1. Introduction
   1. Overview of the workshop
   2. Objectives of the workshop
   3. Prerequisites for attendees
   4. Activity: Have the attendees introduce themselves and share their experience level with Git, Azure DevOps, and x++.
2. Azure DevOps Process Templates
   1. Overview of process templates
      1. Agile - https://learn.microsoft.com/en-us/azure/devops/boards/work-items/guidance/agile-process?view=azure-devops
      2. Scrum - https://learn.microsoft.com/en-us/azure/devops/boards/work-items/guidance/scrum-process?view=azure-devops
      3. CMMI - <https://learn.microsoft.com/en-us/azure/devops/boards/work-items/guidance/cmmi-process?view=azure-devops>
      4. Activity: discuss the pros and cons of each process template (Agile, Scrum, CMMI) and how it might apply to different types of projects.
   2. ~~Customizing and creating process templates~~
      1. ~~Create inherited process~~
         1. ~~To start, you need to clone the existing process that you want to use as the basis for your custom process. This is done from the Azure DevOps~~
      2. ~~Customize work items~~
         1. ~~Next, you can make changes to the cloned process to meet the needs of your organization. This can include customizing work item types, fields, states, and transitions.~~
      3. ~~Add new work items~~
      4. ~~Test~~
         1. ~~Before applying the custom process to your projects, it's a good idea to test it thoroughly to ensure that it works as expected. You can create a test project to do this.~~
      5. ~~Apply~~
         1. ~~Once you have tested the custom process, you can apply it to your projects. This can be done from the Azure DevOps project settings pag~~
      6. ~~Monitor~~
         1. ~~Finally, it's important to monitor the custom process and make any necessary updates or changes. This can be done by regularly reviewing the custom process and making changes as needed.~~
   3. ~~Best practices for process templates~~
      1. ~~Document the changes you make to process templates, so that you can track changes over time and revert back if necessary.~~
      2. ~~Test changes in a separate test organization before applying to production.~~
      3. ~~Avoid customizing the system process templates, instead create a new inherited process template that you can modify.~~
      4. ~~Avoid removing work item types and fields, as they may be referenced by other parts of the system.~~
      5. ~~Consider versioning and branching when making changes to templates, to manage multiple versions and changes.~~
      6. ~~Ensure the changes you make do not violate auditing, compliance or security requirements.~~
   4. ~~Activity: Have the attendees work together to create a custom process template and share it with the group for feedback.~~
3. What about GitHub?
   1. Most parts of what we’ll talk about today will work on GitHub
   2. NuGet packages are an issue because of availability
   3. Overall story here is developing quickly and we will hear more on this later this year
4. Source control options
   1. Overview of Git and Azure Repos
      1. Git is a distributed version control system, which allows multiple developers to work on the same codebase and track changes to the code.
         1. https://phoenixnap.com/kb/wp-content/uploads/2021/09/git-workflow.png
      2. Azure Repos is a cloud-based source code management tool that is part of Azure DevOps. It provides Git repositories for version control, as well as a web-based interface for collaboration, code reviews, and project management. Azure Repos supports both Git and Team Foundation Version Control (TFVC) for version control.
         1. https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/media/azure-devops-cicd-for-azure-web-apps.png
      3. In Azure Repos, teams can create Git repositories for their code, track changes to the code, and maintain different versions of the codebase. The tool also provides features such as pull requests, code reviews, continuous integration, and deployment (CI/CD), and other tools to help manage the development workflow. Additionally, Azure Repos integrates with other Azure DevOps services, such as Azure Boards, Azure Pipelines, and Azure Artifacts, to provide a complete DevOps solution.
         1. <https://www.edmondek.com/images/blue_green_azure_devops_app_service.png>
         2. https://learn.microsoft.com/en-us/azure/devops/pipelines/repos/media/github-structure-mapped-to-azure-devops.png?view=azure-devops
   2. TFVC – Why?
      1. If upgrading code, only supported path
         1. https://learn.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/migration-upgrade/analyze-code-upgrade
      2. Large binary files: TFVC can handle versioning large binary files better than Git, which can slow down or consume large amounts of storage when versioning large files.
      3. Rigid structure: TFVC provides a more rigid structure for version control, which may be desirable for organizations with strict compliance or regulatory requirements.
         1. https://learn.microsoft.com/en-us/azure/devops/repos/tfvc/media/branch-strategically/ic372280.png?view=azure-devops
      4. Familiarity: If your team is already familiar with centralized version control systems like then using TFVC might be a better choice as the concepts and workflow are similar.
      5. Straight forward branching and merging: For teams that require a lot of branching and merging, TFVC can provide a simpler and easier to use branching and merging model than Git.
      6. Stronger permissions model: TFVC provides a stronger permissions model compared to Git, which allows for more control over who can access and modify code in the repository.
   3. Setting up a Git repository
      1. GitHub VS Azure Devops
         1. Pipeline Syntax is different.
         2. Other than that, Git is Git.
         3. <https://acloudguru.com/blog/engineering/azure-devops-vs-github-comparing-microsofts-devops-twins>
         4. <https://levelup.gitconnected.com/ci-cd-for-azure-functions-app-by-using-github-and-azure-devops-c6ccfd0decaf>
         5. <https://www.mssqltips.com/sqlservertip/6557/azure-devops-ci-cd-using-github-repo-and-visual-studio-azure-sql-database-project/>
      2. Setup
         1. Create an Azure DevOps organization: If you don't already have an Azure DevOps organization, you'll need to create one. This can be done through the Azure DevOps portal.
            1. https://learn.microsoft.com/en-us/azure/devops/organizations/accounts/create-organization?view=azure-devops
         2. Create a project: After creating an Azure DevOps organization, you'll need to create a project. This is where you'll store your code and work items.
            1. https://learn.microsoft.com/en-us/azure/devops/organizations/projects/create-project?view=azure-devops&tabs=browser#create-a-project
         3. Install Git: In order to use Git with Azure DevOps, you'll need to install Git on your local machine. You can download the latest version from the Git website. Or you can use VS 2019. Or you can use VS 2015 or 2017 with the Git Add in.
            1. https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2019#clone-an-existing-git-repository-in-visual-studio-2019
         4. Connect to your Azure DevOps project: Once Git is installed, you can connect to your Azure DevOps project by using a Git client, such as the Git command line or a Git GUI client like Visual Studio
            1. https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2019
         5. Clone the repository: To start working with your code, you'll need to clone the repository. This will create a local copy of the code on your machine that you can work with.
            1. https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2019
         6. F&O Specific: Merge with packages local directory
         7. Push changes to the repository: After making changes to the code, you can push the changes to the repository by using a Git client. This will update the code in the Azure DevOps project.
            1. https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2019
         8. Pull changes from the repository: To get the latest changes from the repository, you can pull changes from the repository by using a Git client. This will update your local copy of the code with the latest changes.
            1. https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2019
         9. Start working with branches: If you want to work on a feature or bug fix, you can create a branch in Git. This allows you to make changes to the code without affecting the main branch.
            1. https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2019
      3. Activities
         1. Activity 2: Have the attendees set up a Git repository and practice basic Git commands in Azure DevOps.
         2. Activity 3: Create new project using Git as the source control option, initialize the repo
   4. Best practices for using Git in Azure DevOps
      1. Create a repository for each project: This will help you manage and track changes to the code more effectively.
      2. Use branching and merging: Use branches to isolate changes, and merge changes back into the main branch when they're ready.
      3. Commit often: Commit changes frequently to the repository to keep track of changes and make it easier to revert if necessary.
      4. Use descriptive commit messages: Use meaningful commit messages to describe changes and make it easier to understand the history of the codebase.
      5. Use code reviews: Have a code review process in place, where changes are reviewed by other team members before being merged into the main branch. Include in the Pull Request (PR) process.
      6. Avoid committing large binary files: Large binary files can slow down Git operations and consume storage. Store large binary files separately and track them outside of Git if necessary. Hint: use Azure Artifacts.
      7. Keep the repository clean: Regularly clean up the repository by removing old branches and cleaning up the history. We’re going to talk more about this later.
      8. Use tags to mark releases: Use tags to mark important points in the development history, such as releases, to make it easier to identify and track releases.
         1. What are tags?
         2. https://dev.to/saurabhpro/git-tags-ftw-1i77
      9. Integrate with the rest of your development process: Integrate Git with the rest of your development process, such as continuous integration and deployment (CI/CD), to automate the deployment of code changes.
      10. Educate your team: Ensure that all team members are familiar with Git and understand best practices, such as these, to ensure that everyone is working efficiently and effectively.
   5. Callout: Working with version control for database schemas and data using Azure DevOps
      1. https://www.mssqltips.com/sqlservertip/6212/sql-database-project-code-branching-in-git/
5. Source control branching strategies
   1. Git Concepts
      1. Branches
         1. A branch is a pointer to a specific commit in a Git repository. Branches allow you to work on multiple parallel versions of a repository, such as a development branch and a production branch. When you create a branch, you create a new pointer that tracks changes to the repository independently from other branches.
         2. https://nvie.com/posts/a-successful-git-branching-model/
      2. Forks
         1. A fork is a complete copy of a Git repository that is stored in a different location, typically in a different account or organization. When you fork a repository, you create a new independent repository that you can modify and use as you see fit. Forks allow you to make changes to a repository without affecting the original repository.
         2. https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/command-line-GitHub-fork-CLI-terminal-shell
      3. Pull Requests (PRs)
         1. A pull request is a feature in Git that allows users to submit changes to a repository for review and integration into the main codebase. It's a way to propose changes to a repository and have those changes reviewed by other contributors before they are merged into the main branch.
            1. https://www.atlassian.com/blog/bitbucket/5-pull-request-must-haves
            2. Create work item
            3. Create a branch
            4. Switch to that branch in V
            5. Code, commit, and test, code, and commit
            6. Create PR, optionally received feedback
   2. Overview of branching strategies
      1. Gitflow: A popular branching strategy that involves creating a master branch for production code, a develop branch for integration, and feature branches for individual features. This strategy also involves creating release and hotfix branches for handling releases and critical bug fixes.
         1. Take diagrams from <http://datasift.github.io/gitflow/IntroducingGitFlow.html>
         2. Advice on when to use and when not to use
      2. Trunk-based development: A simple branching strategy where all changes are made directly to the main branch (trunk). This strategy works well for smaller teams and projects where the main branch is frequently integrated and tested.
         1. <https://trunkbaseddevelopment.com/trunk1b.png>
         2. <https://reviewpad.com/blog/github-flow-trunk-based-development-and-code-reviews>
         3. Advice on when to use and when not to use
      3. Feature toggle: A branching strategy that involves using feature flags or toggles to enable or disable specific features in code. This strategy allows teams to work on features in parallel and release them independently.
         1. <https://launchdarkly.com/blog/feature-branching-using-feature-flags/>
         2. Advice on when to use and when not to use
      4. Forking: A branching strategy where a separate branch (fork) is created for each contributor to work on. This strategy is commonly used in open-source projects where many contributors are involved.
         1. <http://cdn.differencebetween.net/wp-content/uploads/2021/02/Fork-vs-Branch.jpg>
         2. Advice on when to use and when not to use
      5. “Task” or Topic branching: A branching strategy where a separate branch is created for each task or bug fix. This strategy helps to isolate changes and makes it easier to revert if necessary.
         1. Todo: create diagram
         2. Advice on when to use and when not to use
   3. ~~Implementing branching strategies in Azure Repos for TFVC~~
      1. ~~Activity: discuss likes and dislikes of using TFVC~~
   4. Pulling and resolving conflicts in Git
      1. Use a merge tool: Use a merge tool, such as Git's built-in merge tool or a third-party tool, to resolve conflicts effectively and efficiently.
         1. Use Visual Studio
         2. Setup merge tool from there if you don’t like the VS one – ie; WinMerge
         3. https://learn.microsoft.com/en-us/azure/devops/repos/git/merging?view=azure-devops&tabs=visual-studio-2019-git-menu#resolve-merge-conflicts-1
      2. Use the command line: Use the Git command line to resolve conflicts if you are familiar with it, as it provides more control over the resolution process.
         1. <https://learn.microsoft.com/en-us/azure/devops/repos/git/merging?view=azure-devops&tabs=git-command-line#resolve-merge-conflicts-1>
         2. More control on process
         3. Should rarely have to use unless you like the command line
      3. Rebase instead of merging: Use rebasing instead of merging if possible, as it helps to keep the commit history linear and reduces the number of merge commits.
         1. https://learn.microsoft.com/en-us/azure/devops/repos/git/merging?view=azure-devops&tabs=git-command-line#understand-merge-conflicts
   5. Implementing branching strategies in Azure Repos for Git
      1. Activity: Create work item, branch for work item, commit changes, create pull request, approve pull request
   6. Working with pull requests and code reviews in Azure Repos
      1. Have the attendees work together to resolve a simulated merge conflict.
      2. Have the attendees work in pairs to review each other's code using pull requests and code reviews in Azure Repos.
   7. Hypotheticals:
      1. Activity: Have the attendees work in small groups to create branching strategies for a fictional project.
      2. Walk through setting up each branching strategy.
6. Managing dependencies and artifacts in Azure DevOps
   1. Activity: Setup artifacts in Azure DevOps
      1. <https://learn.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/dev-tools/hosted-build-automation>
      2. <https://ariste.info/en/dynamics365almguide/azure-devops-artifacts/>
7. Build pipelines
   1. Overview of build pipelines
      1. On MB-204, MB-400 exams
      2. Definition: Build pipelines are defined using YAML or a visual editor in Azure DevOps. You can define the steps required to compile, build, and test your code, as well as configure the environment and tools required to perform these steps.
      3. Triggers: Build pipelines can be triggered automatically in response to code changes or on a schedule. You can also trigger a build pipeline manually.
      4. Stages: Build pipelines consist of one or more stages, each of which performs a specific task, such as compiling code, running tests, or deploying code.
      5. Tasks: Stages consist of one or more tasks, which are the individual steps that perform the actions required to build, test, and deploy code. There are many built-in tasks in Azure DevOps, such as tasks for building code, running tests, and deploying code, and you can also create custom tasks.
      6. Artifacts: Build pipelines can produce artifacts, such as compiled code, test results, and documentation, which can be used by other stages in the pipeline or by release pipelines.
      7. Integration: Build pipelines can be integrated with other parts of the development process, such as source control, testing, and deployment.
   2. Setting up a build pipeline
      1. Use templates from Microsoft or Community
         1. https://github.com/microsoft/Dynamics365-Xpp-Samples-Tools/tree/master/CI-CD/Pipeline-Samples
      2. Update as needed
      3. Templates are a good starting point. Learn and customize for your client / organization / needs
      4. Activity: create a build pipeline in a branch, configure policies for build pipelines
         1. <https://ariste.info/en/dynamics365almguide/configure-pipeline/>
         2. <https://ariste.info/en/dynamics365almguide/azure-devops-artifacts/>
         3. Todo: reference all content on how to do this
   3. Build Pipelines Types and Capabilities
      1. DevOps Build Service / Cloud build Service
         1. Diagram

            Description automatically generated
         2. MSFT Hosted
         3. No Database
         4. Just a compile
         5. Optionally, create deployable package
         6. No Tests
         7. Good for build and/or package
            1. Can be set to run after every commit
            2. Can be set to run as part of a PR, and after PR is complete
      2. Dedicate build Server (cloud or on prem)
         1. Diagram

            Description automatically generated
         2. Dedicated build machine
         3. Has a database
         4. Compile, create package, test, all features
         5. VM on Azure Sub, Dev VM managed like any other build VM in LCS/Azure on prem\*
         6. Good for
            1. Build, sync, and test
            2. Should only be used for activities that require the full app, like testing
            3. … or if you run out of minutes with MSFT on the free pipeline. Currently 1800 minutes for free. Every Visual Studio Enterprise subscriber in your company means you get 1 Self Hosted Pipeline for free
      3. Dynamically create a build machine
         1. Diagram

            Description automatically generated
         2. On demand build machine
         3. Has a database, can be empty, contoso data, or real data
         4. Compile, create package, test, all features
         5. VM on Azure Sub for Dev/Test, Dev VM created on demand from template. Template needs to be updated periodically so builds are on semi current version
         6. <https://ariste.info/en/2021/02/azure-devtest-labs-build-dynamics-365-fno/>
         7. Good for:
            1. Build, sync, test
            2. But it is complicated. Ideal for complex implementations so multiple pipeline instances can be running
   4. Best practices for build pipelines
      1. Use version control for pipeline definitions: Store your pipeline definitions in version control, such as Azure Repos, to enable collaboration, track changes, and revert to previous versions if needed.
         1. Consider: pipelines are code, code can’t be changed in Main, have to create a branch for a pipeline change, the PR will create a build request with the pipeline you made. It is self testing.
      2. Keep pipelines simple: Keep your pipelines simple and focused on a specific goal. Avoid creating overly complex pipelines that are difficult to understand and maintain.
         1. Build in azure hosted
         2. Build and package in Azure hosted
         3. Build and test in self hosted build machine
         4. Build, test, package in on demand build machine
      3. Use templates: Use templates to reuse common pipeline components, such as environment definitions, across multiple pipelines
         1. https://github.com/microsoft/Dynamics365-Xpp-Samples-Tools/tree/master/CI-CD/Pipeline-Samples
      4. Use multi-stage pipelines: Use multi-stage pipelines to divide your build process into multiple stages, such as build, test, and deploy, making it easier to manage and visualize your pipeline.
         1. https://devblogs.microsoft.com/devops/whats-new-with-azure-pipelines/
      5. Automate testing: Automate testing as much as possible to catch issues early in the development process. Use tools such as Azure Test Plans to manage your testing efforts.
         1. ATL
         2. SysUnit
         3. RSAT
      6. Use approvals: Use approvals to require manual approval before proceeding to the next stage, such as deploying to production, to enforce compliance and quality standards. This is a release pipeline.
         1. <https://www.programmingwithwolfgang.com/deployment-approvals-yaml-pipeline/>
      7. Monitor and improve pipeline performance: Monitor the performance of your pipelines and optimize the build process to reduce build time and increase efficiency. \*Only build Build Project\*
         1. https://learn.microsoft.com/en-us/azure/azure-monitor/app/continuous-monitoring
      8. Document your pipelines: Document your pipelines, including pipeline definitions and the reasoning behind pipeline design decisions, to ensure that other team members can understand and maintain the pipelines in the future.
         1. Templates from MSFT come documented, just keep it up to date
   5. “Code Coverage”
      1. Code coverage is a measure of how much of the source code of a program has been executed, tested, and validated during a particular test run. It helps in identifying which parts of the code have been tested and which have not, thus providing insights into the thoroughness of testing.
      2. Tools
         1. Microsoft Dynamics 365 Finance and Operations code coverage: This is a built-in code coverage tool that allows you to measure the code coverage of X++ code within the Microsoft Dynamics 365 Finance and Operations environment. It provides an overview of the code coverage percentage and allows you to view details on which lines of code are executed when tests are run.
            1. https://community.dynamics.com/365/financeandoperations/f/dynamics-365-for-finance-and-operations-forum/399219/how-to-get-the-code-coverage-in-x
         2. Visual Studio code coverage: This is a code coverage tool built into Visual Studio that can be used to measure the code coverage of X++ code. It provides detailed information on which lines of code are executed when tests are run, and allows you to view code coverage results in a visual format.
            1. <https://learn.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/perf-test/testing-validation>
            2. <https://www.powerazure365.com/blog-1/automate-your-tests-for-dynamics-365-finance-and-operations>
            3. <https://www.codecrib.com/2019/02/repost-enabling-x-code-coverage-in.html>
   6. “Automated testing” – ATL
      1. ATL framework enables you to perform various types of tests, such as unit tests, integration tests, UI tests, and data integrity tests, among others. It also supports test execution across different environments and configurations, making it easier to ensure that your applications are tested in different scenarios and conditions.
      2. <https://en.dynagile.com/blogandnews/automate-your-tests-for-dynamics-365-finance-and-operations>
      3. Test Types
         1. Unit Tests: These are tests that validate the individual pieces of code, such as methods and functions, in isolation from the rest of the application.
         2. Integration Tests: These are tests that validate how different components of the application interact with each other.
         3. UI Tests: These are tests that validate the user interface of the application, including the correctness of data displayed on forms and reports, and the behavior of buttons and other controls.
         4. Data Integrity Tests: These are tests that validate the accuracy and consistency of data within the application, including data entered by users and data stored in the database.
         5. Performance Tests: These are tests that measure the response time and throughput of the application, and help to identify potential bottlenecks and performance issues.
         6. Regression Tests: These are tests that validate that existing functionality has not been impacted by recent changes to the code.
   7. Other automated testing
      1. RSAT – too much to talk about
      2. Diagram

         Description automatically generated
   8. Implementing secure and compliant CI/CD pipelines with Azure DevOps
      1. Access control: Azure DevOps provides role-based access control (RBAC) to manage access to pipeline resources, ensuring that only authorized users can perform specific tasks and access sensitive information.
      2. Compliance: Azure DevOps provides a set of built-in compliance policies and auditing capabilities, as well as the ability to integrate with third-party security tools, to help you meet compliance requirements such as GDPR, HIPAA, and PCI DSS.
      3. Secrets Management: Azure DevOps provides secure and encrypted secrets management capabilities, so that sensitive information such as passwords, API keys, and certificates can be securely stored and used in your pipelines without being exposed in the pipeline configuration or code.
   9. Implementing continuous integration and delivery (CI/CD) best practices
      1. Automate as much as possible: Automating your build, test, and deployment processes can help to reduce the risk of human error and speed up the delivery of new code changes.
      2. Test early and often: Use automated testing to validate your code changes as early as possible in the development process. This can help you to catch and fix issues before they become major problems.
      3. Use Continuous Integration: Set up a Continuous Integration (CI) process to automatically build and test your code every time new code changes are committed. This can help you to catch issues early and ensure that your code is always in a releasable state.
      4. Implement Continuous Delivery: Use Continuous Delivery (CD) to automatically deploy your code changes to production once they have passed all tests. This can help you to reduce the time it takes to get new code changes into production and speed up your software delivery process.
      5. Monitor and measure: Use tools to monitor and measure the performance of your CI/CD pipeline, such as build and test duration, code coverage, and deployment success rate. This can help you to identify bottlenecks and areas for improvement.
      6. Regularly review and improve: Regularly review and evaluate your CI/CD pipeline to identify areas for improvement. Use feedback from your team and customers to continually refine and optimize your CI/CD process.
   10. Activities:
       1. ~~Have the attendees run a code coverage report and discuss the results.~~
       2. Have the attendees practice setting up automated testing with Atl or other testing tools.
       3. Have the attendees practice setting up code validation in a build pipeline.
8. Build validation
   1. ~~Overview of code validation tools and techniques~~
   2. ~~Setting up code validation in a build pipeline~~
   3. ~~Best practices for code validation~~
   4. ~~Activities~~
      1. ~~Have the attendees work in pairs to set up a release pipeline and share it with the group for feedback.~~
      2. ~~Have the attendees practice customizing the Azure DevOps build and release experience with custom tasks and extensions.~~
9. Release pipelines
   1. Overview of release pipelines
   2. Setting up a release pipeline
      1. https://ariste.info/en/2019/10/devops-alm-automation-in-microsoft-dynamics-365-for-finance-and-operations/
   3. ~~Best practices for release pipelines~~
      1. ~~Automate deployment processes: Use Azure DevOps Release pipelines to automate deployment processes, ensuring that applications are deployed consistently and reliably~~
      2. ~~Use environment-specific variables: Use environment-specific variables to manage configuration settings, such as database connection strings, that change between environments.~~
      3. ~~Implement approvals and gates: Use approvals and gates to control the flow of releases and ensure that only authorized personnel can deploy applications to production.~~
      4. ~~Monitor and track releases: Monitor releases in real-time and track the status of releases to ensure that any issues are quickly identified and resolved.~~
   4. Customizing the Azure DevOps build and release experience with custom tasks and extensions
      1. Turn on build VM – if using build VM
      2. Dynamics 365 Finance and Operations Tools - Required
      3. Pull Request Merge Conflict Extension - Suggested
      4. Trigger Azure DevOps Pipeline – chain together pipelines
      5. Generate Release Notes
      6. Azure Pipelines for Microsoft Teams – have DevOps post messages in DevOps
   5. Automating testing and deployment with Azure DevOps
      1. Build successful
      2. ATL, SysTest Successful
      3. Release to RSAT specific testing environment
      4. Run RSAT Testing
         1. Todo: Juan’s articles
   6. Activities
      1. Have the attendees work in small groups to research and discuss different automation options available in Azure DevOps.
      2. Have the attendees practice automating tasks in a build or release pipeline.
10. x++ package management
    1. Overview of x++ package management
       1. Should have 1 package with all changes? Or multiple packages? Are those packages friends? What about ISVs?
       2. Only save it to Azure DevOps if you need it later
       3. Can be deployed manually
    2. Setting up x++ package management in Azure DevOps
       1. Setting on the pipeline
    3. ~~Best practices for x++ package management~~
       1. ~~Establish a package management strategy: Define a clear package management strategy to ensure that packages are organized, versioned, and deployed consistently.~~
       2. ~~Use version control: Use version control, such as Git, to manage the source code of packages and track changes over time.~~
       3. ~~Automate the build process: Automate the build process to ensure that packages are built consistently and reliably.~~
       4. ~~Use release pipelines: Use Azure DevOps Release pipelines to automate the deployment of packages to target environments.~~
       5. ~~Implement approvals and gates: Implement approvals and gates to control the flow of releases and ensure that only authorized personnel can deploy packages to production.~~
       6. ~~Manage dependencies: Manage dependencies between packages to ensure that packages are deployed in the correct order and that all dependencies are satisfied.~~
       7. ~~Store packages in a centralized repository: Store packages in a centralized repository, such as Azure Artifacts, to ensure that packages are easily accessible and that versioning is consistent.~~
       8. ~~Use environment-specific variables: Use environment-specific variables to manage configuration settings, such as database connection strings, that change between environments.~~
       9. ~~Monitor and track packages: Monitor and track packages to ensure that any issues are quickly identified and resolved.~~
       10. ~~Test packages thoroughly: Thoroughly test packages in a pre-production environment before deploying to production to minimize the risk of production failures.~~
    4. Activities
       1. Have the attendees share tips and tricks for x++ development with the group.
11. Assorted other considerations for x++
    1. Overview of other considerations for x++ development
    2. Consuming C#
12. Q&A and wrap-up
    1. Summary of the workshop
    2. Q&A session for attendees
    3. Next steps for attendees
    4. Activities
       1. Have the attendees ask questions and share feedback about the workshop.
       2. Have the attendees share next steps or plans for applying what they've learned to their own work or projects.