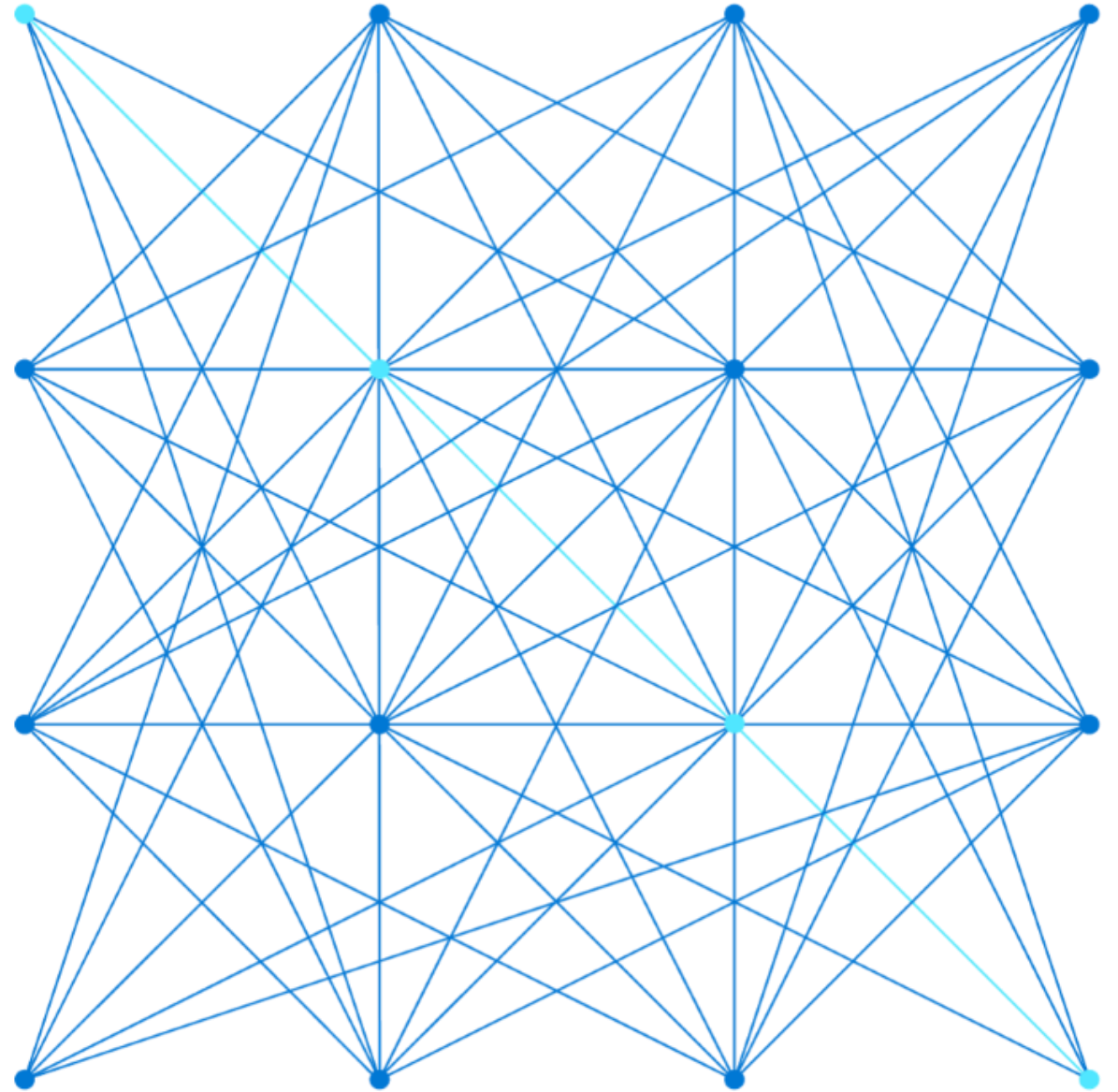


AZ-400.00

Module 03:

Managing Technical Debt



Lesson 01: Module overview



Module overview



Lesson 1: Module overview



Lesson 2: Identifying technical debt



Lesson 3: Knowledge sharing within teams



Lesson 4: Lab



Lesson 5: Module review and takeaways

Learning objectives

After completing this module, students will be able to:

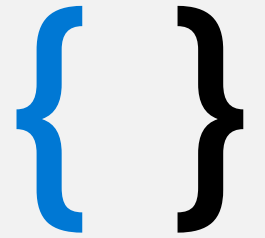


Identify technical debt



Build organizational knowledge on code quality

Lesson 02: Identifying technical debt

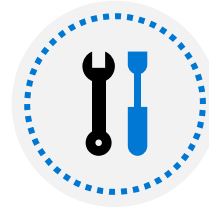


Code quality defined

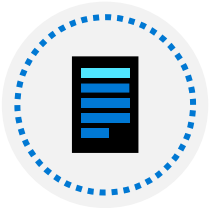
Short deadlines, a lack of coding standards, and poor technical skills can lead to code that is **NOT**:



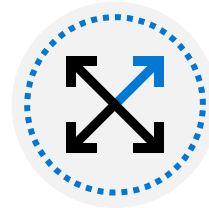
Clear and readable



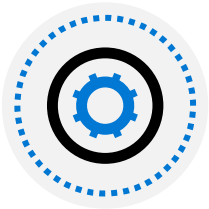
Maintainable



Documented



Extensible



Efficient

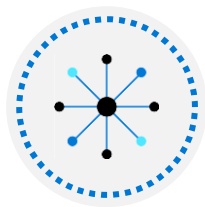


Secure

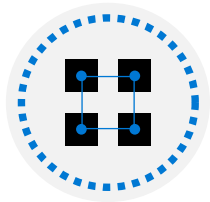
Complexity metrics

A few of the most important complexity-related metrics:

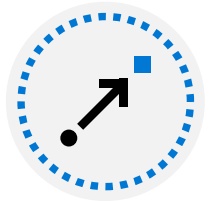
- Program vocabulary (#mw: the no. of **unique** operators & operands used in the program. [link](#))
- Calculated program length (#mw: the no. of operators & operands used in the program. [link](#))
- Volume (#mw: minimum no. of bits required to encode the program. [link](#))
- Difficulty (#mw: You can read it yourself. [link](#))
- Effort (#mw: effort required to read & understand the program. [link](#))



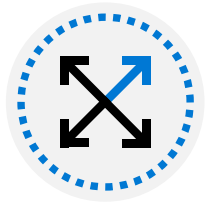
Measuring and managing quality metrics



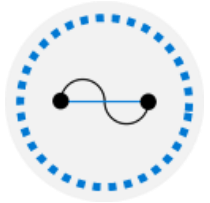
Failed builds percentage



Failed deployments percentage



Ticket volume

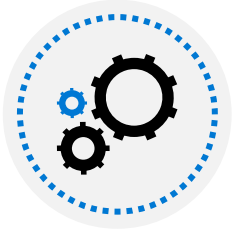


Bug bounce percentage



Unplanned work percentage

Technical debt defined



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

Sources and impacts of technical debt

Common sources of technical debt are:

Lack of coding style and standards

Over-engineering code (az-p-OverEngineeredCode)

Lack of, or poor design of unit test cases

Insufficient comments and documentation

Ignoring or not understanding object orient design principles

Not writing self-documenting code

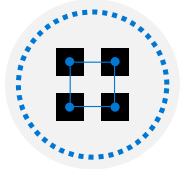
Monolithic classes and code libraries

Taking shortcuts to meet deadlines

Poorly envisioned use of technology, architecture and approach

Leaving dead code in place

Technical debt:



Adds problems during development that makes it more difficult to add customer value



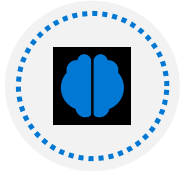
Saps productivity and frustrates development teams



Makes code both hard to understand and fragile



Increases the time to make changes, and to validate those changes



Starts small and grows over time



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

Measuring and managing technical debt

(#mw – this looks like the (older)UI of Sonarqube?)

Overview

Issues

Security Reports ▾

Measures

Code

Activity

Administration ▾

Filters

▼ 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

☐ Bulk Change

🔄

1 / 5,140 issues

58d effort

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

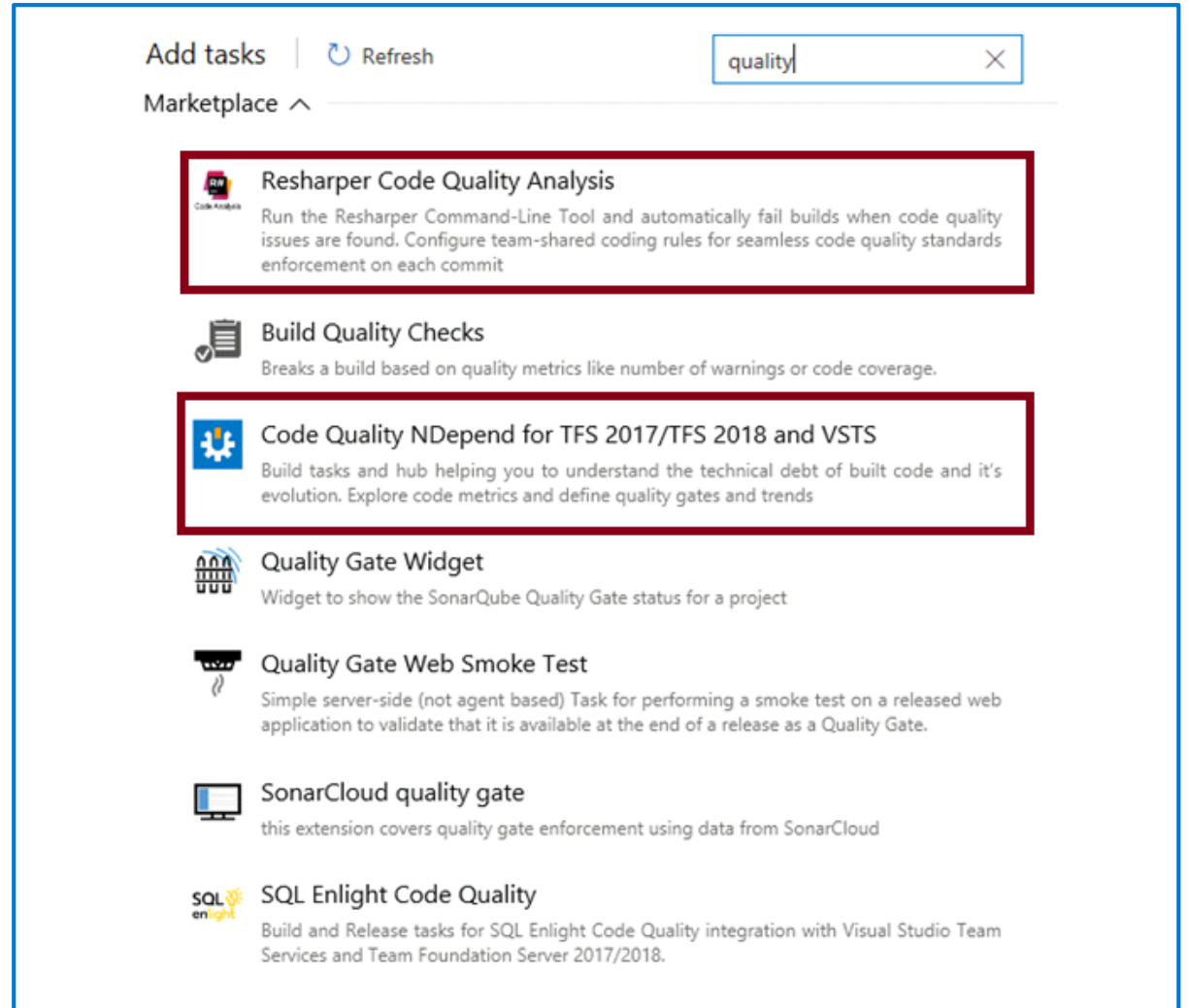
Comment

🔗 cert ▾

Integrating 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.



The screenshot shows the Visual Studio Marketplace search results for the keyword 'quality'. The search bar at the top contains the text 'quality'. Below the search bar, the results are listed. Two items are highlighted with red rectangular boxes:

- 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 TFS 2017/TFS 2018 and VSTS**: Build tasks and hub helping you to understand the technical debt of built code and its evolution. Explore code metrics and define quality gates and trends.

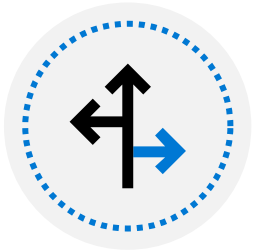
Other visible results include:

- Build Quality Checks**: Breaks a build based on quality metrics like number of warnings or code coverage.
- Quality Gate Widget**: Widget to show the SonarQube Quality Gate status for a project.
- Quality Gate Web Smoke Test**: Simple server-side (not agent based) Task for performing a smoke test on a released web application to validate that it is available at the end of a release as a Quality Gate.
- SonarCloud quality gate**: this extension covers quality gate enforcement using data from SonarCloud.
- SQL Enlight Code Quality**: Build and Release tasks for SQL Enlight Code Quality integration with Visual Studio Team Services and Team Foundation Server 2017/2018.

Planning effective code reviews



Everyone is trying to achieve better code quality

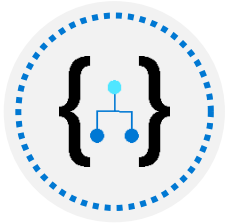


Knowledge sharing

Lesson 03: Knowledge sharing within teams



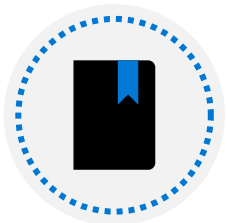
Sharing acquired knowledge within development teams



Organizational knowledge is acquired over time



Organizational knowledge can easily be lost through staff turnover



Relearning old lessons is wasteful and expensive

Discussion: Tools for knowledge sharing

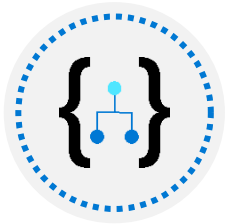


Which knowledge
sharing tools do
you currently use?



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

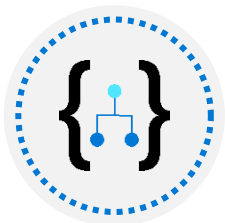
Azure DevOps project wikis



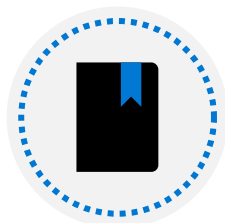
Created in Azure DevOps projects, stored in a repository

The screenshot displays the Azure DevOps web interface. On the left, a sidebar shows the 'Contoso' project selected, with a list of navigation items: Overview, Summary, Dashboards, Wiki (highlighted), and Boards. The main content area is titled 'Contoso Project Wiki'. At the top of this area is an 'Instant Search' message. Below it is a rich text editor toolbar with various icons for text formatting (bold, italic, link, code), list creation, table insertion, and other editing functions. The editor contains a welcome message: 'Welcome to the project wiki for **Contoso**. The aim of this wiki is to record key lessons learned by our project team over time.'

Wiki contents



Written in markdown and can contain file attachments, including videos



Supports insertion of Mermaid diagrams

Welcome to the project wiki for **Contoso**. The aim of this wiki is to record key lessons learned by our project team over time.

```
 ::: mermaid
graph LR;
  A[Wiki supports Mermaid] --> B[Visit https://mermaidjs.github.io/ for Mermaid syntax];
  :::
```

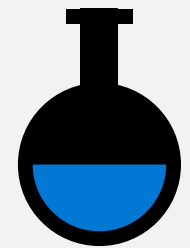
Welcome to the project wiki for **Contoso**. The aim of this wiki is to record key lessons learned by our project team over time.

Wiki supports Mermaid



Visit <https://mermaidjs.github.io/> for Mermaid syntax

Lesson 04: Modernizing development environments with GitHub Codespaces

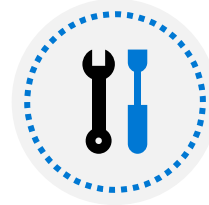


Developing online with GitHub Codespaces

Cloud-based development environment hosted by GitHub



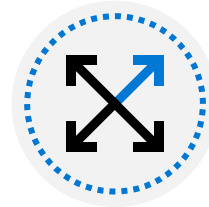
Avoids issues with old hardware/software



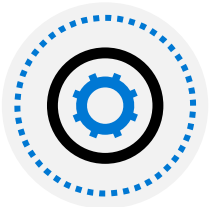
Based on Visual Studio Code



Highly portable



Work from PCs, tablets, Chromebooks



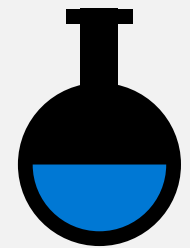
Protect against proliferation of intellectual property



Connect to Codespaces from Visual Studio Code

Lesson 05: Lab

We're skipping this.
Can do outside class time



Lab: Sharing team knowledge using Azure project wikis

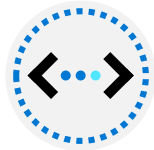


Lab overview:

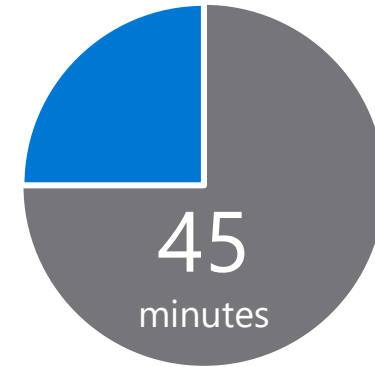
In this lab, you will create and configure wiki in an Azure DevOps, including managing markdown content and creating a Mermaid diagram.

Objectives:

- Create a wiki in an Azure Project
- Add and edit markdown
- Create a Mermaid diagram



Duration:



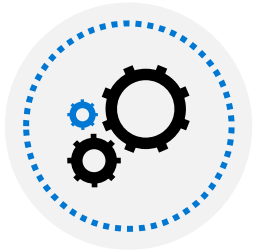
Lesson 06: Module review and takeaways



What did you learn?



Identify technical debt



Build organizational knowledge on code quality

Module review questions

1

What are Mermaid diagrams?

2

What are code smells? Give an example of a code smell.

3

You are using Azure Repos for your application source code repository. You want to create an audit of open-source libraries that you have used. Which tool could you use?

4

Name three attributes of high-quality code

5

You are using Azure Repos for your application source code repository. You want to perform code quality checks. Which tool could you use?