Developing Toward an SLA:
Establishing a Test Plan

Tom Wilson

1 Introduction

The previous installments of this series discussed several facets of evaluating a *Service Level Agreement* (SLA). In [Wil10b], we examined the variation of the response times across transaction types. This was a notional analysis since the data were designed specifically to illustrate the points. In [Wil10a], we looked at the variation in average response time due to the load. The different types of load correspond to different parts of the SLA evaluation period. We also took a real transaction and analyzed its response time distribution. In [Wil11a], we considered how transaction performance varies due to the data referenced by the transaction. We highlighted that the results for the performance test are likely to be influenced by the data chosen for the test. This aspect is complicated by the growth of the user base and the database over the operations lifecycle. In [Wil11b], we considered how user load varies during the SLA evaluation period and the operations lifecycle. We observed that there was a load difference between a mixed system and a closed one.

In this installment, we are going to bring all of these concepts together into a plan that can control both the risk of system performance and its cost. The plan has options that the responsible decision-maker will have to consider in conjunction with the myriad of other system development and maintenance issues. Work avoided in the development phase increases the dangers associated with poor design in the maintenance (or operations) phase. This is a conscious choice.

2 Evaluating an SLA

The system providing example data supports logistics and maintenance for military equipment. For the purpose of anonymity, some general details are provided, and some specific details have been changed. This proprietary system has a response time SLA defined for it. For our purpose, the SLA is:

> The system shall respond to user requests with an average response time of 5 seconds for 98% of the transactions during the monthly charge period.[1] The system shall support 3000 concurrent users. The database shall contain up to 5,000,000 items of equipment.

Other details exist in the SLA that we will omit from this discussion. For example, the database contains other items (e.g., people, users, work orders) that are probably related to the amount of equipment. Also, there are report transactions that must meet a different average response time. These details will also impact performance.

What is missing from the SLA is a description of the growth during the operations lifecycle; only the final goal is defined. Nonetheless, since most of the risk lies with the contractor, understanding this growth is critical to a cost-effective system development and maintenance plan. The system is in a military environment. Unlike in a commercial environment, users will not be showing up when they hear how great the system is or when some interesting function becomes available.[2] The growth concept was discussed in [Wil11a] and [Wil11b].

A user load profile that defines the number of users at certain times during the SLA evaluation period is also missing. The system will certainly not have the maximum number of users at all times. Ignoring this profile causes testing to assume a worst case performance perspective, where, in reality, there is relief in the lightly-loaded parts of the evaluation period because the SLA formula uses averaging for most of the transactions in the interval (reference [Wil10a]). The concept of the user load profile was discussed in [Wil11b]. A user load profile will also establish a session duration distribution and a

[1]This means that after the top 2% of response times are eliminated, the remaining 98% must have an average that is less than or equal to 5 seconds.

[2]Having no competition has both advantages and disadvantages.

think time distribution. The user load profile is strongly related to workload, which is another key aspect that is missing from the SLA specification.

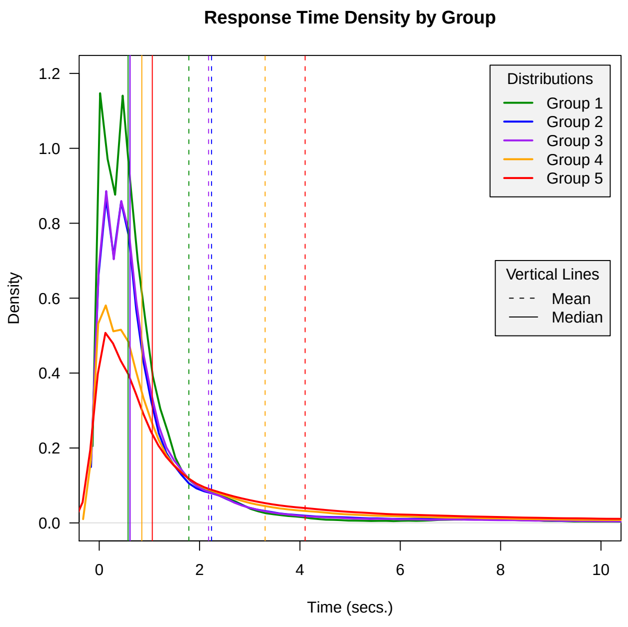
The SLA specification ignores the individual user's experience since individual response times can be very large. Other aspects, which are present in the real SLA, are not fully considered here: system qualities (such as reliability, availability, security, and safety) and additional functionality (such as transaction replay to account for remotely executed work and a mixture of report and standard transactions). Without a complete set of assumptions or explicit specifications, there is considerable risk for the contractor. The customer is focused on the users completing their objectives (i.e., maintaining the equipment), not detailing the SLA.

Three views of an incomplete or vague SLA can be taken. (1) The system can be developed and deployed. Then the performance issues can be addressed, weighing whether or not the penalties are worse than the expense to improve the system. Taking this view is not far from reality. (2) A much more detailed specification can be derived and agreed to with the customer. This is not likely to happen because it transfers a significant amount of burden back to the customer to understand what is being agreed to. (3) A much more detailed specification is derived for internal purposes without explicit agreement by the customer. Customer involvement is necessary but can be limited. This is probably the best of the three options. The contractor controls the amount of detail and much of the risk. This is the view that we will take.

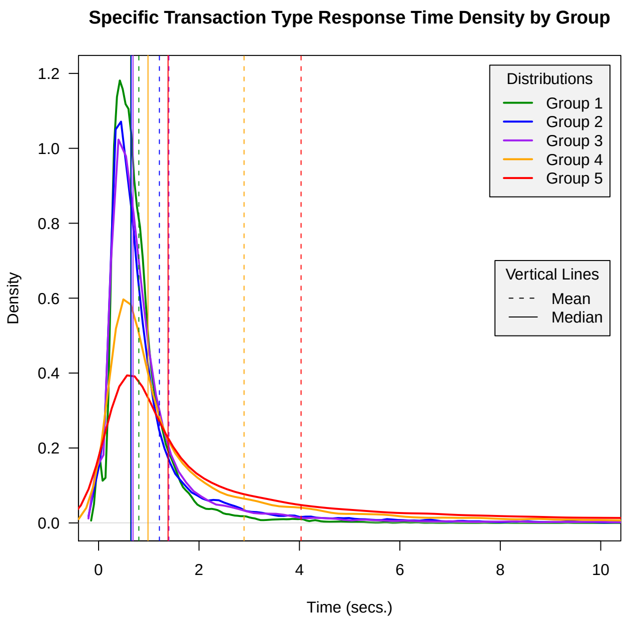
3 Understanding the SLA

Before defining a plan, let's make sure we understand the SLA (as presented here) by defining some terminology. What is a transaction? A *user transaction* is the work accomplished between the submission of a command and the receipt of an appropriate response. A *system transaction* is the work done between the user interface and the system (i.e., application, database, etc.). A user transaction may trigger several system transactions. Which transaction is called out in the SLA? Neither is, but it was agreed that the system transaction will be used in computing the SLA. From a risk standpoint, this definition is advantageous to the contractor. But the users are not going to be happy if many user transactions consist of a few system transactions.

Next, the SLA evaluation calls for the averaging of transaction response times. Figure 1 shows two charts from [Wil10a] where we grouped transactions according to the load on the system. Each chart has the median and mean added for each group of transactions. Because the distributions are lognormally-shaped, the means are an inappropriate choice for central tendency (refer to [Wil10c] for an explanation of appropriate statistics for a skewed distribution). They tend to be about the 80th percentile (note that the 2% transactions have already been removed).



(a) All Transactions



(b) Specific Transaction

Figure 1: These charts show the response time distributions for 5 load groups (load levels). Each group's median and mean are shown as vertical lines. Chart **(a)** combines all transactions. Chart **(b)** isolates a specific transaction.

Table 1 highlights the median and mean for each group as well as the combination of all groups. If the SLA target is the mean for the "all" entry, it is evident that groups 4 and 5 can be well above the target. What is not shown is the maximum response time (a measurement removed by the "2%" clause). For all transactions, it is 76.40 seconds; for the specific transaction, it is 76.36 seconds! Times like these are infrequent, but are very annoying to the user. The loads encountered during these measurements are far below the SLA limit (about 25% of the maximum load).

Table 1: Workload Statistics by Group (times in seconds)

(a) All Transactions

Group	Median	Mean
5	1.06	4.10
4	0.85	3.31
all	0.69	2.72
3	0.62	2.18
2	0.61	2.24
1	0.58	1.79

(b) Specific Transaction

Group	Median	Mean
5	1.38	4.03
4	0.98	2.90
all	0.78	2.12
3	0.69	1.40
2	0.65	1.21
1	0.64	0.80

The evaluation formula gives the contractor some freedom. Poorly performing transactions can be very poor as long as a large number of transactions have response times that compensate in the averaging. This was highlighted in [Wil10b]. The user would like to experience a normal distribution of response times, but this is probably unattainable for an affordable system.

4 The Plan

If we understand the SLA, we should have an appropriate system development and maintenance plan. A comprehensive plan must be defined from the beginning. In fact, it would be best to do this at the proposal stage, regardless of whether or not the customer sees it. Various options can be defined and selected as project execution occurs. The plan presented here is not comprehensive and only touches on key issues.

The overall strategy is to meet the SLA at the beginning of the operations lifecycle and have the capability to improve performance when the SLA is in danger of being exceeded. The latter requires the ability to forecast performance and predict improvements expected by upgrades. Understanding the scalability of the architecture is important for predicting improvements. The key performance items are:

1. An operations lifecycle model
2. An SLA period load model
3. A performance test model
4. Test databases and test data
5. Results analysis

4.1 Operations Lifecycle Model

The operations lifecycle model is discussed in [Wil11a] and [Wil11b]. Basically, milestones should be defined (perhaps at some point each year) with associated database sizes and maximum user loads. The database sizes are computed from the sizes of certain parameters (e.g., number of equipment, number of faults, number of people) and associated data sizes. Hopefully, the customer can project when each organization will employ the system and help determine these parameters. The customer needs this information already in order to do other planning that surrounds the system (e.g., system installation planning, user training).

Figure 2(a) shows how the databases and the number of users might grow during the operations lifecycle. In reality, there might be additional years with no appreciable growth. The quadratic growth we have shown may occur. It is not likely to be linear either. Certain high-level organizations (e.g., air force or navy) will employ the system and will dictate the growth rate based upon numerous factors (e.g., system training costs and schedule, obsolescence of existing systems). The addition of these organizations dictates how data are added; it is neither random nor uniformly distributed. These military organizations do not grow in size; they grow in number (refer to [wik, page: "Military organization", section: "Hierarchy of modern armies", 2011] for examples of military organizations and their sizes).

Figure 2(b) illustrates how the users and data should be scattered across the organizations. While the illustration implies a linear relationship upward, that implication is not true. Almost 97% of the users fall into the lowest two organization

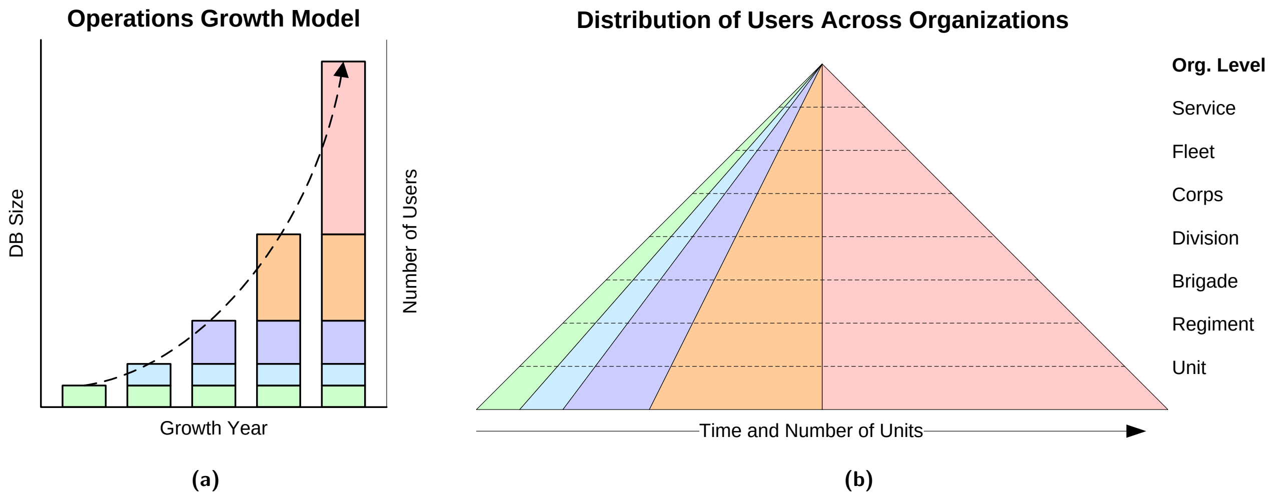


Figure 2: Chart **(a)** notionally shows how the database size and the number of users might grow over the operations lifecycle. Chart **(b)** notionally shows how the users and data are scattered across the organizational levels.

levels in the current system (reference [Wil11a]). Aspects of the new system could alter the distribution, but no analysis has been performed. The distributions of users and data are important for accurate testing.

As defined here, five databases are required for five SLA period load models (these two "fives" are a coincidence). It is intended that each database is a subset of the next larger; the same is true for each user load-level group. This reduces the amount of work required and will allow more meaningful comparisons between years.

Performance needs to be estimated at the milestones and mitigation plans defined for improving performance. Performance can be estimated by defining an SLA period load model for each year and executing the set of performance tests necessary. Predicting performance improvements will help control risk; however, it is not presented here since improvements are design and technology specific.

4.2 SLA Period Load Model

The customer is unlikely to be able to define a user load profile. Since data are available from the existing system, using them is a start (otherwise, a load profile should be estimated). From the load profile comes the SLA evaluation period model. From this model will come a number of performance tests (refer to [Wil11b]).

Figure 3 illustrates how the five load levels are defined for each year (in [Wil11b], a notional drawing used linear growth). Load levels are defined in [Wil10a] from a transaction perspective. The corresponding levels for users is described in [Wil11b].

When several performance tests that correspond to the different load levels are executed, the SLA can be evaluated from the results. We defined the load levels so that each level contained 20% of the transactions. The SLA is evaluated by averaging the results from each test run. If the distribution were different, a weighted average would be used. At this point, the performance of individual transactions can be addressed. We already saw in [Wil10b] that transaction performance is not directly determined by the SLA goal.

Table 2 shows notional SLA period load models over the operations lifecycle. Individual users are denoted by letters. For example, the test for load 3 of year 2 consists of users A-C and F-H. Note that the maximum number of users should really be 3000. Each year would also have an associated database. Setting up the models in this way allows more reuse and comparisons between tests. The actual plan dictates which tests are run.

4.3 Performance Test Model

The performance test model is a simulation. Typically, a simulation is stochastic (i.e., based on randomness). This creates some issues that may not be affordable:

- For a stochastic simulation to have value, it must be replicated many times in order to see the typical performance. With one test being lengthy to set up and execute, there will not be time to do many. If only one replication is executed, an analysis should be done to understand how much it deviates from expectation.

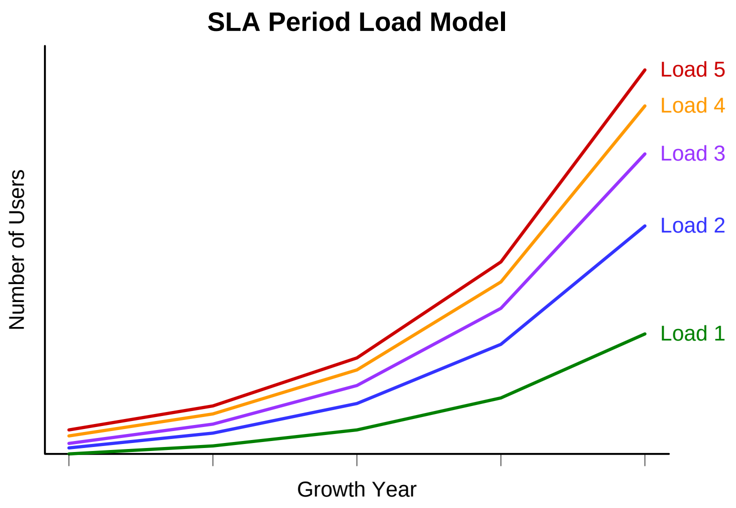


Figure 3: This figure shows the expected relationship between the database size (or amount of equipment) and the number of users for each load level for each year.

Table 2: Notional Example of User Assignments in the Load Models

		User																								
Year	**Load**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**
	1	×																								
	2	×	×																							
1	3	×	×	×																						
	4	×	×	×	×																					
	5	×	×	×	×	×																				
	1	×					×																			
	2	×	×				×	×																		
2	3	×	×	×			×	×	×																	
	4	×	×	×	×		×	×	×	×																
	5	×	×	×	×	×	×	×	×	×	×															
	1	×					×					×														
	2	×	×				×	×				×	×													
3	3	×	×	×			×	×	×			×	×	×												
	4	×	×	×	×		×	×	×	×		×	×	×	×											
	5	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×										
	1	×					×					×					×									
	2	×	×				×	×				×	×				×	×								
4	3	×	×	×			×	×	×			×	×	×			×	×	×							
	4	×	×	×	×		×	×	×	×		×	×	×	×		×	×	×	×						
	5	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×					
	1	×					×					×					×					×				
	2	×	×				×	×				×	×				×	×				×	×			
5	3	×	×	×			×	×	×			×	×	×			×	×	×			×	×	×		
	4	×	×	×	×		×	×	×	×		×	×	×	×		×	×	×	×		×	×	×	×	
	5	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

- For a stochastic simulation to have value, it must be repeatable. This is necessary in order to investigate and solve problems. This is a difficult, if not impossible, task when numerous computers (e.g., load generators) are involved in the testing. We would like to diminish the amount of variability if possible.
- Significant effort is required to avoid unintentionally repeating an action because of randomness. For example, if a user is closing faults on a vehicle, we would not want him to close a fault he already closed. A similar problem can occur if two different users can perform the same activity on the same data. This is called a "collision" and results

in errors during the test. A very large amount of data is required to support randomness.

To have more control of our test, we will generate a random instance, which will not change for any run. This has some drawbacks. For the most part, it is one sample point from a large population. We should do some analysis to make sure that we are not using an outlier as well as doing some model validation. If different runs are desired, they should be accounted for during the planning stage.

Workload is a complex topic that is critical to the accuracy of the system's performance evaluation. All other work may be futile if the workload is incorrect. For the existing system, we can examine the interactions with the system over an interval and summarize the workload by:

- an average number of users: We should note that not all users are the same. They can have different roles and views of the data
- a list of business processes with frequencies of occurrence: What may be hard to define is the data referenced by the functionality
- the average session duration
- the average think time

Workload could change during the SLA evaluation period or during the operations lifecycle. Existing analysis says that it did not change for either in the past for this system (refer to [Wil10d]). The number of users in the mixed or open system must be converted to an equivalent number in the performance test. This was demonstrated in [Wil11c]. That same paper also describes how to define a detailed think time model.

Because we are creating a performance test, we do not need to account for all functionality in the test. [Bar07] suggests looking at (1) business-critical, (2) performance-intensive, and (3) frequently-used business processes. Once the business processes are defined, scripts can be recorded and assigned to users. So in Table 2, each user would have an associated script. Each script is iterated during the testing interval as described in [Bar01, Part 4].

4.4 Test Databases and Test Data

The test databases and test data play a crucial role in creating a realistic workload. Building the database is not a trivial task. [Wil11a] described how new organizations need to be added to the database, rather than growing existing organizations. Most organizations are not directly referenced, but attention should be given to the business processes executed by users in high level organizations because they may access data for many organizations at lower levels. Users associated with later years in the operations lifecycle need to belong to some of these new organizations. For this system, the application must be used to load data. It was not designed to bulk load data. So, this is a time-consuming effort.

There should be some understanding of the database implementation so that test data and other database data are appropriately mixed (refer to Figure 4). It also important to understand the efficiency of the queries being generated within the system. If full table scans are frequently generated, the location of the test data is probably irrelevant and performance is going to be generally bad. If direct access to data occurs (e.g., by an equipment identifier), then caching may be too effective. By this, we mean that more caching occurs in the testing that really occurs in operations. A false view of the performance of the workload results. Figure 4 illustrates this in the third example: When test data are specifically referenced and they are physically near each other in the storage implementation, a higher cache hit ratio may result.

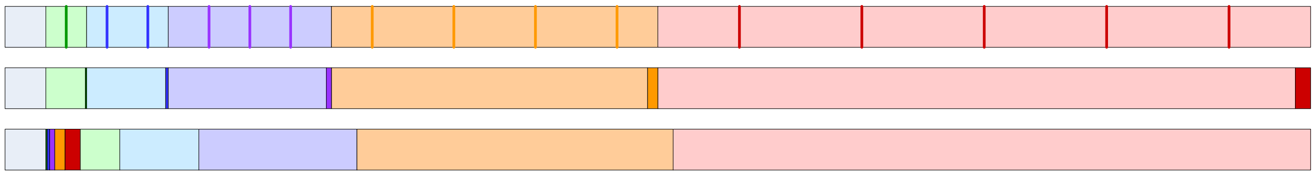


Figure 4: This figure shows 3 possible ways of building up the databases with the test data embedded. Each row is really 5 databases. The first database consists of the grey region (existing data) and the green region (new data). The second database adds the blue region to the first database. The process is repeated for purple, orange, and red regions. However, each row shows where the test data have been loaded (small, dark regions). In the first case, the test data are distributed throughout. In the second case, the test data are appended to each database. In the third case, the test data are appended to the first database. Whether or not there is a performance impact depends on the database implementation and the queries generated within the system. The first case is the least impacted by those details.

While the plan should be defined before any work is done, the bigger databases do not need to be built right away. That work can be postponed until more is learned about the system and the engineering processes that are being applied.

4.5 Results Analysis

When each performance test is run, some analysis of the results should be done. Occasionally, the test should be verified. Just because the test ran does not mean that the system functioned properly. Typical unit and integration testing does not execute heavy loads and so some examination of the correctness of the system should occur.

The results analysis should track the performance of the transactions. Transaction performance should be considered at the different load levels (reference [Wil10a]). The results of five runs are required to execute the SLA evaluation period model. It is at this point that some assessment can be made as to how to improve performance. In some cases, it may be that hardware changes are required. In other cases, transactions can be prioritized for improvements. We may also want to examine our test and make changes to it in the event that it is too light or too heavy.

5 Plan Execution

As sketchy as the plan is, it sounds too easy. Throughout this series we have never talked about true engineering performance requirements. If some can be defined, they may alleviate some of the pressure experienced later in the lifecycle.

5.1 Engineering Requirements

Should there be any engineering requirements? Remember that an SLA is not a requirement. The typical approach is to develop the entire system, test it, and react to bad performance. One option is to accept the SLA penalty. That is why an SLA is not an engineering requirement.

Creating requirements will be viewed as additional development pressure since opportunities for failure have been introduced. This is because of the way that the SLA has been defined. An engineering requirement is a pass-fail concept. Creating a requirement that a test does not pass results in a system that is not accepted. However, in operations, the system would still be used, and only a penalty would be assessed. An inappropriate requirement risks the system not getting accepted without unnecessary cost being expended to pass it. Also, a requirement that is met does not guarantee that penalties will not occur in operations. This can happen when workloads in operations do not match those specified in the requirements.

Getting the customer to commit to requirements may be a problem since that puts risk back in his domain. When performance requirements are met during engineering testing, yet the system is not effective enough, problems may exist for the actual users. Even though penalties are assessed, there is a need for the system to perform. Getting money back may not truly offset the consequences of the performance problems. The customer has a false sense of success because of inappropriate requirements. Informal requirements, or goals, may be an alternative. When goals are not met, the problems can be addressed or temporarily ignored. Perhaps the goals or tests will be adjusted rather than changing the system.

Because the SLA allows a lot of variance in transaction performance, individual component or subsystem requirements are unlikely. Likewise, internal metrics like CPU utilization are a waste of time. Such things can be used to manage the system, but they are not going to be useful as requirements for design.

At a minimum, measurements should be captured and tracked (reference [Bar10]). This is especially helpful during the early phases of development where most of the system does not exist yet. Test scripts, test data, and databases will not be mature either.

One suggestion is to categorize the transaction according to complexity (e.g., simple, intermediate, complex). The complexity would reflect the magnitude of work being done internally. At this point some expectation can be established and some comparison can be made between transactions with similar complexity. Additional complexity categories can be defined as more is learned about the system design.

The only real requirements (or goals) will concern user experience. While none were defined here, they certainly could have been. This will add burden to the development when individual transaction response times become important.

5.2 Development Lifecycle

Testing a completed system is one problem. Testing a changing, incomplete system is another. Depending on the system's user interface, test scripts may break as the development advances. Test data may require changes as new functionality introduces new attributes. Databases require migration as new tables and columns are added to the schema. Deciding values to put in those new fields is an overwhelming task. This work adds serious overhead to the testing. Bulking up databases for "future lifecycle" tests also creates migration work as the design advances. All of this work must be considered. Doing nothing until the system is developed is the standard, high-risk approach.

Early in the development lifecycle, most of the system does not exist. While a test can be run with the available functionality, the SLA cannot be evaluated. Only a percentage of the required users will be present; substituting existing functionality for missing functionality is a mistake in view of the fact that the resources may be improperly stressed.

So, some thought should be given to what testing actually occurs during the early part of the development lifecycle. Likewise, the frequency of testing should be low. The testing environment should be established and matured without focusing on running tests. For the tests that are run, large degradations in performance should be addressed. Scripts should be developed along with the functionality, and they should be maintained by both the developers and the testers. Otherwise, testers spend a lot of time trying to figure out why the scripts no longer work.

5.3 Operations Lifecycle

During the operations lifecycle, the existing models can be matured. The operations lifecycle model should get more accurate data as the system is really employed. The SLA period evaluation model will also become more accurate when real operations data are available. There is expected to be some difference in use between the existing system and this new one. The performance test model can also be improved with a better understanding of workload. One key to having accurate test results will be having a test environment that is identical to the operations environment. How much work is necessary can be driven by the current and projected performance.

6 Learning Opportunities

There were significant learning opportunities on this project. Many things discussed in this series of papers were not actually done! Here are some issues that were encountered:

- While there was some understanding of growth of important parameters, there was no lifecycle model. User counts were not associated with lifecycle milestones nor database sizes. All test users came from an organization that already existed in the database rather than being spread out.
- A few differently-sized databases were built. They were built a few times, but never as described here. Variations included putting all new equipment into (1) an existing organization and (2) a new organization.
- The testing tool was not fully understood. No training was provided for those using the tool. Randomness was used inappropriately resulting in data collisions and errors. Errors were considered acceptable rather than fixed.
- Tests were not repeatable. Changes introduced between runs were not sufficiently tracked. Explaining performance variation was very difficult.
- There was no SLA evaluation period model. Tests were not run for the corresponding load levels.
- Transactions were examined on an isolated basis. Transaction responses were often higher than the SLA value, and problem reports were written up as a result. There was no removal of the equivalent "2%" when computing averages.
- There were no unit level performance tests. There was no tracking of transaction performance.
- A "code coverage" metric was being generated from performance tests. This was computed as a percentage of the transactions in the source code that were occurring in the tests. This was not an objective for the performance tests, but was viewed by leadership as a lack of progress because the percentage was low. A performance test is not meant to test all paths. It is best not to report irrelevant metrics.
- The workload model was inappropriate. It focused on the frequency of system-level transactions and not business processes. Therefore, business processes had incorrect frequencies.

Considering all of the problems could there possibly be success? Yes. The hardware was sufficiently over-engineered so as to compensate for any performance dangers. . . for now. The SLA value was also renegotiated so that it was effectively doubled![3]

7 Conclusion

Developing toward an SLA can be difficult depending on the details within the specification. Often, sufficient detail will be missing, causing the contractor to fill in the gaps with assumptions. This series showed that risk can be better controlled by understanding the SLA and providing an appropriate detailed plan. The details in the SLA were for a future time frame and required scaling to be appropriate for the present. The experience on the actual project was that the

[3]Yes–the SLA goal was increased to 10 seconds! The customer conceded this in exchange for new functionality.

(performance) engineering effort was disorganized, resulting in unnecessary work, repeating necessary work, and not doing some necessary work.

Bibliography

[Bar01] R. Scott Barber. "User Experience, Not Metrics", 2001. 13 parts, specific parts are cited in each reference.

[Bar07] Scott Barber. "Get Performance Requirements Right—Think Like a User". Technical report, Compuware Corporation, 2007.

[Bar10] Scott Barber. "A Performance Testing Story From Conception To Gravestone". *CMG '10 International Conference*, December 2010.

[wik] Wikipedia. `http://en.wikipedia.org`. Specific page and last date referenced are noted in each citation. *Caveat lector: Because contributions to Wikipedia are not necessarily peer reviewed, readers should always validate what they are reading by other established sources. The references are given because of their simple accessibility.

[Wil10a] Tom Wilson. "Developing Toward an SLA: Understanding the Testing Interval". *CMG MeasureIT*, October 2010.

[Wil10b] Tom Wilson. "Developing Toward an SLA: Understanding Transaction Performance". *CMG MeasureIT*, July 2010.

[Wil10c] Tom Wilson. "Statistics for the Performance Analyst". *CMG MeasureIT*, November 2010.

[Wil10d] Tom Wilson. "Workload Correlation and Visualization". *Proceedings of the CMG 2010 International Conference*, December 2010. Reprinted in *CMG MeasureIT*, Issue 5, 2011.

[Wil11a] Tom Wilson. "Developing Toward an SLA: Understanding Data Complexity". *CMG MeasureIT*, Issue 3, 2011.

[Wil11b] Tom Wilson. "Developing Toward an SLA: Understanding User Load". *CMG MeasureIT*, Issue 4, 2011.

[Wil11c] Tom Wilson. "What Were They Thinking: Modeling Think Times for Performance Testing". *CMG MeasureIT*, Issue 2, 2011.