The Cookbook principle for Changes and SOPs

Purpose

To make good Changes and SOP's (Standard Operating Procedures) - its similar to being a chef that need to learn how to make a good dish, or a marine that need to perform a certain function really good, as in: repetition, repetition....

In short, the goal is to end up with a solid recipe, to make the same "good meal" every time - That the Cookbook principle for me

First, lets have a look at the environments we (commonly) deal with in an every days Service Operation environment:

System	Purpose	Who can deploy here?	The good and bad stuff
Development	Basic or custom development, this can be on the developers own systems or a development setup	Developer	A voliatile environment with a potentionally high number of changes and no control. Very flexible and agile for high productivity
Test	Testing what has been released via development	Developer	A voliatile environment with a potentionally high number of changes
Stage	Deployment and testing of the release after test in Test	Sysadmin (And/or Developer if DevOps is used)	A known/controlled environment
Production		Sysadmin (And/or Developer if DevOps is used)	A known/controlled environment - few and controlled changes that has be have been performed in Stage.

The environments from above is not the only ones, in many (large) setups we can have several others:

System	Purpose	Who can deploy here?	The good and bad stuff
UAT	User Acceptance Test	Sysadmin (And/or Developer if DevOps is used)	A known/controlled environment - few and controlled changes that has be have been performed in Stage.
integration	Special integrations tests, typically with 3rd party systems outside our control. Import/Export/Sync	Sysadmin (And/or Developer if DevOps is used)	A known/controlled environment - few and controlled changes that has be have been performed in Stage.

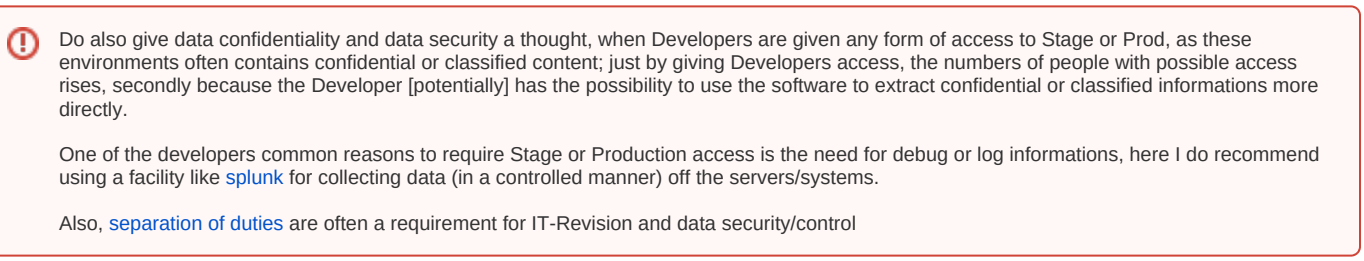
DevOps

DevOps and Continues Delivery concepts extending the traditional concepts of the Developer vs. Sysadmin approach, where a common sence is that Sysadmins deploys, while Developers does not interact with Stage- or Production-environments.

These are not necessarily colliding ways of thinking, but requires some thinking and implementation that enforces special rules and approval-procedures, since (learned over 10 years as Sysadmin):

> The Developers Point Of View on classic virtues and procedures are typically focused in a complete other direction than those of the Sysadmin - The Developer has a focus on getting quickly to the next finish line with features and deployment - sometimes the cost are giving a little slack on test and deployment diciplin.
>
> Giving the Developer access to Deployment in the Stage or Production without enforcing special rules and approval-procedures, do consider the battle lost

During DevOps or Continues Delivery You should implement rules, procedures and technically based access/approval workflows where deployment depends on successfully deployment/test in the previous environments, to prevent unintended or hasty deployment from the Developers side.



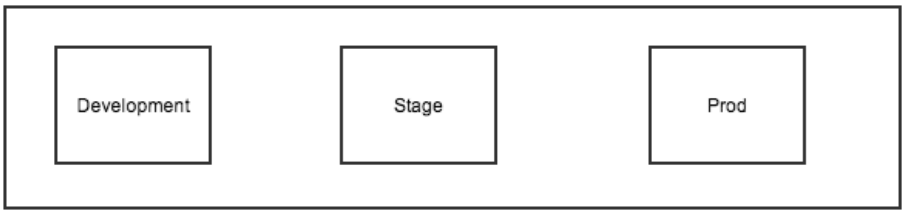
Do also give data confidentiality and data security a thought, when Developers are given any form of access to Stage or Prod, as these environments often contains confidential or classified content; just by giving Developers access, the numbers of people with possible access rises, secondly because the Developer [potentially] has the possibility to use the software to extract confidential or classified informations more directly.

One of the developers common reasons to require Stage or Production access is the need for debug or log informations, here I do recommend using a facility like splunk for collecting data (in a controlled manner) off the servers/systems.

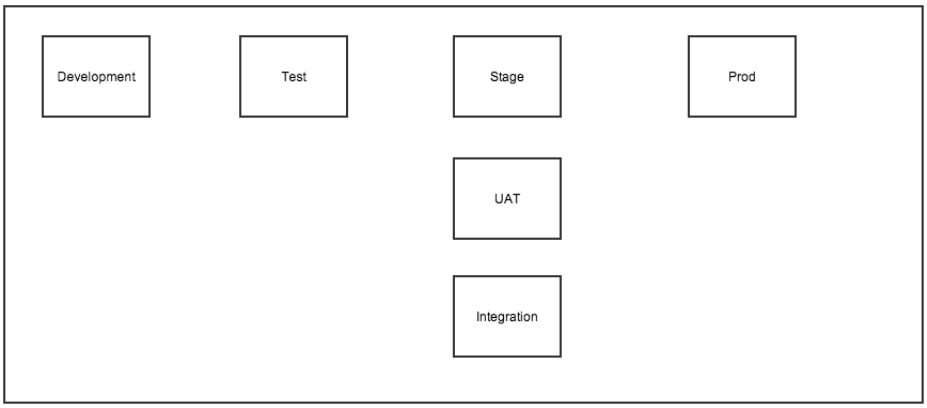
Also, separation of duties are often a requirement for IT-Revision and data security/control

Environments

The minimum should be:

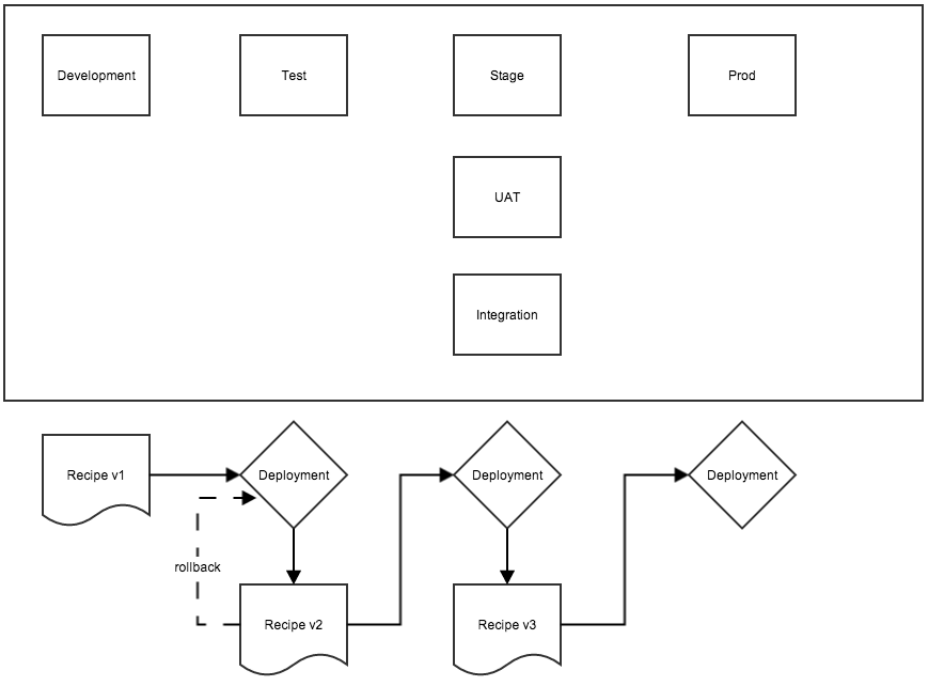


Even better:



Creating the recipe and doing deployment

Seeing beside the number og levels of environments on our platform, we do have the oppertunity to improve our recipe through the environments from Development to Production, and with todays techonologies for virtualizing like VMware or snapshotting filesystems (like ZFS) within a short timeframe - we can repeat and improve the recipe several times, even within each environment:



From recipe to SOP

A recipe in typically going forward from A to B, it does not (always) imply the thoughts of getting back to the base, besides rolling back to a snapshot.

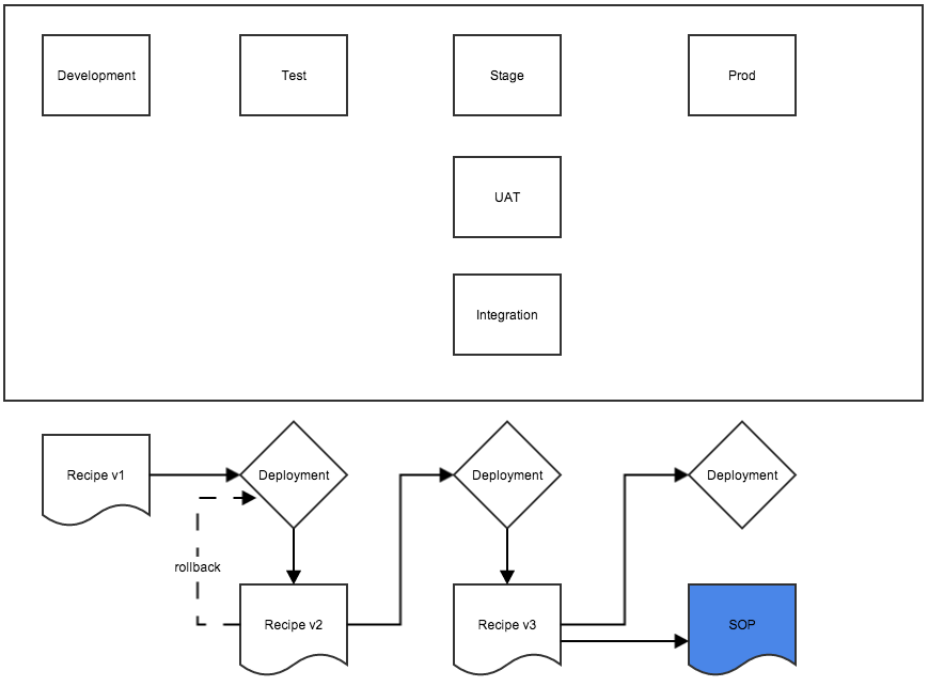
In many situations this type of rollback in not possible, allowed or comply with real-life (prod) - this can be for systems required to have low or zero downtime, or to maintain 100% audit or date/time/id integrity - for example when interacting with 3rd party systems.

At the same time, the recipe is typically very specific regarding objects and entities, with actual host and database names in it.

Hence, we can have a wish for cleaning the recipe and making a SOP - where all specific names and references in the nomenclature are becoming placeholders instead. The SOP then needs to have a section explainng what the placeholders should contain (Pre-requisites).

Se Template for a SOP

So final step is to make the SOP:



Divide and conquer

The Divide and conquer principle is crucial for good Changes and SOPs - as opposite to havin a big Change with many steps and types of implementation (Application, Firewall, Database, OS), You should divide the Change into smaller Changes that are independant and/or sequentiel to be done separately. This also means that a rollback of a Change is not highly critical for the complete scope of the Changes.