Best Practices for Code Control, Versioning, Governance, and Collaboration

'ike (knowledge) level: Advanced

Rev. February 5, 2025

Introduction ↘

As organizations expand their use of ServiceNow, managing development across multiple teams becomes increasingly complex. Whether you're working with internal teams or collaborating with external partners, coordinating multiple developers on the same platform can create significant challenges. Common problems include code collisions, conflicting updates, and inefficiencies in version control and deployment. These challenges can hinder progress and cause delays if not addressed properly.

In this white paper, we'll discuss the key challenges involved in managing multiple ServiceNow development teams, including external teams, and provide actionable best practices for version control, governance, and code management. We'll also include practical recommendations on release management strategies, and how ITIL best practices for release management can help streamline development and deployment.

Challenges of Managing Multiple Development Teams in ServiceNow ↘

1. Code Collisions and Conflicts

Multiple teams working on the same ServiceNow objects or components can lead to conflicts when changes are made simultaneously. If different developers unknowingly overwrite each other's changes, it can lead to broken functionality, downtime, or unnecessary rework.

2. Lack of Version Control

Without version control, it's difficult to keep track of what changes were made, by whom, and when. This can cause confusion when multiple developers are working on the same component. If issues arise, it becomes time-consuming to roll back to a previous stable version.

3. Inconsistent Workflows

Development teams, especially when external partners are involved, may have different approaches to coding, testing, and deployment. This lack of consistency can cause delays, code duplication, or even create a fragmented user experience.

4. Environment Synchronization Issues

ServiceNow has multiple environments, such as development, test, and production. Managing code migration between these environments can get tricky when multiple teams are working in different instances, leading to integration problems and inconsistent configurations across environments.

5. Governance and Compliance

For organizations in regulated industries, there are strict requirements around documentation, approvals, and audit trails. If these governance processes aren't followed properly, there could be risks of non-compliance or security vulnerabilities.

Best Practices for Managing Multiple Development Teams ↘

1. Implement Git-Based Version Control

Best Practices for Code Control, Versioning, Governance, and Collaboration

'ike (knowledge) level: Advanced

Rev. February 5, 2025

Version control ensures that all changes are documented, tracked, and available for easy collaboration. This prevents code collisions and provides a way to track and manage versions across multiple developers.

Key Considerations:

- **Repository Structure:** Each ServiceNow module or application should have its own Git repository.
- **Branching Models:** Set up a clear branching strategy (e.g., feature, release, hotfix) to minimize conflicts.

Tools:

- **GitHub, GitLab, Bitbucket:** Platforms that manage repositories and track changes.
- **ServiceNow Git Integration:** ServiceNow's built-in Git integration for managing code deployments.

2. Code Review and Merge Processes

Code reviews should be mandatory for all changes. This improves quality, ensures consistency, and helps prevent bugs or issues from making their way into production.

Key Considerations:

- **Peer Reviews:** All code changes should undergo peer review before merging.
- **Automated Static Code Checks:** Implement static analysis tools to automatically detect coding errors, security issues, or poor coding practices.

Tools:

- **SonarQube:** For static code analysis to check security vulnerabilities and performance issues.
- **ServiceNow Code Review:** Use ServiceNow's built-in code review features to manage approval workflows.

3. Naming Conventions and Documentation

Establishing naming conventions and maintaining clear documentation across teams ensures consistency and makes code easier to maintain and troubleshoot.

Key Considerations:

- **Standardized Naming Conventions:** Define naming standards for scripts, fields, tables, and other ServiceNow elements.
- **Documentation:** Provide thorough documentation for customizations, configurations, and processes.

Example Naming Conventions:

- **Tables:** u_<module>_<feature>, e.g., u_incident_management_case
- **Scripts:** u_<module>_<script_type>, e.g., u_ticket_creation_script

Tools:

- **Confluence or SharePoint:** Use for centralized documentation, where all teams can collaborate and share development guidelines.

4. Effective Use of Update Sets

Update sets are a key tool in ServiceNow for moving customizations across instances. However, improper use can result in conflicts, redundancies, and errors.

Key Considerations:

- **Separation of Work:** Ensure that different teams do not overlap on the same update sets.
- **Track Dependencies:** Regularly check that dependencies between update sets are managed properly to avoid issues.

Tools:

ServiceNow Update Sets: Built-in tool to track, review, and deploy customizations.

5. Linting Tools for Code Quality

Implementing automated linting ensures code follows established best practices, improving readability, consistency, and security.

Key Considerations:

- **Enforce Code Standards:** Use linting tools to check for styling and syntax issues.
- **Integrate with CI/CD Pipeline:** Automate linting in the deployment pipeline to catch issues early.

Tools:

- **ESLint:** For JavaScript linting to check for errors and enforce coding standards.
- **ServiceNow Linter:** ServiceNow provides a built-in linter to automate the process.

6. Governance and Change Management

Establishing governance processes ensures that all changes are documented, reviewed, and approved before being deployed. This is crucial for maintaining compliance and minimizing errors.

Key Considerations:

- **Change Approval:** Use change management processes to control code deployment.
- **Audit Trails:** Keep a comprehensive record of all changes, approvals, and deployments.

Tools:

- **ServiceNow Change Management:** Manage change requests, approvals, and deployments.
- **ServiceNow ALM (Application Lifecycle Management):** Track and manage applications and their lifecycle stages.

Approaches to Releases: ITIL Release Management Best Practices

Best Practices for Code Control, Versioning, Governance, and Collaboration

'ike (knowledge) level: Advanced

Rev. February 5, 2025

Effective release management is critical for ensuring that new features, fixes, or changes are delivered smoothly and with minimal risk to the production environment. In ServiceNow, following ITIL best practices for release management can help organize and streamline the release process.

1. Release Planning and Scheduling

Release planning should start early in the development process and be updated regularly to ensure a smooth rollout. Planning includes defining what will be included in the release, setting deadlines, and coordinating across teams.

Key Considerations:

- **Release Window:** Define a clear release window that works for all teams and minimizes downtime.
- **Impact Assessment:** Assess the impact of the release on various teams, departments, and stakeholders.

Tools:

- **ServiceNow Change Management:** Integrate your release management process with ServiceNow's Change Management to ensure releases are planned, tracked, and approved appropriately.

2. Automated Deployment with CI/CD

To minimize errors and streamline deployment, automate the deployment process through Continuous Integration and Continuous Deployment (CI/CD) pipelines. This reduces the risk of human error and accelerates the release cycle.

Key Considerations:

- **Automated Testing:** Ensure automated tests are run before deploying new code to minimize the risk of introducing bugs.
- **Deployment Automation:** Automate the deployment process using CI/CD tools to push code to production with confidence.

Tools:

- **Jenkins, Azure DevOps:** Integrate with ServiceNow for automated deployments.
- **ServiceNow ATF (Automated Test Framework):** Use to run unit tests and ensure functionality before release.

3. Release Communication and Documentation

Clear communication and documentation are essential to ensure that all teams are on the same page regarding release changes, timing, and impact.

Key Considerations:

- **Internal Communication:** Ensure that all stakeholders are informed of the upcoming release, changes, and potential downtime.

- **Post-Release Documentation:** Document any known issues, fixes, or new features that were included in the release.

Tools:

- **ServiceNow Release Management:** Track and document each stage of the release process, from development to deployment.

4. Rollback Procedures

Have a rollback plan in place to address any issues that arise post-release. This plan should clearly define steps to revert changes if something goes wrong, ensuring minimal impact on the business.

Key Considerations:

- **Backup and Restore:** Regularly back up your instances so you can quickly restore to a previous state if needed.
- **Version Control:** Ensure that code versions are stored in a version control system like Git to facilitate quick rollbacks.

Tools:

- **ServiceNow Instance Management:** Manage and backup your ServiceNow instances to facilitate easy restoration.

5. Post-Release Support and Monitoring

Once the release is deployed, it's important to monitor the system for any issues and provide ongoing support to resolve them quickly.

Key Considerations:

- **Monitor System Health:** Use ServiceNow's built-in monitoring tools to track system performance after the release.
- **Feedback Loops:** Collect feedback from stakeholders and users to identify any issues or improvements needed.

Tools:

- **ServiceNow Performance Analytics:** Monitor system performance and gather insights post-release.
- **ServiceNow Incident Management:** Manage post-release incidents and resolve issues efficiently.

Leveraging Automated Testing Framework (ATF) to Streamline Development Across Multiple Teams ↘

When multiple development teams are involved in building and maintaining a ServiceNow instance, testing can become a major challenge. The good news is that ServiceNow's Automated Test Framework

(ATF) can be a game-changer in ensuring consistent, fast, and reliable testing, particularly when teams are working on different parts of the platform.

In this section, we'll dive into how ATF can help streamline the testing process, reduce manual effort, and minimize risks when multiple development teams are involved. We'll also explore how using ATF as part of your overall development and release management strategy can significantly improve the quality and speed of your projects.

Why ATF Is Key for Multiple Development Teams

1. **Consistent Testing Across Teams** When different teams are working on various parts of the ServiceNow platform, there's always a risk of inconsistent testing processes. One team may follow a different approach than another, or certain tests may get overlooked entirely. ATF solves this by automating testing, so that every developer on every team is running the same tests to ensure the code works as expected. This helps make sure that no matter who's making the changes, everything gets tested in a standardized way.

For example: Imagine one team creates a custom table, and another team integrates it with an external system. With ATF, automated tests can ensure that all parts of the system interact properly without the need for extensive manual testing.

2. **Faster, More Efficient Testing** In a project with several development teams, getting fast feedback is crucial. Manual testing can take up a lot of time, especially when multiple teams are making changes and pushing code to different environments. With ATF, testing happens automatically after each change, and developers get immediate feedback on whether their code is working. This speed allows for faster iteration and more efficient development cycles.

For example: If an update set is deployed, ATF can run tests automatically to verify that no existing functionality was broken by the changes, saving time and reducing the chance of errors slipping through.

3. **Reduced Manual Testing Workload** Manually testing every change takes up time and resources, especially when teams are working across different modules of ServiceNow. ATF frees up valuable resources by automating the testing process, allowing your quality assurance team (or even the developers themselves) to focus on more complex, higher-priority tasks. This ensures quicker turnarounds without sacrificing quality.

For example: If external vendors are working on specific features or integrations, ATF can automatically run tests to ensure these additions don't break existing functionality, saving you the trouble of having to manually verify every change.

4. **Increased Collaboration and Visibility** When multiple teams are involved, it's easy for things to get siloed. But with ATF, all testing results are stored in ServiceNow, making it easy for all teams to see which tests have been passed or failed, and to monitor the overall health of the system. This transparency makes collaboration easier and ensures everyone is on the same page when it comes to the quality of the code being deployed.

For example: A project manager or product owner can review test results directly within ServiceNow to see where there are issues, helping teams quickly address problems before they escalate.

5. **Seamless Integration with CI/CD Pipelines** To ensure that your code is always ready for deployment, it's essential to automate testing as part of your continuous integration/continuous deployment (CI/CD) pipeline. ATF can easily be integrated into this process, ensuring that tests are triggered automatically whenever code changes are pushed to the repository. This gives teams the confidence that their code is stable and ready for release.

For example: With integration into a CI/CD pipeline (such as Jenkins or Azure DevOps), teams can set up ATF to automatically run tests when new code is committed, ensuring that no bugs or regressions make it into production.

6. **Cost-Effective Quality Assurance** As your ServiceNow environment grows, testing becomes more complex. Manually testing each new feature or fix is time-consuming and expensive. With ATF, you can automate much of the testing process, ensuring that quality is maintained without the need for extensive manual testing resources.

For example: If you're managing a large-scale ServiceNow instance with multiple external development teams, ATF can automate functional and regression testing, reducing the cost and time spent on quality assurance while maintaining a high standard of testing.

Best Practices for Using ATF with Multiple Teams

1. **Define Standardized Test Cases for All Teams** To ensure consistency, create a library of standardized test cases that can be used across all teams. This helps maintain uniformity in testing, regardless of which team is responsible for which module or feature.

2. **Test Early and Often** Start using ATF as early in the development cycle as possible. The sooner you automate your testing, the earlier you can catch issues, which prevents bugs from snowballing into larger problems later on. Running tests as part of your CI pipeline ensures that every commit is verified.

3. **Modularize Your Test Suites** When working with multiple teams, it's a good idea to break your test cases into smaller, reusable modules. This allows different teams to test their individual components independently, while still ensuring that all tests are reusable in the context of larger, end-to-end tests.

4. **Run Full System Tests Before Each Release** Once all individual components are tested, run full, end-to-end tests using ATF before deploying to production. This ensures that all the pieces of the system work together seamlessly and that nothing is overlooked.

5. **Monitor and Address Test Failures Quickly** Make sure your teams are alerted to failed tests as soon as possible. The quicker you address test failures, the less likely it is that small issues turn into larger, more complicated problems.

6. **Test for Different User Personas** ServiceNow is used by a variety of personas (e.g., end users, managers, system admins). Ensure that your ATF tests simulate actions from each type of user to validate the experience across the system. This guarantees a consistent user experience for everyone.

7. **Integrate ATF with Incident Management** Automatically generate incidents from failed tests within ServiceNow. This makes it easier to track and manage issues, ensuring nothing is missed and that all problems are addressed systematically.

ATF Cautions and Limitations ↘

While ServiceNow's Automated Test Framework (ATF) offers numerous benefits, such as automated testing for faster feedback and improved quality, it does have some limitations that organizations should be aware of when considering its use. Here are some key limitations:

1. Complexity of Custom Tests

- **Limitation**: ATF works well for standard processes and out-of-the-box features, but it can become challenging to set up and maintain custom tests for complex workflows, particularly if those workflows involve many custom scripts, integrations, or third-party applications.
- **Impact**: For highly customized ServiceNow instances, creating and maintaining reliable automated tests for every unique configuration or integration can be time-consuming and may require specialized expertise.

2. Limited Support for End-to-End Testing

- **Limitation**: While ATF excels at testing individual components or features, it can struggle with performing comprehensive end-to-end tests, especially when testing cross-module functionalities or workflows that span across multiple applications in ServiceNow.
- **Impact**: For complex, cross-functional processes or large implementations that require extensive end-to-end validation, ATF may not be sufficient to cover all scenarios, requiring additional tools or manual testing.

3. Not Fully Automated for All Use Cases

- **Limitation**: ATF can automate functional testing but is not fully automated for all use cases, particularly in the areas of performance testing, load testing, or security testing. It also doesn't fully automate tests for user interface (UI) responsiveness, accessibility, and other non-functional requirements.
- **Impact**: Teams may still need to rely on third-party tools or manual processes to validate performance, security, and accessibility, adding extra overhead and complexity to the testing process.

4. Difficulty Testing Complex User Interfaces (UI)

- **Limitation**: ATF has limited capabilities when it comes to testing complex UI elements, such as dynamically rendered content, custom widgets, or integrations with external systems where the UI needs to react in specific ways based on user interactions or data.
- **Impact**: For projects with a heavy focus on custom UI components, ATF may not fully cover all testing scenarios, which can lead to gaps in quality assurance.

5. Integration with External Systems

- **Limitation**: ATF is primarily designed to test ServiceNow functionality within its own environment. While it can simulate interactions with external systems, complex integrations with third-party applications or external APIs can be difficult to test fully within ATF.
- **Impact**: Testing the full flow of integrations with other systems may require additional tools or manual intervention, leading to potential gaps in testing coverage and increased testing overhead.

6. Test Maintenance Overhead

- **Limitation**: As ServiceNow updates its platform, there may be changes that affect existing test cases, especially for workflows or functionality that relies on out-of-the-box components. This means that organizations will need to regularly maintain and update their tests to keep them in sync with platform updates and new features.
- **Impact**: Test maintenance can become a significant effort, particularly for organizations with extensive customization or those that undergo frequent platform upgrades.

7. Performance Overhead

- **Limitation**: Running automated tests can add some overhead to the system, especially if multiple tests are executed simultaneously or if tests are resource-intensive. Large test suites may impact the performance of the instance during test execution.
- **Impact**: Running a large number of tests or complex scenarios can lead to longer test execution times or resource consumption that may affect the performance of the environment during testing, particularly in production or pre-production environments.

8. Limited Support for Non-Functional Requirements

- **Limitation**: ATF is primarily focused on functional testing. It has limited or no out-of-the-box support for testing non-functional requirements like load testing, stress testing, security testing, or compliance testing.
- **Impact**: To ensure that the application is scalable, secure, and compliant with regulatory requirements, organizations will need to supplement ATF with third-party testing tools or manual processes.

9. Learning Curve

- **Limitation**: ATF requires a solid understanding of the ServiceNow platform, scripting, and test automation concepts to create effective tests. For teams with limited experience in ServiceNow development or automated testing, the learning curve can be steep.
- **Impact**: Organizations may need to invest time and resources in training team members or hiring specialists to maximize the effectiveness of ATF, which can delay the adoption of automated testing.

10. Limited Reporting and Analytics

- **Limitation**: While ATF provides basic reporting and test result tracking, it may not offer the depth of reporting capabilities that larger organizations require. More advanced analytics, such as visualized trends, failure root cause analysis, or detailed test metrics, may require custom development or integration with third-party tools.
- **Impact**: Lack of advanced reporting and analytics may make it harder for teams to identify patterns or areas for improvement across large test suites or multiple development teams.

Using ServiceNow's Automated Test Framework (ATF) is a powerful way to streamline testing when multiple development teams are working on a project. It reduces the need for manual testing, ensures consistency and faster feedback, and helps teams catch issues early. By integrating ATF into your CI/CD pipeline and using it as part of a broader development strategy, you can maintain high-quality standards, improve collaboration, and ensure your releases are smooth and reliable. ATF is an essential tool for organizations looking to scale their ServiceNow implementations and improve their development process, especially when working across multiple teams.

Checklist for a ServiceNow Environment with Multiple Development Teams ↘

When you're managing ServiceNow development across multiple internal and external teams, consider the following checklist to ensure smooth and efficient collaboration:

1. Version Control and Branching

- Set up a Git repository for each application or module.
- Define a branching model (e.g., feature, release, and hotfix branches).
- Integrate ServiceNow with Git for seamless code tracking.

2. Code Reviews and Automated Checks

- Implement a mandatory peer review process for all code changes.
- Use tools like **SonarQube** for static code analysis and automated quality checks.
- Enforce consistent coding practices across all teams.

3. Naming Conventions and Documentation

- Establish naming conventions for all ServiceNow objects (tables, fields, scripts, etc.).
- Ensure clear and consistent documentation for all customizations.

4. Update Set Management

- Assign update sets to specific tasks or features to avoid overlap.
- Track and validate update sets before deployment.

5. Linting Tools

- Set up automated linting tools to enforce code quality standards.
- Integrate linting into the CI/CD pipeline for consistent checks.

6. Governance and Compliance

- Establish an approval process for all code changes.
- Maintain audit trails for code changes and deployments to ensure compliance.

7. Release Planning

- Plan releases early and coordinate with all teams.
- Schedule and document releases according to the ITIL release management best practices.

8. Post-Release Monitoring

- Monitor system health after release using ServiceNow's built-in tools.
- Collect and address feedback from users and stakeholders.

Conclusion

Managing multiple development teams in ServiceNow requires effective planning, collaboration, and governance to minimize risks and ensure smooth, reliable releases. By following best practices for version control, code reviews, naming conventions, and release management, organizations can successfully manage their ServiceNow environment and deliver high-quality results on time. Implementing ITIL best practices in release management ensures that all changes are carefully planned, tested, and communicated, while minimizing disruptions to business operations.

About Mana'o Pili ↘

Mana'o Pili is a Hawai'i based technology consulting firm specializing in business automation through ServiceNow. Mana'o Pili provides its customers with individualized solutions. We reject the notion of one-size-fits-all solutions. Instead, we partner with you to craft a tailored plan that aligns with your unique needs, budget, and objectives. Our approach focuses on optimizing your existing platform while minimizing customization and reducing technical debt.

Here at Mana'o Pili, we treat our customers as 'ohana (family), listen closely to your challenges and deliver tailored attention with exceptional service.