Contents

1. Introduction ....................................................................................................3
2. Definition of Configuration Management ...........................................4
3. Organizational Considerations ...............................................................4
4. Configuration Planning Design Principles ...........................................5
5. Configuration Identification Design Principles ..................................7
   Design with the End State in Mind ..........................................................7
   Define CI Classes .......................................................................................7
   Define Attributes for Each CI Class .......................................................12
   Define Dependencies ...............................................................................13
   Modify or Extend OOTB CI Classes ......................................................15
6. Configuration Control Design Principles .............................................16
7. Configuration Verification and Audit Design Principles ..................16
8. Configuration Status Accounting Design Principles .......................17
9. CMDB Population Efforts ........................................................................17
10. Integration with Other Processes ......................................................18
    Asset Management ..................................................................................19
    Project Management ...............................................................................20
    Information Security ...............................................................................20
11. Summary .....................................................................................................20
Appendix ..........................................................................................................22
   A. Related References ...............................................................................22
   B. Example Configuration Management Plan Outline .......................23
   C. Example Configuration Control Board Charter ...............................26
1. Introduction

This white paper provides general guidance for designing the CMDB prior to initial implementation and subsequent improvement efforts. Please review this document in its entirety, as there are a number of design elements which overlap and could likely impact your design. The design guidance provided is considered to be in alignment with configuration management and ITIL best practices as it specifically relates to ServiceNow. For more information about how to implement the suggestions, see the Appendix.
A successful configuration management capability will depend upon the ability to continually evaluate and adjust configuration management’s value proposition over time.

2. Definition of Configuration Management

Having a clear understanding of configuration management is a prerequisite to designing your CMDB. Per ITIL v2011, configuration management is defined as:

“...the process responsible for ensuring that the assets required to deliver services are properly controlled, and that accurate and reliable information about those assets is available when and where it is needed. This information includes details of how the assets have been configured and the relationships between assets.”

Consider another definition of configuration management, which provides some additional clarity (source: MIL-STD-3046 dated 6 March 2013):

“...an engineering and management process which ensures the configuration of an item is known and documented and changes to an item are controlled and tracked for purposes of establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information.”

The latter definition provides more of an engineering perspective which emphasizes the underlying value proposition of configuration management: ensure a product or service matches its documentation and meets expectations. The Institute of Configuration Management has published a statistic which indicates having clear, complete, and concise information to manage your products and services which comes from a configuration management capability can reduce the cost of sales by up to 40% by reducing corrective action. Source: Institute of Configuration Management (icmhq.com), white paper CMII-830B – “Monetary Value of Traditional Configuration Management versus CMII,” Revision B, published January 14, 2014. There is a link to this white paper in Appendix A

3. Organizational Considerations

Configuration management provides a single common representation which models the services, applications, and underlying IT infrastructure which the entire organization can leverage. One way to think of the CMDB is as a decision support system for your organization - providing data to aid in strategic, tactical, and situational decisions. The value-add can go well beyond IT service management – there is real value to numerous other business areas including audits, compliance, customer service, facilities, legal, human resources, etc. Insights into what a product or service is expected to do along with the information describing how the product or service is designed, implemented, and performing offers an unparalleled mechanism to govern and deliver your products and services.

Many organizations struggle to find the “right” amount of configuration management. Trying to have an all-encompassing “let’s include everything” capability often proves to be too expensive, time-consuming, and cumbersome, resulting in an inadequate value proposition and failure. Just the opposite is true as well—too little configuration management will not deliver any meaningful improvement to an organization’s ability to deliver products and services to their customers. This situation usually results in organizational apathy and failure. As the business requirements change to adapt to the evolving marketplace, the underlying value proposition for configuration management must also change.
Initial implementation and ongoing evolution of configuration management is best accomplished as part of planned and structured efforts working towards defined goals and objectives (i.e., projects) rather than an ad-hoc or as-needed, organic-growth approach to implementing improvements. Customary practice is to execute planned and structured configuration management efforts which create and maintain a document commonly referred to as a Configuration Management Plan (CM Plan) or a Service Asset and Configuration Management Plan (SACM Plan) if ITIL terminology has been adopted. Refer to Appendix B for an example Configuration Management Plan Outline.

4. Configuration Planning Design Principles

Configuration planning begins with the establishment of goals and objectives for configuration management efforts. Agreement upon configuration management goals and objectives is essential to ensure stakeholders are in alignment and agree upon the priorities established. Your configuration management efforts should be clearly and explicitly tied to the strategic goals of your organization and to the tactical goals of your IT department.

Refer to Appendix A for the strategy and planning white paper “Configuration Management Database in ServiceNow.”

Often there will be some differing opinions on priorities, but this is generally overcome by clear alignment to the eventual goals. Without aligned and engaged stakeholders, configuration management efforts are likely to struggle and ultimately fail. A documented future state roadmap is a very useful tool for communicating the long-term vision for the configuration management capability and helping socialize an incremental approach to implementation.

The ServiceNow CMDB is designed to be flexible. The out-of-the-box (OOTB) CMDB can leverage the discovery data of your infrastructure by enabling the import of details about each of the discovered devices within the infrastructure into the CMDB, map the default OOTB dependencies between these devices, and periodically refresh the data and dependencies within the CMDB. For some very small number of organizations, this minimalistic solution may represent an adequate value proposition. Most other organizations seek a considerably larger value proposition from their configuration management efforts.

While writing the CM/SACM Plan, a clear scope must be defined so that everyone in the organization understands what is and what is not in scope. For example, many organizations have development/testing “labs” where infrastructure changes must occur rapidly and frequently. Initially ruling these areas out of scope for configuration management may be necessary to limit operational impacts to those efforts. Refer to 9. CMDB Data Population Efforts below for additional information.

A CM/SACM Plan usually also defines approved configuration management policies and procedures. Procedures often document not only how configuration management integrates with other service management processes and other IT services, but should also include the processes the configuration management team is going to follow. For example, how are CMDB enhancement requests going to be managed or how will requests to run an out-of-cycle report be handled?
Another common request is for configuration management teams to participate on large strategic technology projects. This is called “project tailoring” in configuration management practices. A senior configuration management team resource is embedded on the large strategic project teams, to assess the changes needed in your configuration management capability and to ensure configuration management service are ready and available as the project requires them. Some of the types of changes commonly seen from technology projects involve discovery, data collection and validation, creation/alteration of CI classes, updates to the CM Plan, requirements for new CI audits and dashboards, etc.

It is also recommended to specify the governance for configuration management. Many organizations create a panel of key stakeholders to oversee and ensure configuration management is delivering the appropriate level of value-add to the organization. Historically, this panel is called a Configuration Control Board (CCB) and is staffed by mid-level management (senior enough to be directly accountable for strategic goals and still retain direct day-to-day oversight of technical IT staff delivering and supporting IT services). It is considered best practice to formally charter your CCB and get senior leadership to sign off on the authorities granted to this board. The more formally this governance is enacted, the greater the chances of cultural success and longevity. Refer to Appendix C for the Example Configuration Control Board Charter.

The CCB is not the same as a Change Advisory/Approval/Review Board, as the focus of the CCB is to ensure configuration management efforts are aligned to the strategic goals of the organization and ensure the day-to-day delivery of configuration management as documented.

The Governance section of your CM/SACM Plan should clearly document what role configuration management has with change management processes, release and deployment management, incident management, problem management, and other governance mechanisms such as an Architecture Board, any Program/Project Management Office, services provided by Privacy/Risk/Information Security organizations, etc.

Clear definition of roles and responsibilities is also often included in a CM/SACM Plan. Roles such as configuration management process owner, configuration manager, configuration management analysts, coordinators, specialists, etc. are often defined as part of the configuration management team. Other roles, such as ServiceNow administrators are usually part of other teams. Segregation of duties is generally considered best practice, as the configuration management team is the “business owner” of the CMDB, and the ServiceNow administrators serve as the IT technical support for the ServiceNow platform. It is likely there will be a much larger group of IT technical support teams supporting the entire Configuration Management System (CMS).
5. Configuration Identification Design Principles

Design with the End State in Mind

One of the most important aspects of a successful configuration management capability is for everyone who uses the CMDB to understand the data. In some respects, this is the most significant critical success factor—after all, what good is a decision support system if no one understands how to use the data within it? Consistency is the key here:

- Terminology and definitions: key words and acronyms are used consistently and there are clear organizational definitions for them
- Consistent data: every field has a defined purpose and is only used for that purpose (i.e. no field overloading)
- Visualizations:
  - Screens and reports have similar design
  - Colors are used consistently
  - Views of configuration item (CI) inter-dependencies have a consistent structure

Before populating the CMDB with data, consider starting with the end state in mind. It is important to have a vision of what the CMDB will look like when it is fully implemented and for the major milestones along the way as part of your future-state roadmap. Each iterative structured improvement effort should also have a clearly defined end-state (for that effort) in mind before beginning implementation efforts.

ServiceNow recommends utilizing use cases to help develop your end-state vision(s).

How will users of the CMDB use ServiceNow?

- What CI classes are needed to support their functional roles and duties?
- What CI attributes are needed for each CI class so the right decision support information is readily available?
- What dependencies between CIs are needed to fulfill the envisioned use cases?

Define CI Classes

Configuration Identification is the most difficult activity of configuration management’s functions. Deciding what CI classes are needed, deciding what to call them, what details to include about each one, and deciding how they should be visualized within the CMDB is going to present some initial struggles.

Another struggle, worth mentioning early on, is trying to model the high-level business services of your organization. More often than not you will find, just like most organizations, there simply isn’t any existing business services model which you can use. With the best of intentions, IT organizations lead efforts to develop a business services model but quickly realize that this is well outside their sphere of influence and is without the business drivers to gain the necessary business unit involvement. This has to be championed from the highest level of your organization if you wish to have the buy-in from your business leaders. Thus, almost by default, most ServiceNow customers fall back on a list of applications to use as their highest level CIs. Quite a few IT organizations also find some success defining IT services and fewer still “invent” their own business service model (something is better than nothing right?). If you create your own business model in the CMDB, be careful how this is used when interacting with your business leaders as there is no agreement the model is correct from their perspective.
When you are defining services, please keep in mind the best practice of naming a service is to ensure their name reflects some action. For example, there is no inherent action for “Payroll.” “Manage Payroll” is a better choice for naming this example service. After all, one definition for the word “service” is “the action of helping or doing work.” Using verbs will aid you later when you go to build your dependency maps.

Please keep in mind that Service CIs (those depicting business services, e.g., “Manage Payroll”) in the CMDB are not the same thing as an item from your service catalog. Think of service catalog items as a list of things you can order from your organization rather than a way to organize the types of work each business area performs.

ServiceNow offers functionality to help you gain visibility of the services defined within your CMDB. By having a service-aware CMDB, discovery can automatically audit your environments, identify discrepancies, and even orchestrate programmatic fixes for some conditions. Please refer to the appropriate version of documentation on the ServiceNow wiki (docs.servicenow.com).

A few words about naming CIs. Most often, configuration items use the object-oriented (i.e. class and instance) nomenclature commonly used in IT. A class can be thought of as a set or collection of similar items; servers as an example. An individual server would be an instance of that class. By having CI class names which are unique among each other, and if all the instance names of CIs in that class are also unique amongst each other—you are guaranteed each CI will have a unique name.

Deciding at what granularity to define as a CI class is called “leveling.” In traditional configuration management practices, there is an axiom which says CIs should be leveled at the lowest level of independent change. That axiom is not cost-effective for the IT industry, however, it is a useful concept when determining the appropriate level for a CI class. Consider, every CI needs the following minimum registration data in your CMDB:

- A unique name which never changes (from “cradle to grave” to ensure traceability)
- A description of its fit and function within the context of the product(s) or service(s) it supports
- A defined set of attributes
- A defined set of allowable dependencies (relationships) to other CIs
- A defined configuration control process for process for managing changes to the CIs’ data within the CMDB
- A business owner (who can approve changes) and a technical owner (who supports this CI and implements changes).

How many parts of a physical server can be independently changed? Consider an example design scenario where physical server hardware is considered an assembly of a motherboard CI, memory SIM card CIs, network card CIs, power supply CIs and disk drive CIs. What benefit is realized from having additional levels of CI detail for the use cases envisioned? In addition, it would be difficult to develop and use a CI-naming schema for these lower level CIs such as power supplies, network cards, etc. Best practice has shown that serial numbers don’t make good CI names because there is no guarantee of uniqueness and using a 10, 15, or 20 alphanumeric character string as a CI name in incidents, problems, and changes is error-prone (not to mention tedious).

For most organizations, a physical server leveled as a single CI class is generally sufficient to accommodate most use cases. Of important note, in a small set of industries there are regulatory traceability requirements which require separate CIs for the operating system software and the hardware as these can be independently changed. Your design efforts should include conversations with the regulatory auditors within your organization to ensure compliance for the CMDB as a system of record.
Let’s use an insurance company to illustrate a few design options. Consider the use case where a caller has informed a service desk agent that the “Write New Business” application is down. The service desk agent needs to record the incident and route it to the appropriate support team. Sounds pretty simple, but how this is actually accomplished is entirely dependent upon the structure of the CMDB.

Suppose the service desk agent can’t find an application CI called “Write New Business” to associate with the incident ticket in order to route it to the appropriate support team. Is “Write New Business” really an application? Or is it just one small piece of functionality as part of a larger software solution? Are all service desk agents, who could be anywhere on the globe, just supposed to have the acumen to determine that “Write New Business” is part of another application?

For this example, the “Write New Business” function is part of the insurance company’s claims application. There are at least a couple of different ways to design a solution which provides the service desk agent with a method for handling this call. Refer to the illustration below.

Figure 1: Example single CI class for application

One solution is to have a single CI class for applications with an attribute which can be added to the layout of the incident form and defines the major functions of the application. This choice list attribute can be used by the service desk agent to help route notifications to the different CI owners and support teams. In addition, the use of an alias attribute can help the service desk agent search for other names (such as Write New Business).

A second design option is shown in the illustration below which might have proved beneficial to this insurance company.

Figure 2: Example CI classes for application and major functions

Claims (Application)

Print Checks (Major Function)

Write New Business (Major Function)
In the previous illustration, the CI class called Application has been defined as a logical CI to represent the major software tools people use to deliver business outcomes. Another CI class called Major Function has also been defined to improve understanding of how large application can have several logical functions. In this case, the Major Function CI class is optional—not all Application CIs could require lower-level Major Function CIs. This design and leveling option can be useful if any of these conditions are true:

- Each function has separate business owners and/or support teams
- Each function uses separate and distinct combination of infrastructure components
- Each function has different monitoring requirements
- Each function has differing OLAs and SLAs and this makes availability and outage reporting more refined
- Business continuity/disaster recovery requirements are different across functional areas

The insurance company reviewed their business recovery capabilities for their major business functions. It was determined the biggest application, Claims, would take too long to recover due to the large scale of the infrastructure needed to drive the application and the time it would take to restore the data. For purposes of this example, the insurance company decided to redesign the Claims application, splitting major functions into separate applications. The illustration below shows two example stand-alone applications which resulted from this redesign.

ServiceNow customers often report difficulty in defining what an “application” is. It seems everyone has a different definition and no single criterion seems to provide an adequate definition. When a CMDB user sees a CI that is classified as an Application CI, do they understand what is meant by the term “application”? Is something like the Claims example considered an application? Is a small database solution written by someone in a business area on their desktop an application? Even if this home-grown “application” is business-critical and heavily used by that department? Or how about an important spreadsheet with macros? Is the internet browser installed on every workstation an application?
What are the characteristics you might use to determine if something meets your definition of what an application is? Consider these possibilities:

- Number of users
- Business criticality
- Requires a set of user ID/passwords to be granted role-based permissions
- Purchased commercial software package requiring licensing
- Software that delivers business outcomes
- Software that delivers business outcomes which is hosted in a data center or in any access-controlled, secured computer closet
- Software deemed important enough to be in scope for change management
- Software the service desk is likely to be contacted about if there is a service disruption
- Software which a critical business service is dependent upon
- Software that has formal availability targets (e.g., OLAs, SLAs)
- Software that has support for it outsourced to another business partner/supplier

Consider this approach:

- Tentatively agree upon a draft definition for what an application is and then proceed into an iterative series of exploratory activities.
- Use your discovery tools to capture the inventory and dependencies of your environment.
- Now using the draft definition for an application, as a group can you and your co-workers clearly define which infrastructure components support which applications?

This tends to be more difficult than it might appear to be because data tend to flow across a chain of applications and there could be numerous shared infrastructure component CIs along that path.

The view of where an application “begins and ends” is often a function of the role a person has within the organization. It is expected that applications will share CIs as they represent shared resources across multiple applications.

At the end of the day, you need a definition of an application which is meaningful to your organization and which allows you to clearly delineate what infrastructure component CIs support each application.

In all likelihood, the original draft definition of an “application” is going to evolve as you explore. There is simply no substitute for taking the time to explore this as an organization in order to formulate your definition of an application.

Keep in mind there is no right or wrong way to model your dependencies, as long as everyone in your organization understands what they are looking at. If you are trying to decide if something should be a separate CI class or an attribute of a CI class, remember, an instance of each CI class will appear as a node on a dependency map. Do you want people to explicitly see that information when performing impact analysis?
Don’t forget there is a minimum set of information needed to define something as a CI. As an example, there is a desire to have Memory SIM cards as a CI class because so many of the applications for your organization have critical memory requirements. In order to have a CI class for SIM cards, there must be a definitive method for how each individual SIM card CI will be named. There are likely to be hundreds, thousands, or tens of thousands of these cards in your environment—what unique nomenclature will be given to each one? This is a difficult prospect at best, and the value proposition doesn’t really support the cost of managing this CI class. You will find that often the requirement to have a unique name for each CI of a CI class eliminates the possibility of a separate CI class.

You still want to ensure that critical memory requirements are met. So given that requirement, consider this alternative design solution:

• Have one attribute of the Server CI class for required memory as approved by change management and one attribute for discovered memory.
• Write an audit script that identifies, reports, and alerts if there are discrepancies every time a discovery event is performed and refreshes the attribute for discovered memory by comparing the as-discovered value against the as-approved value.

ServiceNow is often asked about when a particular OOTB CI class should be used, so a guide to aid you in deciding which OOTB CI class to use has been created. For each OOTB CI class, a general description of what each CI class is trying to represent in the OOTB solution is provided, along with additional details to aid your design efforts. Be sure to read about modifying or extending OOTB CI classes further down in this document for additional information.

Define Attributes for Each CI Class

Once CI classes have been defined, design considerations should be taken when identifying attributes for each CI class. Many configuration management implementation efforts have struggled or failed outright because of poorly designed attribute strategies—it is simply not cost-effective to manage every possible attribute for every CI.

Some attributes are likely important enough to warrant using the proposed changes feature of the Change Management application in ServiceNow so that changes needed to bring the CMDB up to date after a change can reflect the newly changed environment as soon as possible. If you were a change manager, would you want to approve a change request if the implementation team couldn’t tell you what updates to the CMDB will be required after the change is successfully implemented? Shouldn’t the implementation team be able to tell you that before they make that change?

Other attributes may be left open for edit, with data added/changed as the support teams deem necessary (such as a Journal field). Still others may be imported into the CMDB after a discovery operation is performed. Generally speaking, it is best to separate the “critical few from the trivial many.” Only require proposed changes for a small number of critical attributes, severely limit the number of fields left open to ad-hoc updates, and use discovery to populate the CMDB with the (read-only) technical data points as desired. (In practice, the discovered read-only attributes are overwritten with each discovery operation.)

CMDB performance issues have been reported to ServiceNow when viewing a large number of CIs in List view when a large number of attributes are present on that specific CI class table. For example, the Server CI class has hundreds of attributes, and you are trying to display a list of thousands of servers. Thus, before the addition of an attribute, consider the value of that attribute carefully. How does each attribute contribute to support impact analysis for incident resolution, problem management analysis, change management, or other decision support use cases? Best practice is to start with a minimum number of attributes and add them as a valid value proposition is articulated.
In addition, it is best practice to group all discovered attributes together, make the fields read-only on user interface forms, and provide a date/time stamp for when the data was collected by the discovery tool and the name of the discovery tool which performed the discovery. The time/date/source information will prove to be critical for people who are using the CMDB for decision support to manage risk. For example, knowing that discovery data in the CMDB about a specific CI is five days old may impact their decision making process.

Experience indicates there are usually only a limited number of attributes critical enough to warrant proactive management using the proposed changes feature and most of those are not discoverable. For example, an attribute to indicate if a server has a specific start-up procedure associated with it is not a discoverable attribute, but it is important to know if the procedure is changing. Attributes about the manufacturer, model, and serial number of the network interface card in the server will not change (unless the card is replaced) and are good candidates to be refreshed with each discovery operation. Many organizations write additional auditing scripts to inspect discovery data to identify possible unapproved changes.

When finalizing the list of attributes needed for a CI class, check to ensure certain attributes are not the basis for licensing or billing. Some software publishers use some characteristic of an operating environment to calculate costs. For example, software may be licensed for some number of processor cores. You will need to make sure the CMDB can provide for auditing the number of approved cores versus the actual number of cores discovered to be in use.

**Define Dependencies**

With CI classes and attributes defined, the design focus now shifts to dependency mapping. ServiceNow recommends that you design your CMDB to reinforce best practices with your dependency maps. Does your organization leverage an enterprise architecture or reference architecture that is used to provide design principles for new applications and implementing infrastructure changes? One aspect of these types of standards is they imply a structure depicting how technologies are deployed to deliver business solutions.

Consider the illustration below of a generic and highly simplified reference model.

---

**Figure 4: Example simple reference architecture model**

---

*Business Service Layer*  
*Presentation Layer*  
*Business Logic Layer*  
*Data Layer*
Using this example reference architecture model, all CI classes used to model business services would appear at the top of all hierarchical dependency models within ServiceNow. Next lower levels of the model, in descending order, show the Presentation layer, Business Logic layer and Data layer.

The illustration below is a simple example to show how a dependency map might look when implemented using the example reference model.

The top layer in the illustration above represents a CI class to model Business Services. The next layer depicts a CI class used to model Business Functions. Note in the reference model example in the previous illustration, the light blue layer models the business functions, and in this example two CI classes have been defined to depict the organization’s business model. The orange layer depicts the presentation layer (where the accounting staff is presented with two different interfaces to deliver their assigned business function) which includes the Application CI class containing two application CIs (Accounts Payable and Accounts Receivable). In this simple example, all business logic runs on two separate servers as shown in the dark blue layer. Finally, the green layer represents the data layer depicted by a CI class representing databases.

ServiceNow recommends that consideration be given to leveraging a reference architecture model when designing for the dependency models. TOGAF® (The Open Group Architecture Forum), NIST, ISO/IEC/IEEE 42010:2011 and GERAM® (Generalized Enterprise Reference Architecture and Methodology) are some examples, and an internet search on any of these will provide additional information. Aligning your Dependency Maps to reflect your reference architecture provides reinforcement for the desired architecture and ensures consistency in how relationships are modeled in your CMDB.
Regardless whether there is a reference architecture, the allowable relationships between CI classes need to be identified and enforced within the dependency map feature. Using the example above, this means rules should be established to prevent a database from having a direct dependency to a Business Service. In fact, CIs of the CI class Database may only have a parent of a CI of CI class Server in this example. This type of enforced consistency is highly desirable.

**Modify or Extend OOTB CI Classes**

While the OOTB solution has many commonly used attributes, it is often not a complete solution for every customer. It is common practice to use some OOTB attributes, alter some (e.g., OOTB choice list values), not use others, and add some new attribute fields to the OOTB CI classes from ServiceNow. Just remember when in the System Definition tables, inheritance rules apply—anything added or changed to a CI class table will impact every CI of that class and every CI in every sub-class. For example, if an attribute is added to the Server table, then every CI class extending the Server table will also include that attribute. Be careful of class inheritance when making changes to the CMDB tables.

A design question often asked is “when should an existing CI class be modified or when should an OOTB CI class be extended to create a new CI sub-class”? In most cases, using an OOTB CI class is preferred when:

- An OOTB CI class contains more than 50% of the required attributes
- The definition of the OOTB CI class is in alignment with your definition for the fit and function of the CIs of that CI class
- The OOTB CI class is properly positioned in the default ServiceNow CMDB dependency views relationship hierarchy

Sometimes, the name of the OOTB CI class presents a cultural issue for an organization. For example, the common term of “host” is used instead of the term “server” which is an OOTB CI class. Changing this terminology from “host” to “server” can have significant impact—it may require training, adjustments to discovery mechanisms, monitoring, alerting, reports, etc. Carefully consider the cultural impact of any decision to change the common vernacular used inside your organization.

There are two design options for this situation: you can extend the OOTB Server CI class to create a new CI class called Host or change the labels used in ServiceNow when displaying CIs of that class. Changing the CI class database table name is **NOT** recommended and should be avoided in ALL cases.

Extending an OOTB CI class to create new sub-classes is particularly useful if there are a number of variants present which will each possess different attributes. For example, the OOTB Server CI class has already been extended creating sub-classes for Linux, Unix, OS/X, Windows servers, etc. Each of these sub-classes has different additional OOTB attributes from the base CI class of Server.

In most cases, extending an OOTB CI class to create a new base CI class is preferred when

- Several variants exist
- An OOTB CI class contains less than 50% of the required attributes
- An OOTB CI class is not properly positioned in the default ServiceNow relationship hierarchy
- There is no OOTB CI class which aligns with your organization’s definition for the fit and function of the CIs of the new CI class
6. Configuration Control Design Principles

ITIL 2011 defines configuration control as “the activity responsible for ensuring that adding, modifying or removing a configuration item is properly managed—for example, by submitting a request for change or service request.” Notice configuration management practices do not mandate any specific requirements for how changes are made to your CIs, other than to say your organization should be satisfied that your controls meet your operational requirements. That leaves a lot of room for variation, allowing your organization to implement change mechanisms that are cost-effective and a value-add. The Institute of Configuration Management advocates a best practice, stating that your goal should be 80% of changes take the form of a “fast path” process.

For a variety of reasons, ServiceNow customers do not always build and maintain service maps for every CI. This can be likened to a CMDB containing an inventory of CIs which are managed via Change Management. ServiceNow advocates the position that there is a key difference between a CI being under configuration control versus a CI being under change management. From ServiceNow’s point of view, by definition, configuration control requires proactive management of changes (including dependencies) and change management does not. After all, the word “configuration” is defined as “an arrangement of parts.”

To restate, if there is no management of relationships, there is no configuration control. ServiceNow recommends using the Propose Change feature within ServiceNow Change Management to bring a CI under configuration control. The propose changes feature must be the only method allowed to update the data for a CI using the normal user interface while the CIs are deployed into “production” (i.e., actively deployed in support of a defined business or IT service). No CMDB user (not even the Configuration Manager) has permissions to directly “edit” data using the normal user interface if a CI is under configuration control. (Of course, changes with an approved Request for Change can use other update capabilities, besides Proposed Changes, such as an import set to introduce changes.) This is not to say that a CI is not under change management at all times, but the best practice is for CIs deployed into “production” to be under configuration control.

7. Configuration Verification and Audit Design Principles

The Configuration Audit function has two distinct aspects as defined by ITIL: verification and audit. These terms are also commonly used across a number of other practice domains, such as project management and quality assurance.

Verification is an activity that ensures that a new or changed IT service, process, plan, or other deliverable (i.e., a CI) is complete, accurate, reliable, and matches its design specification. This is not to be confused with validation: assurance that a product, service, or system meets the needs of the customer. CI owners are responsible and accountable for validation efforts and this falls outside of the configuration management domain.

Audits are formal inspections to check whether a standard or set of guidelines is being followed, that records are accurate, or that efficiency and effectiveness targets are being met. Another easy-to-remember distinction between verification and audits: verification happens in real time and audits are scheduled (or triggered) events.

ServiceNow supports verification efforts with Proposed Change Verification Rules. If verification rules are defined, the Change Management application can automatically execute the rules (as part of applying a proposed change) to safeguard CMDB data integrity by ensuring invalid data is not introduced by proposed changes. This is the
preferred method for ensuring service mapping rules are enforced and attribute data quality. Some examples of verification rules include:

- Ensuring new CIs aren’t duplicates
- Ensuring an attribute is valid (e.g., the new value is an allowable value of a choice list, a date field contains a date within an expected date range, etc.)
- Ensuring a new server name complies with the desired naming schema
- Ensuring a Database CI is not trying to add an invalid relationship to a Business Service CI

ServiceNow supports audits using health properties and the CMDB health dashboard. These are scripts which can be executed on a predefined schedule or on-demand. User-developed auditing scripts allow an organization to certify the CMDB meets defined controls.

A design question that will also arise is whether the same verification and auditing rules apply to all CIs. Are CIs not under configuration control subject to the same rules? The rules for service mapping won’t apply to CIs with no dependencies. Recall the previously mentioned concept of “production” — are some of the rules different for CIs deployed into production that say other environments (say development)?

When designing your CMDB, best practice is to define these rules at the same time you are identifying the attributes and the structure of your service maps. This effort should not be an after-thought! ServiceNow recommends identifying resources to develop and test auditing rule scripts when planning a structured effort to implement/improve the CMDB.

8. Configuration Status Accounting Design Principles

ServiceNow contains numerous OOTB reports related to configuration management. These reports are generally valuable to the primary consumers of the CMDB—the IT staff and owners of business services. Using the portal to tap into the reporting capabilities of ServiceNow is an excellent way to provide customized information to many additional parts of your organization, without having to sign onto the ServiceNow application. When designing your CMDB and a use case presents itself for areas of your organization which do not generally have need to log into and use ServiceNow, you should be able to accommodate their requirements by using the portal to access ServiceNow’s reporting functionality.

9. CMDB Data Population Efforts

With a CM/SACM Plan in hand, efforts can now be focused on a strategy for importing data into the CMDB. What to import first? There are a number of approaches which can be used and all are equally valid in their approach.

Consider these few examples for data collection and import approaches (not in any implied or preferred order):

- **Vertical**: Collect data for the critical customer-facing/revenue-generating applications. In other words, collect data for every CI for which the application has a dependency—from top to bottom. This is sometimes done in batches of, say, 10 applications at a time. Also, this approach often leverages a business continuity recovery priority order (if established) to determine the prioritized order of data collection. This approach offers the benefit of having a complete and full CMDB definition for the applications which are completed.

- **Horizontal**: Collect data for all CIs of a specific CI class. For example, collect data for all Windows servers. This approach appears to offer a “quick win,” but until data for every CI class are collected and imported, full “end-to-end” service maps for applications cannot be completed.
It is best practice not to import data into the CMDB if there are known quality issues with the data.

- **Environment:** Limit data collection to only CIs which support a specific environment (e.g., production, QA, testing, staging, development, etc.). Some organizations include both the Production and the final environment just prior to operational release (per the software development life cycle). The names of this environment varies from organization to organization, but this approach offers the benefit of focusing on data collection and importation of CIs which are critical to providing stable IT services to your business and customers.

- **Location:** Limit data collection to everything hosted in a single physical location. This has the benefit of localizing the implementation scope but may result in incomplete service maps if services or applications depend on CIs which are hosted outside that physical location.

Most often, combinations of these approaches are employed at the same time. There will likely be several “waves” of data collection and importation into the CMDB over time, as most organizations have too much data to manage in a single effort. Each “wave” may employ different combinations of these approaches as needed.

ServiceNow recommends managing the data collection and importation of data very carefully. The quality of the data initially captured is likely to be fairly low. Missing data, inconsistent fields, incorrect data, and old and unneeded data are common. Review the data, and have CI owners certify the data as accurate (complete, correct, and concise) before importing into the CMDB.

The cultural success of a configuration management capability is directly related to how users perceive the value-add offered. Is there a current culture of “untrusted data”? Most organizations will admit to only having 50%, 60%, maybe 75% accurate data about their CIs in other data sources before they begin implementing configuration management. Simply moving the existing “untrusted” data into a CMDB will not improve trust levels. Have CI owners review the data and have CI owners (both business and technical) certify the data as accurate before importing into the CMDB. If data discrepancies are found later, these should result in an incident, so data quality is an ongoing imperative.

This cannot be overstated: if the data in the CMDB are just as inaccurate, incomplete, and inconsistent as they were in the spreadsheets, databases, and other sources of data where it was before implementing a CMDB, the cultural acceptance of the CMDB will suffer greatly. In some cases, a CMDB’s bad reputation has led to the failure to accomplish the goals and objectives established for configuration management. You have this one chance to remediate the data and break “the culture of untrusted data”—there will not be another chance.

If you add a required attribute to a CI class after it is populated and in use, careful consideration must be given to ensuring this is done without disruption. If there is no data in a required field, the CMDB will try to enforce this requirement if the record is opened in the CMDB. This causes the user to experience a situation where they cannot close that record without providing a value for the required field. If possible, collect data for every CI and load this data at the same time you add the field to the form.

10. Integrations with Other Processes

Configuration management serves as a central tenet of many IT service management processes, such as incident management, problem management, change management, etc. ServiceNow provides OOTB integrations with these processes which can be easily configured to your organization’s specific needs.

This section will provide some high-level design considerations for some other key process integrations.
Asset Management

Asset management is usually considered a financial function where information for certain items is managed and reported. IT departments usually support this effort by providing information into the status, location, and management of IT components to the Finance department. ServiceNow provides an OOTB solution for physical (hardware) assets, software assets (licensing), and consumables (items often below the asset threshold value but widely needed throughout the business).

Configuration management depends on a solid set of basic asset management functions. New IT hardware must first be requested, approved, ordered, and received before the IT department deploys it into an operational state where it becomes a CI. The figure below helps to illustrate that some items are assets (routers, storage arrays, physical servers, desktops and laptops, etc.) and some things are just CIs (business services, documents, subnets, etc.). Some items will be both an asset and a CI.

Figure 6: Asset and CI Overlap

To illustrate this concept, consider this question: is a printer a CI, an asset, or both? The average personal desktop printer is pretty inexpensive nowadays—likely to be under your organizational financial threshold for an asset. So, chances are a personal desktop printer would be treated as a consumable and not an asset.

There is a MICR (magnetic ink character recognition) printer in the Finance department which is used to print checks which are mailed to your customers, print your paychecks, etc. This printer would be treated as an asset and a CI because key business services have a dependency on that printer. If that printer is broken, those related services are impacted.

Now, consider the printer at the end of the hall which serves your entire office floor. If it is not leased, in all probability it has a value over your organization’s financial threshold, making it an asset. But is it a CI? Is any business service dependent upon that printer to deliver services? In all likelihood, no, because there is always the option to use another nearby printer for people to accomplish their work. So this printer is just an asset, using service requests and not change management to manage changes.

An asset usually becomes a CI when it is initially deployed into a live environment, which should then require an approved change via change management going forward. At some later point in time, that CI may be used to support a critical business service and be placed under configuration control.
Project Management

Projects and project managers can greatly benefit from leveraging data from the CMDB and the larger CMS. Often early in the feasibility phase or the phase to obtain funding for an IT infrastructure project, accurate counts of devices or other infrastructure components are needed. In addition, configuration management data can help establish which IT technical support teams and business areas should participate in the project. New or changing licensing fees can be estimated, using the example of core-based licensing model. Transformational IT infrastructure projects should include configuration management project tailoring, as the CMDB and the broader CMS needs to be ready for use when preparation and deployment activities occur. Without this forward-thinking insight, projects could be delayed or implemented without the ability to leverage configuration management capabilities. Policies and procedures for the interaction of project management and inclusion of the configuration management team for IT infrastructure projects should be defined, published, communicated, and adhered to.

Information Security

Just like for project management, the CMDB offers easy access to data which is useful to Information Security, which may require some specialized views of CMDB data and service maps. In addition, you will want to consult with your Information Security team to have them assess the risks and vulnerabilities of having sensitive data (e.g., IP addresses, connection protocols, current patch levels, support staff names and phone numbers, etc.) available to all staff who will be using the CMDB with the ITIL user access role.

11. Summary

This white paper provides general guidance for designing the CMDB prior to initial implementation and subsequent improvement efforts. Below is a recap of some of the main concepts discussed:

- Going beyond the standard ITIL definition of configuration management, consider another definition of configuration management which provides some additional clarity:
  
  “An engineering and management process which ensures the configuration of an item is known and documented and changes to an item are controlled and tracked for purposes of establishing and maintaining consistency of a product’s performance, functional, and physical attributes with its requirements, design, and operational information.” (source: MIL-STD-3046 dated 6 March 2013)

- Configuration management provides a single common representation which models the services, applications, and underlying IT infrastructure which your entire organization can leverage. However, the value proposition goes well beyond the scope of IT. Insights into what a service is expected to do, along with the information describing how the product or service is designed, implemented, and performing, offer an unparalleled mechanism to govern and deliver your products and services.

- A successful configuration management capability will depend upon the ability of your organization to continually evaluate and adjust configuration management’s value proposition over time.

- Take the time to write a Configuration Management Plan before you begin implementing your CMDB. Allowing unmanaged “organic” growth will almost certainly result in outright failure or failure to achieve and maintain a positive value proposition.

- Form a Configuration Control Board—a steering committee for configuration management—to set direction, resolve issues, and prioritize improvements. Staff this group with leadership high enough up in the organization to have a clear stake in the strategic goals of your organization and yet close enough to the day-to-day support efforts to understand the impact of day-to-day configuration management activities.
• Document your use cases and use them to drive data requirements.

• Configuration items require a specific minimum set of information to be registered into the CMDB, one of which is a unique name which never changes. Make use of out-of-the-box CI classes, but do not be constrained by them. It is a normal and expected practice to add CI classes and attributes or change the values in a selection list to suit the needs of your use cases and your organization.

• Have a clear definition for what constitutes an “application.” There are a wide variety of software solutions which leverage a wide variety of technologies and types of operational delivery, support and governance models. Expect the definition of an application to be refined as the scope of configuration management expands.

• Not everything is created equal. Some business services are critical, such as customer-facing/revenue-generating services, while some are not critical, such as the intranet site which has the company cafeteria menu displayed. Attributes are also not created equal, as in most cases attributes which are collected via any discovery technology should be read only in the CMDB. There is no sense editing / maintaining an attribute field which is going to be overwritten the next time a discovery is performed. There will be a small set of critical attributes which must be formally managed using change management and the “proposed changes” feature.

• Define allowable relationships between CI classes. Develop a framework model for your CI to CI relationships which aligns with your reference architecture if the out-of-the-box service maps do not meet your needs.

• Poor data quality must be avoided at all cost. Do not load data into the CMDB if it is known to be inaccurate, unless you take steps to visually segregate the inaccurate data in some fashion so people know it is not accurate but is the best available at that time. People must understand what they are looking at when they view the CMDB and have confidence they understand the context of where the data came from and when it was collected.
Appendix A

Related References

- Example Configuration Management Plan Outline (see Appendix B)
- Example Configuration Control Board Charter (see Appendix C)
Appendix B
Example Configuration Management Plan Outline

1. Introduction
   a. Purpose
   b. Background and Context
   c. Scope
   d. Definitions
   e. Goals and Objectives
   f. Future State Roadmap
   g. References
   h. Applicable Policies and Standards
   i. Hierarchy of Configuration Management Plans

2. Configuration Management Organization
   a. Organizational structure
   b. Current Assignments
   c. Roles and Responsibilities
   d. Configuration Management Job Family
   e. Configuration Control Board
   f. Interfaces to Other Governance Mechanisms

3. Configuration Planning
   a. Overview
   b. Project Tailoring
   c. Continuous Process Improvement
   d. Training Strategy
   e. Communications Strategy
   f. Policies
      i. Configuration Management Plans
      ii. Configuration Management Plan Reviews
      iii. Deviations and Waivers
      iv. Organization and Assignment Changes
      v. Communications
      vi. Service Requests
   g. Procedures

4. Configuration Identification
   a. Overview
   b. Policies
      i. Configuration Item Classification
      ii. Configuration Item Identification and Registration
      iii. Configuration Item Naming
      iv. Decommissioning and Archiving
      v. CI Relationship Modeling
      vi. Configuration Baselines
      vii. Definitive Media Library and Hardware Store
      viii. Relationship with Asset Management
      ix. Data Confidentiality
   c. Procedures
5. Configuration Control/Change Management
   a. Overview
   b. Policies
      i. Required integration with Change Management
      ii. Configuration Control Board
      iii. Baseline Control
      iv. Interface Control
      v. Version Control
      vi. Supplier/Sub-Contractor Control
      vii. Release Authority
      viii. Relationship with Release and Deployment Management
   c. Procedures

6. Verification and Audit
   a. Overview
   b. Policies
      i. Verification of Changes
      ii. Scheduled Audits
      iii. Data Integrity Incidents and Response
      iv. Audit Responses and Closure
   c. Procedures

7. Status Accounting
   a. Overview
   b. Policies
      i. Standard Reports
      ii. Customized Reports
      iii. Dashboard Reporting
      iv. Integration with Other Processes
   c. Procedures

8. Interface Control
   a. Overview
   b. Policies
      i. Life Cycle and Procedural Interfaces
      ii. Organizational Interfaces
      iii. Hardware Interfaces
      iv. Software Interfaces
   c. Procedures

9. Supplier/Sub-Contractor Management
   a. Overview
   b. Policies
      i. Supplier/Sub-Contractor Acceptance of Work
      ii. Supplier/Sub-Contractor Issue Management
      iii. Supplier/Sub-Contractor Periodic Reviews
      iv. Supplier/Sub-Contractor Audits
   c. Procedures
10. Configuration Management System (CMS) and Tools
   a. Requirements for the CMS
   b. Logical Design of the CMS
   c. Physical Design of the CMS
   d. Use Cases
   e. Testing
      i. Test Plans
      ii. Test Scripts
   f. Current Operational Constraints
   g. Maintaining the CMS

11. Document Information
   a. Approvals
   b. Revision Log
   c. Distribution List
Appendix C

Example Configuration Control Board Charter

1.0 – Configuration Control Board
This Charter establishes a Configuration Control Board (CCB) to oversee and direct actions and changes to the <organization> Configuration Management Plan and all related configuration management activities.

1.1 – Goals, Objectives, and Guiding Principles of the CCB
The goal of the CCB is to effectively manage configuration management efforts to improve the effectiveness and responsiveness of <organization>‘s information technology infrastructures to better enable <organization>‘s business efforts in alignment with corporate goals.

General objectives of the CCB are to:
• Ensure the <organization> Configuration Management Plan is in alignment with IT initiatives
• Oversee deviations and waivers to the <organization> Configuration Management Plan
• Provide leadership, managerial oversight, and decision-making process to ensure effective configuration management activities
• Review, approve, and prioritize changes to the <organization> Configuration Management Plan in order to ensure effective deployment of resources
• Promote and encourage interactions with other relevant processes, such as architecture, security, project management, business continuity planning/disaster recovery, internal audits, etc., to improve communication, coordination, and cooperation
• Ensure continuity of the <organization> Configuration Management Plan

The guiding principles of the CCB are:
• Serve as an example of best practices for <organization>‘s IT groups
• Adherence to, and full support of, corporate initiatives, such as information security, privacy, regulatory audits, production assurance, etc.
• Provide for the transparency of configuration management activities
• Make decisions that are consistent with reducing key person dependencies throughout <organization>‘s IT areas
• Assist in maturing <organization>‘s employees, processes, and tools to improve long-term business performance
• Act on requests for changes, deviations, and waivers within established timeframes
• Make decisions and recommendations that are cost-effective, justified, and have engaged other appropriate areas
1.2 – CCB Organization, Roles, and Responsibilities

The CCB has three roles: the CCB Chair, Board Members, and the CM Process Owner.
The following table defines the responsibilities and assignments of each role:

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibilities</th>
<th>Current Assignment</th>
</tr>
</thead>
</table>
| CCB Chair             | • Leads and manages the CCB, including but not limited to chairing/facilitating CCB meetings, publishing pre-read materials and agenda, establishing and prioritizing the agenda, overseeing communication of CCB decisions, arranging for meeting room and equipment, tracking of actions, etc.  
  • Publishes CCB minutes within one (1) working day of CCB meeting date  
  • Escalates issues to the CM Process Owner  
  • Appoints representative to attend/chair meeting/vote on his/her behalf in the event he/she cannot attend  
  • Acts as the single voice for communicating CCB decisions  
  • Manages transitions of Board Members  
  • Represents configuration management best practices and guiding principles  
  • Invites others to participate as a voting Guest CCB Member for agenda item(s) as needed | • Name, Email, and Phone                                                             |
| Full CCB Members      | • Prepares for each meeting by reviewing pre-read materials and additional information gathering as necessary  
  • Appoints representative to attend and vote on his/her behalf in the event he/she cannot attend  
  • Manages transitions with CCB Chair  
  • Represents configuration management best practices and guiding principles | • Name, Email, and Phone  
  • Name, Email, and Phone  
  • Name, Email, and Phone |
| Guest CCB Members     | • Prepares for meetings(s) by reviewing pre-read materials and additional information as necessary for agenda item(s) as directed by CCB Chair  
  • Appoints representative to attend and vote on his/her behalf in the event he/she cannot attend  
  • Manages transitions with CCB Chair  
  • Casts single vote on agenda items as directed by CCB Chair  
  • Represents configuration management best practices and guiding principles | • By invitation of CCB only |
| CM Process Owner      | • Serves as final decision authority on all issues escalated by the CCB Chair | • Name, Email, and Phone |

*CMDB Design Guidance, White Paper*
1.3 – CCB Rules and Contingencies
The following rules and contingencies are in place for the CCB:

1. A Board Member may review agenda items and provide their position/decision in advance of meetings.

2. In cases where a Board Member vote is not cast during the meeting, the vote may be communicated to the CCB Chair within two (2) business days after the CCB minutes are published. In all cases where the vote results in a tie, the vote is automatically escalated by the CCB Chair to the Configuration Management Sub-Process Manager.

3. Actions of the CCB may be appealed and submitted for a second CCB review. Further escalation within <organization> management is the responsibility of the Configuration Management Process Manager.

4. A majority of Board Members is required to conduct a CCB meeting where a vote on an agenda item is taken.

5. Guest CCB Members may cast single vote on the agenda items as directed by the CCB Chair.

6. Special meetings of the CCB, beyond regularly scheduled meetings, are approved and scheduled at the direction of the CCB Chair.

7. CCB meetings must continue until all agenda items have a disposition.

1.4 – CCB Meetings
The CCB shall meet monthly, within the first full calendar week.

The general format of the meeting shall be:
• The CCB Chair convenes the meeting
• Review urgent requests, if any
• Review status of ongoing CCB efforts/action items/issues, if any
• Review new business per agenda and disposition each agenda item
• Call for additional items of business
• Summary reading of minutes
• Close meeting

The table below outlines the possible actions the CCB may take:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign for Study</td>
<td>The CCB requires more information. Requires assignment and follow-up date.</td>
</tr>
<tr>
<td>Approve</td>
<td>The request will be implemented. Requires information regarding budget, priority, assignments, deadlines, and follow-up dates.</td>
</tr>
<tr>
<td>Approve with Modifications</td>
<td>The request will be implemented with modifications stated. Requires information regarding budget, priority, assignments, deadlines, and follow-up dates.</td>
</tr>
<tr>
<td>Defer</td>
<td>Approval/rejection authority is given to another person or entity. Requires specification of transfer of decision authority and follow-up dates.</td>
</tr>
<tr>
<td>Reject</td>
<td>The request will not be implemented. Requires information regarding the decision to be recorded.</td>
</tr>
<tr>
<td>Table</td>
<td>Request will be reviewed at a later date. Requires follow-up date.</td>
</tr>
</tbody>
</table>
1.5 – Triggering Criteria

The following list provides examples of events which could trigger CCB review:

- Authorization of new CI Classes
- Authorization of changes to the CM Plan
- Major changes to Cls
- Changes in relationship with other processes, such as architecture, security, project management, business continuity planning/disaster recovery, internal audits, etc.
- Establishment of configuration management goals and objectives, and alignment/adjustment to other corporate goals
- Review of Configuration Management metrics and scorecard
- Review of incidents or problems related to configuration management
- Other major activities with potential impact to configuration management efforts