servicenow

Define ServiceNow development management practices

What's in this Success Insight

This Success Insight is an introduction to ServiceNow® development management. It provides guidance on how to guardrail development on the Now Platform® by defining development management policies and practices.

It will answer the following key questions:

1. What is ServiceNow development management? Why is it important?
2. What concepts and practices do I need to consider when defining ServiceNow development management?
3. How do I define policies that support specific development management needs at my organization?

Key insights

- With strong ServiceNow development management practices, you also have effectively defined development standards, guidelines for customization and configuration, a development process flow, and a release management process.
- Start by defining a minimum viable set of policies as part of a starter management approach rather than trying to deliver a full, mature approach all at once.

1. What is ServiceNow development management? Why is it important?

ServiceNow development management sets the processes and foundational standards that will support your teams as they develop and configure high-quality applications on the platform while they also minimize technical debt. It also gives your development teams the common frameworks, processes, and tools to use when they develop on and maintain your Now Platform.

Your technical governance board should develop and/or approve your development management policies and procedures because these are key outcomes of solid ServiceNow technical governance.

2. What concepts and practices do I need to consider when defining ServiceNow development management?

Your ServiceNow development management approach should define your development standards, guidelines for customization and configuration, development process flow, and release management process. Let's look at each of these separately.

Practitioner insight: Make sure that your development management practices and policies align with your established golden rules. Golden rules are a set of simple, high-level principles that define how your organization expects to manage and use the Now Platform. Ideally, you'll have golden rules in place before you start defining detailed development management practices and policies. If you do, make sure that your development management practices and policies don't conflict with existing golden rules. If you don't have golden rules aren't yet, consider defining them.

Development standards

Development standards set basic expectations that developers must uphold when they work on the Now Platform at your organization. By creating and enforcing these standards, you'll have consistent development practices, which ultimately makes the platform easier to manage and leads to greater overall platform health.

These three development standards are integral: naming standards, development documentation, and development best practices.

Naming standards

Naming standards set conventions for how to name development objects so all developers can quickly identify objects and their purpose.

Create meaningful naming conventions that are easy to adhere to. This table shows the common objects that you need to define naming standards for.

Object	Considerations
Update sets	• Update set names should always be unique. • Use a naming convention made up of several parameters that allows developers to identify key information about the update set quickly. **Format**: <Story/Defect> - <Application> - <Developer Initials> - <Sequence> **Example**: STRY100151 - Incident - FL - 001
System properties	• Use a customer acronym or abbreviation to be able to quickly separate custom system properties from OOTB system properties. • Specify the application or functionality that the system property is related to. • Use descriptive words to encapsulate the system property's purpose. • Use dot notation to separate parameters. **Format:** <customer ID>.<application>.<purpose> **Example:** xyz.incident.max_reassignment_count
Database (field) names and variable names	• Be descriptive but brief. • Short field and variable names encourage shorter field labels, which is preferred. • Consider how fields and variables will be referenced in code. **Example**: u_target_date
Coding objects (script includes, business rules, client scripts, etc.)	• Use a company acronym or abbreviation as a prefix to quickly identify custom objects from OOTB objects. • Use descriptive words that quickly identify the object's intent. **Example:** XYZ: Calculate project duration on end date change

Development documentation

Many organizations overlook—or decide to avoid—creating development documentation. Don't let this happen at your organization! Make sure your teams create clear, consistent documentation and set the expectation that it's *required*. Also define what *acceptable* development documentation means for your organization.

To determine what development documentation you'll need, consider the questions in this table.

Area	What needs to be defined
Requirements documentation	• What level of detail is required for story descriptions? • What level of detail is required for acceptance criteria? • Who, if anyone, need to approve or sign off on requirements documentation?
Design documentation	• What level of detail is required for solution design documentation (integrations, table architecture, features, and capabilities)? • Are these designs captured in your architecture blueprint or as a knowledge article so other teams can reference them (e.g., information on interfaces, foundational data, etc.)? • Does your architecture review board need to review these design solutions to make sure there's no duplication of effort or reusability across the platform?
Coding documentation	Coding documentation are the notes and comments you put directly into the code. Consider: • When are code comments required? • What detail is required to be captured in code comments? For example: – Description of new and modified code – Story or defect number – Date – Developer name
Build documentation	Build documentation details all code objects developed as part of delivering on a request or story. Consider: • What level of detail is required for build documentation? – Objects created – Intent of the object – Developer name • Who should review build documentation for accuracy?

Equally important to creating this development documentation is ensuring that it's accessible and available when needed. Consider defining a documentation repository and enforcing it with your organization's technical governance policies.

Practitioner insight: ServiceNow Knowledge Management makes an excellent repository for development documentation. It's readily available within the platform, and the development teams will already have access to the application within your instance. If possible, avoid using documents that users can easily download. Consider these development documents as living documents and train your teams to come back to them over time to see the most up-to-date guidance. Managing this information in Knowledge Management can help with this.

Practitioner insight: Creating documentation is a time-consuming activity. To reduce this time, create and provide templates for documentation for your developers so they can focus on the important content.

Development best practices

ServiceNow has defined technical best practices for you. Refer your developers to the ServiceNow-managed documentation to avoid duplicating this information. Additionally, write policies that require all developers to review and become familiar with the documentation. Because we add new ServiceNow documentation and update it regularly, add how often developers should review ServiceNow technical best practices.

Practitioner insight: During code reviews, require reviewers to check work against current development best practices as part of every peer review.

Customization and configuration guidelines

You can customize and configure the Now Platform to modify how your organization uses and/or interfaces with it to meet your specific needs. The flexibility of the platform is one of its greatest strengths, but it's important to control how customization and configuration are managed to avoid technical risk and complexity, which can result in costly technical debt and hinder your future upgrades.

Practitioner insight: Avoid confusion by following strict definitions for customization and configuration. Here's how ServiceNow defines these terms:

- **Customization –** A customization is any change to code that is part of the baseline installation of a ServiceNow instance. For example, when:
  - A product has new requirements to add additional fields to gather data on a form
  - Someone builds a new custom feature into the product (such as custom applications, third-party widgets, etc.)
  - New tables are added to the platform
  - Any modifications to out of the box (OOTB) global glide properties are made

- **Configuration –** A configuration is tailoring an instance using ServiceNow to meet your requirements without making changes to the code that's part of the baseline installation of an instance. For example:
  - When an incident is submitted, the incident must be routed to the correct assignment groups. You can configure an assignment rule to provide this functionality.
  - Many applications have properties that you can modify that affect the behavior of the application or you can enable additional behavior to provide more functionality.

The first thing you should consider when you define your organization's customization and configuration guidelines is how you'll decide when it's appropriate to customize in the first place. A ServiceNow demand board should be responsible for making these decisions, but they'll rely on the technical policies and guidance you craft to make informed decisions on when to invest in customizations.

As an example, we recommend developing a template like the one shown here that your demand board can use to assess the complexity and risk of customizations. This allows them to effectively compare the risk against the potential value of customization requests.

SCENARIO	COMPLEXITY SCORE	WHAT MINIMUM LEVEL OF BUSINESS VALUE SHOULD CUSTOMIZATION DELIVER?
Modification to form layout or design	LOW – This will typically not delay upgrades or new functionality, although poorly planned layouts may disrupt the user experience.	Low
Add fields and/or UI policies to forms	LOW – This will typically not delay upgrades or new functionality, but over configuration of forms (e.g., adding 50+ fields) risks both the service experience and usage.	Low-Medium
Build a simple custom integration	LOW – Integrations that can use IntegrationHub (see below) will not typically delay upgrades or new functionality.	Low
Extend an existing table (e.g., cmdb_ci) with new fields only	LOW – This will typically not delay upgrades or new functionality, but additional fields should be strictly tied to validated business requirements.	Low-Medium
Extend an existing table (e.g., task) with some scripting	LOW-MEDIUM – This will typically not delay upgrades or new functionality, but additional fields should be strictly tied to validated business requirements. Complexity is dependent on the extent of the scripting.	Medium
Extend an existing table (e.g., task) as the basis for a different application	LOW-MEDIUM – This will typically not delay upgrades or new functionality, but additional fields should be strictly tied to validated business requirements. For new applications, this should be preceded by clear process mapping and acceptance testing among knowledgeable users.	Medium
Change the state choice list (e.g., modify the incident process)	LOW-MEDIUM – This may affect time to upgrade or new functionality if the modification changes underlying business logic, rules, and/or policies.	Medium
Build a new scoped application	MEDIUM – This will typically not delay upgrades or new functionality, but scoped applications should be strictly tied to validated business requirements. The complexity is dependent on the extent of the scripting.	Medium
Build a new global application	MEDIUM-HIGH – This may affect time to upgrade due to testing or new functionality if the application changes underlie business logic, rules, and/or policies.	Critical
Change baseline business rules (e.g., modify the SLA process)	HIGH – This may affect time to upgrade due to testing or new functionality.	Critical/Mandatory
Build complex custom integration	HIGH (DEPENDENT ON COMPLEXITY OF INTEGRATION) – This may affect time to upgrade due to testing or new functionality.	Critical/Mandatory
Use the incident table as is for a different use	High – This may prevent use of baseline products in the future. Use of the incident table outside its intended use case (e.g., ITSM) should be preceded by clear process mapping and acceptance testing among knowledgeable users.	Critical/Mandatory

This template may work for your organization but you'll probably need to tailor it to best suit the context and preferences of your organization. Consider these questions when you either modify this template for your own use or when you build your own:

- What demands for customization should we expect to receive?
- How comfortable is our organization with risk? This will inform how you label the complexity and risk of specific demands that the demand board may receive.
- Who would be responsible for implementing approved customizations? Does this change how we want to evaluate the complexity and risk of demands?
- Are there any customization requests that we never want to fulfill? Are there any that should always be accepted?

Once you've structured how to decide on when customizations and configurations are appropriate, consider how your organization will manage them once they're complete. Consider these questions as you define policies for how to manage the customizations and configurations that your organization decides to invest in:

- Who is responsible for managing customizations and configurations?
- How will you track customizations and configurations made in your platform?
- How will you manage these customizations and configurations so they continue adding value? How will you compare them against new functionality delivered in ServiceNow releases?
- What information do you need to document to support the management of customizations and configurations? Consider recording the:
  - Reason for customization, that is, its business justification
  - Planned review date
  - Approver of customization

Additional customization resources

- Customization best practices for ServiceNow

Development process flow

The development process flow defines how teams will conduct development and defines policies for the activities that must occur during each phase of development. Creating and

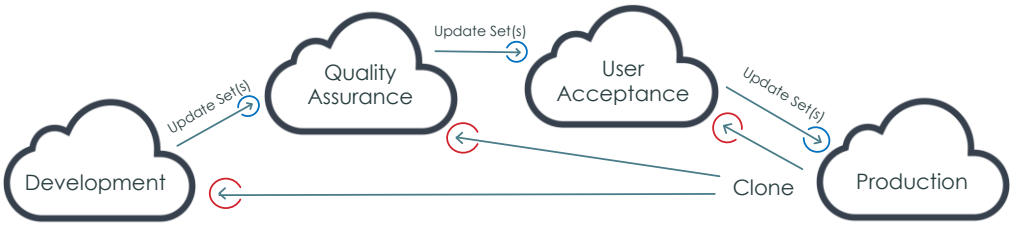
following your development process flow leads to high-quality development that meets requestors' expectations.

Begin by defining phases for development-related activities and include testing phases necessary to validate the work. The next table represents phases that are typical within a development process flow. Most of the phases listed represent different types of testing that's performed during the development process. Though not all phases are required for every organization, it's important that you find the right mix of testing to create a quality product that meets your organization's existing standards.

Phase	Description
Build	In the build phase, the teams complete the development or configuration work to match the acceptance criteria defined.
Unit testing	The developer performs unit testing as an initial check to determine if the development matches the story's acceptance criteria.
Peer testing	A second developer performs peer functional testing on the defined unit tests and evaluates the results to determine if the development meets the story's acceptance criteria. Complete peer testing as a quality check before you pass the functionality to the business for systems integration testing (SIT) and user acceptance testing (UAT).
Systems integration testing (SIT)	You'll complete SIT when you need to test a complete system or an application along with other systems or applications. Perform the level of SIT and specific tests that were defined within the intake/demand process. The SIT level and tests are also an input into the development process.
User acceptance testing (UAT)	Ask the business members who requested the functionality to perform UAT. You can also include testing the end-to-end process of the new functionality within the application that it supports. Once the functionality passes UAT, you have the final signoff that the functionality developed meets the acceptance criteria defined in the story.

Once you've defined each phase that's necessary for your development process flow, align each of the phases with the instance that phase should occur in. To do this, you'll need to know what your instance structure is, including how many instances there are in your stack and what those instances are used for. See our ServiceNow environment management document if you haven't already defined your instance structure.

As an example, consider this four-instance stack structure:



Within this example four-instance stack structure, you might align your development phases to the instances where they should occur according to this table.

Phase	Instance to conduct
Build	Development
Unit testing	Development
Peer testing	Development
Systems integration testing	Quality assurance
User acceptance testing	User acceptance

Note that multiple phases can occur within the same instance.

Practitioner insight: In large environments, you can use a sandbox instance in the development process flow so you can prototype or prove out a concept without fear of impacting routine development activity. But keep in mind that, because of the nature of prototypes and POCs, you might create artifacts that you don't want to migrate into your production stack. To avoid this, once you've created and proven a concept in the sandbox environment, make sure you redevelop the functionality in the development environment rather than migrating the update sets.

Once you've assigned each phase to the instance where teams will perform the work, define the phase requirements. These requirements establish what your teams must complete as part of each phase and defines the conditions they must satisfy before the next phase can begin.

This table shows the common requirements by phase:

Phase	Phase requirements
Pre-development	• Acceptance criteria is defined for each story. • Application architecture is defined. • SIT requirements and test cases are defined, if necessary. • User acceptance tests are defined and participants are identified. • Documentation requirements are defined for the developers. – Examples: code commenting, technical debt documentation, customization description, testing documentation, deployment etc.
Build	• The standards and best practices for ServiceNow development are followed. • The architectural design defined before development is adhered to • All applicable guiding principles and issues are raised to the architecture review board or appropriate technical governance when necessary. • All activities are documented as they were outlined in documentation requirements defined before development
Unit testing	• The developer has conducted all unit tests and all of them passed. • The developer has determined that the functionality created meets the acceptance criteria defined with the story • All unit test results are documented.
Peer testing	• A second developer conducts all unit tests and all of them passed. • A second developer has determined that the functionality created meets the acceptance criteria defined with the story. • All unit test results are documented.
Systems integration testing	• Testers execute all defined SIT test cases and document the results. • All defined SIT test cases pass successfully.
User acceptance testing	• The business users/requesters execute all defined test cases and document the results. • All defined UAT test cases pass successfully.

Practitioner insight: Create a robust testing strategy using tools from ServiceNow. Automated Testing Framework (ATF) can improve your testing efficiency because it automates functional testing of ServiceNow applications so they'll work as expected when you introduce changes. ATF has over 500 quick start tests to accelerate adoption. To make sure you complete testing thoroughly and accurately, use these ATF features to cover as much testing as possible, then use Test Management on the Now Platform to track and manage the test plans and test cases associated with each of your testing phases.

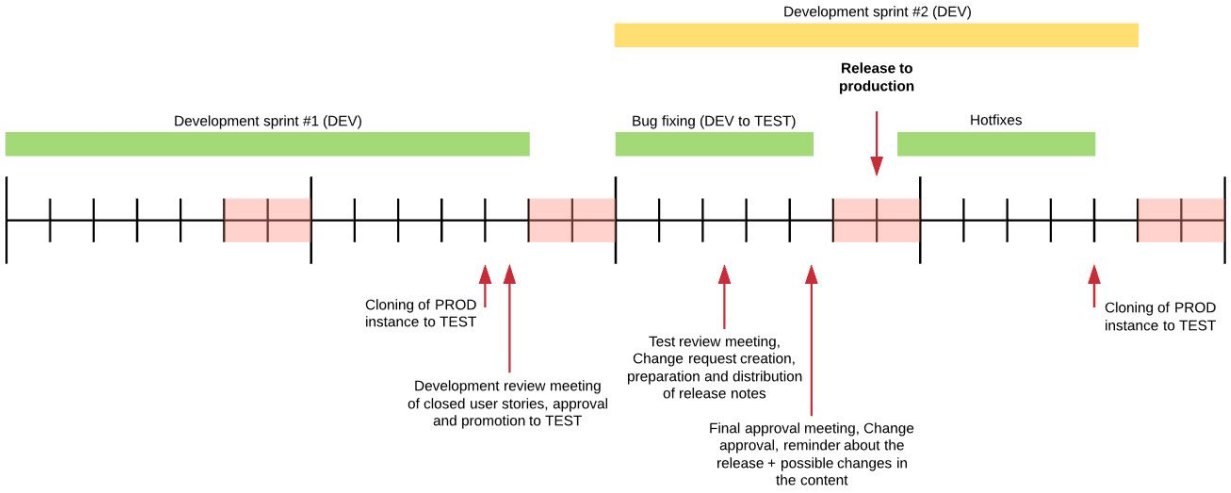
Practitioner insight: HealthScan can help you identify development mistakes and prevent them from impacting your instance health. If you're actively engaged with ServiceNow Customer Outcomes, ask your representatives if these services are available to you and determine how you can best use them as part of your development lifecycle:

- **HealthScan update set scanning –** Update set scanning assists with scanning one or more update sets related to a user story.
- **Sprint scans –** You can use sprint scans during the build or testing phases to make sure your teams are following best practices before you move to production.

Release management process

Release management is the process of planning, scheduling, and controlling the build, testing, knowledge transfer, and deployment of releases to an organization's ServiceNow instance. Release management gives you the assurance that any new release being deployed to an organization's ServiceNow instance delivers the service or functionality required by the business, while protecting the instance's integrity.

Start by defining how your high-level release management model will work. Here's an example release model for a release with a two-week cadence that aligns with two-week sprints:



Note: Most ServiceNow deployment uses a hybrid project management model with a large initial release followed by agile methodology. In this case, the diagram above would represent the release cycle after the initial release.

Practitioner insight: Consider how your organization's development methodology—agile, waterfall, or hybrid—and pace will impact your release process. The example release process shown depicts an agile release cycle. Your process may look very different, especially if your organization is running waterfall. What's important is to consider what the release timeline needs to look like and then map it out so you have visibility into the process.

While developing your release management process, consider defining these areas.

Area	What needs to be defined
Release model	• What steps do we need in our release model to align with how development works at our organization? • What review and approval meetings will we need to organize ahead of planned releases? The example above includes three meetings: a development review meeting, a test review meeting, and a final approval meeting. • How will we plan to avoid blackout windows? • Can we manage multiple release streams? If so, how will we coordinate them? • Will our normal platform release cycle vary from scoped application releases? Will that have a different CI/CD release process from the application repository?
Release frequency	• How frequently will we schedule releases? How do we sync release schedules with development sprints? • How will we bundle releases? ServiceNow recommends that: - All projects or bundled changes that affect the ServiceNow instance (releases containing multiple enhancements and/or fixes) go through the release management process - All projects and bundled changes have a completed change record with the appropriate approvals obtained for implementation - Whenever possible, bundle changes to an existing service, application, or functionality together and release them on a regular basis using the release management process. • Will high-priority changes be released outside of standard our release frequency?
Release ownership	• How will we assign ownership for releases? We recommend identifying a single owner for every release who will be responsible for the successful coordination and execution of the release, as well as making sure all required documentation related to the release exists.
Release testing	• How will we test and verify releases before implementation? • Will there be testing timelines to meet before each scheduled release? • Will we use automated testing, like the Automated Test Framework?
Release documentation	• How will we document and distribute release notes? The best practice is to distribute release notes two or three business days before the actual release. This will give people enough time to get familiar with coming changes. • What do we need to document for each release? We recommend that you document all release deliverables and make sure you transfer the knowledge.
Implementation and validation	• What process should we follow to make sure that all implementation work on a release is completed by the planned end dates and times? • How will we validate that releases have been completed successfully? What post-release testing is required?
Release scheduling	Release to production should happen outside of the peak hours to minimize impact to the users. To find the best day of the week to release to production, consider these questions: • How often will a release contain changes that impact many users? How will we prepare for the change and train them? • If there's an issue in production, when will users be able to detect it? Do users work over weekends or are they more likely to discover as issue the next business day? • Are the developers available for hotfixes during weekend?
Rollback strategy	• What will we do if an update set shows collisions during the preview? • How will we respond to incidents that result from a recent release? • When will we fix forward by deploying a new hotfix update set rather than backing out of an update set?

Practitioner insight: A common release management approach is to publish a new release during the weekend because to avoid regular business hours. Considering the release scheduling questions above, releases over the weekend may bring some challenges:

- Developers must be available during the weekend to react on critical issues if they happen.
- Since users don't work during the weekend, issues may not be detected until Monday morning. Because Monday mornings are typically when you'll see the heaviest usage (since workloads build over the weekend), issues may result in major incidents.

To avoid such situations, some release models schedule the release to production late on Monday evenings. This gives users time to work with known features during the peak times that occur on Monday mornings. Developers also have higher availability during this time than over the weekend. This improves your ability to react to issues and reduces the impact of any incident that might occur.

3. Who should be involved in defining ServiceNow development management?

Your ServiceNow platform owner and ServiceNow technical governance board should oversee the definition of effective development management practices. This board is ultimately accountable for ensuring development management policies are drafted and approved. However, your technical governance board will likely need to enlist support from other groups and/or roles to define development management policies.

Consider involving these roles when you define your ServiceNow development management if they're not already participants on your technical governance board.

Role	Description
Enterprise architect (EA)	The EA creates a single language between people, processes, and technology that allows your teams to seamlessly share and enrich data across business functions. You can also consult the EA when you define your development management policies, but if you don't, make sure you at least inform them of this work.
Platform owner	The platform owner is typically accountable for managing the creation of all development management policies.
Platform architect	The platform architect is typically responsible for the creation of development management policies while consulting members of the organization, including EAs, security administrators, and development leads/SMEs, ServiceNow architects, and process owners.
Security administrator	Security administrators are consulted when the development management policies are created to confirm that the development guidelines appropriately safeguard against unnecessary risk.
Development leads/SMEs	Consult your development leads/SMEs when you define the development management policies because the output of these activities will affect the development flow and process.
IT operations lead(s)	Consult your operations leads when you define development management policies to identify which management activities are feasible and how they'll be done on a recurring basis.

If your organization doesn't have a ServiceNow technical governance board, see our resource on getting started with ServiceNow governance.

4. What's the starter or minimum viable approach to ServiceNow development management?

Ideally, ServiceNow development management should define the development standards, guidelines for customization and configuration, development process flow, *and* release management process.

That's a lot to get right from the start. We recommend starting with a minimum viable policy that defines the most important standards and controls for your specific needs first—instead of trying to define everything all at once.

Take these steps to get started:

1. Assign accountability for managing the development of ServiceNow management policies at your organization to your ServiceNow platform owner and, if it's established, your ServiceNow technical governance board.
2. Work with your platform owner to select one to three of the policy components listed in section 2 (for example, "development standards") that you think are the most urgent and important to enact at your organization. In our experience, at least development standards and development process flow practices are most important to include in a minimum viable policy.
3. Consult with your technical governance board to approve of the components you selected. If necessary, invite SMEs to help your technical governance board consider what's needed in your initial ServiceNow development management process.
4. Build out specific policy standards for your selected components.
5. Make all stakeholders aware of the new polices, train them on them, and set the expectation that they must adhere to them.
6. Expand on your minimum viable approach to include standards for the remaining components in section 3.

5. How should I define specific ServiceNow development management policies?

While this Success Insight covers what to consider when you define ServiceNow development management, you'll need to define the specific policies and standards required at your organization. Account for any special needs required by:

- **Industry –** Companies operating in highly regulated industries (government, finance, and healthcare, for example) will likely require management practices and policies that more strictly enforce platform standards and controls.
- **Existing enterprise governance and/or platform management standards at your organization –** Many organizations already have defined practices for how to manage and maintain IT platforms. Your ServiceNow development management practices will need to align with (and usually be subordinate to) any existing enterprisewide policy at your organization.
- **IT architecture –** Your IT architecture may impact how you need to manage the Now Platform. Specific policies should account for how your IT environments are designed. This is especially true if ServiceNow is highly integrated with other systems.

When possible, we recommend working with your partner or with ServiceNow platform architects to define specific ServiceNow development management practices and policies that will work for your organization. Reach out to your account manager to connect with ServiceNow architects to help with this.

Additional resources

- ServiceNow governance resources on the Customer Success Center
- Get started with ServiceNow governance
- Define ServiceNow technical governance policies
- Defining ServiceNow governance golden rules
- Define ServiceNow data governance
- Govern your ServiceNow environment
- Define ServiceNow platform management practices
- Manage application development on the Now Platform

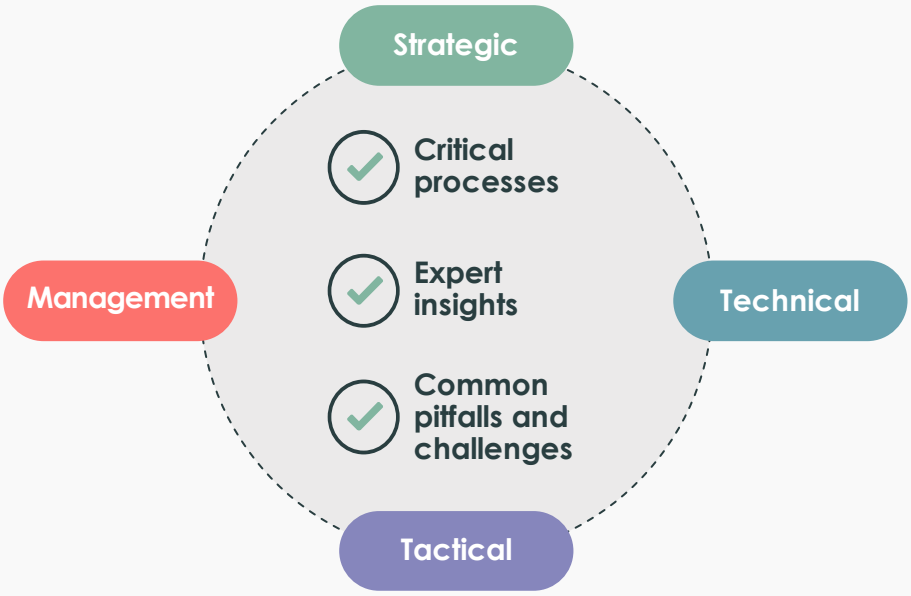
If you have any questions on this topic or you would like to be a contributor to future ServiceNow best practice content, please contact us at best.practices@servicenow.com.

servicenow

Customer Success Best Practices

ServiceNow's Best Practice Center of Excellence provides prescriptive, actionable advice to help you maximize the value of your ServiceNow investment.

Definitive guidance on a breadth of topics



Designed for:

- Executive sponsors
- Platform owners and teams
- Service and process owners

Created and vetted by experts

Best practice insights from customers, partners, and ServiceNow teams

Based on **thousands of successful implementations** across the globe

Distilled through a **rigorous process** to enhance your success

Proven to help you transform with confidence

- Practical
- Actionable
- Value-added
- Expert-validated

Get started today.

Visit Customer Success Center.

Contact your ServiceNow team for personalized assistance.