

2009

PACIFIC NW SOFTWARE
QUALITY
CONFERENCE



MOVING
QUALITY
FORWARD

OCTOBER 27-28, 2009

Conference Paper Excerpt

From the

CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

New Challenges to Quality in the 24x7 Enterprise I.T. Shop: Post-Integrated Business Automation Systems

Al Hooton, CIO
al@hootons.org

August 2009

The traditional 24x7 IT operations organization is often perceived as only needing sub-par technical professionals. This has often not only been the perception but also the reality for many of these organizations because the systems they would purchase and deploy have been “pre-integrated” by the companies providing the systems. All too often this limited the ability for the IT operations professionals to get very far under the hood, relegating them to roles requiring less technical prowess than their product development counterparts. Reality has been changing rapidly in the last several years as new approaches to the design and integration of complex business systems has thrust these operations professionals into a new world of “post-integrated” systems. This paper discusses the four primary forces behind these changes, and provides a few examples of approaches the 24x7 IT shop can borrow from their product development brethren in order to meet the skillset and innovation requirements of this new post-integrated age of business systems.

Al Hooton has been professionally active in the software development and information technology industries for the last 30 years, holding various positions including computer operator, hardware/software engineer, chief architect, quality assurance manager, Director of Technical Operations, Chief Information Officer and CEO/President. His experience is evenly split between software product development positions and 24x7 I.T. operations positions. Mr. Hooton holds a Baccalaureate degree in Computer Science from Purdue University and a Master of Science degree in Computer Science from the University of Louisiana.

Copyright © 2009, Al Hooton. All Rights Reserved.

Traditional Differences Between Computer Engineering and Computer Operations Personnel

For most of the 65+ years that electronic information processing systems have been in use, there has been a distinct difference in perspective regarding the skills necessary for “designing and building stuff” using computers versus “running stuff” using computers. Designing and Building Stuff (otherwise known as one or more areas of *computer engineering*) requires academic foundations, formal techniques, ascendancy through multiple levels of mentored guidance provided by more senior engineers, and various other experiences that result in one possessing a highly respected set of skills. On the other hand, Running Stuff (otherwise known as *information technology operations*, or “I.T.”) is perceived to require few or none of these things, being entirely addressable by smart self-learners who have taught themselves how to fix password problems and call the vendors (or engineers) who created the systems they are running when anything difficult has to be done.

I.T. professionals are understandably quick to defend their abilities, and will argue strenuously that they are expected to have the same levels of competence as their engineering counterparts. When the author has been thrust into debates regarding this set of assertions, the argument is quickly resolved by retrieving several current job postings for both engineering and operations positions for comparison. Here are excerpts from two job postings current at the time this paper is being written, both of which are looking for “mid-level” professionals:

Computer Systems Engineer

- Bachelors degree in computer science, computer engineering or electrical engineering required; Masters degree highly preferred
- 7-10 years experience designing, building and maintaining complex information systems
- High levels of proficiency with the following technologies: UNIX/Linux, Java, EJB, SQL, C/C++, operating system internals
- Prior demonstrated success working as part of an engineering team

I.T. Operations Engineer

- Bachelors degree desired; high school diploma with equivalent experience may be substituted
- 2 years experience in the operation and maintenance of both Windows and Linux systems
- Familiarity with concepts and usage of the following types of systems: ERP, accounting, human resources, data warehousing
- Familiarity with shell scripting and writing administrative tools in perl

This is not an indictment of the operations professionals themselves, rather it simply points out there are different expectations in the formal education and technical experience levels on the part of the information systems industry overall. There are certainly exceptions to this rule, but anybody who is active in this industry will recognize the differences as more common than not. The operations professionals are often viewed as “second-string” resources who do not need to be as technically competent as their engineering counterparts.

Historical Causes for These Differences

The differences in technical skill level between engineering personnel and operations personnel is most often real, not just perceived. Businesses are perfectly happy for these differences to exist because operations personnel are typically paid less than engineering personnel. The accountants and executives see this as an advantage since the computers keep running almost all the time anyway, at lower labor cost.

This state of affairs has developed over several decades. While there are several contributing factors, the author contends that the most influential is the “pre-integrated” nature of typical enterprise-scale information systems.

Successful I.T. shops have brought in business analysts to work with all the groups around the company to understand the relevant business requirements, and would then evaluate large systems from several vendors who each claimed their system could meet the requirements. A choice would be made to select a single vendor for software and a single vendor for hardware (often the same) because this was the only way to ensure all the pieces of the system worked together [1]. These traditional systems lack flexible integration points at their edges and typically utilize proprietary internal integration architectures between their primary functional subsystems, thus are referred to here as being pre-integrated.

These pre-integrated systems typically go through an extended deployment period, during which the vendor(s) provide costly professional services to make changes to the selected system so it actually does meet the needs of the business. The operations personnel would be unable to make these changes even if they had all the correct skills, due to the closed nature of these systems and the lack of external integration interfaces. For these same reasons, once the system is deployed the operations personnel are typically relegated to a role of monitoring the system to ensure it is functioning correctly and calling the vendor for help at the first sign of trouble or when functional enhancements are required.

An obvious issue with these approaches is the Total Cost of Ownership (TCO) for such pre-integrated systems. Any experienced enterprise I.T. executive will tell you this scenario results in the on-going costs for a system far outweighing the initial costs of purchase and deployment. Vendors of these systems know they essentially have their customers stuck with no alternatives but to come back for help or modifications, so those services become very expensive. Until recently this was the path most likely to lead to success for the operations personnel in supporting the needs of the business, regardless of cost, so it was the one most often chosen. However, in the last several years, some fundamental changes in software technology have altered the I.T. operations landscape. These changes are not only driving fundamental shifts in how I.T. shops purchase, build and deploy information systems but they are also driving changes in the skills that operations personnel must possess in order to succeed.

The Dawn of “Post-Integrated” Enterprise I.T. Environments

The realities described above have been shifting in the recent past, with the result that everyone in the enterprise I.T. shop is struggling with new expectations, new technical challenges and a completely different approach to the purchase/build/maintain cycle for the business systems they deploy. The primary reasons for this shift are described in the next several sections of this paper, but the combined result may be described as a move away from “pre-integrated” systems to “post-integrated” systems. This new world is one where the overall set of functionality deployed to meet the needs of the business is no longer procured from a single vendor. Rather, there are systems from many vendors being procured, modified, integrated and deployed, in addition to systems that may be constructed from scratch internal to the I.T. shop. This is a radical departure from the past, and is driving operations personnel to be proficient in enterprise-scale software development, multi-level systems integration, management of long-term architectural roadmaps that drive system evolution, etc. In short, operations personnel now must be proficient in all the skills previously expected only of computer engineers.

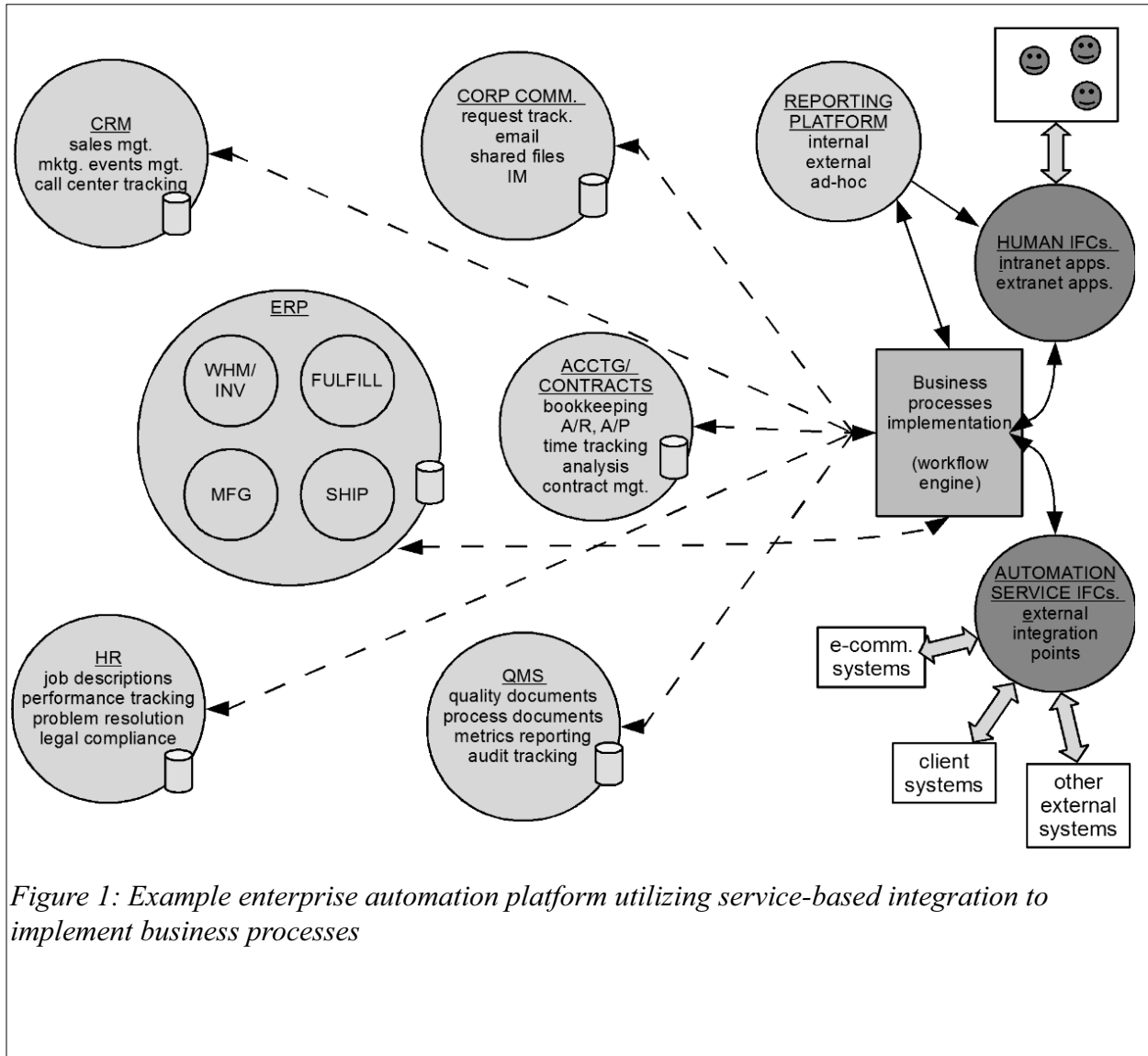
The primary reasons we are entering the post-integration era are discussed in the following sections of this paper. In one way or another they all provide *new ways to think about and implement integration between heterogeneous systems that is driven by business requirements instead of technical constraints.*

Service-Oriented Architectures (SOAs)

With traditional enterprise-scale information systems, it was effectively impossible to utilize functional primitives within the system except through a user interface provided by the vendor of the system. For example, a warehouse management system might implement a functional primitive such as “move N items of part number X from the overstock area to the manufacturing area”, and expose that functional primitive through a series of user interface screens on terminals in the warehouse. However, there were no other interfaces that could be used to trigger the execution of this functional primitive. Thus, even though a business process might exist that requires the value of the moved inventory to be credited to an overstock financial account and debited from a manufacturing financial account, there is no way to integrate the accounting system with the warehouse management system in order to automate the enforcement of the business rule.

SOAs [2, 3] allow systems to be built such that functional primitives communicate with each other, and with

external systems, via standard mechanisms called a “service interface” or “web service” (depending on technical details). By using such a standard mechanism for both internal integration and external integration, functional primitives from across multiple systems (i.e., the warehouse management and accounting systems) may be combined in order to automate cross-functional business processes. Figure 1 shows an example enterprise architecture where a workflow engine is programmed to implement business processes by utilizing functional



primitives across all the back-end systems.

Enterprise-scale SOA approaches allow multiple systems from disparate vendors to be easily integrated by parties other than the vendors. This work is most often done in the I.T. shop by the operations professionals, requiring them to be proficient in many areas of software development, integration and testing.

Core Services versus Non-Core Services

The traditional enterprise I.T. shop runs all automation systems necessary to support the business on-site, or “behind the firewall”. This would include the most critical systems, such as the accounting system and systems vital to

generating revenue, as well as non-critical or niche systems such as email list servers or limited-use websites. Since everything was pre-integrated this was often required, but it resulted in the operations personnel being diverted from ensuring the critical systems were running whenever a non-critical system needed attention because everything was internal and running on the same infrastructure.

In recent years a philosophy has emerged that suggests the I.T. shop does not provide *systems* to the business, it provides *services* [4]. Those services are automated on computer systems, but there may or may not be a one-to-one correspondence. For instance, the business may require automated inventory tracking services in order to be competitive. Internal to the I.T. shop there may be multiple systems that are used together in order to provide these services. This philosophy can provide the I.T. shop a great deal of flexibility in how it plans and deploys systems, but only if the commitments it makes to the business are truly made in terms of services instead of systems.

Once the commitments are defined in terms of services, the I.T. shop has the ability to work with the business to define *core services* and *non-core services*. The definitions will vary between organizations, but in general terms the core services are usually directly related to the immediate generation of revenue and require high availability, high data reliability and high visibility. Alternatively, the non-core services are usually not directly related to the immediate generation of revenue and can operate at lower levels of availability, etc. This allows the I.T. shop to make different implementation and priority choices when determining how to deliver services to the business, and allows operations personnel to more effectively spend their time on systems that deliver core services.

This set of philosophies can significantly alter the demands on operations personnel if it is determined that systems delivering non-core services can or should be physically disparate from those delivering core services. This is a frequent outcome, due to the high cost of hosting systems in operational environments that provide everything required to meet the availability and reliability needs of core services. As soon as separate physical locations are involved between these two sets of systems, operations personnel will be required to master sophisticated network engineering, replication, fail-over and remote management skills.

Software as a Service (SaaS) delivery mechanisms

In recent years the model for delivering non-core services to the business has started to shift toward the utilization of Software as a Service (SaaS) delivery from a mix of external providers. SaaS [5] allows delivery of services to the business from remote data centers on a subscription basis, with the business having no ownership of the systems delivering those services or responsibility for maintaining those systems. There are several established examples of this, such as Salesforce.com, as well as many emerging examples providing a full range of business services. In his current position, the author has directed several initiatives that have resulted in many non-core services being migrated or launched on systems that are delivering the services in a SaaS model, including all corporate email services, shared calendaring services, company-wide issue/request tracking services, document control/tracking services and corporate intranet services. The result has been increased ability for operations personnel to monitor, maintain and enhance the systems remaining behind the firewall that deliver core services to the business.

Many companies that deliver their solutions using a SaaS model have consciously built sophisticated integration interfaces to these services. The emerging approach for these integration interfaces is to use SOAs and expose the resulting service interfaces through secure networking connections. In this way, I.T. shops that are utilizing capabilities being delivered to them via SaaS mechanisms have the same ability to integrate with those systems as they would have if they were running everything locally.

The emergence of SaaS as an enterprise-ready option further demands that operations personnel become familiar with sophisticated approaches to network engineering, heterogeneous systems integration and assembly of end-user services from multiple functional primitives.

Free Open-Source Software (FOSS)

Although Free Open-Source Software (FOSS) [6] has been around since the 1970's, it has traditionally been viewed with a combination of suspicion and skepticism by enterprise I.T. shops. There are a variety of reasons for this, but the most common is the belief that "there's nobody to call when it breaks". For good reason, operations personnel are most often judged on the predictability of the services they provide – the fastest way for an I.T. shop to get in trouble is when system issues cause the services they deliver to be unpredictable in some manner. The best way to avoid this problem was to ensure there was a vendor who could be called on (and blamed, when necessary).

As a general rule FOSS systems have no vendor, or the vendor is releasing a FOSS version of their system with no included support, as a way to introduce the market to their product and hopefully convince people to purchase a full commercial version. Most FOSS systems are built and maintained by a loosely-connected set of developers who may not have ever met in person and do all their work on the system virtually. While this may give traditional operations personnel reason to pause, there are many well-known FOSS systems that are more stable, exhibit fewer bugs, drive a lower TCO and are repaired much more quickly when problems occur than their commercial competitors. For these reasons FOSS systems are increasingly chosen in many I.T. shops to provide a variety of services to the business.

The appearance of FOSS requires operations personnel to not only become competent with the full range of integration skills described previously, but in many cases will require them to also become fully proficient software engineers. This is required when they must make modifications of a FOSS systems to meet their specific business needs, or they are required to make fixes to the system if a bug appears. FOSS systems allow a number of advantages over commercial systems, but operations personnel must have the skills necessary to successfully make use of these advantages.

Leveraging Well-Known Software Engineering Quality Techniques in the I.T. Shop

With the 24x7 I.T. shop being bombarded with these new challenges (and opportunities) it is common to find operations professionals struggling to successfully meet the needs of the businesses they support. The new opportunities provided by the changes discussed above are not fully leveraged in most shops, and are not being used to any degree at all in many. Operations professionals are in a tough spot, and the only way out will be to aggressively pursue new skills and then apply those skills to the design, procurement, integration, deployment, maintenance and operations of their systems going forward.

Like it or not, operations professionals and also the businesses that employ them need to recognize that fundamental preparation in an academic setting is now becoming necessary for the I.T. shop just as much as it is necessary for product developers. College degrees in Computer Engineering, Computer Science or related fields will become mandatory for success in the era of post-integrated systems. The businesses that employ professionals with these new levels of preparation will need to pay them at levels commensurate with product developers as well. This additional expense will be returned to the businesses many-fold in the form of enhanced capability, flexibility and reliability of the resulting business automation systems.

Through the course of formal education, operations professionals will be exposed to many approaches and techniques that have been familiar to software product developers for decades. Most, if not all, of these techniques are applicable to the world of post-integrated business automation systems. Below are a sampling of lessons derived from the experience of the author in applying well-known software engineering approaches to I.T. operations teams.

Lesson 1: Incremental development, integration and deployment (avoid the Big Bang)

The typical approach to deploying enterprise-scale business automation systems looks a lot like the classic waterfall approach to product development that was discarded by product developers decades ago. In the 24x7 I.T. shop there are often very long periods of requirements gathering and sign-off, followed by equally long periods of defining functional specifications based on those requirements. When all this work is done vendors receive Request For Proposal (RFP) documents that are often hundreds of pages long, and they are expected to respond to these RFP documents with proposals that meet all the needs of the business in a single system to be rolled out across the company in a single deployment. After further long periods of selecting a vendor, having the vendor modify their system to meet the needs, and a long period of testing with a small group of people, the day comes to “flip the switch” to the new system. Product developers often refer to this kind of approach as the “big bang” technique, in that everything is supposed to be fully compliant with the initial requirements all at once, when the switch is flipped. This rarely works, resulting in alternate definitions of “big bang” that are much more negative in nature. This kind of approach is, at best, problematic with pre-integrated systems. It is significantly more difficult with post-integrated systems, and it misses a key potential advantage that post-integrated systems offer: the opportunity to build and deploy in phases, allowing the management of complexity to be tackled in smaller chunks.

Over the last two decades there has been great progress in the product development community in the area of *incremental development and integration* techniques. This started with the work of Barry Boehm in the 1980's [7, 8]

and exists today in various forms that include a set of approaches collectively referred to as *agile development* [9]. All of these approaches share a few important understandings:

- It is not possible to know everything a complex business automation system will need to do before it is first built.
- It is not possible to manage all the complexities of a full enterprise-scale business automation system at the same time. Success comes from dividing the system down into pieces that are incrementally developed, integrated and deployed at different times, some earlier and some later.
- By explicitly recognizing at the outset that there will be many outstanding questions, and utilizing an approach that develops, integrates and deploys the entire system over many incremental cycles, it allows the team to focus on giving the business users what is known to be useful and then leveraging those things to further define what is not yet known before attempting to build, integrate and deploy additional functionality.

24x7 I.T. shops can deliver great value to their business users, more quickly, and avoid suffering one of the often-reported large-scale system deployment failures if they adopt incremental development and integration techniques.

In a recent position, the author was responsible for several groups of professionals in both internal development and operations support roles who were building, integrating and running a large real-time financial transactions platform. The teams had been attempting to utilize a variety of agile approaches, but had been quite disappointed in the results because they found themselves significantly over-committing for each build/test/deploy iteration and subsequently failing to achieve 60% to 70% of those commitments. When the author joined the organization, many of the business people internal to the company were prepared to force the technical teams to abandon their “new” agile approaches because these approaches had failed to meet the needs of the business any better than the previous “big-bang” approaches.

After meeting with each of the teams involved individually, and then through a series of cross-functional meetings with all the teams, a critical lesson was uncovered. Even though many aspects of current-day agile techniques were being used, a fundamental requirement for success was missing: the requirement that each iteration culminate in a customer-ready system. This had been the intent when the switch to agile approaches was first adopted, but the operations team found that customers were unnerved by frequent updates to their production financial systems. Due to this the operations team had made a well-founded decision to “skip” several iterations at a time that were released from the developers. While this is a fine approach, it caused the unintended result of QA and testing efforts to be reduced for the skipped iterations. In this organization, responsibility for the QA and testing work fell to operations instead of development, but when the demands for quality dropped for skipped iterations there were serious defects remaining that went undetected.

Everybody thought things were going great until work started on an iteration that was intended for external release. The QA and testing efforts ramped back up and proceeded to find that the last several iterations had not actually resulted in a customer-ready system being produced. This smaller version of the “big-bang” approach would then cause significant commitment and schedule overruns for the customer-visible releases.

The Lesson: Just because the organization determines that it is not going to externally release the output of every iteration in an agile approach you still must truly produce a customer-ready system with each iteration.

Lesson 2: Integration is the other 90% of the work, focus on it from the beginning

A statement frequently heard on product development teams is that “writing the code is the first 90% of the work, integrating the code into a system is the other 90% of the work.” This is fine when it is the development team that is doing the integration but, as discussed previously, it is increasingly the responsibility of the operations team to do the majority of integration work between heterogeneous systems. The act of integrating complex system components together to form a complete user-ready system is difficult because it is during integration that the management of full system-wide complexity is truly required. This made that much more difficult when the systems being integrated are from multiple vendors, and the operations team was not involved in the original development of most or all of them.

Many years ago, researchers in product development techniques identified integration as the place that the most difficult or complex system requirements would either be met or fail to be met. Two important categories of techniques have emerged from this set of understanding that are in wide-spread use among product developers:

- Sophisticated configuration management approaches to track subsystem dependencies, manage multiple released versions, allow easy and rapid roll-back of any given release if problems emerge, and manage how the overall system evolves through time due to incremental development/deployment.
- Utilization of higher-level modeling techniques to manage evolving system complexity. There are a variety of diagram-based modeling techniques such as Unified Modeling Language (UML) [10] that can be used with both technical and non-technical stakeholders during the early stages of projects to understand how business requirements interact to drive technical requirements.

Several years ago, the author was responsible for both the product development team and the 24x7 operations team at a company delivering a web-based complex document preparation system delivered in a SaaS model. At the time he joined the company, there had been a series of accidental configuration, deployment and operations mistakes that disrupted service to customers. While these mistakes were usually identified and rectified quickly, they had a negative impact on the reputation of the company with both customers and prospects.

In this case the root cause was easy to discover. Even though the development team was keeping comprehensive configuration management records during development and testing of the system, no such efforts were being made to track customer-specific configurations at the time of deployment. This caused two problems:

- The development team was unable to readily see what was happening to the system when it was being deployed, preventing them from understanding how to better build future versions of the system to ease operational problems.
- There was no way to apply automation to either maintaining or re-applying customer-specific configurations after an upgrade of the core system. Manual records were kept, but it was error-prone to re-apply the configurations after an upgrade and there was no way to support automated regression testing against anything except the core system configuration.

The development and operations teams worked to extend the configuration tracking system used in development so it would hold configuration profiles, deployment scripts and automated tools that would compare any deployment of the system against the records in the configuration tracking system to identify inconsistencies. As the first version of this new approach became available, customer interruptions due to incorrect deployment/configuration exercises essentially disappeared.

The Lesson: Managing system-wide complexity only at the time of integration is difficult. By managing complexity consistently (in this case with a configuration management system that spanned the entire life-cycle of each product iteration) it is possible to keep the complexity under control.

Lesson 3: Design it like you plan to deploy it

One of the most common refrains in a 24x7 operations shop is “what were these people thinking when they built the system like this?” This is often heard when the operations people find that something they think about every day of their professional lives, such as being able to monitor the system resources being utilized by an application system, never crossed the minds of the people who built the system. This is a frequent reality, especially in the pre-integrated world of the past, but is also found in the post-integrated world that operations professionals are increasingly supporting.

The good news is that the same forces causing the 24x7 I.T. shop to achieve competency in complex systems integrations also provide many of the hooks necessary to make the resulting systems easier to deploy and monitor. The integration approaches used to connect various business systems together, especially SOA approaches, also lend themselves to being utilized by deployment, configuration and monitoring tools.

However, in order to reap these benefits, the operations personnel who are designing and building the integration software between the primary systems must “design it like they plan to deploy it.” This means that the requirements for the overall system must not only be written in terms of important end-user functionality, but in terms of processes for deploying, configuring and monitoring performance of all systems involved. In some circumstances it can even be the case that these additional requirements will have an effect on how the end-user functionality is implemented, allowing for better auditing of activity in the system.

The author would like to report specific success with this approach, but is currently embarking on his first attempt to use it in a comprehensive manner. In his current position, a multi-year project is getting underway to upgrade or replace the majority of core systems operated by the I.T. organization. All of the approaches discussed in this paper are being used, with the intent to incrementally deploy a flexible system that can adapt quickly to changing needs and removes the “single-vendor handcuffs” that come along with pre-integrated systems.

The (Expected) Lesson: Place deployment, configuration and maintenance requirements at the same level of importance as user functionality requirements – the resulting ease of operations and stability of the system over time are just as important to end-user value.

Remember that change is your friend, not your enemy

The traditional 24x7 I.T. shop was typically a strong proponent of the status quo, preferring that the systems they were responsible for did not change frequently (or at all, if they could get away with it). This grew out of the fact that their success was most frequently measured in terms of stability, not in terms of innovation. The policies and procedures put in place were designed to ensure the highest levels of stability in systems, and frequently anticipated changes to those systems as rare anomalies.

Those days are gone, and operations professionals need to understand they are now expected to deliver value to the business through innovation just as much as their product development counterparts. Contemporary software development methodologies can be applied within IT shops and bring value both to the services they provide and to the operations professionals themselves. Being successful going forward will require that the policies, procedures and techniques employed in 24x7 I.T. shops embrace change and use it a central force to be successful in the eyes of their businesses.

References

- [1] – Wikipedia.com, “Fear, Uncertainty and Doubt”, http://en.wikipedia.org/wiki/Fear,_uncertainty_and_doubt, 7-Jul-2009
- [2] – Open Service Oriented Architecture collaboration, introductory material, <http://www.osoa.org/display/Main/Home>, 30-Apr-2009
- [3] – OASIS, “OASIS Committees by Category: Web Services”, http://www.oasis-open.org/committees/tc_cat.php?cat=ws
- [4] – IBM Corporation, “video: Service Management vs. Systems Management”, <http://www.youtube.com/watch?v=guHgiSOHImA>, 23-Apr-2009
- [5] – Wikipedia.com, “Software as a service”, http://en.wikipedia.org/wiki/Software_as_a_service, 23-Jun-2009
- [6] – Wikipedia.com, “Free and open-source software”, <http://en.wikipedia.org/wiki/FOSS>, 6-Jul-2009
- [7] – Academic website for Barry Boehm at the USC Center for Software Engineering, http://sunset.usc.edu/Research_Group/barry.html
- [8] – Wikipedia.com, “Barry Boehm”, http://en.wikipedia.org/wiki/Barry_Boehm, 19-Jun-1009
- [9] – Original Agile Manifesto, <http://agilemanifesto.org/>, 13-Feb-2001
- [10] – Object Management Group, “UML Resource Page”, <http://www.uml.org/>, 10-Mar-2009