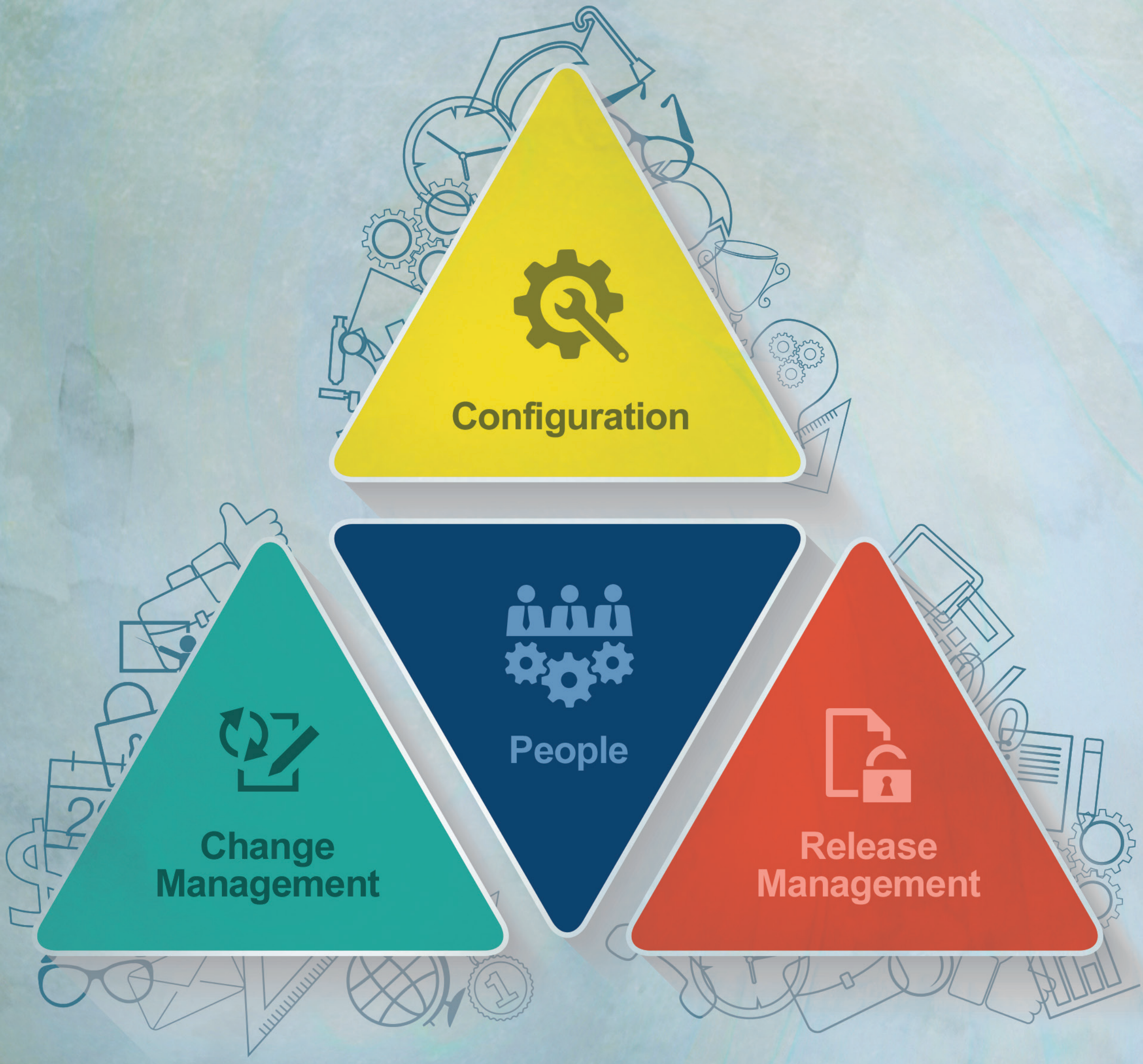
How one telco built the magic triangle to succeed in release management

ITSM Best Practice Lessons



Overview

A mid-sized telecom company lost about 25 percent of its customers over a period of three months due to multiple delayed releases of critical software. Clearly, the existing release management process was not effective, and the company had to troubleshoot and arrive at a concrete process to set things straight.

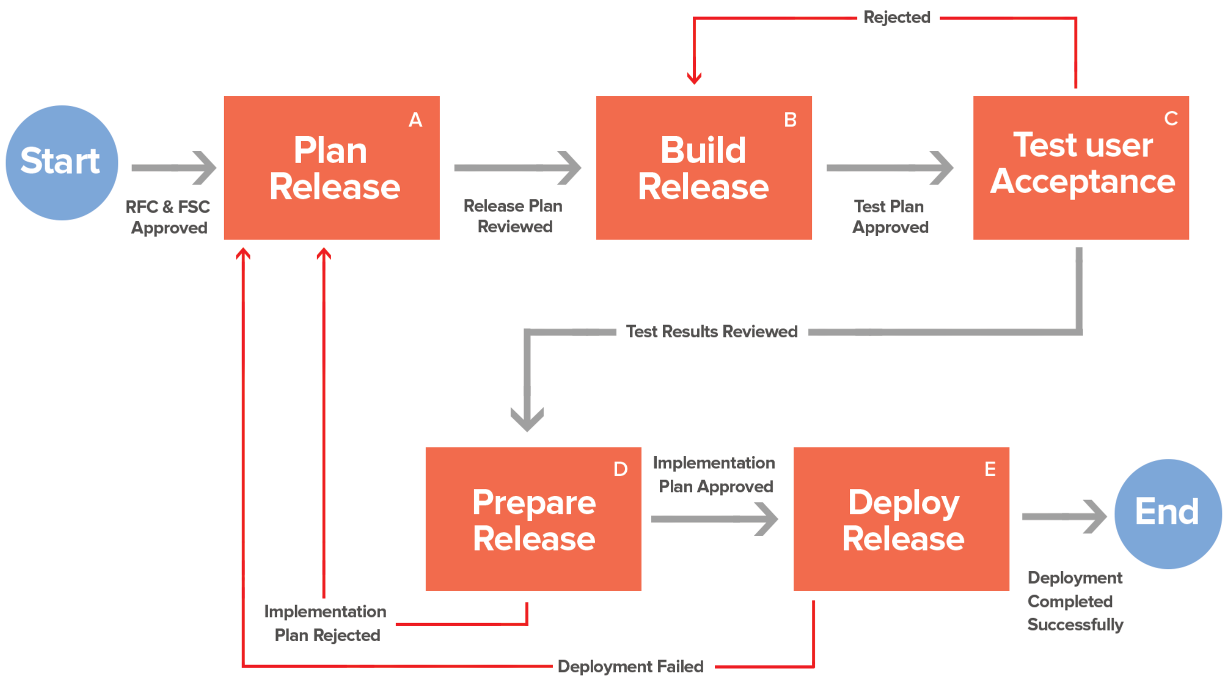
The company formed a new release management team to assess and streamline its existing release management process. In the following three months, the new team was able to put out both the pending releases, and two scheduled releases of re-engineered applications. Below, you'll see how the team streamlined the release management process. The steps the team took reflect best practices that can be employed by any IT organization, including yours.

Assessing the existing release management process

The new team wanted a detailed picture of the current release process. They began with a number of walk-through sessions with key individuals involved in the software process. In these sessions, the team learned that a CRM software had been pending release for two months after completion.

Based on customer`s requirements, the CRM software included claim settlement. It also included service credits and penalties for telecom customers who faced issues in downtime, service degradation, and transfer of new billing plan. The delay in rolling out the CRM software meant that several of the customers couldn't get their service credits or refunds on time, and that there was no tracking system for updated status requests. About 25 percent of retail customers lost patience in the system, and walked away.

In the existing process, the touch points, validation, and verification process were not seamless due to hardware procurement, test environment, and the absence of an agreed upon release cycle.



ManageEngine
ServiceDesk Plus

Consolidated assessment findings

- Release cycles were random, as opposed to having a structured and agreed upon release window and frequency.
- Test environments were out of date, and not available on demand.
- In many cases, regression testing took over three months, and was dropped in many deployments.
- Integrated change control processes, with configuration and release management, were absent.
- The teams' overall morale was down, due to a high number of failed and delayed releases.

Course correction outline

In order to deal with this situation, the new release management team followed a few course correction pointers, as listed below:

- Determining and establishing approved release cycles.
- Simplification of process, and early testing.
- Controlled infrastructure using configuration management system (CMS).
- Streamlining change control process and automation.
- Building the magic triangle (integrated configuration, change, release, and deployment process).
- Taking into account measurements and metrics.
- Training and awareness workshop.

Determining and establishing the release cycle

Once the team got a picture of the current state of the release management process, they decided to focus on defining and establishing an agreed release cycle. To understand the frequency of release in production, the team had to understand the frequency of the non-functional testing. After a discussion with the engineering teams, the team concluded that the project required regression, performance, and integration testing.

The release cycle was developed to achieve the following:

- Create an opportunity to meaningfully discuss any non-functional testing that the software might need.
- Form a timetable to update the expected release of a feature or functionality to the stakeholders.
- Establish a routine with which all teams can align (including marketing and engineering).
- Ensure customers that their orders will be of good quality, and delivered within the agreed timeline.

The release management team started out by experimenting on a weekly cycle, but the plan proved unfeasible. The client's database environment could not be refreshed quickly. The team then tried two-week cycles. There was no immediate objection from the participants, but the two-week cycle failed the first two times. Once they overcame a few environment turnaround bottlenecks and automated some of the tests, the team decided that the two-week cycle was an achievable one.

Finally, the team established a cycle whereby, every two weeks, production-ready code from the engineering team was put into system testing. Two weeks later, they released that code to production.

The release cycle was not about when the customer wanted the release. It was about when the team could deliver it to the market with the desired level of quality. However, when the team engaged the customers, and made them the decision makers, the customers started supporting it!

Simplifying the processes

Lightweight processes are those that do not require lengthy bureaucratic approvals or endless meetings to get agreement. They usually require only the minimum acceptable level of input and output. What they lack in bulk and bureaucracy, they make up for in response to change and popular adoption.

Underpinning this approach was a thorny issue of documentation, however. The team needed to record what was done and how it was done. Otherwise, it would be impossible to review or improve the situation.

Documentation and tooling

The company decided to move towards a documentation standard that would let people (technical and otherwise) read and act upon documents, instead of letting the documents lie on the shelf.

The engineering team chose Confluence, a commercial tool, to collaboratively document their work. They used the software to create minimal but effective documentation of what they agreed to build in every cycle of work. They recorded what they built, how they built it, and what was required to make it work. The new team saw the value in this approach, and rolled it out (both approach and tool) to everyone else involved in the process.

A sequence of tasks was then created to release the software from the engineering teams. The list of tasks included:

- Ensuring delivery, straight from the source control management system.
- Specifying the nomenclature, and defining how each element (executable code, database scripts, etc.) would be run, and on which platforms.
- A dry run was done using a dummy code for each element. The sequence was tested while documenting what the team did and how they did it. This formed the basis of the installation instructions.
- The next step was to take the people who would be deploying the real release through another dry run, using only the documentation created by the team.

- The people involved extended, amended, and improved the team's instructions. The process became more inclusive, and everyone contributed to the documentation. As they were part of its definition, the process became more widely adopted, with better quality.
- After each release, the team reviewed the process, examined the documentation, and identified changes made during the release. They also looked at how the documentation could be improved, and fed the enhancements back into the process.

Establish a controlled infrastructure using CMS

The scope

A release infrastructure ensures that everything is in place to deploy software and enable use. The release management team took time to develop a configuration management system (CMS), and started by building the tree structure and configuration item (CI) hierarchy.

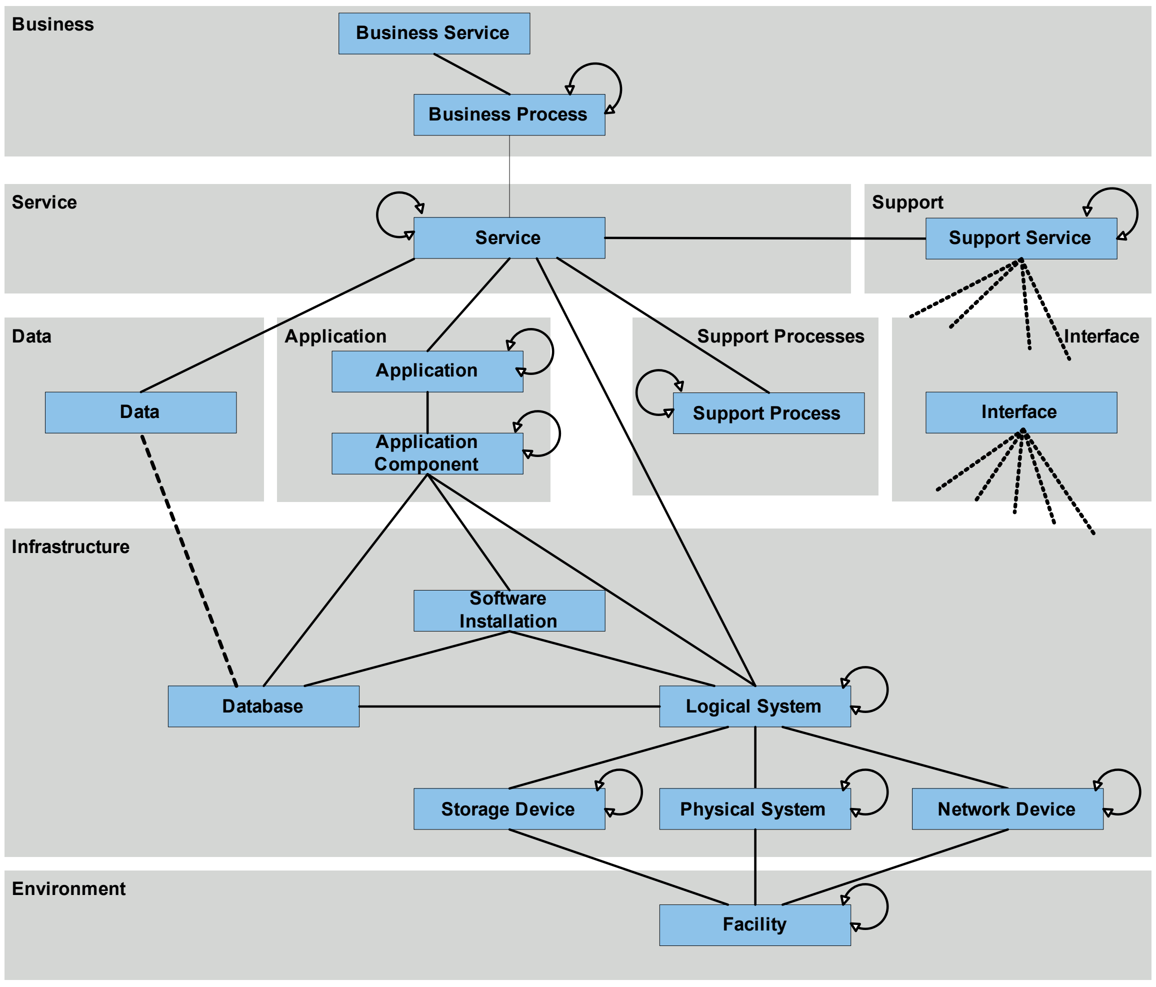
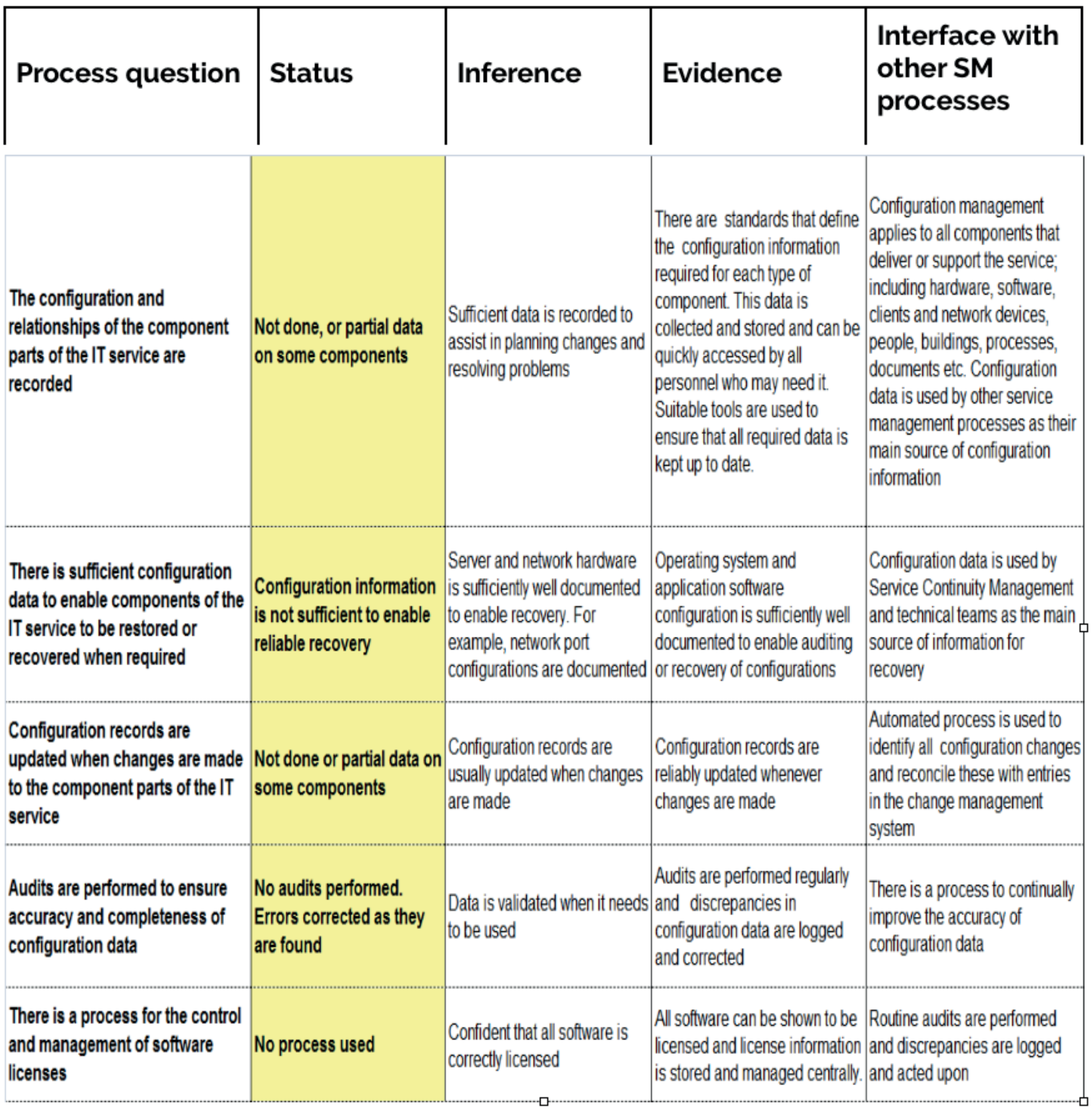


Figure 1: Conceptual CI topology, hierarchy, and relationships.

ManageEngine ServiceDesk Plus

Assessment of the configuration management process



Process question	Status	Inference	Evidence	Interface with other SM processes
The configuration and relationships of the component parts of the IT service are recorded	Not done, or partial data on some components	Sufficient data is recorded to assist in planning changes and resolving problems	There are standards that define the configuration information required for each type of component. This data is collected and stored and can be quickly accessed by all personnel who may need it. Suitable tools are used to ensure that all required data is kept up to date.	Configuration management applies to all components that deliver or support the service; including hardware, software, clients and network devices, people, buildings, processes, documents etc. Configuration data is used by other service management processes as their main source of configuration information
There is sufficient configuration data to enable components of the IT service to be restored or recovered when required	Configuration information is not sufficient to enable reliable recovery	Server and network hardware is sufficiently well documented to enable recovery. For example, network port configurations are documented	Operating system and application software configuration is sufficiently well documented to enable auditing or recovery of configurations	Configuration data is used by Service Continuity Management and technical teams as the main source of information for recovery
Configuration records are updated when changes are made to the component parts of the IT service	Not done or partial data on some components	Configuration records are usually updated when changes are made	Configuration records are reliably updated whenever changes are made	Automated process is used to identify all configuration changes and reconcile these with entries in the change management system
Audits are performed to ensure accuracy and completeness of configuration data	No audits performed. Errors corrected as they are found	Data is validated when it needs to be used	Audits are performed regularly and discrepancies in configuration data are logged and corrected	There is a process to continually improve the accuracy of configuration data
There is a process for the control and management of software licenses	No process used	Confident that all software is correctly licensed	All software can be shown to be licensed and license information is stored and managed centrally.	Routine audits are performed and discrepancies are logged and acted upon

Figure 2: Assessment on the status of configuration items, relationships, and interfaces.

ManageEngine ServiceDesk Plus

Discovery and tools workshop

The team conducted workshops with the infrastructure, application development, and management teams to decide on the configuration, change, and release processes. The agreed conceptual flow, from a tool perspective, is depicted below.

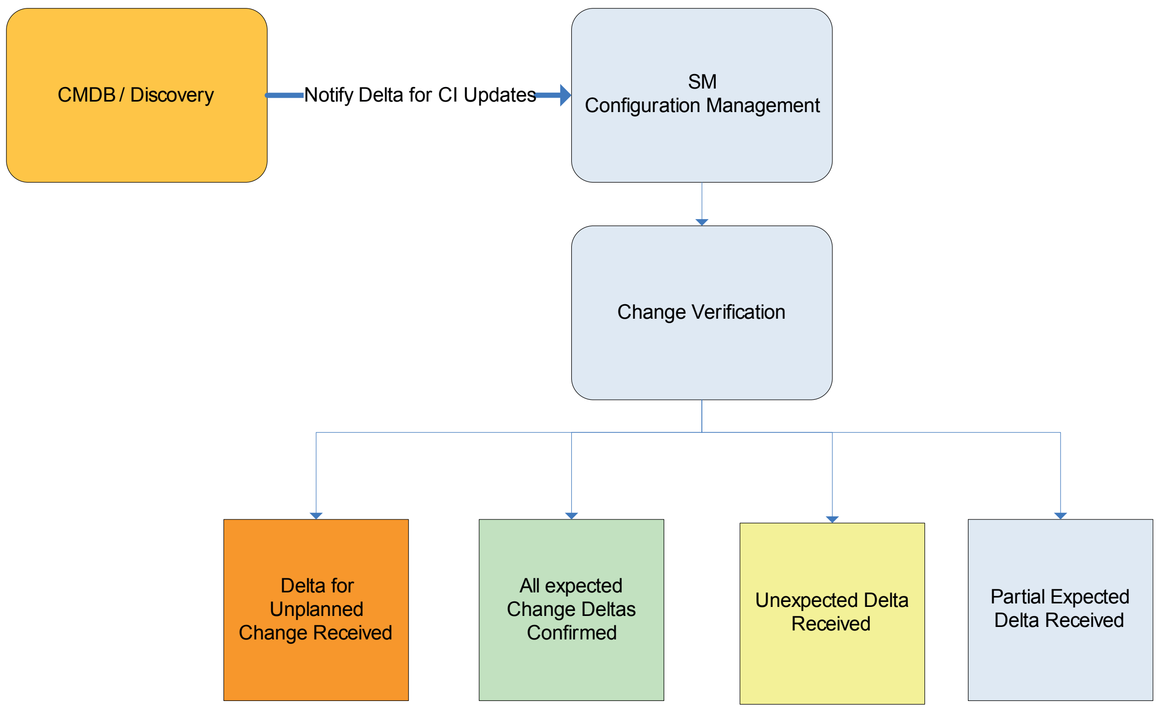

CMDB / Discovery
Notify Delta for CI Updates
SM Configuration Management
Change Verification
Delta for Unplanned Change Received
All expected Change Deltas Confirmed
Unexpected Delta Received
Partial Expected Delta Received

Figure 2: Conceptual flow from discovery to configuration, and change management.

Hardware procurement and matching skills

The release infrastructure covered the hardware, storage, network connections, bandwidth, software licenses, user profiles, and access permissions. Human services and skills were also part of the release infrastructure. For example, if a specialist software needed to be installed and configured, they could not exclude the availability or cost of getting such skills into the infrastructure plan.

It is critical to discover any hidden bottlenecks in procuring the required hardware or missing skills (e.g. configuration of secure networks). The team had to resolve these issues before delivery.

However, that standard was not easy to uphold. The team strove to get the release infrastructure in place as soon as they started on the project. Even after six weeks' lead time, however, they were still waiting on specialized memory and hard drives for the test servers.

Hardware procurement and matching skills

Identification of automation tasks

The team identified the build and test tasks that could be automated, and came up with the criteria and definitions for normal, standard, and emergency changes. Automation enabled them to perform repetitive tasks without tying up valuable human resources. They also qualified a number of standard changes based on risk, repeatability, and time involved in execution.

Changes were bundled to align with the release cycle of two weeks. Change teams worked with the release and deployment teams synchronize with the release calendar, types of release, and early life cycle support.

Manual versus automated package

Prior to their involvement in the project, the engineering teams manually crafted deployable packages. Due to this, each new package was not guaranteed to be the same as the previous one. In fact, it was not guaranteed to even be the software they had been building, much less guaranteed to work! It often took the tech staff days to create a package with the features, in a structure that could be deployed.

Service acceptance and verification

The release management team immediately drew up a structure and service acceptance criteria for the deployable package the engineering team was delivering, and helped them standardize the packaging. This triggered the implementation of automated processes to build the software in that consistent structure for every release point.

As the team had automated the verification process of the acceptance criteria, the execution was guaranteed. For example, code must be unit tested prior to delivery and test deployed to ensure that it could be. As a result, the release management team was able to package, version, test, and deploy the finished code with a single command, in a very short time.

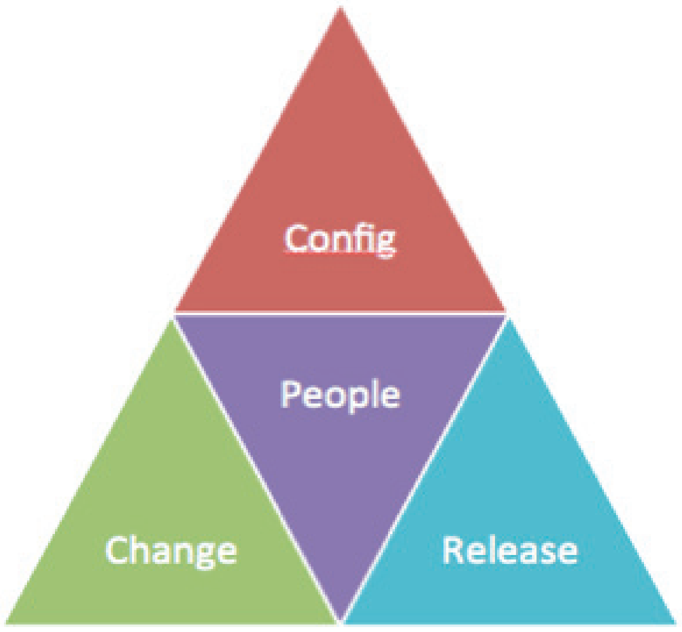
The new norm of release cycles

The newly established release cycle meant that a release had to be tested for regression, performance, and integration in just two weeks, for the team to be able to release it into production. The team was able to overcome different types of testing (integration and performance) by having separate environments for each type. However, the challenge of accommodating two months of regression testing into a two-week window seemed impossible. Here's how they did it.

- First, the team initiated a prioritization exercise. The customer identified the high priority regression tests—the minimum they would accept as proof that the old functionality existed.
- The team then started automating these tests. Subsequent acceptance tests were also automated to ensure that the team could regression test every release in just hours, rather than days.

Building the magic triangle

While the combination of configuration management, change control process, and release and deployment management processes were integrated seamlessly, the entire process required the vertex of people in order to be possible.



Roles and responsibilities

The release management team backed up this importance by establishing that the designated release manager would expect the software to be ready at the time that the teams agreed upon. The team made an arrangement that the program manager (in this case, the end usercustomer) would explain to the teams why the release was important.

The team requested that engineering teams ensure that the software they delivered conformed to a standard (versioned, tested, documented, and packaged). The team also established that they would request this standard package for every release cycle. This made the automated process easier and more consistent, and the allowed for feedback integration.

Release management metrics

The following release management metrics were measured continually to tune the release management process.

- Number of changes pending future system releases (backlog).
- Number of successful changes in a release.
- Number of failed changes in a release (percentage of failed changes).
- Number of outages caused by a release.
- Number of incidents caused by the release.
- Percentage of releases delivered on time for QA testing.
- Percentage of releases delivered on time for production.
- Percentage of releases by priority or type.
- Total release downtime, in hours.

Snapshot of metrics captured, from August 2014 – Feb 2015

Metrics	August 2014	Nov 2014	Feb 2015
Number of changes pending future system releases.	14	6	2
Number of outages caused by a release.	6	1	0
Percentage of releases delivered on time for QA testing.	60 %	90%	100 %
Percentage of releases delivered on time for production.	75 %	85%	95 %
Total release downtime, in hours.	120 Hours	54 Hours	10 Hours

The power lies on investing in people

During the course of the entire transformation, the biggest asset was the people. The release management team took their perspectives, issues, and challenges in a fair and transparent manner, and informed the management of the same. They even listed a few of the early adopters as change agents. They invested heavily on training, awareness, and communication, thereby instituting a reward mechanism for acknowledging positive behavior and knowledge sharing.

Training and awareness workshop

The following workshops were conducted for the configuration, change, and release management teams, without exception. The line managers were also available to validate the effectiveness of the workshops.

- Fundamentals of the release management workshop (one day).
- DevOps foundation workshop (two days).
- Metrics and measurement workshop (one day).
- Release and deployment action workshop, including roles, responsibilities, timeline, and deliverables (one day).

The outcome of the five-day workshop was increased clarity for the involved teams on various touch points, deliverables, and overall business impact. A quick cheat sheet was distributed to summarize the key learning points of the training.

Lessons Learned

- The management of organizational changes was fundamental to getting all the teams to work on a shared vision and goals that would achieve defined business outcomes.
- The decision of the release readiness had to be based on product or application, even more than the urgencyneeds of the end usercustomer.
- Realistic criteria of the release cycle, in consensus with stakeholders.
- Integration of process and tools (including end user needs) aligned with configuration, change, and release and deployment management processes.

Bottom Line

Based on the release management team's experience in working with others on the project, they realized that good release management takes hard work, resolve, and great communication. However, the greatest skill is the ability to review, learn, and adapt improvements with effective collaboration of people in conjunction with the magic triangle—configuration, change, and release and deployment management processes.