Agile Deployment: Lean Service Management and Deployment Strategies for the SaaS Enterprise

Robert Benefield
British Telecom plc
robert.benefield@bt.com

Abstract
The Software as a Service (SaaS) delivery model has recently become an acceptable business model for software delivery. However, the attractiveness of SaaS to customers, with their low upfront costs and seamless delivery, changes the game in ways that challenge traditional software development methodologies. As customers begin to treat software as an on demand utility, expectations of high uptime, predictable quality, strong security and privacy, and lower total costs become at least as critical as time-honored feature functionality requirements. The efficiency and accuracy of delivery can make or break a business. As demands upon the service increase and change, the resilience and transparent scalability of the operational architecture becomes as important as that of the software running within it. This paper explores how these new challenges parallel modern demands within manufacturing, and how many key Lean manufacturing techniques might be applicable within this emerging field.

1. Introduction

The advent of the Internet created an entirely new mechanism for information, software and business delivery that has revolutionized society. The ubiquity of the web browser and Internet connectivity has become a huge draw to businesses of all sizes as it allows for immediate and unencumbered access to the consumer. Entire multi billion dollar industries, from search and media conglomerates like Yahoo! and Google to commerce platforms like Amazon and EBay to business tools and process companies like Salesforce.com and Netsuite depend entirely upon the Internet to exist. More and more companies, from the IBMs of the world down to sole proprietorships, rely upon systems that are part of, connected to, or built on top of other services entirely contained in this emerging “Internet platform” to conduct their day to day business.

For the software industry, the “Internet platform” introduces a means to replace the traditional bottlenecks of delivering shrinkwrapped media, a model that requires manual installation on customer supplied hardware, with immediately accessible web pages and APIs connected to the Internet from supplier hosted applications. This distribution mechanism allows for incredible speeds to market and rapid adoption. It also lowers costs through eliminating the need for additional equipment on the consumer side. Other cost lowering tactics such as customer multi-tenancy are also possible. As the application upgrades are in the control of the supplier, there is also an opportunity to seamlessly narrow the support band for a product. While the benefits are numerous, this newfound efficiency comes at a cost.

2. Availability Demands

The increasing dependence on this new “platform” world has begun to create a requirement by customers that these services should be no different than electricity or telephone service, a world where consistent availability is traditionally both expected and highly valued. As a response Software as a Service (SaaS) businesses must often compete with each other not only on the relative feature functionality merits of their product, but also on the service availability. They have begun to vie for the most number of ‘9s’ of service availability, with some touting availability numbers near or at the “telco grade” standard of 99.999%. Service availability is typically calculated as some variation on the formula:

$$\text{Availability} = \frac{\text{service time} - \text{downtime}}{\text{service time}} \times 100$$
Each increase in the number of ‘9s’ dramatically reduces the amount of downtime allowed. Most high end hardware vendors, having years of experience with the telecoms industry, typically aim for a Mean Time Between Failure (MTBF) that achieves the telco 5-9s of availability (99.999%) target.

The challenges of high software uptimes are many. Software must run on hardware, and thus must deal with the cumulative MTBF of all the underlying components. In addition, while development and quality control practices within the software industry are relatively nascent compared with that of hardware, such practices are still embryonic on the Internet delivery platform. In order to overcome these challenges, some providers have begun to borrow from the tenets of Lean Manufacturing that were first pioneered by Toyota for automobile manufacturing. The Toyota Production System (TPS) combines management philosophy and practices to form an integrated socio-technical system that, like the SaaS environment not only can affect service quality but also availability. Variances also challenge service integration efforts. Many useful systems should be put in place that not only reduce the possibility of inconsistencies, but also can help make it easy to understand and reproduce environments quickly across the entire software lifecycle. These systems should span the technology stack, as well as the software lifecycle. Operating system installation tools such as Solaris Jumpstart or Linux kickstart, and automating build tools like CruiseControl or Hudson are already commonly used tools for fully automating commonly repeated tasks in an easily annotated, reproducible and versioned way. Asset databases that may already exist within an organization can be used to track hardware components and variances, when paired with monitoring and incident data they can help find potential problem areas or bottlenecks in infrastructure. Investments in better atomic packaging, automated deployment and configuration tools, and intelligent test automation take this even further. Atomic packaging provides versioned, self contained packages that can be independently released and rolled back easily and cleanly. Automated deployment and environment management tools should understand versions and role relationships between packages and environments. It should deploy and configure packages atomically and in an automated fashion, and should have the ability to regularly audit software manifests and configurations to ensure compliance between what is installed and the models stored in the environment management database. When both are combined with test automation, changes can be made, tracked and tested quickly and in a focused manner. Errors and inconsistencies can be located quickly and corrected.

### Table 1: Availability as a unit of downtime allowed per measured period

<table>
<thead>
<tr>
<th>Availability (%)</th>
<th>Downtime Per Week</th>
<th>Downtime Per Month</th>
<th>Downtime Per Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>95%</td>
<td>8.4 hours</td>
<td>36 hours</td>
<td>18.25 days</td>
</tr>
<tr>
<td>99%</td>
<td>1.68 hours</td>
<td>7.2 hours</td>
<td>3.65 days</td>
</tr>
<tr>
<td>99.9%</td>
<td>10.1 mins</td>
<td>43.2 mins</td>
<td>8.76 hours</td>
</tr>
<tr>
<td>99.99%</td>
<td>1.01 mins</td>
<td>4.32 mins</td>
<td>52.6 min</td>
</tr>
<tr>
<td>99.999%</td>
<td>6.05 secs</td>
<td>25.9 sec</td>
<td>5.26 secs</td>
</tr>
</tbody>
</table>

3. Poka Yoke – Mistake Proofing

Most common quality problems are caused by inadvertent mistakes. The concept of Poka Yoke is to put mechanisms in place that either makes mistakes impossible to perform or otherwise make it impossible to not immediately notice them when they occur. A common example in the physical world is that of 3.5” floppy disks, which are designed so that the disk can only be inserted into a drive one way.

Lack of reproducibility is one of the biggest scourges to ensuring software quality. Unfortunately, much of this has been brought upon by many common practices in the industry. These include software often being developed, compiled, tested and deployed manually in environments that were at least in part built by hand. Even in more mature organizations it is not uncommon for development, test and production environments to vary in sometimes unknown or irreproducible ways. Dependencies across environments are often hard to track if tracked at all. Software is sometimes designed to modify itself and/or the environment it is on upon installation. Worst of all, traditional development methodologies like Waterfall encourage releasing large bundles consisting of multiple components all at once. These big “birth-like” events often have so many moving parts that deriving the effects of a change make troubleshooting problems difficult, and without careful and considerable effort often make rollback prohibitive.

There is considerable value in “mistake proofing” the whole software development lifecycle, especially for an SaaS. Unknown or unnecessary variances in an SaaS environment not only can affect service quality but also availability. Variances also challenge service integration efforts. Many useful systems should be put in place that not only reduce the possibility of inconsistencies, but also can help make it easy to understand and reproduce environments quickly across the entire software lifecycle. These systems should span the technology stack, as well as the software lifecycle. Operating system installation tools such as Solaris Jumpstart or Linux kickstart, and automating build tools like CruiseControl or Hudson are already commonly used tools for fully automating commonly repeated tasks in an easily annotated, reproducible and versioned way. Asset databases that may already exist within an organization can be used to track hardware components and variances, when paired with monitoring and incident data they can help find potential problem areas or bottlenecks in infrastructure. Investments in better atomic packaging, automated deployment and configuration tools, and intelligent test automation take this even further. Atomic packaging provides versioned, self contained packages that can be independently released and rolled back easily and cleanly. Automated deployment and environment management tools should understand versions and role relationships between packages and environments. It should deploy and configure packages atomically and in an automated fashion, and should have the ability to regularly audit software manifests and configurations to ensure compliance between what is installed and the models stored in the environment management database. When both are combined with test automation, changes can be made, tracked and tested quickly and in a focused manner. Errors and inconsistencies can be located quickly and corrected.
At Yahoo, such systems not only condensed the duration of testing and integration cycles from often weeks to hours, but reduced the number of cycles required while significantly contributing to adding over one ‘9’ of measured uptime across business units that adopted them.

As such systems are deployed two benefits become immediately apparent. The first is that it becomes far easier to permit a steady stream of product improvements, as there is both confidence in the configurations of development, test and production environments, as well as it becomes easy and straightforward to release and, if required, roll back changes quickly. Secondly, the automation and configuration databases dramatically reduce the amount of staff overhead required in order to manage the infrastructure. A single engineer can build and release to thousands of systems instantly and accurately, as well as later verify that the services are operating correctly and rollback as necessary. Companies such as Google and Yahoo! have been able to obtain ratios of thousands of servers running hundreds of services handling millions of requests per engineer through such mechanisms, allowing their organizations to scale in ways that are otherwise conventionally impossible.

4. Jidoka – “Stop the Line” Automation

While the tools of Poke Yoke go a long way to preventing errors, failures are still inevitable. To preserve high availability, protect customers’ businesses and data, and to bolster customer confidence that their interests are being well looked after, SaaS providers must find and quickly correct any faults. Traditional manual recovery methods are expensive and error-prone, and without the right protections do not necessarily find and resolve the root cause. Anyone who has been told to reboot or restart an application without any explanation or further resolution only for the problem to return later can certainly relate.

Jidoka is best described as “automation with a human touch”. Simple automated restarting of failed processes, or tools that only provide heuristic data alone rarely provide sufficient information to find and eliminate the root causes of errors. Methods such as code hooks and other instrumentation within the software that provide transparency into the overall health of the application, as well as defensive programming techniques that encourage graceful failures and provide effective artifacts for both troubleshooting and future testing, not only help improve the speed of problem resolution but also provide valuable data that can be fed back into the software lifecycle to improve “mistake proofing”.

There are other incredibly valuable data sources that are often not tracked or otherwise carelessly ignored. A professional technical operations team that manages the running of a service can provide incredible depth of knowledge of the operational health of a product. These staff members are the closest thing to having customers on staff, as the quality of the operational health of the product will directly affect these people first. Continued collaboration during product development can provide clues to areas of concern that can be proactively targeted. Incident tracking and trending, as well as site performance numbers, operational load to break and failure mode testing can be compiled into a “health dashboard” that can help spot problem areas and technical debt. Teams that implemented such measures typically added one to two ‘9s’ of availability to their products. They were also able to confidently increase release velocity from months to weeks or days, and were able to target far more effective functionality goals for these releases.

5. Kaizen – Continual Improvement

Tracking failure only provides one useful dimension to the relative quality of a product. If implemented well, the SaaS model also provides unique insights that, when leveraged with the big picture of other metrics across the software lifecycle, can provide a plethora of information to better target resources and improve the customer experience. Having the data is not enough, and many organizations are already overwhelmed by what they have or improperly focus on minutiae. Instead, a systematic approach that encourages continual experimentation, and a holistic view across the entire product and the organization around it is the key to a high performing SaaS.

The philosophy of kaizen is that of continuous, proactive improvement through all aspects of life. It considers both the results as well as the process so that the actions that achieve the effects, whether good or bad, are surfaced. It is also non-judgmental and non-blaming. It is critical that staff or customers should not fear that they will be censured as it breaks trust and often causes people to hide or obfuscate facts.

A huge advantage of the SaaS model is that it allows for unobtrusive observation of how services are being used. A provider can understand what flows and features are more or less popular, and can observe performance and interaction characteristics that can be used for business and technical planning. If the architecture accommodates and the requisite work has been done to gauge software quality, experimental
“bucket tests” can be run that seamlessly redirects a small number of targeted users towards newer or different functionality or configurations and receive immediate feedback. This is of tremendous value over waiting for feedback after a release as it can allow for engineers or business people to change course and reduce waste far more rapidly.

Haphazard data analysis and experimentation can be both dangerous and cause businesses to move in a direction counter to their best interests. The entire organization, from technical engineers to business analysts to Finance, needs to work together collaboratively using a deliberate scientific approach. Key metrics across the software and business lifecycle should be exposed and correlated. Incidents and performance issues often correlate to build health and bug counts. Revenue from certain features may not justify the capital or operational costs associated with them. If installations or upgrades are slow or painful, it might provide certain insight on how quickly the organization can react to new business demands or major shifts in the industry. The term “Internet time”, where radical industry shifts and events happen in highly compressed periods of time, has been battled around for a while and is very real. Scientific experimentation and correlation allows organizations to be more nimble, understand their strengths and weaknesses, and provide measures to technical, business, financial and procedural decisions that can be made readily available to key organizational stakeholders.

There are some basic practices that nearly any technology company can start with. The first is to look for opportunities to obtain fresh perspectives on the state and sanity of key elements of the business. The Toyota Production System actively looks to eliminate anything that does not advance the process. It does not settle for eliminating the waste that everyone recognizes as waste. Many people resign themselves to certain problems, or have become hostage to routine and abandon the practice of problem solving. There are many ways to see if complacency has grown in an organization. Can someone unfamiliar with the business come in and with little direction quickly understand the health of the codebase, infrastructure and current projects? Can they quickly understand the value the business provides to its customers? One test that can be implemented is to have new hires install, configure and use product(s) with little to no assistance beyond what documentation might be available, and afterward capture their impressions, focusing particularly on what things were bad, weird or unexpected. Similar things can be done by having employees trade duties, even for short periods of time. Providing channels for people to share ideas, understand the challenges of the business, and try to find ways to fix them is critical.

Finally, having the means to experiment in nondisruptive and nondestructive ways is key. It is perhaps the most difficult piece to get right as it requires alignment across the organization and an acceptance that not all experiments will succeed.

6. Just In Time

Few SaaS have traffic patterns that are flat and predictable. They will typically experience regular load peaks and troughs, with occasional traffic spikes that seem to come out of nowhere. A SaaS must deal with these traffic shifts transparently in order to ensure acceptable service responsiveness and availability. A business can decide to provision ahead of time for the worst case scenario. Unfortunately, their best guess might be wrong, and the costs of acquiring and maintaining the usually idle resources can be prohibitively expensive. Also, traffic patterns and the demands across the infrastructure will change as software and the customers using it are added or change their behavior, meaning that any specialized configurations tightly tied to key components might only provide transient value and not be flexible enough for their added costs. Means must be put in place to minimize the waste while maximizing the ability of the SaaS to respond quickly to these demands.

The concept of Just In Time is to flexibly deploy resources as they are needed and no sooner. Amazon and Google have both introduced the concept of cloud computing to the mainstream. Clouds are in essence similarly configured servers that work in unison to provide computing resources to the applications running on them quickly and efficiently. This building block approach allows for services to be scaled up or down quickly, much like Lego bricks. While Amazon and Google have built rather sophisticated software to manage their clouds, the underlying concepts of modular architecture in its simplest form can be applied to any SaaS. Software architecture decisions that require lots of specialized systems should be carefully studied as they greatly reduce flexibility and introduce waste.

Just In Time is particularly effective when used in conjunction with Jidoka, Kaizen and Poka Yoke. As seen earlier, Jidoka and Kaizen provide useful mechanisms to understand and respond to changes quickly, and Poka Yoke can instill useful mechanisms that speed the accuracy and delivery of deployment. When used together with a modular infrastructure, signals of traffic imbalances can trigger automated or guided intelligent reconfiguration of the production
environment that shift resources from other parts of the environment to alleviate load and mitigate failures. An added benefit is if traffic patterns follow a daily sine-like curve, batch and offline jobs can be run on dynamically reconfigured production systems in off hours. Other services which have inverse load profiles can also similarly benefit.

As organizations become much more sophisticated, this model can allow for resources to be brought to bear only as they are needed. Services and resources could be requested and leased for defined periods. This can be useful for infrequently used services, and is ideal for development and test where resources and environments might be “dialed up” from a reconfigurable pool where they can be returned afterwards on demand. Decisions can also be made at the last moment of how to optimally manage resources as which ones to run serially or in parallel. At Yahoo, such measures allowed in several cases for a reduction of hardware footprint of 60-80%, while allowing for far more dynamic and complete testing and more timely response to radical traffic spikes, both planned and unplanned.

7. Conclusion

The challenges of the SaaS model will continue to force organizations to look for new ways to produce high quality services more efficiently and more rapidly than ever before. The techniques found in the Toyota Production System and Lean Manufacturing provide a useful guide for both large and small businesses that can be adapted and followed throughout the software lifecycle.

10. References