Microsoft

AZ-400.00 Learning Path 02: Development for enterprise DevOps

Agenda

- Module 01: Structure your Git Repo.
- Module 02: Manage Git branches and workflows.
- Module 03: Collaborate with pull requests in Azure Repos.
- Module 04: Explore Git hooks.
- Module 05: Plan foster inner source.
- Module 06: Manage Git repositories.
- Module 07: Identify technical debt.
- Labs & Learning Path review and takeaways.

Learning Path overview

Learning objectives

After completing this Learning Path, students will be able to:

1. Explain how to structure Git Repos
2. Describe Git branching workflows
3. Leverage pull requests for collaboration and code reviews
4. Use GitHub Flow for collaboration

Module 01: Structure your Git Repo

Explore monorepo versus multiple repos

1. **Monorepos –** Source control pattern where all the source code is kept in a single repository.

2. **Multiple repositories –** Refer to organizing your projects each into their separate repository.

- A repository is a place where the history of your work is stored.
- Azure DevOps projects can contain multiple repositories. There are two philosophies on organizing your repos: Monorepo or Multiple repos.
- The fundamental difference between the monorepo and multiple repos philosophies boils down to a difference about what will allow teams working together on a system to go fastest.

Implement a change log

1

Track changes to a project over versions.
For each version, record:

- New functionality
- Changed functionality
- Removed functionality

2

- Preference is to always avoid dumping log entries into a change log.
- Logs are "noisy," and so it's easy to generate a mess that is not helpful.

3

- Can be manually created
- Can be automatically populated
- Can be a combination of both
- Auto:
  - git log
  - gitchangelog
  - github_changelog_generator

```
git log [options] vX.X.X..vX.X.Y | helper-script > projectchangelogs/X.X.Y
```

Module 02: Manage Git branches and workflows

Explore branch workflow types

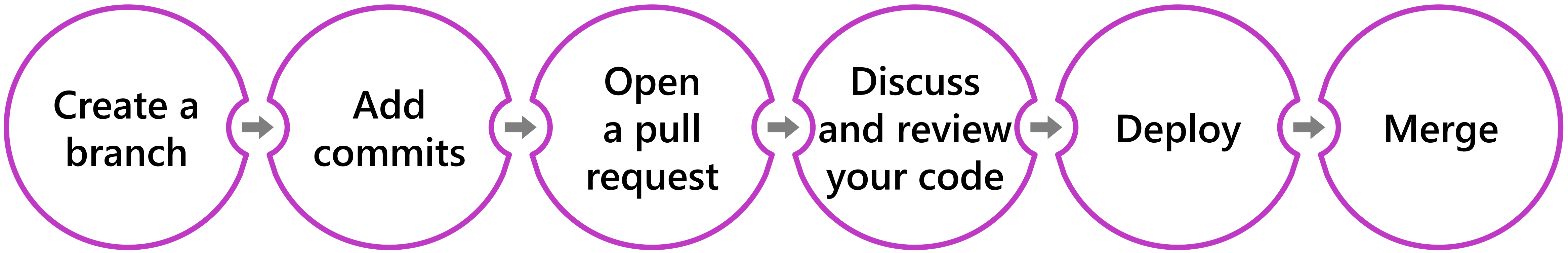
1 **Feature branching** – All feature development should take place in a dedicated branch instead of the main branch.

2 **Forking Workflow** – Every developer uses a server-side repository.

3 **Evaluate the workflow:**

- Does this workflow scale with team size?
- Is it easy to undo mistakes and errors with this workflow?
- Does this workflow impose any new unnecessary cognitive overhead to the team?

Explore feature branch workflow



All feature development should take place in a dedicated branch instead of the main branch.

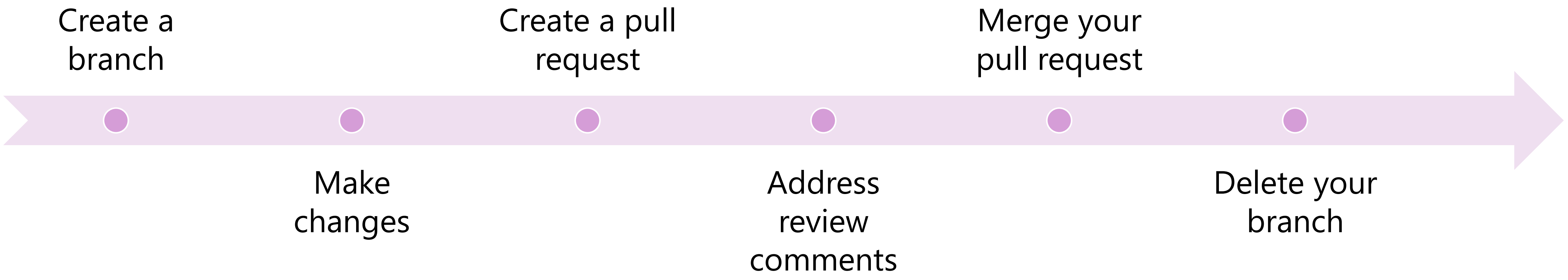
Encapsulating feature development leverages pull requests, which are a way to initiate discussions around a branch.

Share a feature with others without touching any official code.

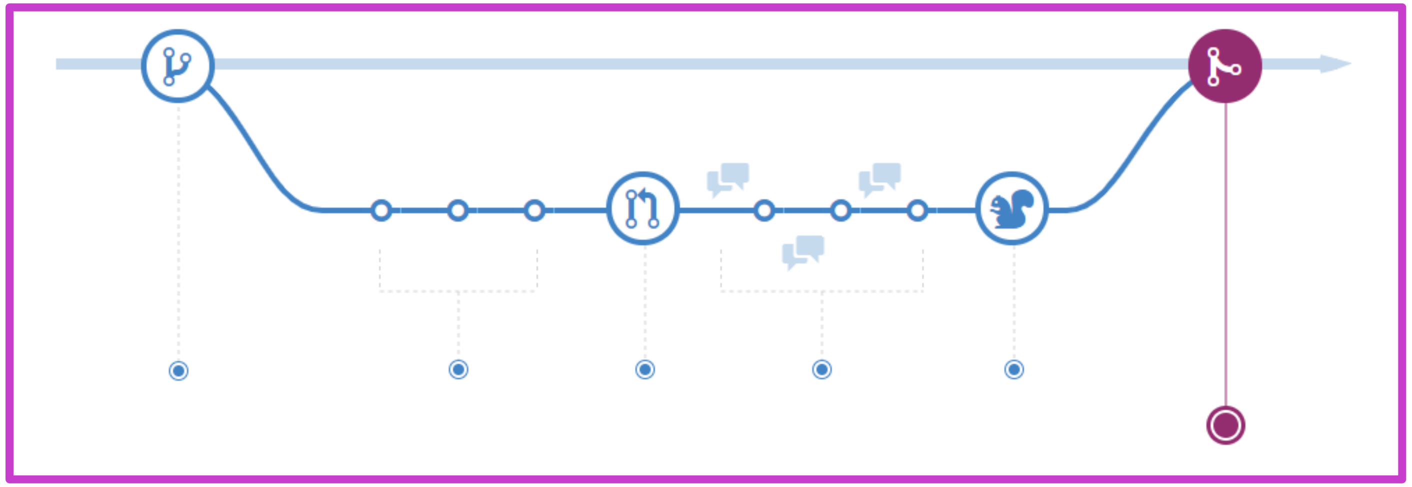
Explore Git branch model for continuous delivery

DEMO

Explore GitHub flow



GitHub Flow is a lightweight, branch-based workflow. The GitHub flow is useful for everyone, not just developers



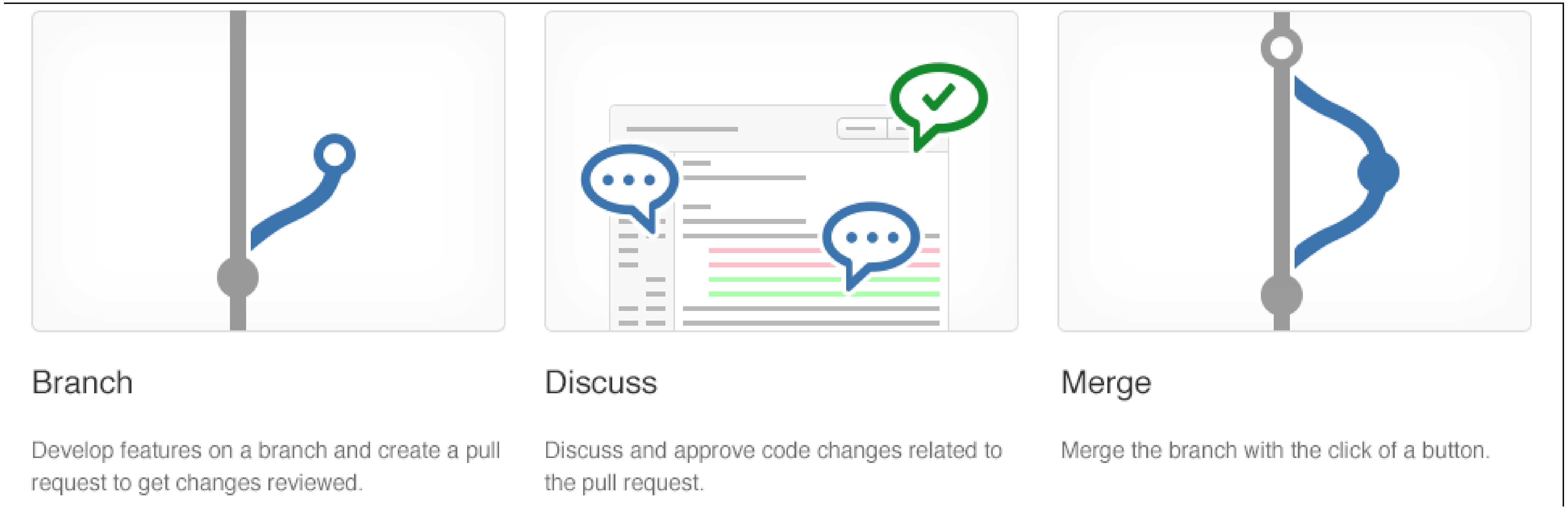
Explore fork workflow

1. Forking workflow gives every developer their own server-side repository
2. Each contributor has not one, but two Git repositories: a private local one and a public server-side one
3. Most often seen in public open-source projects
4. Contributions can be integrated without the need for everybody to push to a single central repository

✓ Forked repositories use the standard git **clone** command

Module 03: Collaborate with pull requests in Azure Repos

Collaborate with pull requests



Branch

Develop features on a branch and create a pull request to get changes reviewed.

Discuss

Discuss and approve code changes related to the pull request.

Merge

Merge the branch with the click of a button.

Pull requests let you tell others about changes.

Collaboration using the Shared Repository Model

Review and merge your code in a single collaborative process.

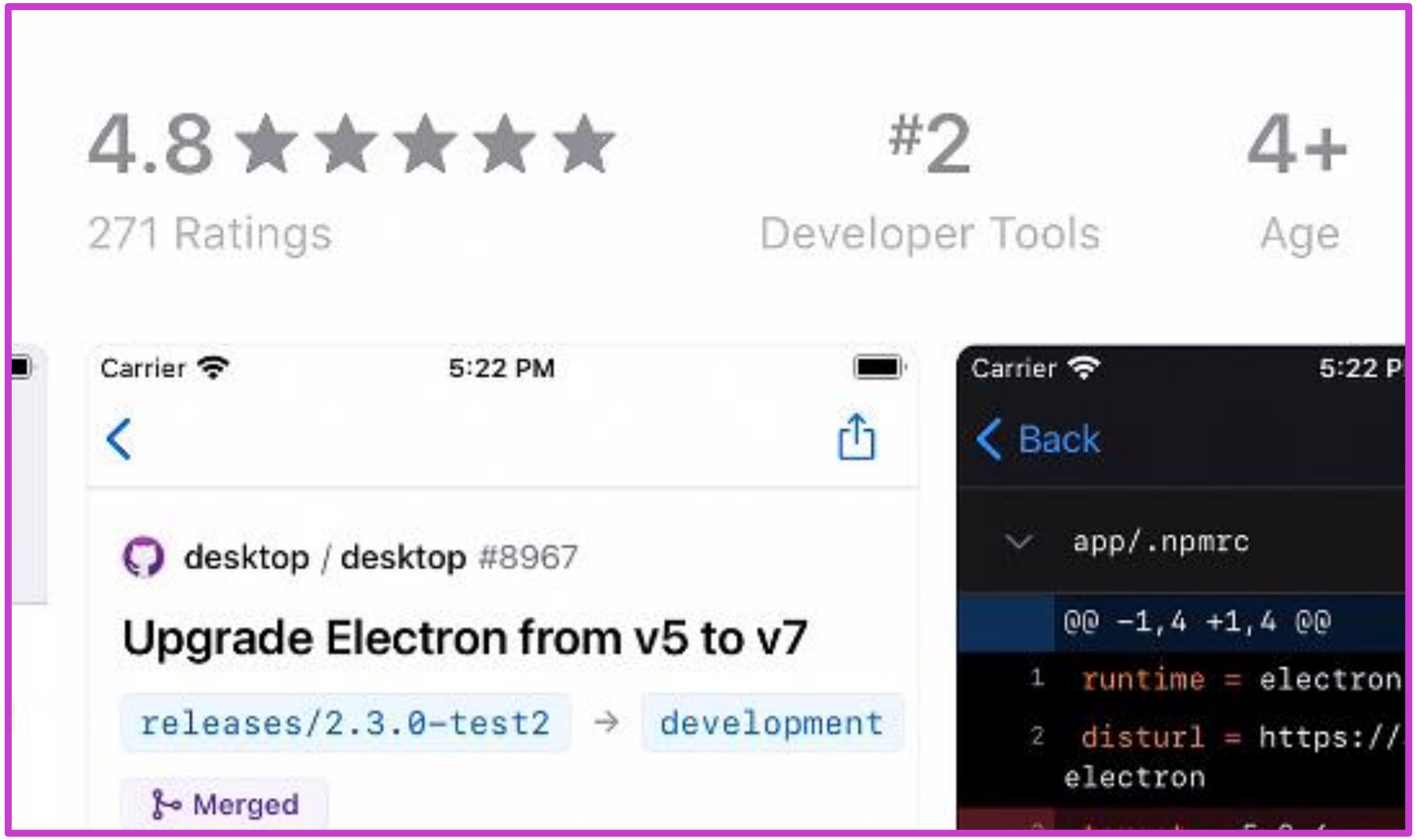
Be sure to provide good feedback and protect branches with policies.

Azure Repos collaborating with pull requests

DEMO

GitHub mobile for pull request approvals

- App can render Markdown, images, PDF files
- Manage pull requests directly within the app
- Comments can be added (including using emoji short codes)



desktop / desktop #8967

Upgrade Electron from v5 to v7

releases/2.3.0-test2 → development

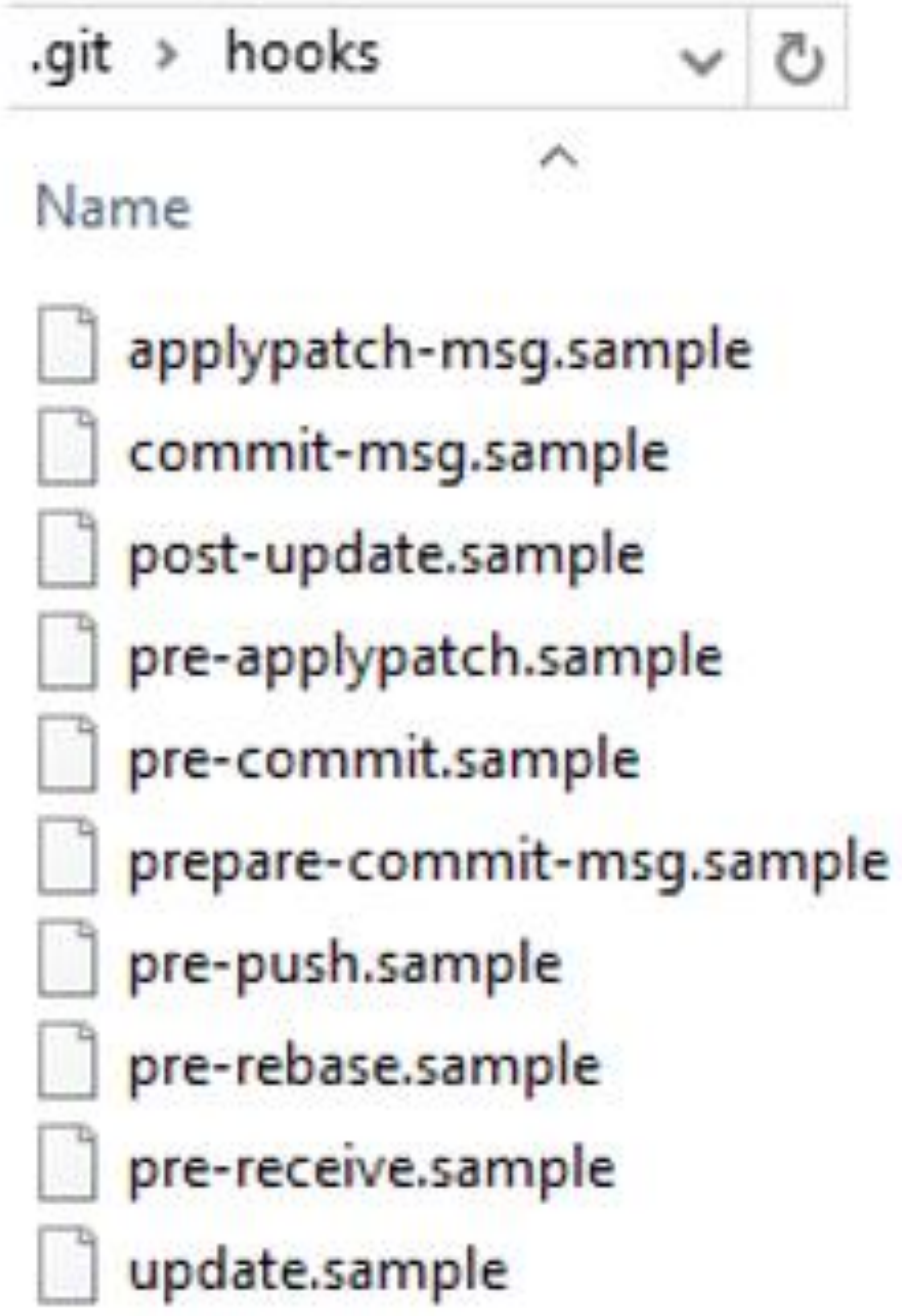
Merged

```
app/.npmrc
@@ -1,4 +1,4 @@
runtime = electron
disturl = https://
electron
```

Module 04: Explore Git hooks

Introduction to Git hooks

- A mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur
- Use Git hooks to enforce policies, ensure consistency, and control your environment
- Can be either client-side or server-side



applypatch-msg.sample
commit-msg.sample
post-update.sample
pre-applypatch.sample
pre-commit.sample
prepare-commit-msg.sample
pre-push.sample
pre-rebase.sample
pre-receive.sample
update.sample

Implement Git hooks

Will my code:

- Break other code?
- Introduce code quality issues?
- Drop the code coverage?
- Take on a new dependency?

Will the incoming code:

- Break my code?
- Introduce code quality issues?
- Drop the code coverage?
- Take on a new dependency?

Module 05: Plan foster inner source

Explore foster inner source

1. Fork-based pull request workflows allow anybody to contribute.
2. **Inner source** brings all the benefits of open-source software development inside your firewall.
3. We recommend the forking workflow for large numbers of casual or occasional committers.

Implement the fork workflow

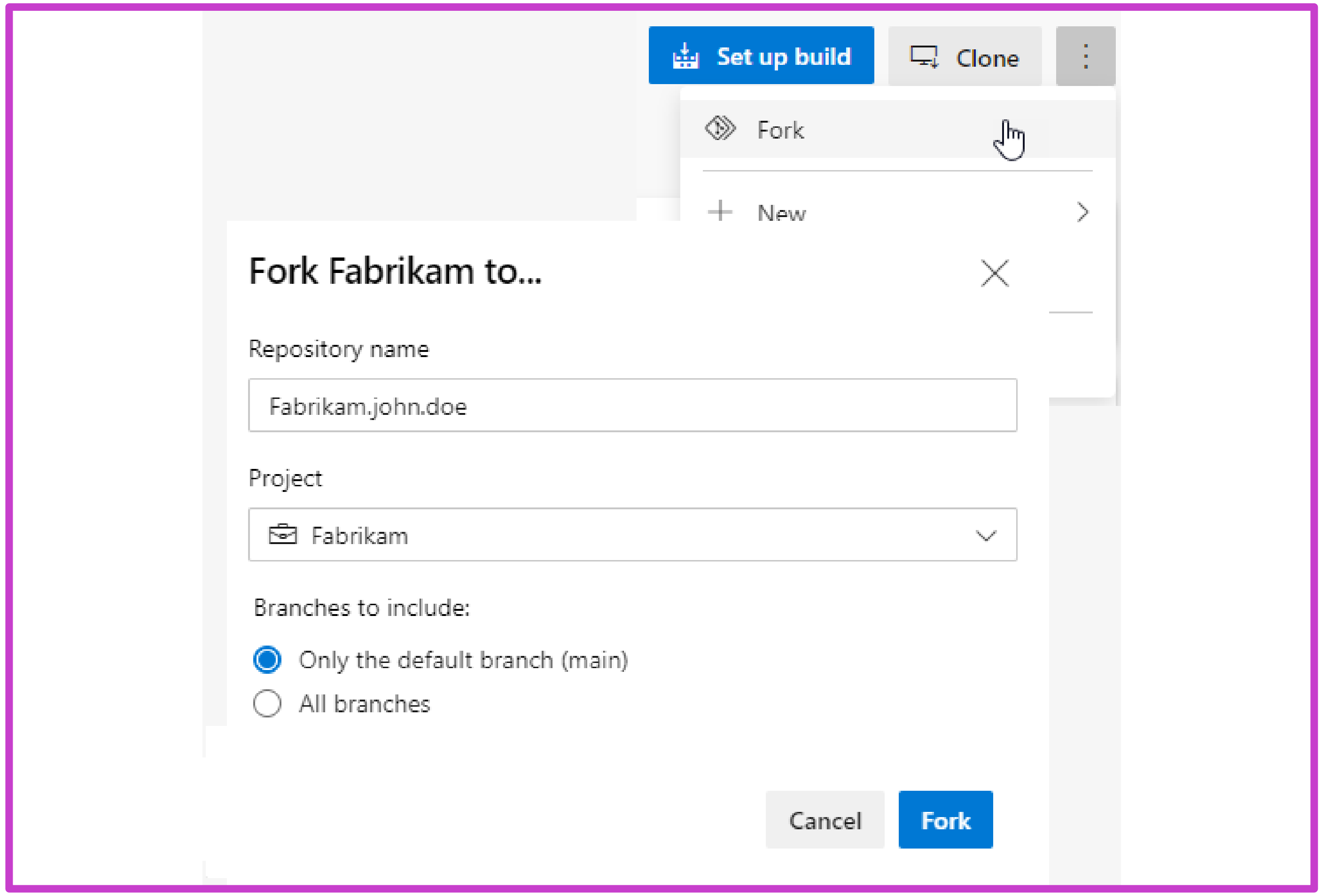
What's in a fork?

Sharing code between forks

Choosing between branches and forks

The forking workflow:

- Create a fork
- Clone it locally
- Make your changes locally and push them to a branch
- Create and complete a PR to upstream
- Sync your fork to the latest from upstream



Fork Fabrikam to...

Repository name: Fabrikam.john.doe

Project: Fabrikam

Branches to include:

[✓ Only the default branch (main)]
All branches

Describe inner source with forks

DEMO

Module 06: Manage Git repositories

Work with large repositories

1. Two common causes: Long history and large binary files

2. Long history: Use shallow clones
```
git clone –depth [depth] [clone-url]
```

3. Large binary files: Use Git Large File Storage (LFS)

Work with large repositories using Scalar

1. .NET Core application available for Windows and MacOS.

2. Maximizes Git command performance.

3. Register, Pause, Unregister:
   ```
   scalar [register][pause][unregister]
   ```

4. Run and Clone:
   ```
   scalar [run all], scalar [clone URL]
   ```

Scalar

Purge repository data

Might need to delete files from repository:

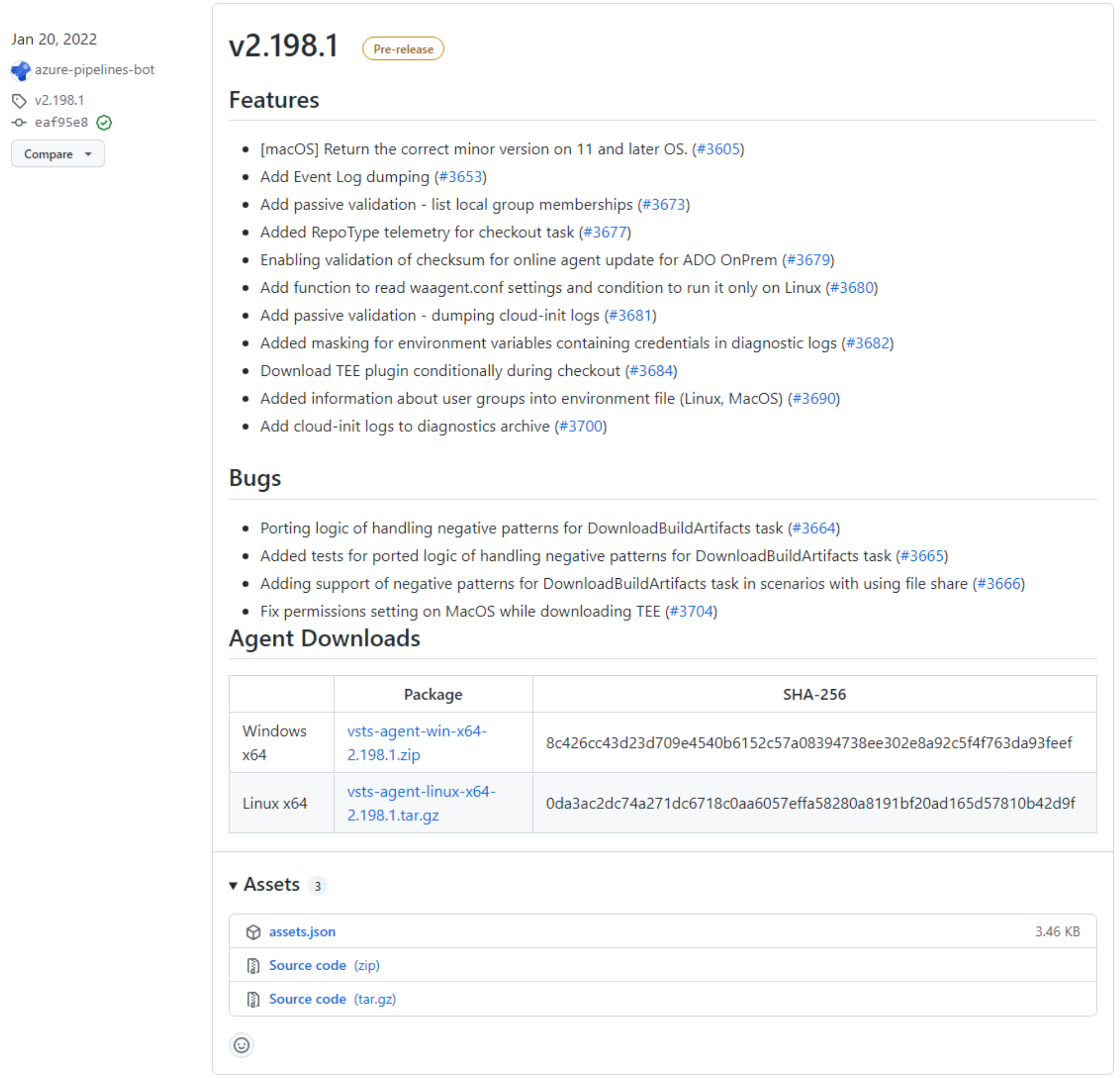
- Reduce repository size
- Remove accidentally committed large file
- Remove committed file with sensitive data (passwords, keys)

Use:

- git filter-repo tool
- BFG Repo-Cleaner
- Example:
  - $ bfg --delete-files file_I_should_not_have_committed
  - $ bfg --replace-text passwords.txt

Manage releases with GitHub Repos

- Releases in GitHub are based on Git tags.
- Tag is a photo of your repository's current state.
- Create a release tag:
  `gh release create tag`
- To create a prerelease with the specified title and notes:
  `gh release create v1.2.1 --title`



Jan 20, 2022
azure-pipelines-bot
v2.198.1
eaf95e8
Compare

v2.198.1 Pre-release

Features

- [macOS] Return the correct minor version on 11 and later OS. (#3605)
- Add Event Log dumping (#3653)
- Add passive validation - list local group memberships (#3673)
- Added RepoType telemetry for checkout task (#3677)
- Enabling validation of checksum for online agent update for ADO OnPrem (#3679)
- Add function to read waagent.conf settings and condition to run it only on Linux (#3680)
- Add passive validation - dumping cloud-init logs (#3681)
- Added masking for environment variables containing credentials in diagnostic logs (#3682)
- Download TEE plugin conditionally during checkout (#3684)
- Added information about user groups into environment file (Linux, MacOS) (#3690)
- Add cloud-init logs to diagnostics archive (#3700)

Bugs

- Porting logic of handling negative patterns for DownloadBuildArtifacts task (#3664)
- Added tests for ported logic of handling negative patterns for DownloadBuildArtifacts task (#3665)
- Adding support of negative patterns for DownloadBuildArtifacts task in scenarios with using file share (#3666)
- Fix permissions setting on MacOS while downloading TEE (#3704)

Agent Downloads

	Package	SHA-256
Windows x64	vsts-agent-win-x64-2.198.1.zip	8c426cc43d23d709e4540b6152c57a08394738ee302e8a92c5f4f763da93feef
Linux x64	vsts-agent-linux-x64-2.198.1.tar.gz	0da3ac2dc74a271dc6718c0aa6057effa58280a8191bf20ad165d57810b42d9f

Assets 3

assets.json	3.46 KB
Source code (zip)	
Source code (tar.gz)	

Automate release notes with GitHub

DEMO

Module 07: Identify technical debt

Examine code quality

Short deadlines, a lack of coding standards, and poor technical skills can lead to code that is NOT:

1. Clear and readable
2. Documented
3. Efficient
4. Maintainable
5. Extensible
6. Secure

Examine complexity and quality metrics

A few of the most important complexity-related metrics:

1. Program vocabulary
2. Calculated program length
3. Volume
4. Difficulty
5. Effort

Introduction to technical debt

Technical Debt describes the future penalty that you incur today by making easy or quick choices in software development practices.

Measure and manage technical debt

1. Failed builds percentage
2. Failed deployments percentage
3. Ticket volume
4. Bug bounce percentage
5. Unplanned work percentage

Sources and impacts of technical debt

Common sources of technical debt are:

- Lack of coding style and standards.
- Lack of or poor design of unit test cases.
- Ignoring or not understanding object design principles.
- Monolithic classes and code libraries.
- Poorly envisioned use of technology, architecture and approach.
- Over-engineering code.
- Insufficient comments and documentation.
- Not writing self-documenting code.
- Taking shortcuts to meet deadlines.
- Leaving dead code in place.

Using automated testing to measure technical debt

Technical debt:

1. Adds problems during development that makes it more difficult to add customer value
2. Saps productivity and frustrates development teams
3. Makes code both hard to understand and fragile
4. Increases the time to make changes, and to validate those changes
5. Starts small and grows over time

✓ **One way to minimize the accumulation of technical debt, is to use automated testing and assessment**

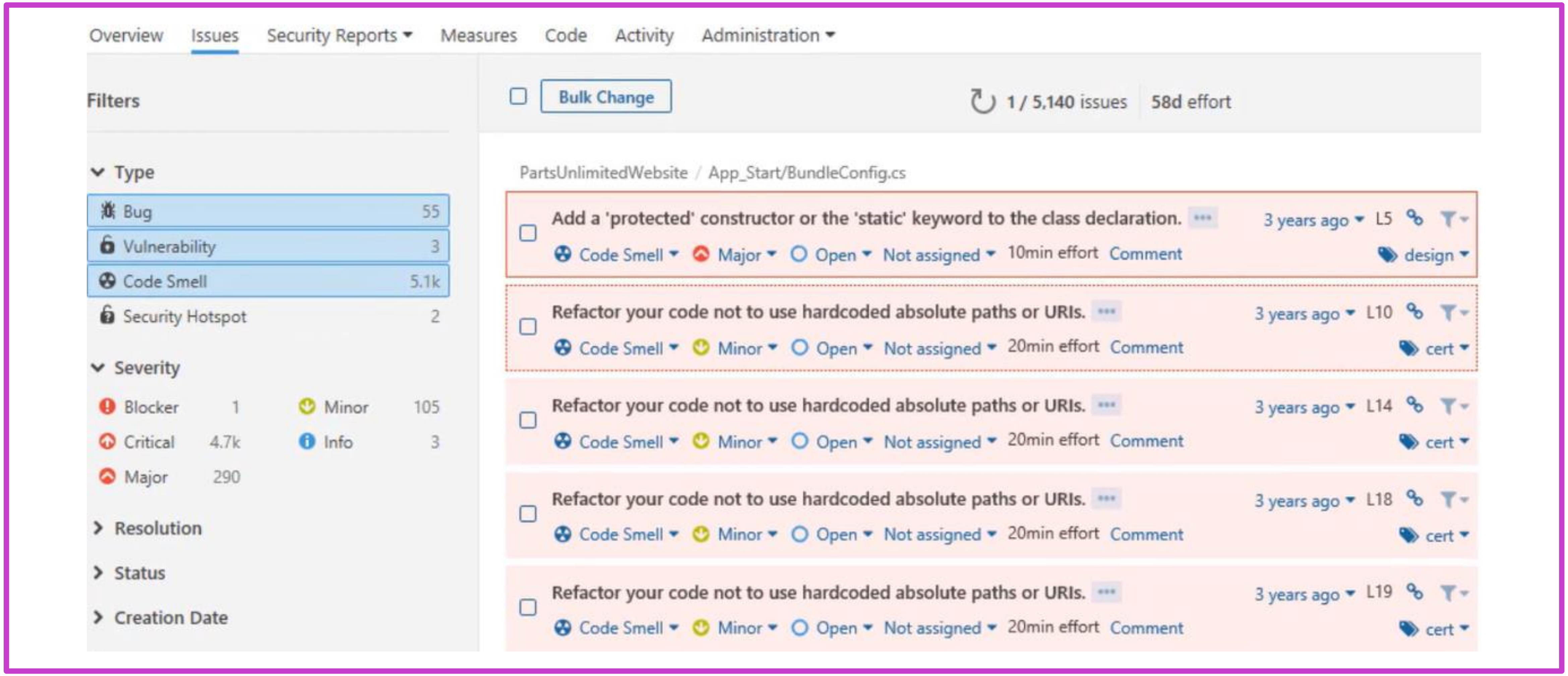
Discussion: Code quality tooling

Azure DevOps can be integrated with a wide range of existing tooling that is used for checking code quality during builds

Which code quality tools do you currently use (if any)?

What do you
or don’t you like about the tools?

Measure and manage technical debt



Overview Issues Security Reports Measures Code Activity Administration

Filters

Bulk Change 1 / 5,140 issues 58d effort

Type

Bug	55
Vulnerability	3
Code Smell	5.1k
Security Hotspot	2

Severity

Blocker	1	Minor	105
Critical	4.7k	Info	3
Major	290		

Resolution

Status

Creation Date

PartsUnlimitedWebsite / App_Start/BundleConfig.cs

Add a 'protected' constructor or the 'static' keyword to the class declaration. — 3 years ago L5
Code Smell · Major · Open · Not assigned · 10min effort · Comment · design

Refactor your code not to use hardcoded absolute paths or URIs. — 3 years ago L10
Code Smell · Minor · Open · Not assigned · 20min effort · Comment · cert

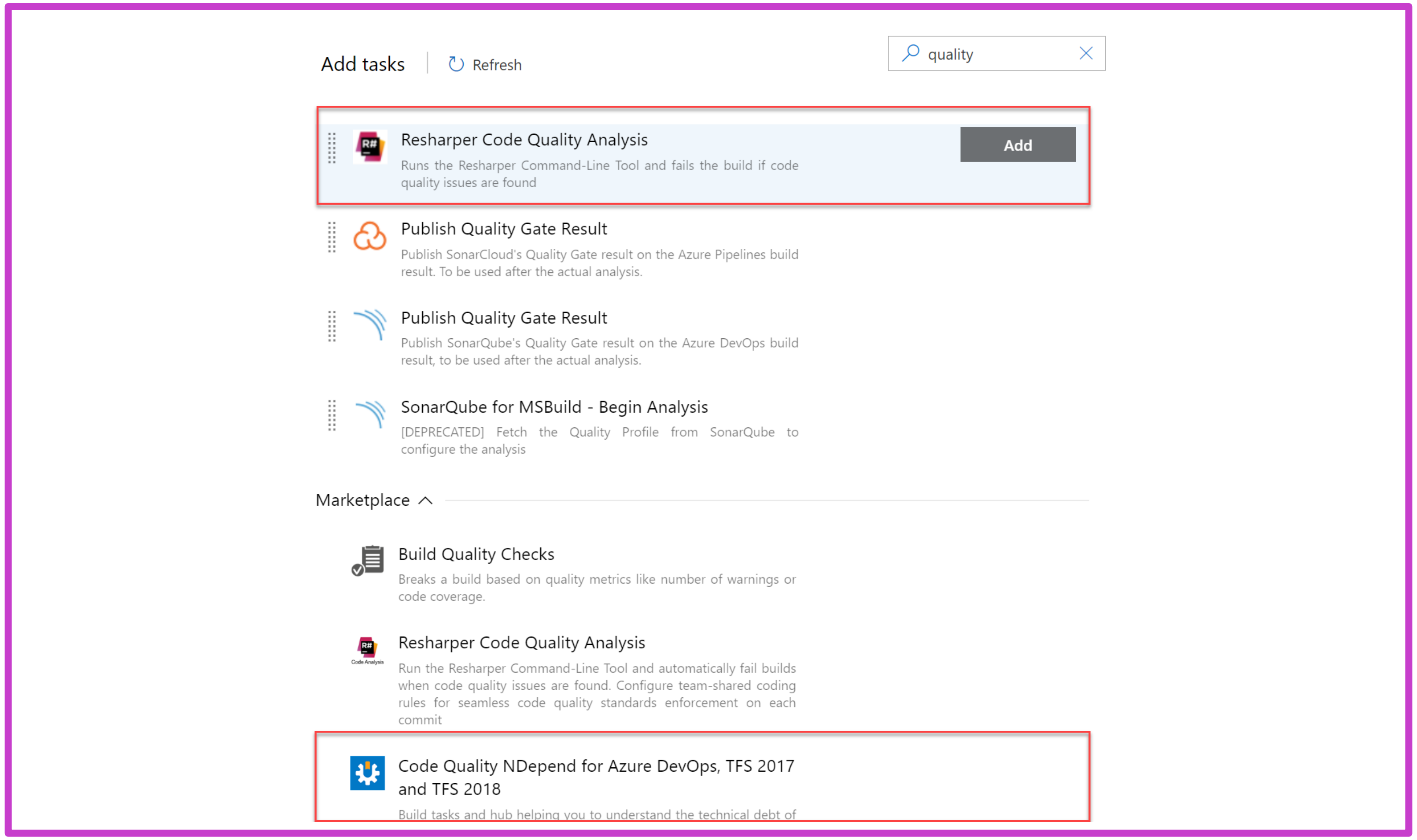
Refactor your code not to use hardcoded absolute paths or URIs. — 3 years ago L14
Code Smell · Minor · Open · Not assigned · 20min effort · Comment · cert

Refactor your code not to use hardcoded absolute paths or URIs. — 3 years ago L18
Code Smell · Minor · Open · Not assigned · 20min effort · Comment · cert

Refactor your code not to use hardcoded absolute paths or URIs. — 3 years ago L19
Code Smell · Minor · Open · Not assigned · 20min effort · Comment · cert

Integrate other code quality tools

- NDepend is a Visual Studio extension that assesses the amount of technical debt that a developer has added during a recent development period, typically in the last hour.
- Resharper Code Quality Analysis is a command line tool and can be set to automatically fail builds when code quality issues are found.



Resharper Code Quality Analysis
Runs the Resharper Command-Line Tool and fails the build if code quality issues are found

Publish Quality Gate Result
Publish SonarCloud's Quality Gate result on the Azure Pipelines build result. To be used after the actual analysis.

Publish Quality Gate Result
Publish SonarQube's Quality Gate result on the Azure DevOps build result, to be used after the actual analysis.

SonarQube for MSBuild - Begin Analysis
[DEPRECATED] Fetch the Quality Profile from SonarQube to configure the analysis

Build Quality Checks
Breaks a build based on quality metrics like number of warnings or code coverage.

Resharper Code Quality Analysis
Run the Resharper Command-Line Tool and automatically fail builds when code quality issues are found. Configure team-shared coding rules for seamless code quality standards enforcement on each commit

Code Quality NDepend for Azure DevOps, TFS 2017 and TFS 2018
Build tasks and hub helping you to understand the technical debt of

Plan effective code reviews

1. Everyone is trying to achieve better code quality
2. Knowledge sharing

Labs

Lab: Version controlling with Git in Azure Repos

Lab overview:

In this lab you will establish and work with a local Git repository, and how to work with branches and repositories in Azure DevOps.

Objectives:

- Clone an existing repository
- Save work with commits
- Review history of changes
- Work with branches by using Visual Studio Code

Duration:

Learning Path review and takeaways

What did you learn?

1. Explain how to structure Git Repos
2. Describe Git branching workflows
3. Leverage pull requests for collaboration and code reviews
4. Leverage Git hooks for automation
5. Use Git to foster inner source across the organization

Learning Path review questions

1. What are two types of branching?
2. What are Git hooks?
3. What are some best practices when working with files in Git? What do you suggest for working with large files?

Microsoft