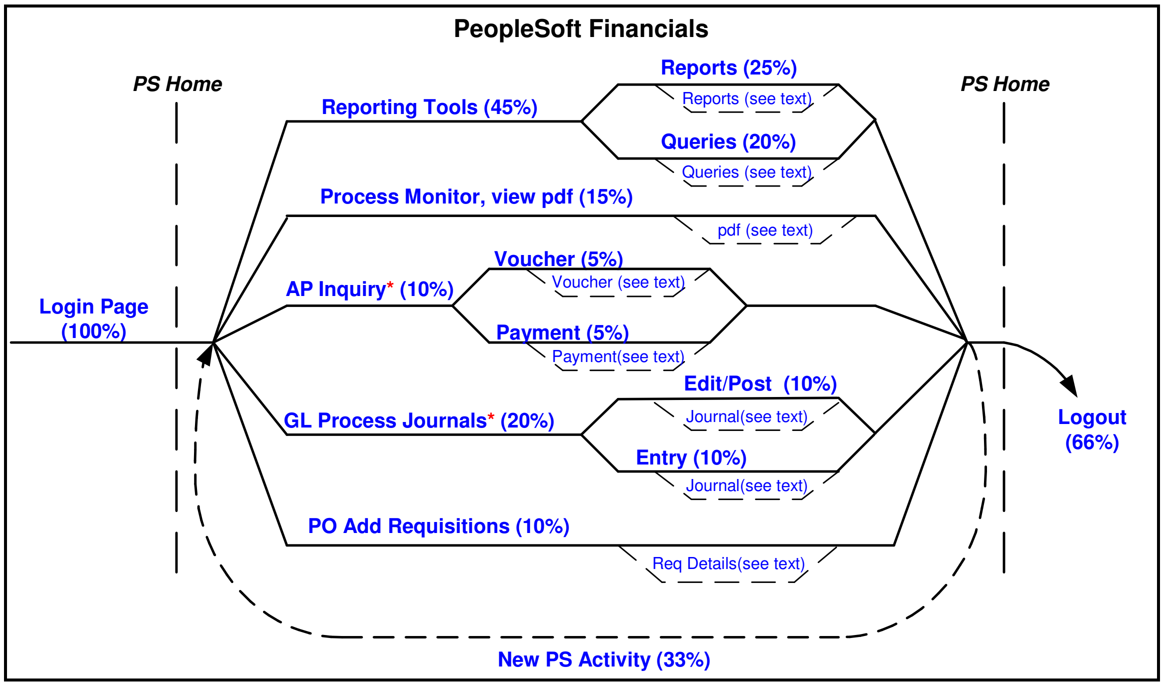
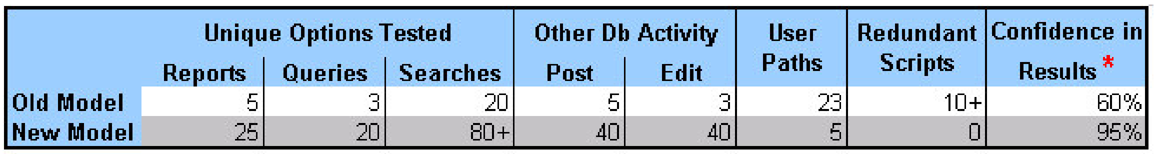


**Figure 2 – Financials Final Model**

## Results of Lessons Applied

Using the new model, we were able to see immediately what we all had suspected from the beginning, that each of the areas of concern (Search, Query and Reports) were slow, bottlenecked at high loads and needed to be tuned. The new model allowed us to quickly isolate those areas for tuning. The main difference between the old and the new models was that instead of the results appearing that particular Searches, Queries and Reports were slow, it clearly showed that those activities were areas of performance concern overall. We are currently in the process of gathering information about and tuning these areas. Items of concern that we identified quickly included, poorly optimized queries, an inadequate report server, and improper and/or missing table indexes.

Simply put, by eliminating the seldom used activities that were being "forced" to occur more frequently simply to add measurements, we were taking virtual users away from the critical activities and creating an unrealistically low load on those activities. Additionally, the limited volume of test data used for the early models had allowed for caching on the application, datatbase and reporting server to skew results. Figure 3 shows a sampling of some statistics between the old model and the new model.



| | Unique Options Tested | | | Other Db Activity | | User Paths | Redundant Scripts | Confidence in Results * |
|---|---|---|---|---|---|---|---|---|
| | Reports | Queries | Searches | Post | Edit | | | |
| Old Model | 5 | 3 | 20 | 5 | 3 | 23 | 10+ | 60% |
| New Model | 25 | 20 | 80+ | 40 | 40 | 5 | 0 | 95% |

**Figure 3 – Selected Results Old vs. New Model**

* Confidence in results determined by questionnaire completed by stakeholders. Actual production validation will be documented when tested.