THE ART OF MEASURING THE PERFORMANCE OF THE SOLUTION
In the context of a web site, some assume that performance is the measurement of page response time. In the context of a messaging system, some measure performance in terms of transactions per second. We prefer to take a holistic approach to performance, looking at all the characteristics and varying the weighting according to the nature of the solution.

WHEN EVALUATING IF THE SOLUTION PERFORMS WELL, WE OBSERVE THE FOLLOWING CHARACTERISTICS

Business Goals

Business goals are arguably the most fundamental and important attribute, and also the most likely attribute to fall off the list. Software that doesn’t meet business goals is synonymous with Joe, the musical performer, singing songs from the wrong musical. In fairness, attainment of business goals is a characteristic that must be evaluated much earlier in the process. We prefer to continue tracking this characteristic to ensure ongoing changes don’t impact the final solution’s ability to deliver the intended value.

Functionality

Many solutions do not fulfill all the requirements initially defined for them. That’s certainly not to say the solutions are flawed or incomplete. It’s simply the reality of the dynamics introduced by time, and proof that software development is truly dynamic. Even with a dynamic requirement list, we track requirements and their implementations continually so we can maintain an ongoing known state to evaluate against the goal.

Quality

Most software contains defects, it’s true. Some are known, and some are not. The goal, of course, is to know about all of them so we can manage the severity, impact and effect and meet pre-determined quality metrics.

Cost

Money is rarely an unlimited resource. Therefore, cost is a factor of every solution. It does happen that we see solutions where a simple, reduced feature set is desired
for various reasons. More often, however, a richer feature set is desired with constrained costs. The trade-off in these cases is seen in the granularity of the features or the level of customization that is possible.

This is where a deep understanding of the options available and the relative impact of constraints come into play. Understanding the options available can only be accomplished with deep understanding of a number of platforms as well as hands-on experience implementing customizations. This knowledge is acquired with research and regular participation on delivery teams. Understanding the impact of potential restrictions, on the other hand, is very complex and seems best achieved through not just experience, but living with the requirements through many implementation cycles. Deeper experience brings increased understanding of the underlying business needs that drove the original requirements, and allows SA’s to make recommendations that will meet the intent as well as the expressed requirements in balance with cost.

**Experience times**

Experience time is a measure of speed as perceived by a user of a website or an application. There are experience techniques that can affect human perception. However, these are not necessarily quantifiable or consistent across an entire site. More commonly, the primary factor evaluated for a website is latency, or how long it takes for the request to be sent, the data processed, any work performed, and the data sent back. Often we’ll break this down further into multiple measures such as how long the first bits take to come back, and how long a whole page takes to render. In most cases, for websites we evaluate these metrics for many browsers. There is certainly an entire topic here, but for the purposes of this paper suffice it to say that latency and the feel of speed by the client are critical components of performance criteria.

**Throughput**

Throughput is critical for all solutions, but takes on a similar importance to experience times in a web solution. Throughput is a metric that is highly dependent on the scalability metric and must be measured with various loads, especially for higher load solutions. For example, if each message takes an average of $x$ time units, and the deployment is capable of handling $y$ instances at a time (parallelism), we can calculate at what point queuing will begin. In most cases, this is the threshold we strive to maximize.
Scalability

Scalability is less a measure of the solution and more a measure of the behavior of the solution as it's expanded. Some solutions have bottlenecks that limit the effect of any added hardware. Such bottlenecks may not present an immediate performance concern, but may raise architectural concerns as the solution is expanded. Other solutions may respond well to added servers (or instances) but don't benefit from added resources. Still others consume resources and improve latency or throughput metrics. Ideally, these effects are not accidental but rather are designed for, and of course we always want to limit bottlenecks.

Reliability

I often see requests for 5 9s, and we do have models that we can populate with average software and hardware times-to-fail, as well as levels of redundancy and a number of other inputs to come up with deployment architectures that can indeed yield an effective 99.999% up time. In many cases, reaching this reliability goal in not in line with the revenue aspirations of the solution. However, when it is, we can use our models to design the software and hardware infrastructure, select appropriate data centers (yep, you need more than one), and monitor every aspect of the system with incredible care.

Manageability

Even the simplest software requires care and feeding. Most of us have had the experience of huge downloads and rebooted computers with periodic updates, just for our personal systems. Extrapolate that to an enterprise system made up of many operating systems, databases, application servers, web servers and even multiple platform technologies. If this sounds challenging, it can be if it's not accounted for and designed for up front. When we evaluate the manageability of the solution, we are looking to gage the level of effort and skills needed to keep the system running smoothly.

Upgradability

Earlier, we alluded to the effect of time on any system, not only as requirements evolve but also as the capabilities of the underlying platform change over time. Designing for and evaluating upgradability is critical, especially for enterprise systems that are designed to last for many years. There is often a cost to additional performance characteristics when designing for upgradability, since more upgradability often takes the form of increased decoupling. Like
all attributes of performance, upgradability plans must be balanced according to need when designing.

**Extensibility**

Extensibility is an attribute often associated with an enterprise-grade product offering, since we often don’t know all the details of the additional components that will make up an end solution. That said, when evaluating a solution we’ll often look at the extensibility of the entire solution. Are all the integrations centralized, or do we have to integrate with every component individually? Can we add new components to the solution, or are the possibilities to do this closed? These are important questions to answer in the definition phase of a solution, since it’s inevitable that changes will be necessary. However, like all these attributes, it’s a balance. You don’t want to compromise other factors while trying to yield a highly extensible solution that no one can afford, or that becomes unmanageable or can’t be upgraded.

All these characteristics, as well as others specific to a given solution, must be evaluated jointly to answer the question: Does this system perform well?
The Art in Measuring the Performance of the Solution

Insights

Learn More at www.smith.co

SMITH

©2014 SMITH