CONNECTING THE CONCEPTS:

FROM CONFIGURATION ITEM TO RELEASE POLICY

Version:	1.0
Date:	March 2007
Author:	Pierre Bernard
	Manager, Product Development
	Pink Elephant

Table of Contents

1 INTRODUCTION

Your Release Management Process Project team is struggling with the definition of the Release Policy. Through discussions, two views of the Release Policy have come up:

- One Release Policy for each IT service
- One generic Release Policy

Which view of Release Policy will be more effective with your Release Management Process design? Is one better than the other? What does ITIL® say? These are all very valid questions that many organizations ask themselves.

A good philosophy is to start with the end in mind. To answer the first question, you will have one overall release document with a section for each service. This way you will not have to repeat everything for each service. Whatever is common to all services should be placed in a common area in the policy document.

Actually, no option is better than the other – both are valid and the actual answer lies somewhere in the middle. As for guidance from ITIL, best practice says that you need a release policy; but, ITIL does not provide much information as to its format, its content or how to go about creating one.

The intent of this document is to provide insight into the various concepts required for an effective and efficient release policy. We will travel through various processes, revisiting some key concepts and linking them together in a more comprehensive and cohesive manner. Both IT and not-IT examples will illustrate some key concepts and how the concepts are linked together.

To start, in order for your release policy to be successful, you need to agree on a few definitions. Then, you need to stitch these definitions together in a cohesive way. Think of it as building a cube. One axis is represented by Configuration Management. The second axis is represented by Change Management, while the third axis is represented by Release Management.

Let us first go back to some concepts from the Configuration Management process.

2 CONFIGURATION MANAGEMENT CONCEPTS

This section does not cover the Configuration Management concepts or process in detail. Please refer to the core ITIL literature for a more extensive review of this process. This section only covers the practical application of some Configuration Management concepts as they pertain to the release policy.

In order to understand the Release Policy, we have to go back to the source – the first axis – the Configuration Item (CI).

ITIL describes a CI as:

"*…any Component that needs to be managed in order to deliver an IT Service. Information about each CI is recorded in a Configuration Record within the [Configuration Management Database] CMDB and is maintained throughout its Lifecycle by Configuration Management. CIs are under the control of Change Management. CIs typically include hardware, software, buildings, people, and formal documentation such as Process documentation and SLAs.*"

Classification is used to ensure consistent management and reporting. CIs, Incidents, Problems, Changes, etc. are usually classified. Simply put, classification is, as per ITIL, "*the act of assigning a Category to something.*"

This leads us to the concept of the CI type, which is defined as:

"*…a category that is used to classify CIs. The CI type identifies the required Attributes and Relationships for a Configuration Record. The common CI types include but are not limited to hardware, document, user, etc.*"

Based on this, we can see that there is a logical hierarchy of CIs to assist in better managing the services offered by IT. It is important to note that IT must adopt a service lifecycle approach and philosophy based on business and IT alignment as opposed to simply providing infrastructure components such as applications to the business.

Hierarchy Of Configuration Items

In order to support the service lifecycle, it is important to properly structure your CI hierarchy. Too many organizations still equate applications and systems with service. The truth is that applications and systems may be components of a service. There are four layers to consider, of which two are logical and two are physical.

The first layer is the service layer, which describes all relevant attributes for each service. This is a logical CI. It is best practice to align the IT services with the business services.

For example, many use e-mail as a service, whereas email is more of a system than a service. A more comprehensive service would be Corporate Communication, which would include e-mail, remote connectivity (e-mail on mobile devices), internet, intranet and extranet access and others. The customers of the service can then subscribe to the whole or part of the service.

The second layer is the system layer, which will describe all relevant attributes for each service. Again, this is a logical layer. In the Corporate Communication example above, the systems would be e-mail, remote connectivity (email on mobile devices), internet, intranet and extranet access.

The third layer is the component layer, which describes all relevant attributes for each component. Here each system will be broken down into three basic component types of hardware, software and documentation.

The fourth and final layer is the attribute layer. An attribute is defined as a piece of information about a CI. Examples are name, location, version number and cost. Attributes of CIs are recorded in the CMDB.

Attribute Types

Wouldn't it be nice to have all the information about each CI at your fingertips? Before you answer yes, think about it. First, you must know who will use the information and why. The following section is not about management information, but about the need to take a top-down approach to management information. Too often management information is dictated by the tools and what they can collect or by historical elements such as "we've always done it this way." The top-down approach to management information goes like this:

1. What business decisions do we have to make this year to support our business objectives?
2. What strategic information do we need to support our business decisions above?
3. What tactical information do we need to support our strategic information above?
4. What operational information do we need to support our tactical information above?

In general, a metric is a scale of measurement defined in terms of a standard, i.e.: in terms of well-defined unit. The quantification of an event through the process of measurement relies on the existence of an explicit or implicit metric, which is the standard to which measurements are referenced.

Metrics are a system of parameters or ways of quantitative assessment of a process that is to be measured, along with the processes to carry out such measurement. Metrics define what is to be measured, and are usually specialized by the subject area, in which case they are valid only within a certain domain and cannot be directly benchmarked or

interpreted outside of it. Generic metrics, however, can be aggregated across subject areas or business units of an enterprise.
Metrics are used in several business models including CMMI. They are used in Knowledge Management (KM). These measurements or metrics can be used to track trends, productivity, resources and much more. Typically, the metrics tracked are key performance indicators.

The following figure illustrates the relationships between corporate objectives down to measurements.

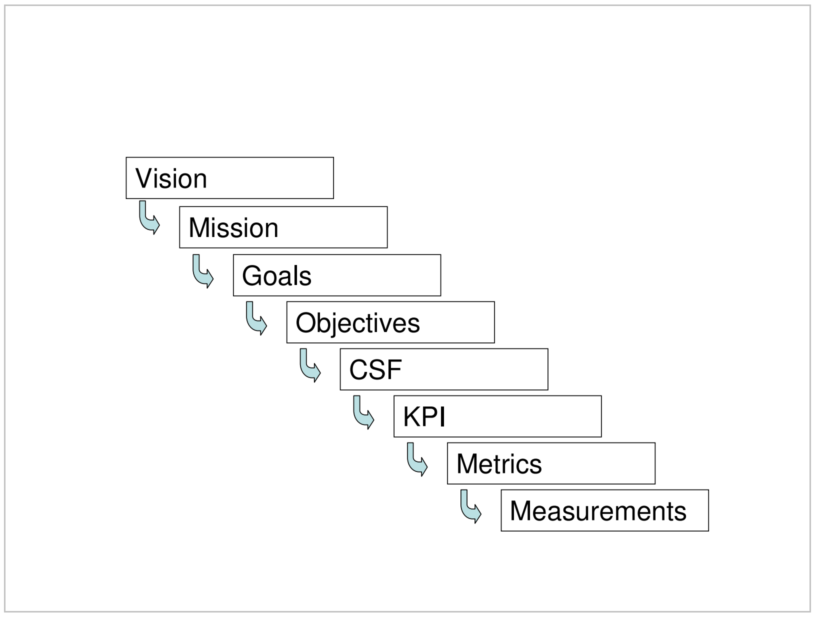

Figure 1 – From Vision to Measurements

Once you have answered the above four questions, then you can determine the attributes for each CI. There are six major attribute categories:

Generic attributes include but are not limited to unique identifier, name, version, owner, make, model and serial number current, previous and future statuses.

Class attributes include but are not limited to service, system, hardware, software and documentation.

Financial attributes include but are not limited to costs for acquisition, depreciation, ongoing maintenance, parts and support.

Record trail attributes are used to identify who created, modified, updated, deleted or archived the CI record.

Relationship attributes are used to describe the relationships between this CI and others.

Historical attributes are links to events related to the CI such as Incidents, Problems, Changes and Releases.

Be careful not to track too many attributes, as you will need to keep them up-to-date. Always ask yourself the following questions: Why do I need to track this? Who will use this information and why? Please remember that the more information in your database, the larger it will be, and the longer it will take to maintain, back-up and search it. Yes, tools are sophisticated; but, if you are not going to track the information, why track it in the first place? The following diagram illustrates a possible hierarchy of CIs down to the various attribute types.

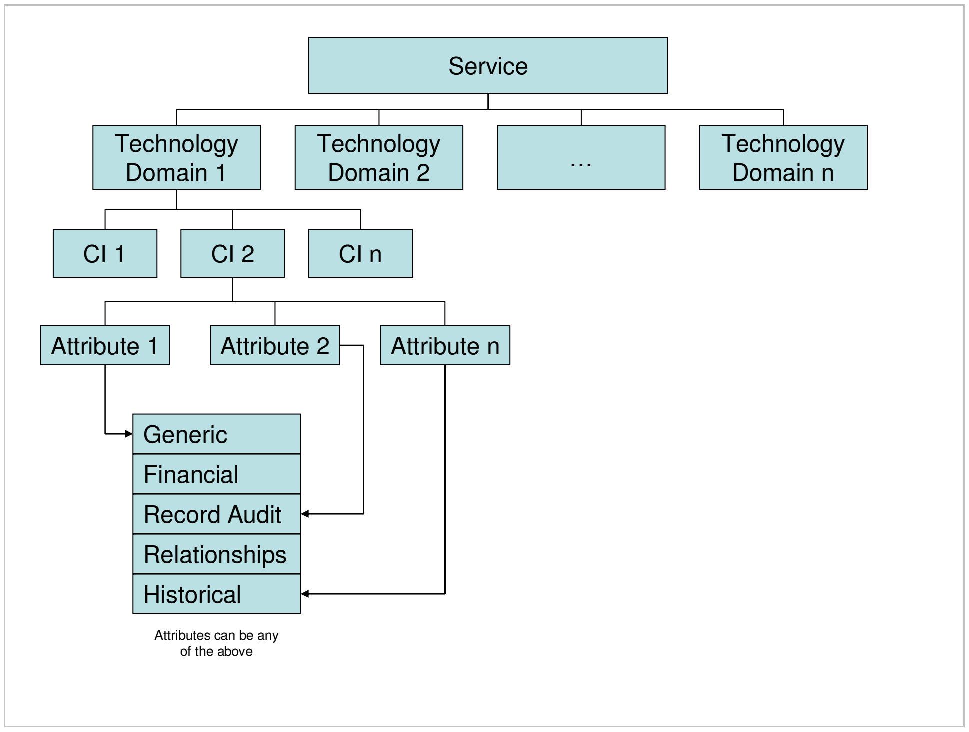

Figure 2 – Component hierarchy

Do you still want all the information about each CI? This information needs to be kept up-to-date. Sure, there are self-discovery tools that can point out and even update the information without user intervention, but you also need to determine the reason for the discrepancy in the first place. Don't rely exclusively on tools to do the work. There needs to be a process in place. The tool must support the process, and not the other way around. Additionally, not all CIs can be verified with a self-discovery tool – service and documentation come to mind immediately.

As you can see, the structure of your CMDB will greatly impact your processes. The more granular the CMDB, the more granular your Configuration, Change and Release Management processes will need to be. How so? By defining more levels in your CMDB and by defining mundane items as CIs instead of attributes, you will have to define and classify your change types accordingly. Make and keep things simple. It is important for your organization to properly list, define and communicate the attributes that it needs.

Summary

Why have we covered this Configuration Management concept? Because we need to link the various concepts together in order to understand the big picture. What are the components to use on our first axis? Use the CMDB hierarchy of: service, technology domain and component. Since all are CIs in their own right, then each will have their distinctive attributes. Our first axis is represented by the following diagram

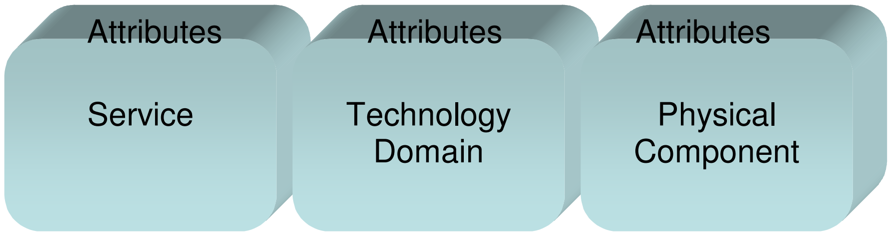

Figure 3 – Attributes in relation to the component hierarchy

The next link in our chain and is simply called a “change”.

3 CHANGE MANAGEMENT CONCEPTS

This section does not cover the Change Management concepts or process in detail. Please refer to the core ITIL literature for a more extensive review of this process. This section covers the practical application of some Change Management concepts as they pertain to the release policy.

ITIL defines a Change as:

"*…the addition, modification or removal of anything that could have an effect on IT Services. The Scope should include all CIs, processes, documentation, etc.*"

What does this mean to our CIs? It means that anytime a CI is added, modified or removed, the Change Management ***process*** is involved. The lifecycle of a Change must be documented in a Change Record, which must then be linked to a CI record through the historical attribute for Changes.

A Change Record is a record containing the details of a Change. Each Change Record documents the lifecycle of a single Change, and is created for every Request for Change (RFC) that is received (even those that are subsequently rejected). Change Records should reference the CIs that affected by the Change. Change Records are often stored in a CMDB.

Of course, Changes are also categorized. The categories of changes are standard, minor, significant and major. Let us revisit these definitions in order to understand how they are impacted by the CI and how they affect the CI.

> **Standard**: A standard change is a Change to the infrastructure that follows an established path. It is relatively common, and is the accepted solution to a specific requirement or set of requirements. The tasks are well-known and proven, authority is effectively given in advance, the train of events can often be initiated by the Service Desk, and budgetary approval will typically be preordained or within the control of the Change requester.
>
> **Minor**: A change that has a minor impact only, and few 'build' or additional 'runtime' resources required.
>
> **Significant**: A change that has a significant impact, and/or significant 'build' or 'runtime' resources required.
>
> **Major**: A change that has major impact, and/or a very large amount of 'build' or 'runtime' resources required, or impact likely upon other parts of the organization.

Urgent: Occasions will occur when Urgent Changes are essential; however, they should be kept to a minimum.

Let us discuss Standard Changes a little bit more. A standard (and pre-approved) change still falls under the control of Change Management, and the Change must be documented. There may not be a need to go to the Change Advisory Board (CAB) with the Change Request, but the fact that the change is pre-approved means that a committee has evaluated the Change. Because it is low risk, well understood, well executed and well documented, staff are empowered to execute it. In some cases, the technical staff may only need to seek approval from their immediate line manager.

It is important to mention that different organizations will apply the above definitions differently. Although the services will be very similar from organization to organization, the impact of a change to a particular type of component may have a minor impact on the service (and the business) in one organization and a significant impact in another organization. Here, we have to quote the infamous consulting answer of "it depends."

For an organization with 10,000 offices, adding a server for a new office is probably, in the grand scheme of things, a relatively minor thing to do. On the other end of the spectrum, for an organization with only three servers, adding a fourth one is probably a major undertaking.

It is therefore very important for your organization to define the Change categories and to provide examples of what will fall under each category. You will not be able to come up with an exhaustive list, but the more comprehensive the list, the better for your organization. You will be able to add to the list by comparing the new change to others in the list when the situation arises. So far, we have a classification for CIs and a categorization for Changes. Here is how they relate. For this, we have to look at the change model. ITIL defines a change model as:

"…a repeatable way of dealing with a particular Category of Change. A Change Model defines specific pre-defined steps that will be followed for a change of this Category. Change Models may be very simple, with no requirement for approval (e.g. Password Reset) or may be very complex with many steps that require approval (e.g. major software release)."

All Changes can potentially affect one or more services. That said, a Change to a document may not necessarily have the same impact to a service as a hardware change. Furthermore, the people involved with developing, testing and implementing the Change to the document may not be the same people involved with the hardware change. This is where the instruction of the Change Model really makes things a lot easier.

The service and the technology domains represent "logical" CIs, while the hardware, software, and documentation CIs are tangible or physical. We have the list of CIs on one axis, and we have the Change Category on the other axis. So, you will describe what Changes are considered to standard, minor, significant and major for each CI category. Here are a few examples:

CI Type Affected	**Change Type**	**Description**
Documentation:	Minor	Correcting the grammar and syntax to an Operational Level Agreement (OLA)
Documentation:	Significant	Modifying the hours of support section of an OLA for one support group
Documentation	Major	Annual review and negotiation of the Service Level Agreement (SLA)
Hardware	Minor	Installation of a printer for financial controller
Hardware	Significant	Installation of an additional router
Hardware	Major	Installation of a new backup storage device for the servers
Software:	Minor	Installation of a graphics software on a desktop
Software:	Significant	Installation of an update to operating system for a group of servers
Software	Major	Installation of a new version of an application

Table 1 – Defining Change Types

The Change Model will document the "who, what, where, why and how" for each type of Change. A diagram can more easily illustrate this concept.

			Standard	**Minor**	**Significant**	**Major**
SERVICE	**Technology Domain 1**	**Hardware CIs**	Policy on what constitutes a <…> change to this platform Who will be assessing <…> changes to this platform Lead times for building, testing and implementing a <…> change to this platform Policy on updating the appropriate records/documentation Who is authorized and empowered to execute this modification *where <...> can be any combination of the matrix*			
		Software CIs				
		Documentation CIs				
	Technology Domain 2	**Hardware CIs**				
		Software CIs				
		Documentation CIs				
	Technology Domain 3	**Hardware CIs**				
		Software CIs				
		Documentation CIs				
	Etc.	**…**				
		…				
		…				

Table 2 – Mapping Change Types to Component Hierarchy

With this information in hand for most changes, we can determine who is involved in the following:

- Assessing the Change
- Building the Change
- Testing the Change
- Implementing the Change

From this, we can start building the procedures for each type of Change, the amount of detail, and what needs to be tracked / documented for each Change.

Be forewarned! The following activity will require a significant amount of work. Here is how you can develop your Change Models. Some activities can be done in parallel.

- List all the Changes that are done on a regular basis

- For each Change, identify the following:

    - Roles (not the actual person) involved in:
        - Assessing the Change
        - Approving the Change
        - Building the Change
        - Testing the Change
        - Implementing the Change
        - Documenting the Change
        - Communicating the Change
        - Doing the training

    - Process considerations:
        - Do you need to order something? If so, what is the procurement process (including timeline)?
        - Typical times when this Change can be performed
        - Who submits the Change request?
        - Who assesses the Change?
        - Who approves the Change?
        - What happens if the Change fails?
        - What happens if the Change needs to be backed out?
        - Can the Change be backed out? (Have you ever tried to remove the cream from your coffee?)
        - How much time is required to build, test, implement, document, communicate and train?

- Steps, procedures, and/or work instructions involved in:
    - Building the Change
    - Testing the Change
    - Implementing the Change
    - Documenting the Change
    - Communicating the Change

- Technology considerations:
    - Service affected
    - Technology domain affected
    - CI affected
    - Attributes affected

Looking at the above, we may realize that we have similar Changes, but that we will end up with different Change Models. Let us look at a vehicle analogy to illustrate. The steps for replacing a transmission on a car, or a farm tractor or a locomotive may be similar, but the details are not the same. The people involved will need different specialized knowledge, tools and parts. The transmission parts for the car cannot be used for the transmission on the locomotive.

Let us now look at an IT example. The mainframe, the server, and the desktop teams may all have a Change Model for adding a storage device in their respective technology domains. Although in concept the steps are similar, the real and potential impacts to the business are not the same. The Change Model will be different, and that is OK.

You must develop your own Change Models. A Change Model is a document outlining the Change to be performed, who does it, when is it being done, where, what is involved in terms of planning, preparation, implementation, and clean-up. I always try to use a non-IT analogy. Let us say you need a transmission. A simplistic Change Model for this could be:

	Car	Farm Tractor	Locomotive
Make			
Model			
Year			
Number of people required			
Lead time to order parts			
Removal of transmission			
Who			
What			
How			
Building new transmission			
Who			
What			
How			
Installing new transmission			
Who			
What			
How			
Testing new transmission			
Who			
What			
How			

Table 3 – Example of a non-IT Change Model

In the IT world, the above example could be translated into adding a storage device on a desktop, server or mainframe. As you can see, the knowledge and skills will be different based on what is to be changed.

Summary

What are the components to use on our second axis? Use the Change types of standard, minor, significant and major. Our second axis is represented by the following diagram

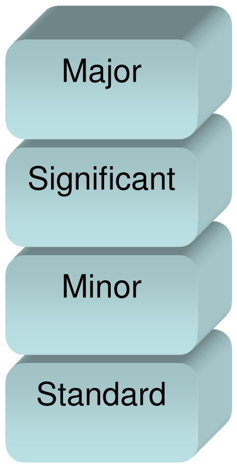

Figure 4 – Change types

Combining our first and second axis, we have:

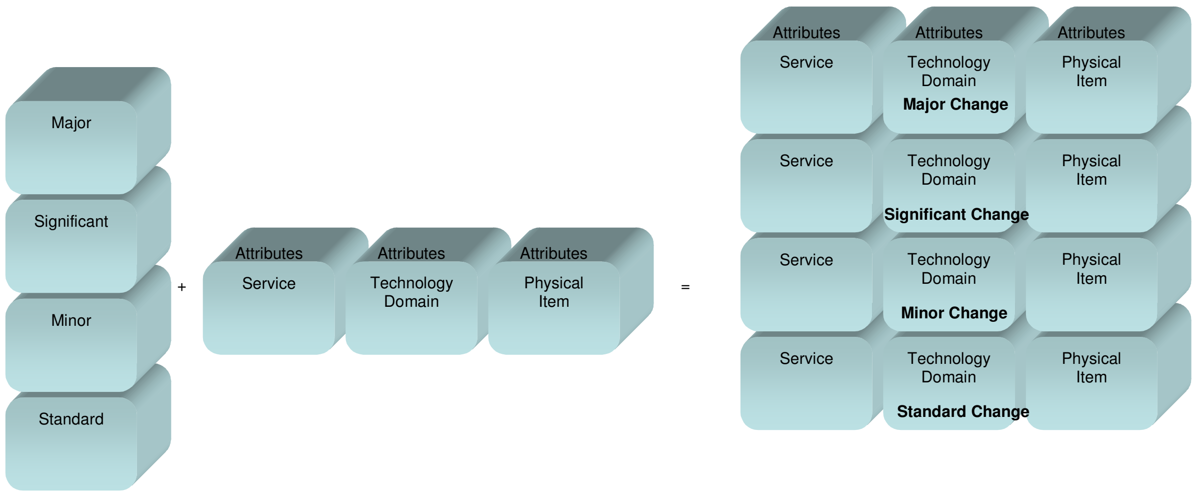

Figure 5 – Mapping Change Types to Component Hierarchy

At this point, we have a better understanding of the impact to the business because we have linked the Change to the affected service through the CI hierarchy. How do we go about linking this to a Release containing multiple Changes? Let's start with understanding the concept of Release Management first.

4 RELEASE MANAGEMENT CONCEPTS

This section does not cover the Release Management concepts or process in details. Please refer to the core ITIL literature for a more extensive review of this process. This section covers the practical application of some Release Management concepts as they pertain to the Release Policy.

ITIL defines a release as:

"…*a collection of hardware, software, documentation, Processes or other Components required to implement one or more approved Changes to IT Services. The contents of each Release are managed, tested, and deployed as a single entity.*"

In order to understand the Release definition better, we need to look at ITIL's definition of the Release Unit:

"…the Components of an IT Service that are normally Released together. A Release Unit typically includes sufficient components to perform a useful Function. For example, one Release Unit could be a Desktop PC, including Hardware, Software, Licenses, Documentation etc.; a different Release Unit may be the complete Payroll Application, including IT Operations Procedures and user training."

Are all Releases created equal? Of course, they are not. Just as we have a hierarchy of CIs and different Change categories, we have different categories of Release known as Release Types. ITIL defines a Release Type as:

"…*a Category that is used to classify Releases."*

There are three basic categories of Release Types: Full, Delta, or Package.

ITIL defines a Release Policy as a document covering:

"…*Release numbering, frequency and the level in the IT infrastructure that will be controlled by definable Releases. The organization should decide the most appropriate approach, depending on the size and nature of the systems, the number and frequency of Releases required, and any special needs of the Users - for example, if a phased rollout is required over an extended period of time. All Releases should have a unique identifier that can be used by Configuration Management.*

A Release Policy may say, for example, that only strict 'emergency fixes' will be issued in between formally planned Releases of enhancements and non-urgent corrections."

Additionally, each Release Type can be qualified as minor, major or emergency. As an example, updating the virus definition table on a regular basis (e.g. weekly) would be considered a Minor Release, while upgrading to the latest version of the anti-virus software for all desktops would be considered a Major Release. An Emergency Release would be to download a specific virus definition now to counteract a possible new virus threat or attack that's making the news currently. Of course, your organization has to define each Release Type and qualifier to suit your needs. For example, you can define that you will or won't have a major-emergency release.

How does this relate back to the Configuration and Change Management concepts? The answer is in the way you define each layer. As mentioned earlier, the more detailed and granular your Configuration hierarchy, the more complex your Change Type and the more complex your Release Types and Units. In the above example, the Release Units were the virus definition table, the anti-virus software and a specific virus definition. Both the table and the definition would be a Delta Release unit, while the new software version would be a Full Release unit.

Trying to connect the Configuration hierarchy, the Change categories and the Release types, we can get many combinations; however, we need to remain realistic and practical. Not all combinations need to be denied in your organizations.

The possible combinations are:

			Configuration Hierarchy			
Change Category	Release Type	Release Qualifier	Service	Technology Domains	Components	Attributes
Standard	Delta	Minor				
		Major				
		Emergency				
	Full	Minor				
		Major				
		Emergency				
	Package	Minor				
		Major				
		Emergency				
Minor	Delta	Minor				
		Major				
		Emergency				
	Full	Minor				
		Major				
		Emergency				
	Package	Minor				
		Major				
		Emergency				
Significant	Delta	Minor				
		Major				
		Emergency				
	Full	Minor				
		Major				
		Emergency				
	Package	Minor				
		Major				
		Emergency				
Major	Delta	Minor				
		Major				
		Emergency				
	Full	Minor				
		Major				
		Emergency				
	Package	Minor				
		Major				
		Emergency				

Table 4 – Mapping Change Categories to Release Types

The above table tells us that we have the potential for 144 different combinations of CIs, Change Categories and Release Types. Does defining all of them make sense? Realistically, can you have a Minor-Delta Release that is a Major-Change to an attribute? Of course the above table will take on a much more complex look when you expand the technology domains for each service, then expand the components for each technology domain. At first the task may appear daunting, but if you have defined your CMDB properly, this is already done. Once the above mapping is done for each service, it becomes a lot easier to populate the Release Policy.

The Release Policy is a document outlining the scope (Delta, Full, Package), the scale (Minor, Major, or Emergency), and the amount of Changes that can fit realistically into the timeframe. When completed, the document will now incorporate a calendar outlining not only the planned release dates, but also the deadlines for submitting a Change Request for each Release as well as all milestones, such as the completion dates for development, procurement, testing, documentation, training, etc. The policy also outlines the periods when a Release to a particular service can or can't be done. For example, many retail organizations have a freeze period from November 1st to January 1st. Government Revenue departments have a freeze period during tax filing season.

It is not the intent of this article to cover governance or process design; however, it is important to remember that too many well-publicized failures are caused by situations where a last-minute change was added to a Release. Or, a failure was caused by a decision not to assess the impact because of the "we know what we are doing" syndrome. It is important to properly design your processes; define and agree to policies; and document the processes, procedures and work instructions. It is also primordial to educate and train the staff on the policies, processes, procedures and work instructions.

The release policy has to consider the following elements:

- How busy are the people doing the development, testing, implementation, documentation, training and documentation? Are they on vacation?
- How much development, testing, implementation, documentation, training and documentation are associated with each change making up the Release?
- What is the influence of a Change upon another Change within the same Release?
- What are the business priorities?
- Do we need to purchase anything? If so, what is the procurement process/cycle?
- Do we require assistance from 3rd party vendors?
- What else in ongoing in the business?

You will have to define for each service:

- What is a Delta, Full and Package Release?
  - How many Changes per type can fit into each Release?
- What is a Minor, Major and Emergency Release?
- Who builds the Release?
- Who tests the Release?
- Who implements the Release?
- Is documentation required?
- Is training required?
- What is the level of the communication plan for the Release?
- What is the frequency of each Release type?
- What are the deadlines and milestones?
- Etc.

It is important to define deadlines for submitting Changes that will be incorporated into a scheduled Release. The reasons include but are not limited to:

- What is the impact on the developers?
- What are the real and potential impacts of the new Change on the ones already part of the Release?
- Do we still have time to procure the appropriate material?
- Are the timelines still valid and achievable?
- How will this impact the amount of and the ability to test the Release?
- How will this impact the training?
- How will this impact the communication?
- How will this impact the deployment?

As you can see, adding what looks like a simple change can have unforeseen implications to the ones already approved. Too often, Releases fail because someone decided to add one more Change to the Release without proper impact assessment. It is recognized that business requirements are often driving these requests, but it is okay to say no and it is okay to implement the Change separately (maybe as part of an Emergency Release).

You can't be everything to everyone. The following table provides some examples of mapping the services to the Release Types and their qualifiers. In this example, the organization decided to not define certain combinations as they did not make sense to them. They decided against having Minor-Full or Major-Delta Releases, for example.

				Delta	**Package**	**Full**	
		These are examples ONLY		Last Sunday of every calendar month	Second Sunday of every quarter	Once a year as per schedule	
SERVICE	**System 1**	**Minor**	Hardware				
			Software	☐	☐	☐	
			Documentation				
		Major	Hardware				
			Software	☐	☐	☐	
			Documentation				
		Emergency	Hardware				
			Software	☐	☐	☐	as defined in the policy
			Documentation				
	System 2	**Minor**	Hardware				
			Software	☐	☐	☐	
			Documentation				
		Major	Hardware				
			Software	☐	☐	☐	
			Documentation				
		Emergency	Hardware				
			Software	☐	☐	☐	as defined in the policy
			Documentation				
	System 3	**Minor**	Hardware				
			Software	☐	☐	☐	
			Documentation				
		Major	Hardware				
			Software	☐	☐	☐	
			Documentation				
		Emergency	Hardware				
			Software	☐	☐	☐	as defined in the policy
			Documentation				
	Etc.	**…**					
		…	**…**	**…**	**…**	**…**	
		…					

Table 5 – High level release policy

Standard Changes & The Release Policy

Are Standard Changes subject to the Release Policy? Of course they are. The Release Policy should simply mention that by their very nature, Standard Changes could be done whenever required. The procedures to handle the various Standard Changes need to be documented. Staff executing the Standard Changes must be trained and appropriate records updated on a timely basis as per procedures and policies.

All aspects of the Standard Changes should be randomly verified and audited to ensure compliance to established procedures. Key performance indicators regarding the various aspects of staff involved with the execution of Standard Changes should be incorporated in job descriptions and performance appraisal.

Additionally, the Change Management process should review the standard changes to ensure that they still belong in that category, just as the Minor Changes should be reviewed, and to determine if the Minor Changes should be re-classified as Standard Changes instead.

Summary

Connecting the concepts of Configuration, Change and Release Management, we come up with a virtual tri-dimensional construct. One axis is represented by the component hierarchy, the second axis by the Change Classification and the third axis by the Release Types. Each little cube represents one possible combination of component, Change Category and Release Type. It is possible to increase the number of layers for each axis. For example, an organization may decide to have four layers in their component hierarchy instead of three. You may decide to account for the Minor, Major and Emergency Release in addition to the Delta, Full and Package definitions. The important point is to make a decision and to start somewhere. A suggestion is to start small and increase the complexity later. Before you "keep it simple", you need to "make it simple" first.

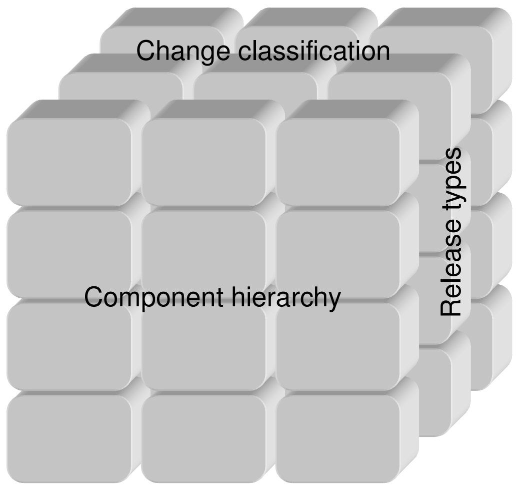

Figure 6 – The Tri-Dimensional Construct

5 SOME FINAL THOUGHTS

Everything is inter-connected. You cannot develop one process while ignoring the others. You cannot build your component hierarchy without understanding your services, just as you can't build your Release Types without understanding your Change Models.

The following is a small list of what needs to be done because of the tight integration between the services, the components making up those services, the Change Classification and the Release Types.

- Define your services[1]
- Define your component hierarchy
- Map your services to your component hierarchy
- Define your change classification
- Map your change classification to your component hierarchy
- Define your Change Models
- Define your Release Units
- Map your Release Units to your component hierarchy
- Define your Release Types
- Map your Release Types to your Change Models
- Map your Release Types to the Component Hierarchy
- Define the Release Policy to incorporate your Change Models

From here onwards, creating your Release Policy will be a whole lot easier.

[1] For assistance on this, please read the book *Defining IT Success through the Service Catalog* available through Van Haren publishing (www.vanharen.net)