BEYOND ITIL:

A MODEL FOR EFFECTIVE

END-TO-END RELEASE MANAGEMENT

(THE SEVEN HABITS OF HIGHLY EFFECTIVE RELEASE MANAGERS)

INTRODUCTION

There is no universal definition of what Release Management is. Different standards and publications define Release Management, or Release Management activities, differently and Release Management is often an adjunct to a standard or process rather than a standard or process itself.

- Prince 2 talks about controlling "release packages" within the Configuration Management section.

- The V-Model (German government standard), which takes a more holistic view of the development process, again only mentions Release Management as a relatively minor activity performed as part of Configuration Management.

- ITIL talks about Release Management within the context of "managing changes to IT services" and states that it should be used for "large or critical hardware roll-outs, major software roll-outs and bundling or batching related sets of changes" implying that it is not used for managing all of the software that goes out of the door.

Release Management has a different meaning depending on where you're looking at it from. If you're looking from a production operations or service management viewpoint then, according to ITIL, Release Management is all about managing change to IT services.

However, if you're viewing Release Management from the development organisation then Release Management is about the definition, co-ordination and delivery of releases (new versions) of the organisation's products[1].

This can be very confusing and is often misunderstood by people, who assume that there is a single Release Management process at work within the IT organisation.

If I am responsible for delivering IT applications and solutions where do I go to find best practice Release Management processes? Prince 2 barely mentions Release Management (and is only focused on those changes delivered by projects), the V-Model includes little mention of it and ITIL only considers Release Management from an IT operations perspective.

At the present time ITIL is very much in vogue and therefore when one thinks of Release Management one almost inevitably thinks of ITIL. However, this is only part of the story, ITIL is really only concerned with the final implementation of the release. Release Management in this respect is predominantly a co-ordination role. There is an analogy of an airport where the Release Manager is in the control tower making sure that each aeroplane (release) arrives on schedule and that the correct ground handling etc is in place to meet it.

If we want to manage releases end-to-end then we need to look further than ITIL; ITIL does not tell us how we should manage a release throughout the development lifecycle.

[1] The term 'product' refers to an IT deliverable, such as an application, not a business product such as a pension or a mobile phone contract. From the definition "manufactured in response to requirements"

Figure 1 illustrates the roles and relationships in a typical organisation that develops its own IT solutions.

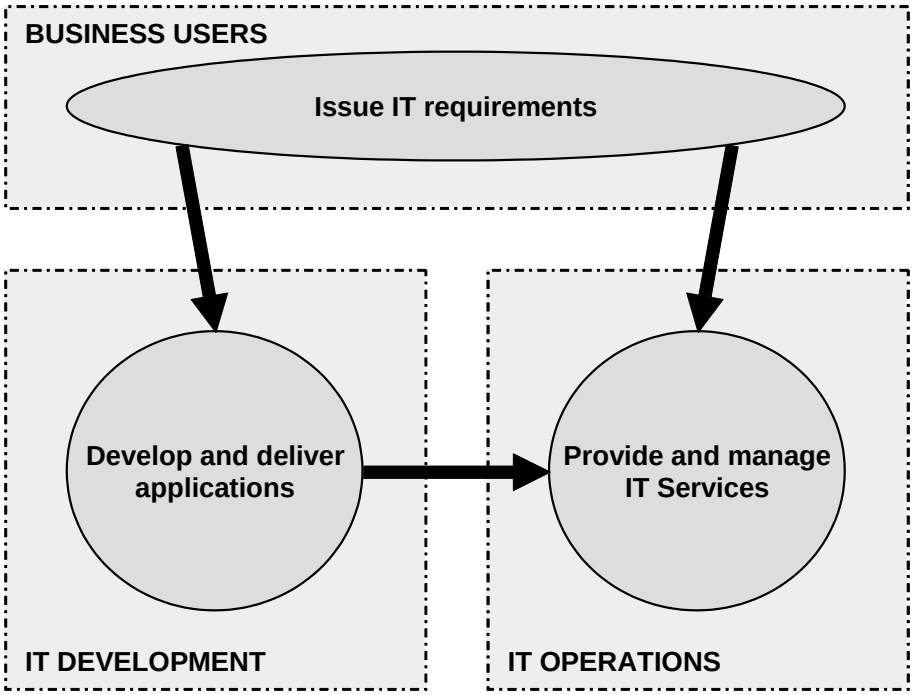


Figure 1 IT within a typical organisation

The scope of this white paper is Release Management within the IT development organisation and covers all products delivered by IT development to the IT operational organisation. This paper does not replace ITIL, we are staunch advocates of ITIL and it's a great framework. Rather, it provides a model that works alongside ITIL's Release Management.

The scope must also include aspects of Change Management because of the close ties that Change and Release Management have and we will see that Release Management has certain requirements of Change Management.

This paper does not attempt to tell you how to measure release risk or how to run release meetings. Instead it describes a proven framework for Release Management and describes an approach to implement it.

This is best practice Release Management that you can actually implement to manage the delivery of IT products.

RELEASE MANAGEMENT IN IT DEVELOPMENT

Whilst ITIL states that Release Management should be used for "large or critical hardware rollouts, major software rollouts and bundling or batching related sets of changes into manageable-sized units" from a development perspective, we want to apply Release Management to *all* of the changes we deliver regardless of whether they deliver a small bug fix or a whole new application.

DEVELOPMENT ARCHITECTURE

Figure 2 illustrates where Release Management fits in the development environment. You'll notice that there is a Release Management function sitting in the operational area. This is ITIL's Release Management function. In the development environment Release Management has been split into the two important functions: Release Planning and Release Control.

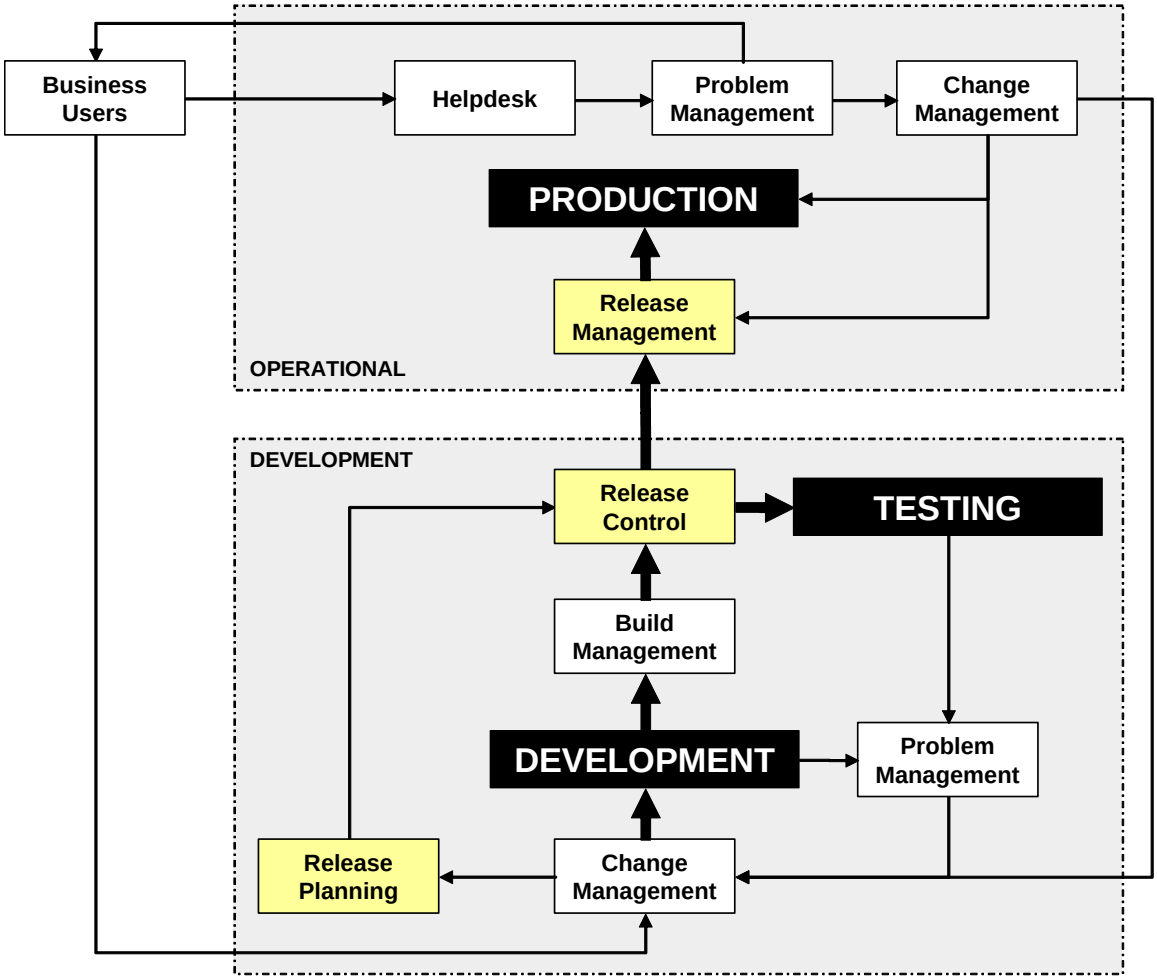


Figure 2. Where does Release Management fit?

Operational Release Management is concerned about release into the production environment but in the development organisation we also want to control releases into formal testing. The term "formal testing" is used to represent testing where the environment is strictly controlled; the contents are documented and all changes go through a process where they are recorded, approved and communicated in advance.

The transition between informal and formal testing will occur somewhere in the testing lifecycle after unit testing and before pre-production testing. Where this transition is set represents a trade-off between lower cost of change and lower risk of failure. Certainly the user acceptance testing phase would normally be considered formal.

Whilst ITIL is busy managing changes (RFCs) to IT services, this is not how changes to IT products are delivered: a business change (what the user wants) is broken down into technical changes (what actions are required to give the user what they want). These technical changes are implemented by new product versions which tend to be bundled into releases.

TYPES OF RELEASE

When people talk about a release they are generally referring to the simultaneous delivery of new versions of a number of IT products on a particular date. To confuse things we then call each of these new versions a release as well.

We need to manage both these types of release so we have to set some terminology so that we can differentiate between the two. Therefore for the rest of this document we will use the terms “Product Release” to refer to the release of a new product version and “Composite Release” to refer to the concurrent release of a set of product releases.

A composite release may exist for two reasons:

- A number of product releases must be released at the same time because they are interdependent, either technically (have an interface) or from a business perspective (both needed to implement a change), or
- Policy dictates that product releases should be grouped where possible to minimise risk or reduce cost.

Figure 3 illustrates the development life cycle of the different types of releases:

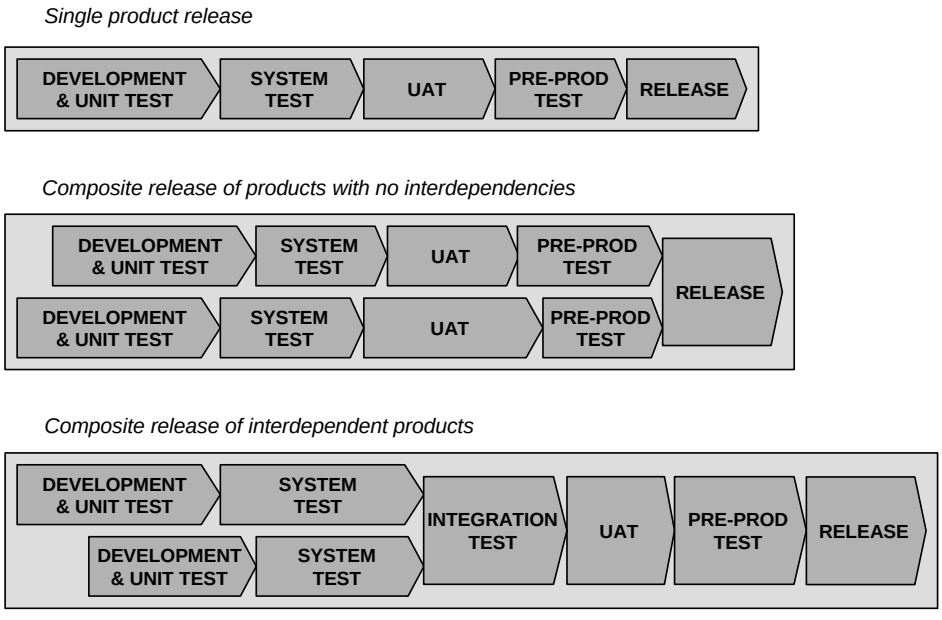


Figure 3 Development life cycle of different types of releases

Release policy and interdependencies between products will determine whether there is one composite release for all products, multiple composite releases (one per related product group such as Pensions, CRM or HR) or on-demand composite releases that are only created when there are interdependencies between product releases.

THE SEVEN HABITS OF HIGHLY EFFECTIVE RELEASE MANAGERS

1. DEFINE RELEASE POLICIES

Release policies are created to enable a common, consistent and communicated approach to Release Management. The policy will dictate the rules that must be followed and reflects the organisation's attitude to risk.

There is a balancing act between frequency of releases and the number of changes in a release. Increasing the frequency of releases increases the risk of production problems. Increasing the number of changes in a release increases the complexity of the release which in turn increases the risk of not being able to deliver it.

Multiple release policies will normally exist. Policies exist in a hierarchy mapped over the organisation, becoming more specific at lower levels.

The top level release policy, applicable across the organisation, might include some overall principals, such as grouping changes together where possible to limit the number of releases required and describing the release policy hierarchy, the process for managing the release policy and the process for dispute resolution.

Lower level policies applied to groups of products, or to individual products, will contain more specific statements whilst remaining in accordance with, and adhering to, the higher level policy. This policy hierarchy is illustrated below in figure 4.

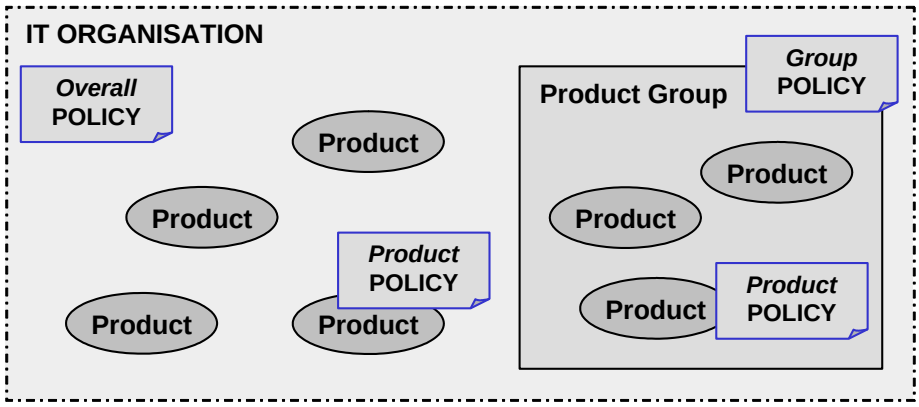


Figure 4 Multiple policies operating at different levels

2. ASSIGN ROLES AND RESPONSIBILITIES

Release Management does not look after itself. It is not simply a case of setting a release policy and then sitting back and watching as all the workers deliver releases on time and to budget without adverse impact.

Release Management is very much a hands-on activity. Whilst policy gives direction when making decisions, there are often difficult choices that need to be made taking into account a complex set of factors. There is often a great deal of negotiation required by the Release Management process. It is not possible to plan for every outcome within the process and to document what must be done in each case. Instead there needs to be an experienced Release Manager at the helm capable of

facilitating the negotiation between the various stakeholders whilst adhering to policy. There are no hard and fast rules, intelligence is definitely required.

The following roles are required for Release Management. With the exception of the overall Release Manager these roles are generally not full time activities but there still needs to be one person assigned the responsibility. A single person could of course be assigned many roles.

Role	Responsibilities
Release Manager	Managing Release Policy Managing the Release Control team Resolving disputes Availability and accuracy of the Release Schedule
Product Release Manager (for each product)	Determining the product's Release Policy (if required) Defining new Product Releases Approving product release requests
Release Owner (for each release)	Managing the content of the release Processing Release Gates Maintaining status information on the Release Schedule Ensuring conformance with relevant Release Policy Raising release request

3. PLAN RELEASES

There are two distinct activities within IT development Release Management: planning and control.

Release planning manages the content of releases. An owner is assigned to each release and part of the owner's responsibility is to manage the list of changes that are included within the release or, in the case of a composite release, the product releases that are included.

The earlier release planning starts the better. When a change request is received from the business it should appear very quickly, at least provisionally, on the release schedule. Regular release dates may be set a year or more in advance whilst releases for particular long-running projects may be set even further in the future. A central release schedule should be easily accessible to everyone involved in the development and delivery process and should list the releases by date and show the status and composition of each one (what changes are included and the status of the changes).

Release planning is closely related to change management: part of the change assessment activity is to negotiate a release that a given change can be included within. The release owner will need to organise meetings with the development and testing teams to determine the feasibility of implementing the change within the time frame and available resources of a particular release.

4. CONTROL RELEASES

Release control is about reconciling the plan with reality. It's about making sure that releases (and builds) are delivered according to plan with appropriate documentation, are properly authorised and contain the changes that they were planned to contain.

Release control will also make sure that builds and releases are securely stored, preferably in the organisation's Definitive Software Library (DSL).

Release control would normally be operated by a single team for the whole organisation. There is no specific knowledge about any of the releases required, it is simply a case of following the prescribed procedures when release requests are received

5. DEFINE RELEASE GATES

Release gates are pre-defined checkpoints in the lifecycle of a release and are used to verify satisfactory progress of the release at key points. The earlier we can spot problems with a release then the more opportunity we have to mitigate the problems or exclude problematic changes that could otherwise threaten the delivery of the whole release.

It's obviously undesirable to pull a change out of a release but if you have to do it then you want to do it as early on as you can to minimise the disruption it will cause to the release. Complications associated with removing changes from a release include backing out documentation and code changes, amending test plans and retesting to ensure the changes have been successfully backed out.

What release gates exists and where they are positioned very much depends on organisation-specific details such as the development and testing lifecycle and the number of product interdependencies. A first gate would usually be placed before development starts. At this point all changes in the release would need to be approved otherwise they will be removed from the release. Other release gates may relate to phases within the development lifecycle. For example: to require high-level design to have been completed by a certain time.

6. SEPARATE BUSINESS CHANGE FROM TECHNICAL CHANGE

Business changes and technical changes are separate entities. Often an organisation will try to manage technical activity under the auspices of the request logged by the business users but this is not the best solution as it does not allow for effective Release Management.

A business change documents what the users require of IT to meet a particular business need. This will normally be expressed at a high level, for example "Enable dual currency handling", and the change document is used as a vehicle to record and manage the interaction between the business and IT and, once agreed, it becomes the contract between the two parties.

A business change is not, however, an appropriate vehicle for recording the detail or progress of the technical changes that will deliver that business change. We need a way of tracking the individual technical changes otherwise how can we ever assess the status?

Figure 5 shows how technical changes are raised during the impact analysis phase of the business change. The technical changes' impact assessments are aggregated to

form the impact assessment at the business change level. Business change approval is cascaded to the technical changes which will then be implemented. Once all the technical changes have been implemented then the business change can be closed as complete.

One business change will be implemented by one or more technical changes but one technical change will only ever apply to a single business change.

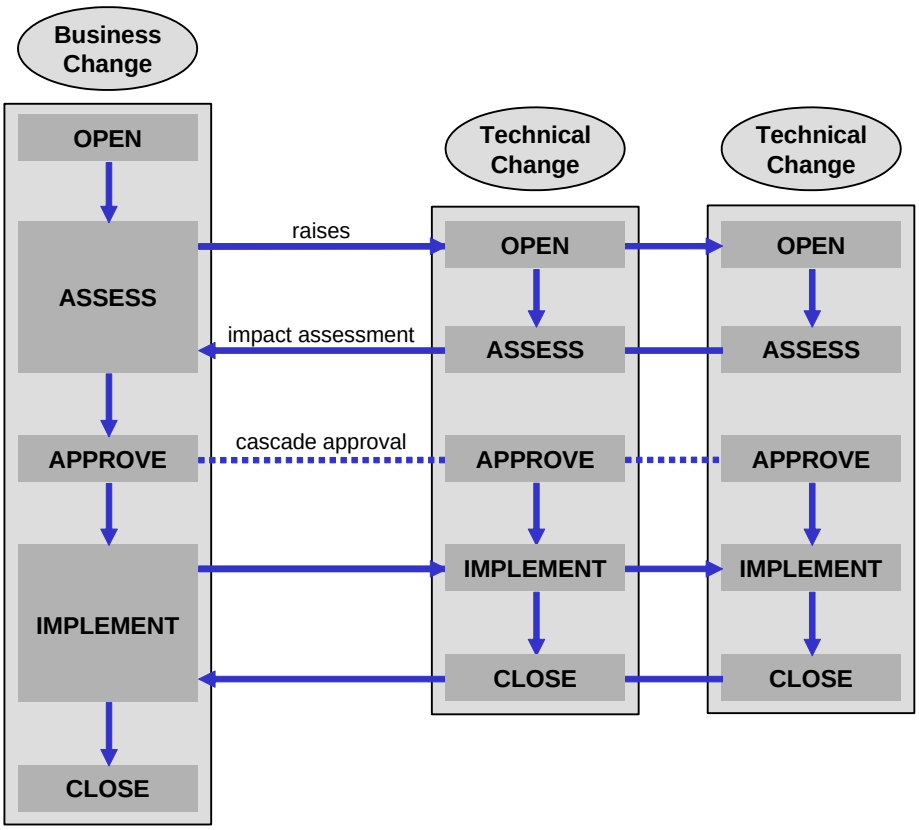


Figure 5 Lifecycle of Business and Technical Changes

For effective Release Management we also need to restrict a technical change to a single release of a single product. In other words a technical change is implemented by a product release and each product release implements one or more technical changes. This enables us to very quickly see what the status of a release is: look at the status of the associated technical changes.

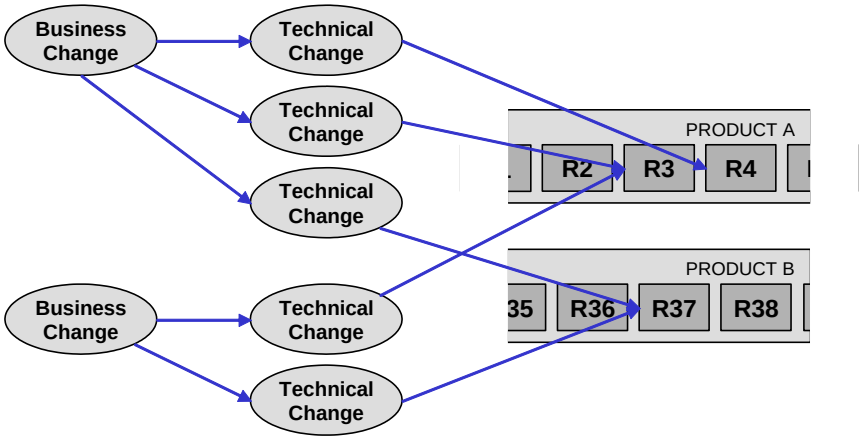


Figure 6 Relationship between business and technical changes and product releases

Figure 6 illustrates the relationship between business changes, technical changes and product releases. It is clear that we can easily find the status of a business change (what is the state of each technical change) and we can see exactly what a given product release delivers.

7. MANAGE BUILDS

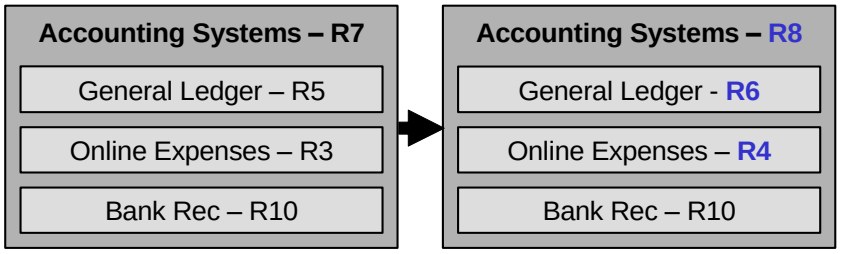
A build is a candidate for a release. If the build passes testing then it will be released otherwise it will be failed and a new (improved) build will be requested. This continues until a build passes all of the required tests or only contains errors that we are happy to live with in production.

Release Management should control the builds that are passed to formal testing. The reason for this is that should the build succeed then this build is what will be released and should be deployed, unchanged, in production. We need to make sure that we capture the build and secure it, ensuring that what is signed-off in the test environment is what will be deployed in production. This last point is important so I'll repeat it: *ensuring that what is signed-off in the test environment is what will be deployed in production*.

Composite releases (with interdependent products) are more complicated to manage. When a build of a composite release is created we need to record the builds of the product releases within it.

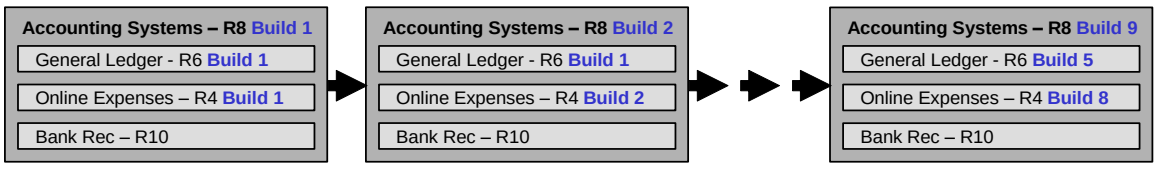
In the following example we are planning a new release of the "Accounting Systems" group of products. This group of products have a number of interfaces and must be tested and released together.

In this new release, R8, we need to make changes to the "General Ledger" and "Online Expenses" products and therefore we will deliver new releases of them. The "Bank Rec" product is not changing and we will continue to use release 10.



During the development of the release, builds will be released for formal testing.

Formal testing is performed at the composite level so we will be testing the combination of the products. Each time we receive a new build of one or more products the new combination represents a new build of the composite product:



Logically, we only deploy builds of the composite release into the testing environments, we never deploy individual product builds. The testers, or environment owners, should raise requests asking for “Accounting Systems – R8 Build 9” to be deployed in their environment, not for individual product builds. This way we know that only pre-defined and approved combinations of the product builds are being tested. It’s no use testing one combination of builds and then releasing another.

Physically we deploy individual products. A test environment will have a build of the composite release recorded against it. The physical machines within the test environment will have the individual product builds recorded against them. In the above example updating a test environment from “Accounting Systems – R8 Build 1” to “R8 Build 2” only actually requires computers hosting the “Online Expenses” system to be physically upgraded from “R4 Build 1” to “R4 Build 2”. Of course if we have an environment where we want to test the implementation of the release (such as pre-production) then each time we upgrade the environment we would probably refresh it to be as production first.

Build release notes are generated for each product build but there is also an aggregate build release note at the composite level, describing the difference between the previous build and this one.

Builds, like releases, should be planned. The testing team(s) should know in advance what changes and fixes are going to be delivered in the build and when it will be delivered.

DEVELOPING A RELEASE MANAGEMENT PROCESS

The objective is to develop a Release Management process that is negotiated between:

- **The implicit requirements, or constraints, of the current process**. Once the as-is position is mapped out it often becomes apparent that certain actions which at first sight, or when considering best practice, would be dismissed, hide site specific requirements or constraints that should not be ignored in the process design.

- **The stated goals of the process review**. The goal may simply be "to align with best practice". Alternatively there may be some particular requirements or constraints to consider, perhaps in accordance with future plans.

- **Best practice**.

Out of this process we would expect the following deliverables:

- **Process flows showing the as-is picture**; how Release Management is currently performed and how it interoperates with other disciplines such as Change Management.

- **Process flows showing the to-be picture**.

- **A roadmap** of how Release Management can be implemented in self-contained steps, each with benefit and cost.

There are a number of ways to document process workflow but the simplest and most effective is to use swimlane diagrams, which show how a process flows between the different participants, supported by lower level "use cases" or flowcharts where necessary to expand into operational procedures.

A common mistake when developing new processes is to make the person who will be responsible for operating the process also responsible for defining the process. It seems like an obvious decision to make, however the mindset required to architect and engineer a process is different from the mindset required to manage and operate a process. The ability to architect the process is a rarer skill than managing a documented process so if you do have someone capable of architecting your process also managing your process then you're probably wasting their talents!

THE PROCESS – STEP 1

Determine the as-is position

The first step is to understand the current processes that are being operated. Even if there are no documented processes and even if these processes are inefficient or ineffective there must be some existing processes otherwise changes would not be getting implemented in production or sent to customers.

The as-is position should not be dismissed, people may have good reason for the process they currently follow or at least they may *think* that they have good reason, even if they're wrong because they don't fully understand the subject.

Ignoring the existing process users' views and opinions is a sure-fire way to build in resistance to any new processes that are implemented.

THE PROCESS – STEP 2

Understand the bigger picture

Release Management does not operate in isolation. Release Management has relationships with many other disciplines including: Change Management, Problem Management, Testing, Deployment, Project Management, Configuration & Build Management.

The most important link is the one with Change Management. The key to effective Release Management is to have effective Change Management surrounding what changes go into the release and changes to the release itself. If the Change Management process is weak then it will result in impact to and additional work for Release Management

Release Management processes will need to be designed such that they interoperate with these peripheral processes, undoubtedly there will be some changes required to the peripheral processes to enable this interoperability and to ensure an efficient and effective end-to-end process.

THE PROCESS – STEP 3

Develop the to-be vision

The to-be vision is where you ultimately want to get to. It may be that the vision is not immediately achievable. It may never be achievable. However, it is important to determine that vision because even if you never achieve it at least you can ensure that you're heading in the right direction. At times you may have to deviate for practical reasons.

THE PROCESS – STEP 4

Develop roadmap to achieve vision

Implementing the vision would normally be broken down into bite-sized chunks, each being complete and providing a level of benefit.

Normally you would want to start with the area that will give the greatest benefit (remove the greatest pain) or achieve the greatest benefit possible in the shortest time. It's determined by how you want to implement the vision. The benefit delivered by each subsequent step will probably decrease. This is one of the reasons why visions do not get fully implemented: the final steps aren't worth it.

Usually the most sensible first step to take is to install a gatekeeper between the development and production environments. At the very least start recording what is passing through the gate, but better still put a control process in place, even if rudimentary.

Final Thoughts

It is impossible to provide an exact recipe for Release Management as so much depends on the environment and existing processes that it will operate within; it is not a case of one-size-fits-all. Yet here we have presented a core model, a model that can form the basis of a sound Release Management process regardless of the environment. The work now is in implementing the model within a particular context.

We need to look beyond ITIL for Release Management. Release Management will continue to confuse people until the Release Management of IT products is seen as an independent function that interoperates with ITIL, rather than being a part of ITIL.

About Propel Systems

Propel Systems develops software and process solutions to enable organisations to work more effectively. Propel Systems' flagship product, Cimera, enables bespoke management solutions to be built in a fraction of the time of conventional systems. Cimera comes pre-configured to support Release Management and can be used to manage releases, builds, release notes, release requests, changes, problems, projects etc.

About the Author

Gwyn Carwardine is one of the founders of Propel Systems. He is passionate about making computer and human systems work more effectively. He has 20 years experience of delivering bespoke pragmatic solutions to companies, specialising in the areas of Configuration, Change and Release Management.

propelsystems
bespoke IT solutions ... off the shelf

197 Cooden Sea Road
Bexhill on Sea
East Sussex
TN39 4TR

t: +44 (0)8456 447 554
e: info@propelsystems.com