International Journal of Computer Applications Technology and Research
Volume 6–Issue 1, 19-22, 2017, ISSN:-2319–8656

Performance Lifecycle in Banking Domain

Vijay Datla

Abstract: Performance assurance and testing plays a key role in complex applications and is an essential element of the application development life cycle. This case study is about integrating performance at a large national bank. Learn how custom monitoring, Six Sigma techniques, performance testing, and daily production reports played an important role in identifying production issues. This paper illustrates and examines the challenges and successes of performance planning, testing, analysis, and optimization after the release of X Bank's CRM application.

Keywords: Performance, Lifecycle, Banking, Vijay Datla, Vijay, Datla

1. INTRODUCTION

Software Performance Lifecycle (SPL) is an approach that can be applied to all types of technologies and industries. This solution allows the true possibilities of the system and software under test to be examined. It allows for precise planning and budgeting. The SPL approach can begin at any stage of the Software Development Life Cycle (SDLC) and will mature as the wheel turns (or the life cycle progresses). However, the earlier performance is evaluated, the sooner design and architectural flaws can be addressed and the faster and cheaper the software development life cycle becomes. The wheel in this case is the development life cycle as a whole, not just one application release but all releases from the start of the application. The SPL steps include planning, testing, monitoring, analysis, tuning and optimization. These steps will be discussed in conjunction with this case study. The idea is to begin the SPL approach at any point on the wheel (or development life cycle). As the wheel turns the SPL approach will position itself to start earlier and earlier in the development life cycle for future release levels. In the example explained below, SPL started at the end of the first release of the application to be tested. Due to the late introduction, we ran into different issues and problems, but we jumped on and started the performance lifecycle. The introduction of the SPL approach will save significant time and money, while ensuring end user satisfaction. Our organization used these set of techniques and procedures and called it Software Performance Lifecycle (SPL)

2. CASE STUDY

This performance case study involves a major national bank with over 2,000 branches, fictionally named X Bank for this study. The bank was facing performance issues with various portions of their CRIVI application. They were experiencing high response times, degraded throughput, poor scaling properties, and other issues. This caused un-acceptance from their end users and customer base.

During the first round of implementing performance at X Bank, our responsibilities as consultants was to help elevate their performance issues. We were in charge of managing and executing the Software Performance Life Cycle, which included items such as planning, scripting, testing, analyzing, tuning, and managing the performance lab. As the application grew in size, so did the team. lt started with two Senior Performance Engineers and evolved to a Senior Performance Engineer, a Senior Application Developer, two Scripting Resources, Environment Team Resource, and a part-time Database Admin.

The production environment consisted of five ACS (Application Combined Servers) and one NT database server. The rollout plan for X Bank called for 500 branches every 6 months until reaching the goal of 2,000 branches.

The CRM application technologies consisted of an ASP front end on IIS Web servers, C++ middle tier on MO Series and an Oracle database.

2.1 Planning & Setup Phase

The first step in the SPL process is planning, this entails planning for the entire process, creating a performance test plan, and setting up the performance environment. The initial responsibilities of the two Performance Engineers was to interact with the bank resources, business analysts, developers, system administrators, database administrators, application engineers, and others to gather enough information to devise a performance test plan. To help create the performance test plan, we needed to fully understand the application behavior at X Bank. First, we sat down with business analysts to understand the major pain points and learn the application usage at the bank. We also made trips out to different branches and spoke to actual end users of the application to analyze their user experience and performance concerns.

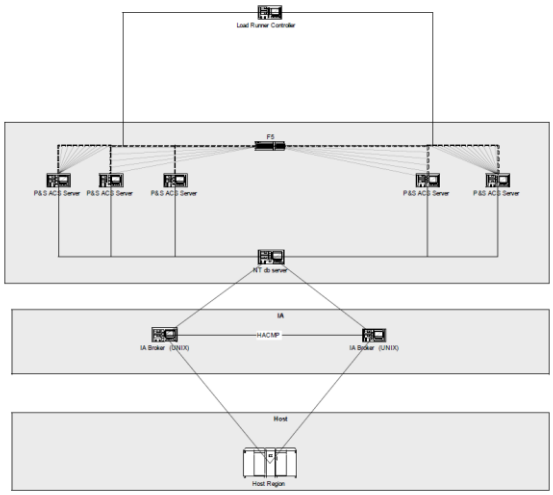
Next, we needed to get a better understanding of the database volumes in production to allow us to properly populate the perfom1ance test lab database. To do this we received database row counts from the production database administrator for the previous three months, and we used that information in conjunction with the growth projections to appropriately populate the performance test lab database.

All the information gathered during the planning phase enabled us to get a better understanding and positioned us to create a performance test plan. The test plan included actual use case! business process steps, SLA goals, database sizing information, performance lab specifications, and exit criteria for this round.

Next we set out to create an onsite performance test lab. The test lab included load testing servers, application servers, a database server, an Integrated Architecture (IA) server, and a host system. The perfom1ance lab environment mimicked production in terms of the application servers but lacked in terms of number of CPUs on the IA Server. The load testing software of choice was LoadRunner and Win Runner. We utilized one load testing controller, two load generators, and two end user workstations. The workstations were used in conjunction with WinRunner to truly understand end user experience under load. Lastly a custom dashboard application was written to monitor application transaction response times at the web and application tiers and to provide server statistics information

The last step was to install the application on the servers and load the appropriate data volumes into the performance database server. We used Perl scripts to generate the

appropriate volumes of test data, which gave us the flexibility to increase volume as needed.



Above picture shows Performance Setup

2.2 Performance Testing & Monitoring Phase

At this point, we had devised a performance test plan, had an understanding of the performance concerns, and created a performance test lab. Now the next step was to begin performance testing and application system monitoring.

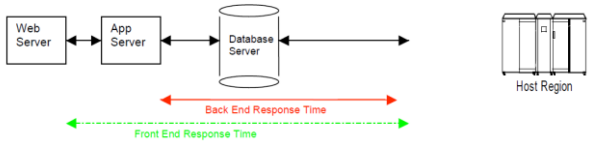
The first step was to create test scripts for all the outlined business process in our test plan using

LoadRunner. The scripting process included recording the script, enhancing the script and running configuration

audits. Enhancing the scripts included items such as parameterization, correlation, verification, logic, extra

coding, and anything else that was required to mimic a real end user. Recording, enhancing, and single playbacks were all performed under the LoadRunner Vugen utility. The Vugen utility is LoadRunner's scripting engine. The next step was to run configuration audits, with multiple users, using the LoadRunner Controller for all defined test scripts. The LoadRunner Controller is the utility utilized to generate virtual user traffic. The reason for running configuration audits was to make sure issues, such as data dependencies and concurrency problems, did not arise in multiple user mode. After we completed the configuration audits, we executed load tests separately against each of the five AC8 servers. The separate tests were conducted to make sure all servers were behaving exactly the same in terms of response times, throughput, and utilization. Next we ramped up to two servers where we calculated BP (Business Process) throughput and compared them to our goals. Other testing goals, used for comparison, included concurrent virtual user mark, business process throughput for other business processes, and transaction response times. After the two AC5 server tests, we scaled up to five AC8 servers, and were not able to meet all our throughput goals.

The logs showed us which application transaction was being executed, along with specific information of each transaction. lt showed the back-end response time which is depicted as a solid line in the diagram below. The next entry in the log file was the front-end time which does not include GUI rendering time (shown as dotted line below). As the front end time includes the back-end times, the difference provided us with just the web server response time, giving us another data point for our analysis. The last two entries provided the exact request size and response size of each transaction, thus allowing us to verify the correct data sizes for the appropriate business processes.



2.3 Tuning & Optimization Phase

The next steps in the SPL include tuning and optimization of the application. Through an iterative testing cycle, which included testing, database resets, monitoring, and analysis we made significant changes to the application. These changes included items such as the login cache mechanism, which originally cached unnecessary content, and changes to a third-party party DLL that was affecting database queries. We also changed the application configuration file to start the correct number of application server instances, which in turn maximized the server resources.

2.4 Results

We measured the round trip application response times a several different ways. As we ran our tests we utilized

LoadRunner to give us end-to-end response time, which did not include GUI rendering. So we utilized WinRunner to get true end-to-end response times which included GUI rendering time. The way we approached this was by running a full load with LoadRunner and placing two workstations on a emulated branch circuit running WinRunner. The WinRunner statistics provided response times that included the WAN emulation

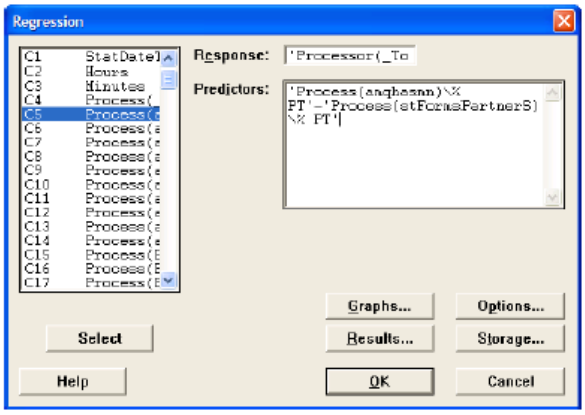
Of a remote branch circuit, as well as GUI rendering. This in tum provided us with a true user experience and end-to-end response times. Also to allow the business users to understand the feel of the application under load we had them walk through the application while we were running a full load test. The business users performed the business process steps from a lab that emulated remote branches. Lastly the business users provided response times from the branch circuit lab when no load was emulated on the system. This provided a true picture, meaning the best we can expect from our end user response times on system. The changes made allowed us to drastically reduce response times for most transactions while increasing the throughput of the application

Before we started SPL process at bank, they are facing performance issues at hundreds of branches.

Response times were high for key transactions, and marketing days were not acceptable by end users. X Bank did not truly understand the application scaling properties, capacity planning, hardware sizing, nor was the bank able to identify the performance issues they were having. But after the first pass through of the SPL process of planning, testing, monitoring, analysis, tuning, and optimization the bank was meeting SLAs, had proper hardware sizing in place, decreased costs, and gain confidence on marketing days.

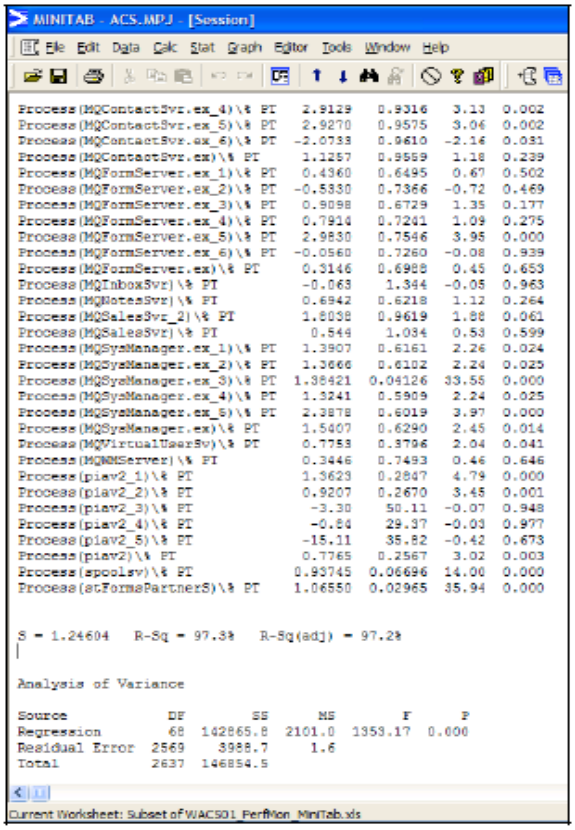
3. PRODUCTION MONITORING

After we rolled all the changes into production and completed the first iteration of SPL at X Bank, we began monitoring production on a daily basis. The application production team provided daily server statistics for all of production. We used six sigma techniques such as regression analysis, processor capabilities charts, Xbar-S charts and boxplots to aid in our production monitoring. First, we performed daily regression analysis on server processes to isolate any top consuming processes. The regression analysis consists of gathering process information from all servers, which was supplemented by Perfmon, Minitab, and Perl. Perform is a windows monitoring solution, Minitab is a statistical computing system, and Perl, a scripting language, was used to parse and format Perfmon data to fit Minitab. Next, the formatted data was imported into Minitab and a Minitab worksheet was created. After the Minitab worksheet was created, a regression analysis was performed to find the top consuming processes [M|Nll]3]. For this analysis, Processor Total was the 'Resp ' processes, to be analyzed, were the 'Predictors' (x variable). See picture below.



After indicating the response and predictors in Figure 1 the next step was to execute the regression analysis. The Output created from the regression analysis is shown in Figure 2. Figure 2 provides a value called R-Sq, the Higher the value of R-Sq the more relevant this data set is to our analysis. ln our example the R-Sq value was 9i'.2%, indicating our data having a really good fit to the model.

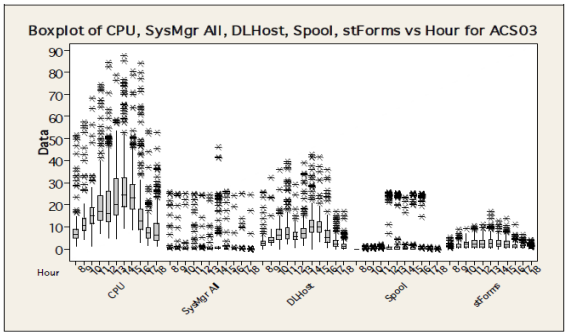
Below Picture shows Regression output, Processor vs Processes



After performing the above steps for all the production servers, the analysis provided us with a list of top processes of each production server. Below picture shows the high consuming processes. The higher T value the higher controlling factor the predictor, or the process.

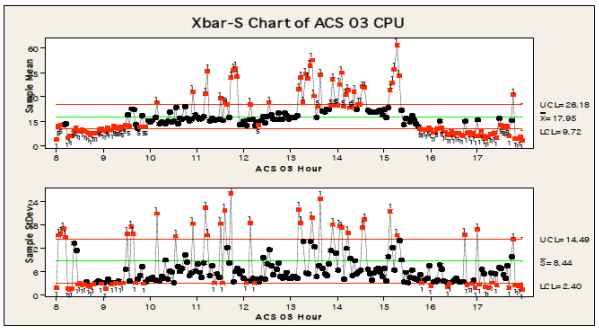
ACS01		
Predictor	T	P
DLLHOST_1)\% PT	89.62	0
MQSysManager.ex_1)\% PT	84.13	0
MQSysManager.ex)\% PT	73.96	0
MQSysManager.ex_2)\% PT	64.52	0
MQSysManager.ex_3)\% PT	63.25	0
MQSysManager.ex_5)\% PT	50.08	0
stFormsPartnerS)\% PT	38.54	0
MQSysManager.ex_4)\% PT	29.16	0
MQBLServer)\% PT	28.35	0
cqmghost)\% PT	11.53	0
spoolsv)\% PT	9.95	0
Inetinfo)\% PT	8.55	0
LSASS)\% PT	7.82	0
piav2)\% PT	7.8	0
amqzlaa0)\% PT	7.48	0
piav2_2)\% PT	7	0
piav2_1)\% PT	6.99	0

Below table shows list of top CPU consuming processes :

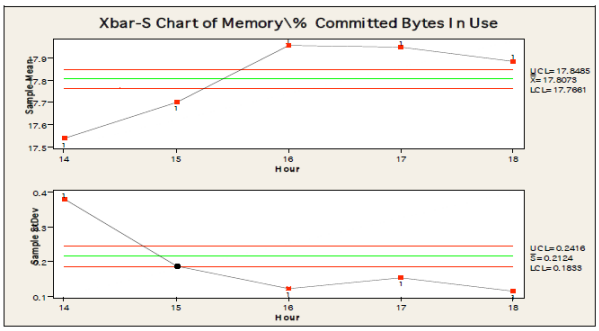


The CPU and memory graphs below displays a control chart for subgroup means (an X chart) and a control chart for subgroup standard deviations (an S chart) in the same graph window. The X chart is drawn in the upper half of the screen; the S chart in the lower half. Seeing both charts together allows you to track both the process level and process variation at the same time [M|Nll]3]. The x-axis of the graphs represents duration (time) while the y-axis represents sample mean or sample standard deviation. We primarily used this information to detect trends overtime. Minitab draws the average (center line), the upper control limit (UCL), and the lower control limit (LCL) lines by default.

CPU Processor control chart:

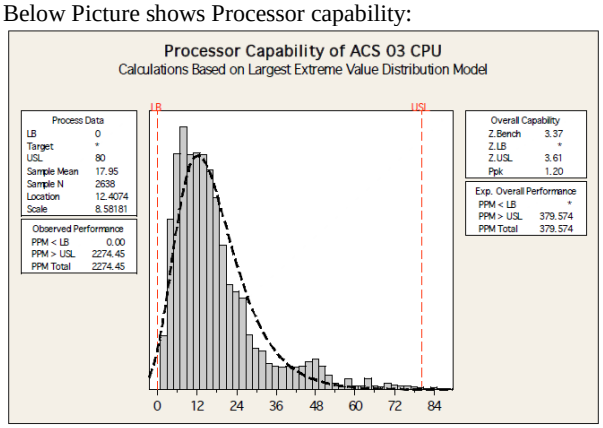


Memory Usage during peak



The processor capability graph [MINI03] below shows the processor distribution model around the CPU utilization
for the server on a given day. We used this information to see how many times (or parts per million, PPM) the data exceed our upper specifications limit (USL). ln our case any data point outside the 80% USL mark is considered defective because the application degraded after the CPU utilization hit 80%. In this graph there are a few things to keep in mind, Left Boundary (LB), Upper Specification limit (USL), and parts per million (PPM). Ln Graph 1, PPM > USL is 2274, indicating that for every 1 million data points of CPU utilization we are following outside our acceptable range 2274 times.

Below Picture shows Processor capability:



We used the above tables and graphs to create daily production reports. From the daily reports we created weekly and monthly trends to isolate any long-term problems. Specifically, it helped us isolate a few high processes that were behaving differently in production than in our test lab. The reason some the processes were behaving differently in production was due the fact that we could not test all business processes during this round of testing and only limited our testing to the business processes that produced 80% of the volume. The 80% was used based on the rule of thumb that20% of the transactions produce 80% of the volume. This allowed us to get a good indication of production volume without spending months scripting many different business processes. It seemed that transactions that were a part of the untested business processes were the top consuming processes to show up in production, and not in the performance lab. Due to our daily monitoring and reports, we were able to resolve these types of issues prior to any production downtime. ln conjunction, we utilized a custom dashboard application that provided response time information at the web and application layer for every transaction. It also provided real-time server and network statistics for all production servers. The dashboard alerted the application support team if any transaction or server network statistics breached the SLA thresholds.

3.1 Conclusion

Start the SPL approach at any stage of the SDLC. The bank faced performance issues in production, which caused un-acceptance from end users, bank personal, and the possible loss of future product upgrades. This could have cost millions of dollars in product revenue and maintenance licenses. But it did not, even though we started the SPL process of planning, testing, monitoring, analysis, tuning, and optimization after release l was in production. Since we were already on the wheel, we were able to include SPL earlier and earlier in the Software Development Life Cycle as the product grew, or as the wheel turned. The SPL approach saved significant time and money, ensured production readiness, improved performance and scalability, and built confidence.

4. REFERENCES

[1] MINITAB Statistical Software