Transition to Software Performance Engineering

Joby Joy
Associate Consultant-TCS
BE- Electronics & Communication Engineering
MVJCE, Bangalore, India

Abstract: Software Performance Engineering is a holistic and quantitative approach towards identifying performance bottle-necks early in the development cycle and hence not compromising on system qualities like flexibility, maintainability, reliability and usability. This paper address processes that may be implemented by IT to transition itself from testing to engineering and implementing best practices in providing value adds and improving the overall QoS to the product delivery.

Keywords : Software Performance Engineering, SPE, Performance Testing, Load generators, Dynatrace, Soasta Cloud Test, Jenkins, build pipeline, virtual user load, performance tuning and profiling, network analysis, database testing, PCOE, monitoring tools

1. INTRODUCTION

In a world where the demand for technology is ever rising, with a growing base of consumers relying heavily on technology for their daily tasks, with a swift competition between companies to attract the customers, one field of interest among the IT hub has always been to monetize on the growing consumer demands in the ecommerce world. And now with Google search results loading instantly, big data analytic tools tracking every actions of the customer and hand held devices becoming more powerful by the day, the bar for website performance is at an unprecedented levels. It just isn't enough that an application has cool features, tempting designs and offers, easy and multiple payment options etc.., but also that the application meets the performance requirements to scale the user load at any point of time. The performance requirements are determined using an extensive engineering approach built right into the SDLC phase of the application.

Software performance engineering is becoming increasingly important to businesses as they look to improve the non-functional performance of applications and get more out of their IT investments. By leveraging performance engineering techniques, IT professionals can be indispensable in building and optimizing scalable systems.

This calls for implementing an approach to stabilize applications through Performance engineering. This paper covers the need for Performance engineering and the methods to be implemented by performance testers that could enable an organization to make its needed transition into Performance engineering.

2. Software Performance Engineering

Software performance engineering (SPE) is a systematic, quantitative approach to constructing software systems that meet performance requirements. With SPE, you detect problems early in development, and use quantitative methods to support cost-benefit analysis of hardware solutions versus software requirements or design solutions, or a combination of software and hardware solutions.

SPE is a software-oriented approach: it focuses on architecture, design, and implementation choices. The models assist developers in controlling resource requirements by selecting architecture and design alternatives with acceptable performance characteristics. They aid in tracking performance throughout the development process and prevent problems from surfacing late in the life cycle. SPE also provides principles, patterns, and antipatterns for creating responsive software, specifications for the data required for evaluation, procedures for obtaining performance specifications, and guidelines for the types of evaluation to be conducted at each development stage. It incorporates models for representing and predicting performance as well as a set of analysis techniques.

If the software does not meet its performance objectives, the application is unlikely to be a success. If we do not know our performance objectives, it is unlikely that we will meet them.

Performance affects different roles in different ways:

- As an architect, you need to balance performance and scalability with other quality-of-service (QoS) attributes such as manageability, interoperability, security, and maintainability.
- As a developer, you need to know where to start, how to proceed, and when you have optimized your software enough.
- As a tester, you need to validate whether the application supports expected workloads.
- As an administrator, you need to know when an application no longer meets its service level agreements, and you need to be able to create effective growth plans.
- As an organization, you need to know how to manage performance throughout the software life cycle, as well as lower total cost of ownership of the software that your organization creates.

As the connection between application success and business success continues to gain recognition, particularly in the mobile and the web space, application performance engineering has taken on a preventive and perfective role within the SDLC to decide how important performance is to the success of the project. The more important you consider performance to be, the greater the need to reduce the risk of failure and the more time you should spend addressing performance.

2.1 Engineering for Performance

To engineer for performance, we need to embed a performance culture in our development life cycle, and we need a process to follow. When we have a process to follow, we know exactly where to start and how to proceed, and also know when we are finished. Performance modeling helps us apply engineering discipline to the performance process. The fundamental approach is to set objectives and to measure your progress toward those objectives. Performance modeling helps us set objectives for our application scenarios. Measuring continues throughout the life cycle and helps us determine whether we are moving towards our performance objectives or away from them.

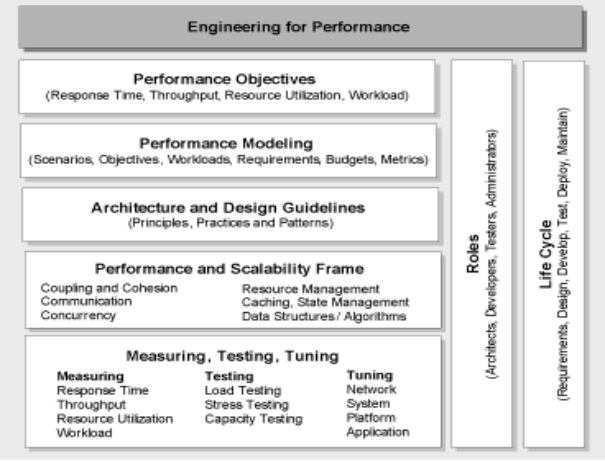


Figure 1: Engineering for performance

Engineering for performance is broken down into the following actionable categories and areas of responsibility:

2.1.1 Set Performance Objectives

Performance must be given due consideration from the beginning including measurable performance objectives. Performance objectives are usually specified in terms of the following:

- Response time: Response time is the amount of time that it takes for a server to respond to a request.
- Throughput: Throughput is the number of requests that can be served by your application per unit time. Throughput is frequently measured as requests or logical transactions per second.
- Resource utilization: Resource utilization is the measure of how much server and network resources are consumed by your application. Resources include CPU, memory, disk I/O, and network I/O.
- Workload: Workload includes the total number of users and concurrent active users, data volumes, and transaction volumes.

We can identify resource costs on a per-scenario basis. Scenarios might include browsing a product catalog, adding items to a shopping cart, or placing an order.

2.1.2 Design for Performance

Performance plays a vital role in determining the design of an application. Factors namely the architecture, technology, design, system configuration needs to be determined with regards to the overall performance of the application. Few key categories that can be called into consideration in the design of an application are:

- Coupling and cohesion: Loose coupling and high cohesion.
- Communication: Transport mechanism, boundaries, remote interface design, round trips, serialization, bandwidth
- Concurrency: Transactions, locks, threading, queuing
- Resource management: Allocating, creating, destroying, pooling
- Caching: Per user, application-wide, data volatility
- Data structures and algorithms: Choice of algorithm-Arrays versus collections

2.1.3 Life Cycle

Performance Engineering needs to be added to the various stages in an application life cycle. Performance attributes that needs to be considered at different stages are:

- Gathering requirements: We start to define performance objectives, workflow, and key scenarios.
- Design: Working within the architectural constraints, we start to generate specifications to construct the code. The design should be reviewed from a performance perspective.
- Development. Start reviewing the code early in the implementation phase to identify inefficient coding practices that could lead to performance bottlenecks.. Be careful to maintain a balanced approach during development; micro-optimization at an early stage is not likely to be helpful.
- Testing. Load and stress testing is used to generate metrics and to verify application behavior and performance under normal and peak load conditions.
- Deployment. During the deployment phase, validate the model by using production metrics. Workload estimates, resource utilization levels, response time, and throughput are few attributes that needs to be validated.
- Maintenance: Continue to measure and monitor the application deployed in the production environment. Changes that may affect system performance include increased user loads, deployment of new applications on shared infrastructure, system software revisions, and updates to your application to provide enhanced or new functionality.

3. A Typical Ecommerce Architecture

A typical ecommerce application comprises of many interconnected systems that performs various functionalities. An end to end business flow for an ecommerce application begins with customer visiting the web-site, placing an order through the payment gateways provided, validating for the order fulfillment, generating invoices to finally supporting the customer till the order is successfully delivered. This typical business flow encompasses multiple systems depended on each other to ensure that the order is be placed successfully.

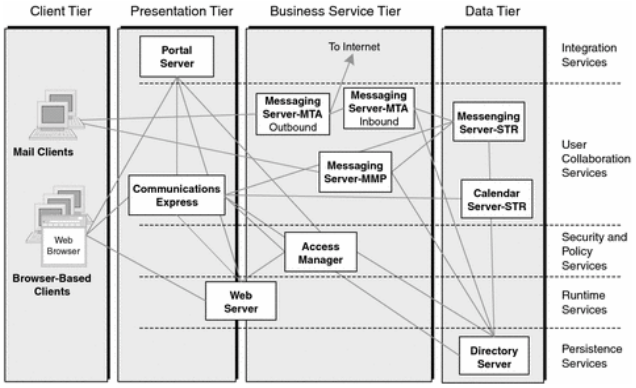


Figure 2: Ecommerce Systems

These ecommerce systems are implemented using a N-tier architecture comprising of web servers, application and database servers. The web servers are used to store, process and deliver web pages to the client using an http/https protocol.

A web server has defined load limits, because it can handle only a limited number of concurrent client connections (usually between 2 and 80,000, by default between 500 and 1,000) per IP address (and TCP port) and it can serve only a certain maximum number of requests per second depending on its settings, the http request type, whether the contents are static, dynamic or cache enabled, hardware or software limitations of OS. Application servers are system software upon which web applications or

desktop applications run. Application Servers consist of web server connectors, computer programming languages, runtime libraries, database connectors, and the administration code needed to deploy, configure, manage, and connect these components on a web host. An application server runs behind a web Server and front of an SQL database that function as the database server storing all the required information using RDMS.

Additionally to all these systems, ecommerce applications also leverage other systems like SAP, Oracle, Microsoft dynamics etc, that consists of several modules, including utilities for marketing and sales, field service, product design and development, production and inventory control, human resources, finance and accounting that collects and combines data from the separate modules to provide the company or organization with enterprise resource planning(ERP).Recently companies have also started using big data technologies to track the consumer sentiments in the ecommerce eco space.

With all these systems, compromise for any downtime within the application is unacceptable and hence performance to assess these application becomes inevitably of paramount importance to meet the expected SLA's.

Figure 3: System SLA compliance

With the number of systems predominately being used in any application it becomes difficult to understand and apply the whole process of SPE to the systems. This calls for every organization to transition into performance engineering to monitor and analyze the potential bottle necks. Although the scope of this paper is limited to imbibing performance engineering methodologies only at the front end, however the underlying concepts can be expanded to other sub-systems.

4. Transition process to SPE

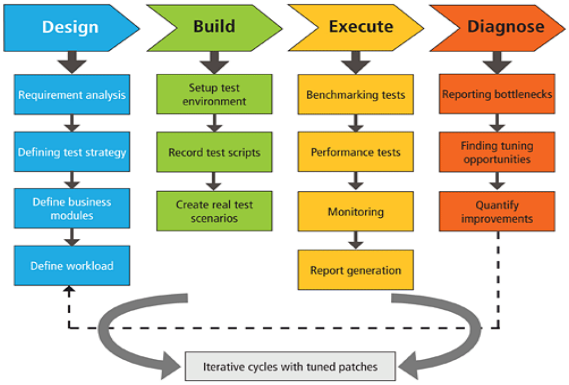
The transition to SPE requires a systematic approach comprising of varied practices:

4.1 Performance Testing

Performance testing (PT) is a non-functional testing technique performed to determine the system parameters in terms of responsiveness and stability under various workload. Performance tesing measures the quality attributes of the system, such as scalability, reliability and resource usage. The different services provided under performance testing are load, stress, volume,endurance,scalabilty and spike testing.

Figure 4: The PT cycle

SPE becomes predominant in the execute cycle of PT where the application is monitored for performance bottle-necks.PT is performed by using load generator tools like Jmeter,



LoadRunner, Soasta CloudTest, RPT etc. These tools provide the flexibility of generating load from different geographical locations hence simulating a virtual user load on par with the real time load. These tools also help in providing data on the transactions response times, throughput, error rate, bytes sent and received etc that can be used for performing a superficial analysis of the applications.

4.2 Performance Monitoring

Application performance monitoring (APM) is all about delivering business applications that meet customer satisfaction by deep monitoring, quick troubleshooting and tracking end user experience. This is the most important phase in SPE cycle.APM consists of tracking performance metrics of applications and servers during the load test and ensuring optimal usage of servers. This deep monitoring helps in planning capacity, troubleshoot quickly and view utilization reports for the various systems in an application.

APM is usually achieved by APM tools that provide a wide range of monitoring capabilities in an application. They are used to analyze the response times for the transactions at different systems, response times between the sub-systems, reason for spikes in the response times, memory usage of the applications, number of threads allocated, information for garbage collection, number of JDBC connections, network latencies, image compression and caching etc. These tools also help in finding out the root cause of a reported problem. For example a Java Transaction Monitoring tool could get in to the details of the Java Transactions executing in an Application Server and help identify which SQL Queries are taking time to execute or which methods in the Java class are slowing down the application. These help save precious time for the application team to resolve a problem. Some of the popular tools for APM are AppDynamics, New Relic, DynaTrace, Extrahopetc

Figure 5: The APM analysis

4.3 Performance Tuning

As performance bottlenecks are identified during performance testing and monitoring, these issues are commonly rectified through a process of performance tuning. Performance tuning can involve configuration changes to hardware, software and network components

A common bottleneck is the configuration of the application and database servers. Performance tuning can also include tuning SQL queries and tuning an applications underlying code to cater for concurrency and to improve efficiency. Performance tuning can result in hardware changes being made. This is a last resort; ideally tuning changes will result in a reduction of resource utilization

4.4 Other SPE Strategies

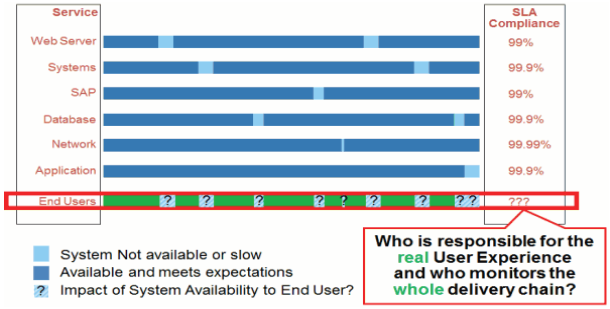
SPE also include troubleshooting for database and networking issues that are identified as a part of the performance monitoring strategy. This testing is usually performed by using database testing tools like Data Generator, Datatect, Toad etc. by validating the data mapping and integrity, business rule conformance, atomicity, consistency, durability and isolation.

Network analysis or testing is performed to address for network issues like latency, number of open connections, bandwidth utilization etc. Tools like wire-shark, Network monitor, Http analyzer can be used for the same. These tools function by capturing the network trace between the client and the server

request and analyzing the response contents. They can also be testing to see if the code supports any compression and cache techniques to enable better bandwidth optimization.

5. SPE-A Practical Guide

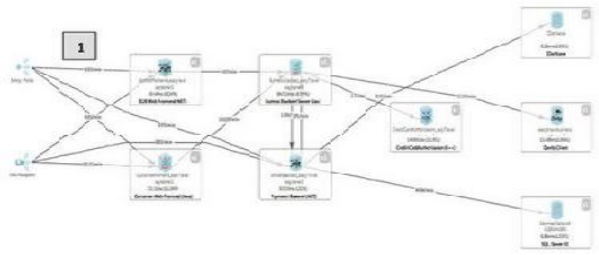
A practical application to the approach of SPE that is generally implemented in an application performance management is discussed here. A feasible gist to the concepts in SPE to help testers and developers track performance metrics follows. The analysis was conducted on the staging environment having two nodes on which the application is deployed. The performance test is conducted on a web-service API (Product Cxf services) that are inherently used in



production to list the products of a popular ecommerce site.

5. 1 Performance Testing

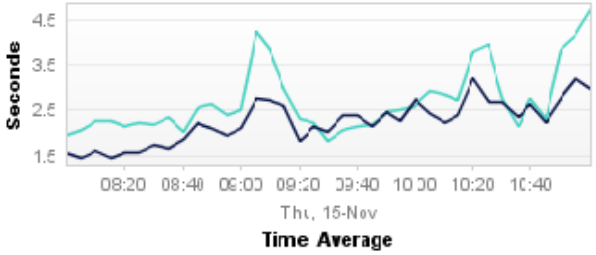
PT of the web service was initiated with a variable user load up to 500 users for 30 minute duration using Soasta CloudTest tool and AWS instances hosted in San Jose. During the course of the test, performance monitoring using Dynatrace tool also



was enabled. Typical performance metrics noted are 90th percentile response time, throughput, bandwidth usage, bytes sent and received and errors on the load generator tools

5. 2 Performance Monitoring (PM)

Performance monitoring require detailed analysis of the



service under load. Different tools come with many options to enable us analyze the performance metrics effectively. However the underlying concepts of usage between the tool remain the same. Typically these tools function by installing agents on the servers that saves the performance data in the server edition of the tools and performing the required analysis.

PM starts by viewing the transaction flow diagram which gives an overview of all the systems that the request serves within the test environment.

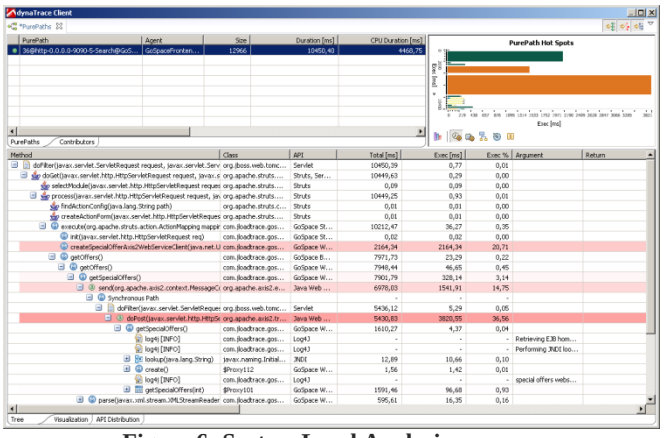
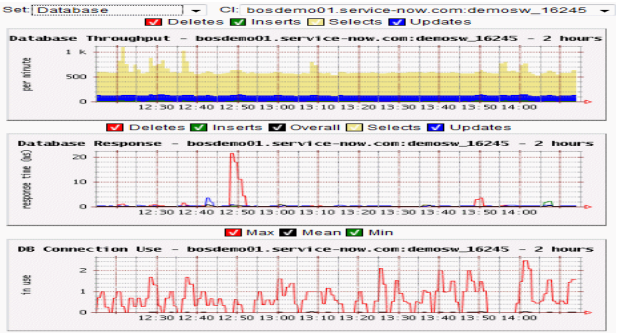


Figure 6: System Level Analysis

The next step is to analyze the performance metrics of the web-service. These results can be compounded with the metrics obtained from the load test tools. These results have to be monitored to check for spikes, resource consumption etc.

Figure 7: Response Time Metrics

Noting the duration of the spikes we can drill into the request that is observed to be consuming higher response times. This can be achieved by using appropriate filters among all the requests that were being processed during the load test. Tools provide with a myriad capabilities for effective filtering of the application based



on the response times. Post filtering we use the tools to drill deep into the source code of the application to analyze the reason for the failure. These failures can be at the thread, routine, method level of the application and can be rightfully identified in the monitoring process. By repeating the test we can observe if the errors are consistent with the previous run and thus track for bugs in the code.

Figure 8: Source Code Analysis

Depending upon the nature of the error other strategies in SPE can be implemented accordingly. To elicit further on this, the error that was analyzed from the tool relates to connectivity issues while writing to the data base. Based on the results collected from two or more tests, the root cause analysis was narrowed towards the database and hence an exclusive approach to database testing had to be incorporated.

In this type of database testing the query was repeatedly executed over a period of time to measure the database throughput, response time and connectivity to further conclude if the issue is persistent for a prolonged period of time.

Figure 9: Database Testing Metrics

Similar approach can be tested depending upon the error observed during the load test. Errors can be occur even at the network layers and effective measures has to be implemented to check for consistencies in the failures. These strategies of performance engineering when implemented at a sprint level can lead to identifying the performance bottle-necks at the sprint level itself and thus leading to less or no production issues with time.

5. 3 Performance Profiling and Tuning

Other key important activity during the SPE monitoring process is performance profiling. This implies to monitoring the memory, CPU, disk IO operations consumed by the processes during load test execution. For applications programmed using java technologies, monitoring the JVM becomes a necessary function in SPE. During the load test the number of threads allocated, the number of socket connection made and in pooling, the CPU and other resource consumed by the application needs to be monitored. For errors such as OutOfMemory exceptions typically triggered when the application reaches the maximum allowed heap configured , then memory dumps needs to be collected and analyzed (typically performed by the developer).

The end initiative of the SPE phase after tracking all the required performance metrics and bugs is performance tuning. A preliminary snapshot of these analyses is sent to the development group to fix the issues or performance tuning recommendations. Once fixes or tuning is applied on the test system, the tests are re-executed and hence, test execution and test analysis processes are iterative processes and need to be executed until the desirable performance is achieved. Typically tuning of the code can be done using developer support tools such as profilers and code coverage analysis tools

6. SPE-Best Practices

These best practices identify techniques for working effectively with others including developers, management, and other divisions within the organization.

6.1 Implementing SPE in Scrum

SPE should be an essential activity in Agile because it provides immediate feedback about the system in each sprint. Performance engineering helps in ensuring that the system is designed, built and validated against the required 'Quality of Service' requirements. This process of validation helps us in tracking for performance bottle necks in the early sprint cycles thereby giving avoid the tedious process of code refactoring in the future.

6.2 Produce timely results for Performance studies

Timely formulation and presentation of results and recommendations is vital, especially when corrective action is likely to be required. If a significant amount of time elapses between when the information is needed and when it is provided, key architectural or design decisions may have already been made. If this happens, the required changes may be more difficult to make, or they may no longer be feasible.

6.3 Adding Performance Test cases to build pipeline.

Continuous tracking of the performance results is crucial to identify performance bugs if any. Building performance test jobs using Jenkins and adding it to the build pipeline of an automation suite in an organization enables in continuous tracking of performance related bugs along with functional bugs. This process can also be mandated by certifying builds before every code freeze and release. Each jobs on the build pipeline needs to be tracked and certified before deploying the code to the production environment.

6.4 Creating a Performance Centre of Excellence.

Every IT organization needs to have its PCOE to provide recommendation and advises on performance test strategy , communicating and coordinating with product teams, identifying high priority test cases and approaches to services like capacity planning, code refactoring etc.

7. Conclusion

System quality attributes and performance are as important as functional correctness and completeness. However, performance is usually tuned into applications as an afterthought and is often inadequately addresses in the development lifecycle process.

Performance engineering calls for specific skills in model building, monitoring, benchmarking, optimization and tuning. Several activities have been identified through the development lifecycle and beyond to meet the skill needs. The widespread adoption and maturity of global software development has provided access to skills in model building and also monitoring, benchmarking, optimization and tuning. However, collaborative framework is needed to carry out performance engineering activities in an efficient and cost-effective way.

The process as mentioned are adequate to enable an organization to gradually transition itself into performance engineering thus enabling companies to take advantage of their resources and never ever asking to compromise on the quality of the product.

8. ACKNOWLEDGEMENT

I would like to thank the performance engineering team at Adobe Bangalore for providing me all the resources and the support that was crucial to conduct the research. The Managers and the Tech leads in the team have constantly shared their thoughts and experiences in aiding to the research. I would extend my gratitude to the development team for providing their value adds in the research. Last but not the least I would like to thank TCS for guiding me through the entire research process.

9. REFERENCES

[1].B. Boehm, “Software Risk Management: Principles and Practice,” IEEE Software, vol. 8, no. 1, pp. 32-41, 1991.

[2]. P. C. Clements and L. M. Northrop, “Software Architecture: An Executive Overview,” Technical Report No. CMU/SEI-96-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[3]. C. U. Smith and L. G. Williams, Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software, Boston, MA, Addison-Wesley, 2002.

[4]. JP Lewis, "Limits to software estimation",ACM Software Engineering Notes ,Jul 2001

[5].https://blog.codecentric.de/en/2008/07/memory-analysis-part-1-obtaining-a-java-heapdump/

[6]. http://apmblog.compuware.com/2012/07/12/proactive-or-reactive-a-guide-to-prevent-problems-instead-offixing-them-faster/

[7].http://www.xoriant.com/blog/software-testing-and-qa/best-practices-in-software-performance-engineering.html.

[8].http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6772693.

[9]. D. J. Reifer, Making the Software Business Case: Improvement by the Numbers, Boston, Addison-Wesley, 2002.

[10].ISO. (2010). DIN ISO/IEC 25000 software product Quality Requirements and Evaluation (SQuaRE).