Extreme Deployment

Context: Large IS organizations that are building and modifying interconnected mission-critical applications.

We've solved the coding problem. Developing applications is still difficult, but we know how to design and write high-quality code as efficiently as possible. Capturing technical requirements is still a challenge, but gathering requirements through use cases and ancillary techniques works pretty well.

Here's where large organizations struggle: application integration and deployment. It's not unusual for an organization to take twice as long to deploy an application as it took to develop it. Developers often sacrifice their personal time and emotional well-being to complete a project quickly, only to see it see it sink below the surface of a molasses-like deployment process.

What are some of the problems that can occur up to and during an application deployment?

1. Dependencies on and between undeployed systems cause significant schedule delays. For example, Application A depends on Application B, which can't deploy until System C deploys. System C won't be ready for at least a month. By the way, it depends on data generated by Application A. This may sound idiotic, (and maybe it is), but it's not unusual.
2. Users and/or customers are denied service during installations, upgrades, and rollbacks
3. Deployments are scheduled late at night or on weekends, when people are tired, distracted, or unavailable. The red flag is that deployments are treated as exceptional (and scary) events
4. Applying changes and upgrades are usually a long and error-prone manual process.
5. Dependent systems and applications are halted or degraded.
6. Throughput and performance capabilities are found to be inadequate, forcing an emergency rollback.
7. A serious functional defect is discovered, forcing an emergency rollback.
8. The organization is unable to quickly back out undesired changes.
9. Data migration is complicated by changes to content, format and storage technology.
10. Deployment works on some platforms but not others.
11. Deployment is a bandwidth hog, especially to remote sites, and virtually shuts everything else down.
12. The users require training in the new system, and training means the training department doing at least a two day training course for everyone.
13. The system being delivered is part of an overall solution involving organizational or procedural changes to a business.
14. The system being delivered is part of an overall solution involving delivery or development of hardware components.

One of the debilitating effects of these problems is that everyone involved begins to dread and delay deployments. Consequently, the backlog just gets longer and longer.

Suggestions:

1. When possible, define an acceptable default behavior for when a needed system has not been deployed or is unavailable.
2. Switch over rapidly. This can be aided by installing changes on a ParallelProduction system, thoroughly testing it, then switching over to it. There is still risk of an outage, but at least the original production system is available while changes are made.
3. Reduce risks so that deployments can be done reliably during normal business hours
4. a. Use an automated deployment system, whenever possible. Ant can be especially useful for this. b. Installing to a ParallelProduction system means that installations can be done carefully without incurring a prolonged outage.
5. See Suggestion #1.
6. Test production level volumes in a ParallelProduction environment. Routing live transactions through a ParallelProduction system (and ensuring they don't interfere with the "real" transactions) is not easy, but it can help to avoid unpleasant surprises.
7. Extreme Programming and other rigorous testing processes should eliminate this problem.
8. Never alter the live system, always switch to an updated Parallel Production system. After the new live system is validated, update the old live system.
9. Where was the missing user story? XP etc. should eliminate this problem.
10. Where were the platform unit tests? XP etc. should eliminate this problem.
11. Whose story was this supposed to be? Surely we can rent some bandwidth?
12. Drip feed training by frequently issuing a few user-friendly release notes. Lessen the need to train by improving SoftwareUsability.
13. Employ a field-trial group who will test migration to the new system
14. Why don't we have the stories from the (actual or potential) hardware users? XP etc. should eliminate this problem.

Challenges:

- Ensuring that the parallel system is a replica of the current live system (except for deployment changes)
- Expense of redundant systems
- Not confusing ParallelProduction systems with backup or fault-tolerant systems.
- Handling transactions that are occurring during the switchover
- Requires a dynamic "switch" in front of the parallel systems
- How many subsystems need to be parallel? In other words, how deep is the parallelism?

Suggestion--Look at how eBay does it, with a phased feature rollout every two weeks. During working hours, without taking down any part of the system, including all of the issues above.

Everything described above requires a lot of hard work. At least it's work that can be done while people are awake and alert.

The goal is to make each deployment no more exciting than a daily build.

This only begins to address the problems of deploying mission-critical applications in a sane fashion. Suggestions and insights are invited.

See CostOfDeployment, EaseOfSoftwareDeployment, HowXpPlansDeployment

---

Last edit September 16, 2005, See github about remodeling.