

Should performance be an integral part of the SDLC?

By Vaidyanatha Siva and Sridhar Sharma

Poorly performing IT apps are bleeding corporations~\$60B/year. What can we do about it?

INTRODUCTION

Diagnosing, fine tuning and fixing poorly performing IT applications is both a science as well as an art. A detailed and dispassionate analysis of our experiences lead us to postulate that there are two underlying principles that project teams would be well advised to follow to build software that “performs”.

v

UNDERLYING PRINCIPLES FOR PERFORMANCE

We hold the following positions with regard to delivering applications that “perform” -

- We posit that performance is a mindset, just like quality is, and that we need to ensure it in every stage of the SDLC, in fact right from the time we make a proposal for a given project. Performance awareness, planning and execution should be baked into each of the phases in the SDLC

- We also believe that technology should be used as an enabler and enforcer to ensure performance.

We will use the above two principles as the leitmotif for the rest of this paper by walking through the life-cycle of a typical software project engagement, right from the pursuit stage.

THE PROPOSAL PHASE: PLANNING FOR PERFORMANCE

“The creation of genuinely new software has far more in common with developing a new theory of physics than it does with producing cars or watches on an assembly line.”

- Bollinger [1]

The success or failure of any software project rests heavily on the estimation methodology followed for that project. Any exercise to “accurately” estimate the cost and effort required to deliver a quality software product is only as good as our interpretation of the program size and the implementation complexity. What

makes this exercise more challenging are certain subjective and seemingly random factors - like the capability and productivity of the team, developer turnover, development schedules and social calendars etc.

There is enormous pressure on IT service providers to codify the software engineering process and deliver custom software as a packaged service. Several estimation models and methodologies have evolved over the last few years to cater to this need, and software organizations are increasingly relying on these models to quickly arrive at the cost-schedule matrix. Most, if not all, of these models rely heavily on past empirical data and attempt to derive numbers based on past experience on similar engagements within the organization. Such is the case with performance estimation too.

While these estimation models are, without doubt, quite mature and reasonably accurate, there are quite a few factors that tend to impair their effectiveness.

- Rapid emergence of new/parallel technologies and frameworks coupled with obsolescence of the old

This rapid rate of obsolescence of technology makes it inherently unreliable to use past empirical data (built on older technologies) for estimating new projects (using new technologies). For instance, we cannot apply the metrics of a J2EE project executed a year ago to a new project that mandates the use of transactional or UI frameworks that are just released as part of the normal JCP process. As the market for distributed component-based technologies matures (examples are

J2EE and .NET), new releases have significant performance fixes built in. These are easily offset by the increased complexity of applications (e.g., new demands placed on collaboration and SOA) and enhanced end-user performance expectations required by seemingly insatiable business needs (to enable business at the speed of thought).

- Emergence of “pure-play” project manager positions without sufficient emphasis on current technology exposure

It is imperative that the estimation exercise be conducted, or at least be validated by someone who has hands-on technology exposure, is in touch with the latest technology trends and can visualize and assess the program implementation complexity based on personal experience. In short, someone who has “been there” and “done that” and has the wisdom built on the bedrock of personal experience.

Good estimation requires experience and judgment. We should continue to value human experience, intuition and wisdom over and above what our “processes” define.

We also emphasize on the importance of using the elaboration phase of the SDLC to tackle potential performance issues and to nip those in the bud.

THE ARCHITECTURE DEFINITION PHASE: ARCHITECTURE CONSIDERATIONS

You’ve won the engagement - congratulations! Now, how do you plan for performance and

ensure that you do not get caught in the classic performance trap? Get the right people with the right skills at the right time!

This is the time to revisit the performance SLA and ensure that

- They are objective and have the right level of detail
- They are feasible and valid – e.g., a requirement that states that each web page should load within 5 seconds is impractical – different pages will load in different times and the load time is dependent on multiple considerations – such as the work each screen does, behind the scenes, the amount of payload each screen has to render, any multithreaded considerations such as locking, etc.
- They are indeed achievable, given the composition of the technology stack – hardware, system software, application software, and network.

Most lead software engineers, project managers or beginner architects either do not realize the impact of each of the possible performance considerations or if they do, are overwhelmed by the seemingly vast and diverse nature of agents that impact performance.

You need the experience of someone who – (a) has the theoretical background and framework (e.g., TOGAF, ATAM) to comprehend and analyze all the issues that can significantly impact performance, and (b) the practical experience and insight to take the right decisions and navigate through the performance maze.

The architect should also be “au courant” with technology and be in a position to make recommendations based on newer technology – e.g., if a client is on Weblogic

8.1 and Oracle 8i and the current technology stack cannot conform to the performance SLA required, the architect should be able to articulate the magnitude of performance improvements possible by an upgrade to a later version of Weblogic and Oracle.

An important but often overlooked aspect is the distinction between “latency” and “throughput.” Latency is the time taken for the performance of a user transaction while throughput is the total amount of work done over a period of time.

A good architect is able to take a judgmental call on both latency and throughput of applications during the architecture definition phase. The architect uses the workload model and target hardware platform to define performance capabilities.

Assuming an unconstrained (by processing, memory or I/O) system the architect is able to define performance bounds (for latency and throughput) and apply sufficient “damping factors,” based on environment factors (hardware, network, contention on shared resources, pipelining of operations, etc.) to arrive at feasible performance numbers that need to be met. This is the input for the design team. For example – for a screen that is expected to take 20 seconds in production, the architect may define that it has to complete in 12 seconds in a single user, unconstrained mode (without CPU, memory or I/O bottlenecks) to be able to achieve the performance SLA.

THE CONSTRUCTION PHASE: CRYSTAL BALL - PREDICTING PERFORMANCE

“If you have to forecast, forecast often.”

- Edgar R. Fielder (Assistant Secretary for Economic Policy, U.S.A)

We are able to define certain (predictive) attributes and goals for a project even before the

first line of code is written. For instance, project managers are asked to 'predict' the number of delivered defects, number of defects that will slip through to the end users, effort required to fix them etc. Many of these predictive attributes are derived from analyzing empirical data of past projects.

Unfortunately, qualitative performance attributes have never been easy to predict. Unlike standard program features, which can be tested using a comprehensive set of test cases or automated tools, the huge Δ between the development and production environments (hardware differences, configuration of our software, the app-server stack, the database, underlying network etc.) makes it very difficult to predict the behavior of the software product on the target environment.

This is where we recommend taking from where the architect left off and designing for performance.

Let us take the above example where the architect predicted that to meet the feasible production performance SLA of 20 seconds the single-user performance should be 12 seconds in the development environment. The tech lead splits the transaction into various logical constituents, based on the computation and data involved. For example, the 12 seconds might be spread into 1 second for UI, 4 seconds for application layer processing, 5 seconds for database access and 2 seconds for display rendering. The lead then consciously designs the application to meet these criteria and has these metrics to :

- Test for performance for each tier, often and
- Refactor for performance, as necessary, in the design phase.

The magic crystal ball for predicting and assuring quality performance revolves around adhering to some very basic principles:

- Automate enforcement using plug-ins wherever possible - for e.g.,
 - PMD for static code analysis
 - VTune Performance analyzer.
- Mandatory use of Profiling tools—for e.g.,
 - TPTP framework for Eclipse
 - Radien profiler.
- Usage of tried, tested and optimized frameworks and common components
 - Radien
 - SPEED .NET
- Repeated validation - once a module, however small, is complete, it should be packaged, deployed and tested for performance on the target environment (either a production mirror, or any environment that comes closest to the target deployment platform). Such repeat tests provide us with a fairly good indicator of the difference in performance between the development and target environments. Once we have tested the smaller pieces, it is possible for us to predict the performance of the larger pieces based on empirical data from the smaller modules. While these plug-ins and tools serve to remove a lot of the drudgery involved in reviewing code and ensuring that they adhere to an ever increasing number of standards and guidelines, they are no substitute for human code reviews and walk through. The expectation from human code reviews is to catch things that the plug-ins cannot - for e.g., usage of sub-optimal sort routines, using improper data structures etc.

QUANTIFYING PERFORMANCE RESULTS: PERFORMANCE REFERENCE VALUES (PRV)

Every software release is accompanied by a rigorous testing phase to not only validate new functionality, but also to ensure that existing routines are regressed properly. But what about degradation in performance because of changes in the code? It is very important for projects to invest sufficient time and effort in putting together a (re)usable framework to quantitatively measure application performance as often as possible, and make it a mandatory part of the release notes that accompanies every software release. This is where the performance reference value framework is applied.

Deriving the performance reference framework for an application involves:

- Defining the load model : The load model documents the “day-in-the-life-of” kind of scenario from the System perspective. How many transactions, how often, how much think-time between transactions, number of users, ramp-up and ramp-down times etc.
- Automation: Automated load testing tools like Mercury Load Runner® are used to mimic the load model and generate sufficient system load. In order to make such load tests repeatable and self-sufficient, it is essential that required test data be set up automatically by the test framework itself for every iteration
- Define a template that captures the transaction timings from load tests and compares these against the expected SLA values. Roll-up the timings across all transactions and we arrive at the overall system performance metric
- In general, it is a good idea to deploy

probes to monitor the performance parameters of the underlying operating system stack - CPU, disk IO, paging, memory profile and network traffic for the duration of the performance test to assist in diagnosis.

Figure 1 shows the PRV Summary sheet for an assortment planning application for a large apparel retailer in the U.S. Every release to production is load tested and the PRV is validated for conformance.

The benefits of having a PRV framework go beyond just absolute application performance. The PRV is a statement that tells the business users how long it takes them to complete a given business process. If the expected PRV stands at 11 hours and the actual application PRV is 7 hours, it directly translates to a 4 hour gain to the business process. This can be positioned as a tremendous value-add to the business process from an IT standpoint.

CONCLUSION

Our experiences in addressing performance issues in projects converge to a few basic tenets:

- Performance is a mindset, just like quality and should be an integral part of each phase the SDLC
 - **Architecture definition phase**
 - distinguish between “latency” and “throughput,” define feasible SLAs for both; translate these into “unconstrained” limits for the design team
 - **Design phase** - Using the performance limits provided by the architect, split the module of work into constituent tiers (separation of concerns) and estimate for each tier,

General requirements based on:		Prod	Test	
Users:		200	100	
# of items & colors:		60	60	
# of items:		20	20	
# of Colors:		3	3	
Type of plan for tests is:		fashion/basic		
Subclasses:		Yes	Yes	
Primary store set:		Climate	Climate	
Test information:				
Test number:		377		
Build number tested:		381		
Standalone or combined:		Standalone		
Results type:		80 percent		
Date of test:		2/7/2006		
Performance Results				
last updated: 03/07/2006 09:13:29pm		Expected performance reference value (hours) 11:90	Actual performance reference value (hours) 7.38	
		11.90	7.38	
		Extended requirement time (frequency X requirement) (minutes)	Extended response time (frequency X performance) (minutes)	Variance
Create plan subtotal		1.40	1.78	(0.38)
Create mock/actual items subtotal		8.50	1.63	-
Define calc parms subtotal		54.92	26.31	-
Create assortment plan subtotal		19.90	3.67	-
RPA4 flag subtotal		18.83	10.57	-

Figure 1: PRV Summary view


Source: Infosys Experience

based on quantum of work; adhere to those limits (in initial design and refactoring)

- **Construction phase** – test for performance often; use tools to automate the tests and analyze results

- Use of frameworks, plug-ins, tools and profilers to repeatedly measure performance
- Define a repeatable (automated) test framework, define and derive the Performance Reference Value (PRV) framework for every project.

REFERENCES

1. T. Bollinger, "The Interplay of Art and Science in Software," IEEE Computer, Oct. 1997, pp. 128, 125-126.
2. JPLewis, "Limits to software estimation", ACM Software Engineering Notes, Jul 2001
3. <http://www.eclipseplugincentral.com/>,
4. Steve McConnell, "Code Complete", Microsoft Press, 2nd Edition, 2004 .
5. Joshua Bloch, "Effective Java ProgrammingLanguageGuide", Addison Wesley Professional, 1st Edition, 2001 

Authors in this issue

SRIDHAR SHARMA

Sridhar Sharma is a Senior Technical Architect in the Retail, CPG and Distribution business unit, Infosys. He can be contacted at sridhar_sharma@infosys.com.

VAIDYANATHA SIVA

Vaidyanatha Siva is a Principal Architect and Head of CPG Technology Consulting the Retail, CPG and Distribution business unit, Infosys. His interests are in distributed component systems and architecting mission-critical enterprise-class applications. He can be contacted at siva_vaidyanatha@infosys.com

For information on obtaining additional copies, reprinting or translating articles, and all other correspondence, please contact:

Telephone : 91-80-41173878

Email: SetlabsBriefings@infosys.com

© SETLabs 2006, Infosys Technologies Limited.

Infosys acknowledges the proprietary rights of the trademarks and product names of the other companies mentioned in this issue of SETLabs Briefings. The information provided in this document is intended for the sole use of the recipient and for educational purposes only. Infosys makes no express or implied warranties relating to the information contained in this document or to any derived results obtained by the recipient from the use of the information in the document. Infosys further does not guarantee the sequence, timeliness, accuracy or completeness of the information and will not be liable in any way to the recipient for any delays, inaccuracies, errors in, or omissions of, any of the information or in the transmission thereof, or for any damages arising there from. Opinions and forecasts constitute our judgment at the time of release and are subject to change without notice. This document does not contain information provided to us in confidence by our clients.

Infosys[®]

POWERED BY INTELLECT
DRIVEN BY VALUES