DEVELOPING THE TEST WORK LOADS: USE OF OPERATIONAL PROFILES

Ross Collard
August 22, 2003, Version 0.3

Introduction

This presentation is based on my experiences in building, maintaining and using operational profiles on three projects. These projects had different characteristics and varying degree of success with operational profiles. Using the right mix of demands (the test work load) is vital to producing realistic test results. There are several issues involved:

- How to design a benchmark or test work load.
- Allowing for background noise.
- Developing peak loads.
- Developing stress loads.
- Using operational profiles, and using industrial engineering to formulate them.
- Handling load variations during testing.
- How to determine the duration of the performance test.
- Weighing the pros and cons of operational profiles versus benchmarks.
- How to decide if we can re-use test loads.
- How to select test suites from libraries of test cases.

Test Load Design

Performance testing is an area that is notorious for lying, especially by vendors who are striving for competitive advantage. The test load is the representative work load used during the performance test run. It should be representative of the likely real-world operating conditions. Sometimes people use the term test work load `B` this is the same thing as a test load. The term benchmark means a standard test load which the testers use to compare performance across different systems, system versions or support environments. The benchmark can be an industry-wide standard or our own into private standard. Based on how we design the test load or benchmark, any turkey of a system can look super slick and any slick system can run like a snail. If we do not know what the users are doing, there is no point in conducting a performance test!

DEVELOPING THE TEST WORK LOADS
(Continued)

Performance testers need to determine the suitable test loads to use (i.e., representative models of the average load, peak load and possibly also peak-peak load ("Tsunami wave testing").

It is not enough to test under average conditions, but also to test under peak conditions. For example, a peak work load may be only 15% higher than an average one for a particular work situation. We cannot assume, though, that the system will degrade gracefully and run 15% slower under peak conditions than it does under average conditions. The performance characteristics probably do not vary linearly with load, and the system may even break under peak load instead of degrading gracefully.

How do we design a realistic test load? The best way is good old-fashioned industrial engineering. In other words, we need to understand what is actually happening in the system's planned environment, in terms of the mix of demands on the system and the frequency of occurrence of the various demands. In other words, we need to build an operational profile or usage model.

For example, suppose we need to develop a test load for an on-line banking application. We could count the actual mix of transactions that occur in a typical 15 minutes of on-line banking activity. Let=s say that the bank processes 132 account balance inquiries, 83 ATM cash withdrawals, 1 new account opening and 3 customer name-and -address changes in a typical 15 minutes of banking activity. We would then design the test load to replicate this same mix of transactions, and run this test load over 15 minutes while we measure the system=s response time and throughput.

If the application is a new one and there is no current activity to count, we need to extrapolate from existing activity with the systems that the new application will supersede.

If there is a state-transition diagram and an operational profile for the system we are testing, then Markov chains are a useful way to design the load needed for testing.

DEVELOPING THE TEST WORK LOADS
(Continued)

Determining the Number of Test Loads Needed

How many different test loads or benchmarks do we need in performance testing? Since the load on the system is continually changing, we can make a case that we need many test loads. However, doubling the number of test loads usually doubles the time and resources needed for testing, and there is a practical limit on how much performance testing we can afford to do.

The answer is that we need a separate test load for each significant and distinct work load the system may encounter.

A reasonably comprehensive performance test project employs a few (and often at least five) test loads: (a) a typical or average work load, (b) a work load which represents the daily period of peak demand on the system, (c) specialized demands, such as the occasional need to print or electronically distribute high volumes of documents, or to back up the database, (d) senior managers= or other high-priority users= utilization of the system, such as for computations of Awhat if?@ queries during key business meetings, and (e) the likely future growth in demand over the next 3 to 5 years.

Determining the Size of a Test Load

A performance test is not complete until we have collected a sufficient quantity of trustworthy information, in order to answer the senior managers= questions about the system performance.

The size of the test load (e.g., the number of transactions or events in a test load) much be large enough to allow for vagaries and variations, but not unnecessarily huge. The questions are:

(a) How large a volume of transactions is needed to give a reliable sense of performance (e.g., measure response precisely enough to meet the target accuracy?

(b) How large a volume is needed to reach steady state behavior, not just the behavior during initial use of the system?

Any volume of transactions or events greater than this minimum adds no useful new information, and so wastes test resources.

DEVELOPING THE TEST WORK LOADS
(Continued)

The sample size needed to achieve a target level of accuracy increases with the variability or degree of inconsistency of the population. The numbers in the earlier table are typical, based on the degree of variability found in most information systems.

Concurrent, Interacting Work

Let=s say that we design a load test which places 1,000 concurrent demands on a system (e.g., there=s 1,000 test cases, each of which performs one task). Let=s also say that these tasks are all independent of each other. Assuming the system has sufficient capacity and no bottlenecks, all tasks can be processed concurrently, and the throughput will be 1,000 and the response time will be 1 (in some appropriate units of measure).

Let=s now say instead that the 1,000 test cases are interdependent. In fact, they all must use the same single-threaded path through a software component. In other words, each task will have to queue up and wait its turn. In this situation, the performance will be radically different B the throughput will be 1 and the response time will be 1,000 in the same units of measure.

To be realistic, the test work load for any multi-user system must include users which interact and possibly interfere with each other. These interactions may affect the system performance differently than a test load with a similar volume of unrelated transactions. Examples of these situations include:

- Two or more users attempt to access the same database record at the same time.
- Two or more users attempt to update the same database record at the same time.
- Two or more users place a load on a shared peripheral, such as a printer.
- Two or more users are using a common resource simultaneously, such as sending e-mail.

What is a realistic rate of interaction among users? For example, how many users on a network are likely to be sending e-mail simultaneously or printing simultaneously? Depending on the situation, with most business applications and databases the amount of interaction among multiple users ranges from a low of 1% to a high of 20% or more.

DEVELOPING THE TEST WORK LOADS
(Continued)

Background "Noise"

An important factor in a performance test is deciding what other background work load, if any, should be sharing the facilities during the test.
To provide a realistic simulation of the real-world operating conditions, a mix of background activities needs to occur at the same time as the performance test is executing. If we want honest results, we cannot test in an empty box or empty network. The design of the test load includes the simulation of a representative and on-going background activity.

I am including this discussion of background noise, which is also called background load, as a part of performance testing. However, it also plays a useful role in functional or feature testing. The background noise can help find many errors in the interaction among features and in the time dependencies among events.

While simulating the background noise during performance measurement sounds like an obvious good idea, often there are practical difficulties in implementing it. In complex networks, there could be 50 or 100 discrete applications running in background mode. We can spend a considerable effort to compile an inventory of these applications (ideally, of course, an inventory is already available for the asking), identify the system owners, gain a minimal understanding of what these other applications do, and understand their load profiles and resource consumption patterns.

We may need this effort just to understand the background noise; we have not as yet considered how to simulate it. (Generally, the simplest way to simulate the background is to run the mix of applications right in the test environment -- if this is feasible.)

To understand the make-up of the background noise, the first stop by the testers is in the office of the system administrators or network administrators. Hopefully, these administrators will be able to whip out logs of system traffic and resource utilization, and explain them to the testers.

DEVELOPING THE TEST WORK LOADS
(Continued)

With a background load running, it is important to provide methods for monitoring the activities during testing. If a defect occurs in the interaction between the system under test and the background activities, it may be impossible to understand without a comprehensive log of events, together with tools to browse and analyze the log. The monitoring probes themselves will impact the performance, so we need to allow for their impact in performance computations.

With a heavy amount of background activity, the load becomes more suitable for a stress test (defined later) than for a pure performance test.

Questions to address in designing the background noise:

- Should we conduct the performance test in an "empty box", i.e., with no background noise and the hardware and network facilities isolated and dedicated exclusively to the test?

- Or should a "normal" mix of other background work be running simultaneously with the test? If so, how do we determine what this work mix is, and monitor and control the actual demands during the test?

- How do we determine if "the playing field was level" during the test, and ensure comparability from test to test? What if a mix of incompatible demands cause the system to "hang" so that we cannot complete the test and adequately measure performance?

- Should the facilities be tuned to maximize response time and/or throughput for this particular test, or de-tuned to representative typical real-world inefficiencies? Vendors usually tune the system and test environment to show their products in the best possible light.

Tools are available which can simulate concurrent work loads or background noise. As well automated load testing tools, these include tools which gobble up resources. The Microsoft software development toolkit (SDK) contains a utility called Stress that consumes a designated amount of RAM, disk space, system resources, etc., to test application performance under low resource conditions.

DEVELOPING THE TEST WORK LOADS
(Continued)

User Activity Ratios and Blocking Factors

There is a huge difference between measuring performance with a load of 100 on-line users and a load of 100 concurrent users. If we do not understand this difference, we run the risk of seriously mis-designing the test load.

To understand the difference, we first have to consider what the terms Aon-line user@ and Aconcurrent user@ mean. Unfortunately, there is no universally consistent terminology in use, but most people use the terms as follows:

- An on-line user is connected to a network (which may be a client/server network or a Web site), but are not necessarily active at any given moment in time.

- A concurrent user is one who is active at a given moment in time, e.g., is transmitting a message (to be active, this user first has to be on-line).

The user activity ratio is the ratio of concurrent users to on-line users. This ratio is usually in the range of 5% to 15%, though occasionally we see ratios below 1% or above 25%.

A different but related term of importance is the blocking factor or blocking ratio. The blocking factor is a designed-in characteristic of the system, which is intended to exploit the fact that most users are not concurrently active. The blocking factor is the maximum percentage of on-line users that the system can accommodate concurrently.

We can think of the blocking factor as the maximum user activity ratio that the system is designed to service. If the blocking factor exceeds the actual user activity ratio by a comfortable margin, then the provision of services is unlikely to ever be blocked.

A typical blocking factor for a voice telephone network, for example, is 20%. In other words, if a telephone network has 100 subscribers, then up to 20 of them can concurrently have telephone conversations. The 21st subscriber who attempts to make a call is blocked (he or she usually hears a busy signal).

DEVELOPING THE TEST WORK LOADS
(Continued)

A typical blocking factor for an Internet service provider (ISP) is more like 10%. In other words, the ISP provides enough ports on its gateway servers to accommodate up to 10% of its subscribers at the same time. In these two example, the blocking factor is not the ratio of on-line users to concurrent ones, but instead is the ratio of concurrent users to total subscribers (for the voice telephone switch), or the ratio of on-line users to total subscribers (for the ISP).

So why do we care -- what=s all of this got to do with someone who is performance testing? There are two connections:

(1) The performance tester needs to (a) know, and (b) allow for the user activity ratios, when designing performance test work loads.

(2) The performance tester needs to know what the blocking factors are, as part of identifying likely bottleneck and planning for scalability testing (see later for a discussion of scalability testing).

Designing the Peak Test Load

As a rough rule of thumb, a peak load is three times higher than an average load, though it is not unusual to encounter situations where the peak is ten times higher than average or more.

The purpose of a peak test load is to test the system under high load or worst-case conditions. If the average test load simulates 15 minutes of typical on-line banking activity throughout a day, the peak might be the busiest 15 minutes of activity in the day. Or the busiest 15 minutes in a week.

Or the busiest 15 minutes in a typical year. Or century? The longer the horizontal time line, the more spiked or intense will be the peak, and the more demanding will be the test load.

Should we model our test load on the busiest 15 minutes per day, per month or per century? The answer depends on the performance and robustness specs. of the system. If the customers are willing to tolerate one service outage per month (for example) because of a demand overload, then the time duration used to set the peak work load for testing is one month (the same amount of time).

In other words, if the robustness objective of the system, stated in terms of mean time between failure (MBTF), is one month, then the peak test load is set to the busiest 15 minutes of demand in a typical month. On the other hand, if the robustness objective is more stringent, let=s say an MBTF of one year, then the peak test load must be much more spiked, i.e., the test load is set to the typical busiest 15 minutes of demand in a year.

The following example of a peak load test may be just a tall story, though beer-loving sports fans in Seattle swears it is true. When Seattle replaced its Kingdome sports stadium, one concern about the new stadium was its toilet capacity. The stadium officials organized a stress test, where 1,200 toilets were flushed at the same time. (This test required 1,200 people to carry wireless devices which signaled everyone when to flush together.)

The facilities passed this test, but later it was found that there were still problems in live operation. How could this be? Well, during a real game, the visits to the facilities become more frequent as the game progresses, and the (ahem) load per visit becomes heavier. In the test, all the toilets had indeed been flushed, but with zero real load.

Average-to-Peak-Load Ratios

Another method of determining the peak load, which is cruder but simpler, is by rule of thumb.

Fore most client/server networks, average-to-peak-load ratio is about 3 to 1. In other words, to determine the peak load, we simply take an average load and triple it.

As a rough rule of thumb, this tripling corresponds to a robustness (mean time between failure or MTBF) goal of about one week. In other words, the peak load encountered in a week is triple the average demand. Of course, depending on the operational profile for the specific system we are testing, the MTBF could be much longer than or much shorter than one week.

The more variable, inconsistent or bursty the mix of demands on a system, the higher this ratio should be. For example, the variability in Web-based applications is usually higher than with client/server ones. The rule of thumb used by many Web load testers is 10 to 1 or 25 to 1.

DEVELOPING THE TEST WORK LOADS
(Continued)

Designing the Stress (Overload) Test Load

On the face of it, developing a stress load appears dead simple. All we have to do is use a load which is sufficiently heavy that the system will undoubtedly fail, and we can then observe whether it recovers and continues to operate as intended. Or instead of using one extreme load, we can steadily increase the load until we reach the system breakpoint. Unfortunately, stress testing is not as simple as it may first seem.

Using an unrealistically high load in testing may lead to cries of unfairness from the system developers, and possibly from the users (and in vendors, the marketers). They will argue, with some justification, that the stress test is holding the system and its environment to an impossibly high standard, leading to unnecessary over-engineering and delays in system delivery to accommodate additional testing, debugging, tuning and re-work. These doubters may challenge the stress testers to show evidence that the users have ever encountered this extreme level of load, or are ever likely encounter it in live operation.

One counter-argument applies to Aalways-on systems@, such as many commercial Web sites, where the mean time between failure (MBTF) goal is effectively infinite. In this situation, there is no such thing as a load that is too heavy, because the system must be able to handle any load thrown at it, without limit. However, this argument may sound obstructionist to managers who are struggling to meet tight deadlines for system delivery (and the deadlines are invariably tight).

In addition, the test equipment and tools have physical limitations on the work load they can handle, and they can fail or exhibit bizarre behavior well before they reach their thresholds (for example, the theoretical maximum rate of throughput).

A more fundamental limitation of stress testing derives from the nature of system failure. Stress testing is concerned with availability, robustness, recovery and survivability, and the system has to fail in testing for us to examine these aspects.

The problem is that systems can fail in many different ways, especially complex systems, and the expected set of follow-up actions (recovery) may differ based on how the system failed. These modes of failure are often not all identified, and the ones which we do identify are not well understood even by the system experts.

DEVELOPING THE TEST WORK LOADS
(Continued)

The types of system failure include:

- A crash, the cessation of all activity. This is the most dramatic and obvious form of failure.
- Thrashing, where excessive overhead consumes too many processor resources and leads to performance problems.
- Inability to scale.
- Unacceptably poor performance caused by bottlenecks.
- Timing and synchronization problems; which are sometimes called race conditions.
- Memory leaks.
- Buffer and database overflows.
- Excessive levels of dropped or lost data packets.
- Partly disabled operation, where some features continue to work but others do not.

I=ll provide a more extensive list later in this book, in the section entitled: AModes of Failure@.

We could make the system to fail in the test lab in ways which can never happen in live operation (and we testers might not know the difference). Worse, the opposite can happen B the system may fail in live operation in ways which we cannot reproduce in the test lab.

One reason for this inability to replicate the system behavior is that most information about the state of the system is lost at the point of failure. Other reasons are that we can=t re-create the cause of failure with the limited test lab facilities, or that the system behavior in failure it is too dangerous to people or equipment.

More difficult still than trying to replicate the behaviors we have actually observed in live operation, is attempting to predict B before the system goes live B the ways in which it could fail, and what loads we need to cause these different modes of failure.

These considerations lead us into robustness testing, as distinct from performance and load testing. Since understanding the modes of failure is crucial, an important source of information is the records of field-reported incidents B where and how has the system, or other similar systems, failed in live operation previously?

I will pick up again on this topic of developing stress test work loads, in the later sections of this book which discuss testing for robustness.

Stress Testing Methods

There are several methods of stress testing, such as brute force stress, hot spot stress, incremental load increases. I=ll describe these below. Related types of robustness testing, such as degraded mode of operation and live re-configuration testing, are described later in this book in the section entitled ATesting for Robustness@@.

Brute Force Stress

In a brute force test, a sufficiently heavy load is placed on a system to cause it to fail. It does not matter particularly what load we use, as long as the system fails or comes close to failure.

This method assumes that the system can fail in only one way, regardless of how we overload it, or that all the modes of failure are equal in terms of what they tell us about the system=s behavior. These assumptions are often not true B systems can fail in different ways, and the follow-up behavior depends on the manner in which the system has failed.

DEVELOPING THE TEST WORK LOADS
(Continued)

Hot Spot Stress

In hot spot testing, we focus on stressing a specific portion of the system which we believe is vulnerable, in order to probe for weaknesses and cause a specific type of failure.

Let=s say, for example, that a system has mathematical computations (which can stress the processors), page swapping (where pages of data or instructions must be fetched from slow memory), memory-intensive functions (these read and write high volumes of memory within short durations), database queries and updates, and network traffic.

In this situation, we could develop five different hot spot loads to stress the system: (a) intense use of features which require heavy computations, (b) use which require intense page swapping, (c) memory-intensive use, (d) intense database use, and (e) intense network use.

Incremental Load Increases in Stress Testing

Let=s say that we have a new Web site where the peak number of concurrent visitors is predicted to be 50,000. With new Web sites, we might uncover many problems with only 100 to 500 virtual users -- we don't have to go to 50,000 users, at least not initially. If the first load used in testing is a stress overload, the system will very likely fail immediately, but not in a way that provides much useful information about where the bottlenecks and problems are.

So it usually is a good idea to start small, incrementally increase the load, find and remove the bugs encountered at each level of load, and only then increase the load and repeat the process.

Of course, if the predicted peak load is 50,000 users, and we have tested the system with 500 users and removed the bugs (those found with 500 users), this does not necessarily tell us much about its scalability to handle 50,000 users.

In the case of the insurance company I had mentioned before, problems surfaced with 50,000 users (such as 143 seconds to download a Web page) which had not been visible when testing with fewer users (500, 5,000 and 10,000).

DEVELOPING THE TEST WORK LOADS
(Continued)

Another reason for incrementally increasing the load is the gather data points to be able to draw a trend chart between the load level and performance characteristics such as response time, and to extrapolate the trend beyond the domain in which we can feasibly test and measure.

This type of testing is also called breakpoint testing.

Concurrent Web Site Visitors

To perform a load or stress test on a Web site, we need a realistic sense of how many visitors the site may have to handle under peak conditions.

With a private network, such as an internal client/server network or an Intranet which is implemented as a VPN (virtual private network), there is a known upper bound on the peak load that can occur. This peak load is constrained by the number of devices (e.g., personal computers) connected to the network, which is a fixed number on any given day.

There also is an upper limit to how fast users can bang away on their machines. If we know that the network contains (let=s say) 500 users, we can calculate the maximum possible load on the system.

This calculation of the maximum possible load is not possible in the public Internet -- there is no upper limit to the number of possible visitors, unless we set it at 6.5 billion (the world=s current population). Of course, building a Web site to handle this many visitors may be a sign of megalomania -- it would be so over-engineered that we could not afford it.

In practice, bottlenecks in the Internet provide an effective upper limit on the load. The Internet service provider (ISP, or the Web host site or the proxy server firewall), place an upper limit on how may visitors can get through and request service at any one time.

Nevertheless, we still need to ask: What is a realistic peak to use as a test load for testing purposes?

DEVELOPING THE TEST WORK LOADS
(Continued)

The peak loads which we are likely to experience are very different for amazon.com or yahoo! than for your own personal site which shows three-year-old pictures of the family dog. The variation in the range of loads is very broad. The peak loads to date which have actually been experienced by Web sites have exceeded 10 million hits within a 4 hour period, and the lows are 1 hit per week or less.

Based on empirical data, the numbers in the following are typical of the loads experienced and thus are appropriate for Web load testing:

Type of Web Site	**Peak Load**	
	On-line	**Concurrently Active**
Major Portal (E.g., Amazon.com or Yahoo!)	50,000 to 75,000	10,000 to 15,000
Large International Retail Site (E.g., Dell.com, Schwab.com)	20,000	4,000
Moderate-size Nationwide Retail Site	5,000	1,000
Moderate-size Local Retail Site	1,000	200
Small Retail Site (E.g., local gourmet food store)	250	50
Small Non-Retail Site (E.g., solo practitioner consultant)	50	10

DEVELOPING THE TEST WORK LOADS
(Continued)

The Operational Profile

If there is an adequate operational profile for a system, most or all of the guessing about load size is unnecessary. The profile, which I have described in section tbd, lists the major demands on a system, and the probabilities of these various different demands actually occurring in live operation.

The operational profile is essential in test work load and benchmark design, because it provides a model of the patterns of expected use of the system. Unless we know the profile, we cannot be sure that the demands on the system are representative of the real world (in other words, that the operational profiles match in testing and in live operation).

See later in this section for a discussion of how to build operational profiles, plus the discussion of profiling tools, in the section entitled: ABuilding an Operational Profile@.

DEVELOPING THE TEST WORK LOADS
(Continued)

What if We Don=t Have an Operational Profile?

The best answer is to persuade either the system developers or the system users to build an operational profile for us (the testers). If this will not work, the next best answer is to build one ourselves, or the best approximation to the profile that=s feasible with the time, information and resources that we have available.

The logistics of building a profile can be a time-consuming, difficult hassle. Many vendors, for example, have little data on how their customers use their products, and the usage patterns are likely to vary highly from user to user.

To complicate the picture, most software products today are configurable, and can be installed at different sites in a very large number of different configurations. (We discuss testing for compatibility across configurations in a later section.) In addition, the configuration settings of a site usually evolve over time, as the system=s on-going processing of the work load automatically changes internal switches.

These variations from site to site are important, as the system performance and robustness often varies dramatically based on the work load and how the system is configured. With Windows 2000, Microsoft reported a 5-to-1 range in reliability (as measured by mean time between failures, or MBTF), in just a small sample of Windows installations. In total, several hundred million copies of Windows are installed (in all versions, not just Windows 2000), and the number of unique Windows configurations is probably a high percentage of the total number of installed sites. So the overall variability in the reliability that users experience with Windows, even with trhe same version such as Win 2000, is likely to be much broader than 5-to-1.

This means that an individual customer=s experience with a vendor=s product may have very different performance, robustness and reliability than the vendor=s experience in the test lab.

The vendors who have been most successful at building operational profiles across distributed customers sites have built-in data collection facilities. These facilities monitor system activity, collect usage data, and periodically and automatically report it back to the vendor from the remote sites. This reporting is done with the customers= knowledge and by prior agreement, and there are incentives for the customers to be involved.

DEVELOPING THE TEST WORK LOADS
(Continued)

One incentive that vendors use is to provide their customers with powerful data analysis tools so that they can understand the data. As well as contributing to the overall pool of data and thus to the common good, each customer can use its own data for system tuning and capacity planning. The vendor that is considered the most advanced in sophisticated in automated data collection from customers is Lucent Technologies.

What can a vendor do if it did not have the foresight to build automated data collection facilities into its products? I suggest these steps on the part of the testers:

(1) In conjunction with vendor=s market research, technical support or fielding engineering staff, develop a taxonomy of customer types (if a suitable one does not already exist). Each category in the taxonomy groups together similar customers from the perspective of how they will use the system. For example, one type of customer might be people who install the software product on single-user workstations, while another type is organizations which use networked versions of the product to support at least 1,000 individual users.

(2) From the inventory of all customers, identify a few Afriendly@ ones within each category. These customers have good working relationships with the vendor, are amenable to participating in research projects (which may require some effort on their behalf), and willing to share data.

(3) Determine what the measurement goals are, what decisions we (or managers) will reach based on the conclusions, what conclusions we=d like to derive from the gathered data, and how we will analyze the data to reach conclusions. It is pointless to gather data without knowing how we intend to use it.

(4) Identify what data ideally we want to collect, in order to support these goals and decisions.

DEVELOPING THE TEST WORK LOADS
(Continued)

(5) Determine how to remotely gather the data needed to build the operational profiles. Consider these issues:

- Do most customer or user sites already have software tools installed which we can use for data collection? For example, operating systems (OS), database management systems (DBMS) and network management system usually have event logging capabilities. A third party might supply these tools, and we could require their permission in order to proceed.
- If so, what pertinent data is being gathered and what are the shortfalls? How can we accommodate these shortfalls?
- If the remote sites do not have suitable data collection tools already installed, what tool should we use?
  - -- Can we purchase it or must we build it?
  - -- Can we seamlessly integrate it into future upgrades of the system before delivering those upgrades to the users?
  - -- Who pays for the tool: us or the user site?
  - -- Who installs and maintains it, including training the user or customer staff in how to use it correctly?
- What about the users= or customers= willingness to participate?
  - -- What incentives can we provide for them?
  - -- What are their concerns, e.g., sharing private data with competitors, risk of foreign tools de-stabilizing their environment, extra work tasks imposed on their staffs, etc?

(6) Prepare a presentation for these customers, to explain what=s going on and sell them on the program.

In the absence of concrete measurement data, here is a tendency to use mixes of demands in performance testing which we hope are representative, where the relationship of the test mixes to results in actual use is unknown.

As I claimed before, though, it is essential to use a reasonably trustworthy profile in testing. I have seen too many performance, load and stress test projects produce results which -- with hindsight -- were useless or worse, misleading. This poor outcome will happen if the profile used in testing was not realistic.

DEVELOPING THE TEST WORK LOADS
(Continued)

Even if building the profile is a difficult undertaking, this is no excuse to skip it. If measuring the actual mix of demands in live operational is infeasible, we should make a serious effort to Aguesstimate@ it and then to validate the guesstimate with knowledgeable professionals.

How Accurate does the Profile Need to Be?

Some people decide that building an operational profile is infeasible because it will not be accurate. In fact, trying to squeeze the last few percentage points of accuracy into the profile can easily triple the effort to build it.

So how accurate is good enough?

This equation computes the accuracy:

Accuracy of the final measured results =

[Accuracy of the operational profile * Collective accuracy of all the other factors]

>= Target accuracy established at the beginning of the project.

In an earlier section, we discussed how to establish the appropriate target accuracy for the test results.

Let=s say that we established the target accuracy at 80% (i.e., the allowable tolerance in the results is +/- 20%). We believe that we can measure the other factors which affect the results with an accuracy of 90%.

According to the equation, the accuracy of the operational profile must be 90% or better:

[0.09 * 0.09] = 0.81 >= 0.80.

DEVELOPING THE TEST WORK LOADS
(Continued)

Working Without an Operational Profile

I have emphasized the importance of realism in evaluating system performance. We can reach almost any answer we want, in terms of performance characteristics such as response time, based on the mix of demands we choose to place on the system during the performance measurement.

Nevertheless, there are circumstances where the operational profile or usage model has only a minor relevance in performance testing. These circumstances include:

(1) We do not need very precise performance data, but only a crude approximation, so that it is not very important if the test load only roughly resembles the live operational loads.

(2) There is good reason to believe that the system performance is relatively stable regardless of load, in other words, it is relatively insensitive to the mix of demands.

(3) There is reason to believe that system is so malleable, configurable or tunable that we can adjust or optimize it to provide a comparable level of performance, no matter what mix of demands are placed on it.

(4) We can dictate, or at least recommend, how the system will be used in operation `B` what hardware and networks will be used to run it, how the system is installed and configured, and what mix of demands we expect to encounter in normal use and in exceptional use. If users choose to disregard these recommendations, we provide no performance guarantees.

(5) While we cannot dictate how people will use the system, our role is limited to providing performance data only under a limited and well-defined set of circumstances. We do not provide performance data for other circumstances, and we make no representation that the mix of demands used in testing is likely to be representative of reality. It is up to the system users, if they so choose, to either interpret how our performance data may or may not apply to them, or to conduct their own performance measurements.

DEVELOPING THE TEST WORK LOADS
(Continued)

(6) We are interested in measuring performance against a benchmark, either our own one or an industry standard benchmark like TPC-D. How the system will actually be used in live operation is irrelevant to this measurement.

(7) In a stress testing, we are interested primarily in the system=s behavior after it fails because of an overload. It not matter exactly how the system fails, provided that it does in fact fail. In other words, it does not matter what the mix of demands are in this overload, as long as it is sufficiently intense. (Caution: this assumption that we don=t care how the system fails, provided it does fail, is often not valid. Systems often have many possible modes of failure and do not react consistently in these modes.)

(8) We are interested in comparative performance, not absolute performance, as explained in the example below.

Comparative Performance

Let=s assume that we work for a telecom vendor which is bringing a new product line to market.

Each member of this product line is a device which resembles a traditional router, but with some brand-new features. There is no point in comparing the devices= performance to other competitors= devices, because nothing else like these devices is available on the market as yet.

Each device in the product line offers exactly the same set of features, and functions in the same way as the other members of the family. Though they work the same way, the devices use different DSP chips (digital signal processors), with different processing speeds and amounts of memory, and the devices can interface with communications channels of varying bandwidth.

We wish to provide performance information to prospective customers, such as the device throughput in packets per second, and the percentage of packets which result in error messages. The prospective customers are interesting which particular gadget within the product line to buy, based on this comparative information.

We (and the customers themselves) do not have operational profiles, because the device is new and the prior usage patterns with traditional routers may not apply here.

DEVELOPING THE TEST WORK LOADS
(Continued)

In this situation, though, we do not need to have a highly accurate operational profile for testing. We can run the same benchmark, across the different device configurations. The situation is similar to running a race on a track which tilts slightly uphill or downhill. The order in which the runners cross the finish line is presumably the same as on a totally flat track.

So we quickly develop a swag benchmark which we think is not too inaccurate (swag stands for Ascientific wild-ass guess@). Of course, even if we do not need a precisely correct model of the work load, we still need at least a rough idea of the likely work load in order to design a benchmark that is Ain the ballpark@.

When we compile the performance measurements, we disclose the test load and its underlying operational profile as part of the published test results. The published results also contain a disclaimer, stating that the performance data measured with the test load may not hold with other loads.

The User Classification Scheme

A common problem for vendors is an insufficient familiarity about how their customers use their products, despite extensive market research. The customers use their products in different ways: for example, users install a software package on different hardware platforms, configure it differently, use different sets of features and in use them in different ways, and have different mixes of demands. Exacerbating this situation, many vendors are not intimately familiar with their customers= operations, have inadequate records on the customer behavior, and no easy way to get that information. Large enterprises which deploy their internal systems to hundreds or thousands of individual sites have a similar problem.

Even if we have detailed and accurate operational profiles, it is impractical to test with hundreds or thousands of separate test loads. And since the operational profiles change over time, we need a daunting maintenance effort to keep a large set of profiles up to date

A practical way around this problem is to develop a user classification scheme which categorizes the universe of users into a few manageable groups. We then assign prominent or typical customers to the groups. Customers can be selected from each group, based on the ease of collecting data from them, and analyzed, and we develop a test load for each group.

DEVELOPING THE TEST WORK LOADS
(Continued)

A good place to look for a user classification scheme, or the basis for developing one, is market research. Marketers categorize users in order to develop a comprehensible model of the marketplace, understand a product=s competitive position and develop sales strategies. The market research available for adaptation can be internal B developed within the vendor B or external, developed by information technology market research firms like the Gartner Group and Giga.

One advantage of using the existing internal classification schemes is that the executives are probably already familiar with the user taxonomy which the marketers use, and will be able to relate to it. It also is easier connect the performance and robustness testing to the marketing strategy through a common model, and it is easier to sell executives on the need for performance and robustness testing if we can tie this program to the marketing goals.

A disadvantage to using externally developed models is that they may be proprietary, which raises issues of plagiarism and licensing fees. The external models also may not fit a particular vendor=s situation as well as its own model. However, we have to use the external market research models, at least as a starting point, if we have not developed suitable classification scheme internally.

IDC, a market research firm, provides an example of a user classification scheme. Their model addresses the server market, and contains seven categories. The numbers next to each category is the market share for each category (i.e., the percentage of the total server dollar sales and service), according to IDC, as of 2002.

- Business processing (24.1%)
  - ERP (enterprise resource planning software, such as SAP)
  - CRM (customer relationship management software, such as Siebel)
  - OLTP (on-line transaction processing software, such as CICS)
  - Batch

- Decision support (12.7%)
  - Data warehousing
  - Data mining

DEVELOPING THE TEST WORK LOADS
(Continued)

- Collaborative (10.2%)
  - E-mail
  - Workgroup

- Infrastructure (33.1%)
  - File and print
  - Networking
  - Proxy, caching
  - Security
  - Streaming media
  - System management
  - Web services

- Scientific / engineering (computing only) (5.9%)

- Other (1.5%)

IDC provides explanatory examples and scenarios of the users in each of these categories.

DEVELOPING THE TEST WORK LOADS
(Continued)

Load Variations during Testing

Bridge builders test highway bridges with both static and dynamic loads. (These days, computer simulation replaces much of this testing so we don=t risk real bridges.) In a static test, the testers park a group of vehicles on the bridge and left there for a few days. In a dynamic test, a steady flow of traffic crosses the bridge. The response of the bridge to the static and dynamic loads can be quite different. We can apply the same idea of static and dynamic loads to testing computer systems.

A popular testing technique, especially in telecommunications network testing, is to vary the load over time. We run the test for an extended period (hours or days) with a typical work load, and from time to time during this period the test load surges or ebbs to simulate the real-world changes in demand. Load variations usually provide a more realistic test of a system than a constant, static load. In actual use, few systems run at continuous peak demand.

If a stress test is done successfully for a short duration, we know that the system can handle the spike in demand.

If we exercise a stress test for an extended duration and the system fails, this fact may only have a limited usefulness. The extended test tells us that the system cannot handle a continuous peak load, for a duration of let=s say eight hours. However, we also may know that it is extremely improbable that the system will ever be called on to handle this load in actual use.

In testing bridges, civil engineers use both static and dynamic loads. In a static test, the engineers park a few vehicles on the bridge and measure the degree of deformity (bending). They could conduct an extended duration stress test, perhaps by piling vehicles ten deep on the bridge overnight, but this is unlikely to ever happen.

So in addition to the static test, the engineers test the bridge dynamically. This means they have a series of vehicles drive over the bridge, varying the load, while they measure the deformity.

This approach combines endurance or duration testing with stress testing. For example, a system may be able to handle the first surge of load, but not behave the same way when a second surge occurs some seconds or some hours later.

DEVELOPING THE TEST WORK LOADS
(Continued)

We sometimes refer to the process as elasticity testing. This name comes from the idea of stress a system to the maximum, then backing off for an interval to a lighter level of load, and then increasing the load to the maximum again. The test work load has a certain elasticity, similar to real-world variations in load. The question is: as well as handling peak demand, can the system handle changes in demand? This is similar to exercising by alternately stressing and relaxing a muscle.

The Duration of the Performance Testing

A system=s performance, as measured in the first 10 seconds of operation, may be different than the performance measured over 10 minutes because of transient overheads during the system initiation. This is especially likely if there is latency, such as the time required for a modem to make a connection through the dial-up network.

A system=s performance, as measured in the first hour of operation, may be different than the performance measured over 24 hours because buffers can become fill, memory leaks occur, and so on.

These variations of performance over time mean that an important part of performance test planning is to determine the duration(s) for which to test. How long is it sufficient to load a system when we want to measure its performance? This answer is that we need to sustain the load at least until the system reaches a steady state. When the performance measurements remain stable for a reasonable period, the system has progressed beyond any temporary start-up fluctuations in its behavior.

So once the system has achieved steady state, how long does the performance test need to persist? The answer is: until we are confident that the response time, throughput and availability data being collected in the test is reasonably accurate and acceptable; again, if the performance have remained the same, during a period that is at least 3 to 5 times longer than the start-up time needed to reach steady state, then these measurements should be valid.

Another way to determine the appropriate duration of a performance is to look at the business processes and Anormal@ cycle times. For example, consider an order entry process. Each order takes anywhere from 30 seconds to 2 minutes to process, depending on its complexity. Ten order entry clerks are simultaneously entering orders.

DEVELOPING THE TEST WORK LOADS
(Continued)

How long should the performance measurement last? Seconds? Hours? Days?

We want to simulate a load where each one of ten clerks enters at least a handful of orders: let=s say 5 each. Entering 5 orders will take about 5 minutes, let=s say 10 minutes to be prudent. We should also allow 5 minutes up front to reach steady-state operation before measuring the performance, so the total duration of this test will be about 15 minutes.

If performance is critical to the system so we need more precise measurements, or if the costs (including the delays) of performance testing are low, we would increase the test duration (and also re-run it, to see if we get similar results the second time).

There are sometimes good reasons to run extended duration tests, for much longer than is required to process all the test cases in the work load. For example, extended-duration testing tries to detect memory leaks. These extended duration tests are discussed later in the section entitled: ATbd@.

The longest duration test I have heard of, and which found a significant bug that otherwise would have escaped into live operation, occurred ast Ericsson, the Swedish telecom company. (Most extended duration tests do not run for more than 72 hours.)

The duration of the test was two months. According to the Ericsson test engineers, they detected the bug towards the end of the two months. The bug they found could have seriously disrupted telephone service in a major city.

Growth Projections

It is important to include projections of likely growth in the design of test loads. A growth in demand of only 0.5% per month, compounded monthly, means a load increase of over 15% within two years -- enough to seriously degrade the performance of many systems.

A related problem is the level of growth which is we assume. The use of an existing system might have been growing at a steady 3% per year, but after we have enhanced or replaced the system the growth may suddenly accelerate to 30% per year. In the late 1990s and early 2000s, some Web tool vendors and Internet service providers (ISPs) were experiencing growth rates of 300% per year.

DEVELOPING THE TEST WORK LOADS
(Continued)

On the other hand, building and testing systems to accommodate hypergrowth which never occurs means that most systems will be over-engineered and ridiculously expensive.

Some people make the mistake of testing for growth projections beyond the likely useful life of the system. Many systems are used for decades. If the system uses hot new technology, though, we could replace or substantially overhaul it in three to four years. In this case, there=s no point in testing with the projected demand seven years into the future.

Unnecessary Precision in Performance Measurement

A rule of thumb in usability testing is that people usually cannot detect differences in hardware, software and network speed of less than 10%. Many performance testers have engineering backgrounds, and Anaturally@ want to measure performance to two significant digits or more. But this degree of precision is almost always irrelevant.

In addition, marketers may want to show that their products are faster than the competitors= products, even if only by 2%.

False precision can be misleading.

Sampling in Performance Testing

When we measure response time and throughput, statistical sampling has a role. For example, let's say there is a performance requirement that query response must be less than 2 seconds 90% of the time. We measure the responses to two typical queries, and we get response times of 1.3 and 1.8 seconds respectively. Have we shown that the system meets the requirement? No: a sample of two transactions is too small to be trustworthy.

The field of statistical sampling addresses the question of how big a sample is enough, and the probability of the performance characteristics of the sample applying to all possible transactions.

DEVELOPING THE TEST WORK LOADS
(Continued)

Statistical sampling implies that, to obtain trustworthy results, we should represent each major type of transaction or demand on the system at least 30 times in the test load load. The results obtained from a very small sample (i.e., under 30) of performance test cases may not be representative.

In addition, it is a good idea to develop two or more separate test loads (mix of representatives) for the same situation, and to execute them both. If the two different test loads are truly representative of the real situation, then the performance results will be consistent. (Say within plus or minus 10% for response time, throughput, availability, etc.) If the results are not reasonably close, then the test loads are suspect.

Simply repeating the same test case 30 times under the same initial conditions is not a valid sample. There needs to be a reasonable variation (for example, following a Gaussian or bell curve) in the input data, initial conditions and background noise from test case to test case.

DEVELOPING THE TEST WORK LOADS
(Continued)

Developing the Test Data

The testers need to determine where the test data for the performance work load is going to come from. One method is to capture the transactions entered by each single user on the system, or multiple concurrent users, by using a capture/replay automated tool. The tool then replays these transactions while we measure the system performance. This method has the advantage of being representative of reality, assuming that the captured data matches the operational profile. However, it can be cumbersome or infeasible to collect the mouse clicks and key strokes of each user.

In addition, if we are timing the transactions as they access and update a database, then the size, indexing, organization and record mix in the database all can affect performance. This means the testers need to ensure that they use a realistic database in performance testing.

Instead of capturing individual user transactions, we can generate a test work load using a test data generation tool. This may be more convenient but probably is less reality based than capturing actual user data.

Assessing the State of the Test Load Libraries

Over time, organizations who initially succeed with performance testing can become victims of their own success because of how they originally built and now maintain their test load libraries.

Nobody specifically designed the test loads in the first place, and for expediency they were simply collected en masse from convenient sources, such as copies of large transaction streams. When I was a kid, you could start a stamp collection by buying a packet of 1,000 stamps for a few dollars. You did not know what you were getting in the packet, but you knew you sure were getting a lot of them.

If the system behavior and usage patterns are not well understood, designing the test loads is difficult. We can make a useful analogy here to feature testing: if a set of features is not well understood, designing feature test cases is difficult and it=s much more satisfying to energetically bang away on the keyboard.

DEVELOPING THE TEST WORK LOADS
(Continued)

When no one designs the items in a test load but we simply copy or generate them, we testers may be tempted to go overboard. If copies of 20 transactions are good, then 30 are better. This has been called the Asling mud at a politician@ style of test load development B throw enough mud and some must stick.

Test libraries tend to grow and become more embellished over time as people continue think of new plausible (and implausible) new scenarios, and then add them to the existing inventory. The larger the volume of items, the more unwieldy the test run B and the more tendency to view the test results as an undifferentiated aggregate. For example, the response times of different transactions in a Agrab bag@ of transactions may simply be averaged, blurring the distinctions among the various types of transactions.

As they evolve, the test loads can lose the traceability from each member of the load (a test case or suite of related test cases), back to its justification for being included a particular the test run. So, we cannot relate the test loads to the test needs.

For example, a test case may be part of a user scenario which we want to run, operational profile, standard test load, type of test (e.g., hot spot, so that they lose sight of the rationale for each test case or group

There may be lots of redundancy, but we cannot easily identify this.

Streamlining the Test Load Libraries

Tbd

Monolithic vs. Selectable Test Loads

Sometimes we design the test load to be executed as a monolithic whole, in a single all-or-nothing run, and we can deduce few or no meaningful findings until after the full load has been processed to completion.

Even if could choose to select and run subsets, do not know which subset to run.

Tbd

DEVELOPING THE TEST WORK LOADS
(Continued)

Question

Are the loads monolithic or selectable?

Are there clear and justifiable criteria for selecting test cases from the library for a particular test run?

How viable (trustworthy, efficient and repeatable) is the process for selection? The process has to compute the sizes of statistically valid samples, generate random numbers, and so on.

Is there a workable mechanism for the physical selection, extraction and storage of test data? Is it efficient and repeatable?

Tbd