

The O of SOLID

The Open/Closed Principle

Maximilian Meffert

The Open/Closed Principle

- Probably around since the late 1980's
 - Bertrand Meyer, *Object-Oriented Software Construction*, 1st edition. Prentice-Hall 1988, ISBN 0-13-629031-0
- What does it say?
 - “Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.”
 - A module will be said to be open if it is still available for extension. For example, it should be possible to add fields to the data structures it contains, or new elements to the set of functions it performs.
 - A module will be said to be closed if [it] is available for use by other modules. This assumes that the module has been given a well-defined, stable description (the interface in the sense of information hiding)” (Bertrand Meyer)
 - “You should be able to extend the behavior of a system without having to modify that system.” (Robert C. Martin)
- What does it mean?
 - Software entities should be **both** open for extension **and** closed for modification
 - Open for Extension = It is possible to extend behavior
 - Closed for Modification = It is not necessary to modify manifestation (i.e. source code, compilation, etc.)

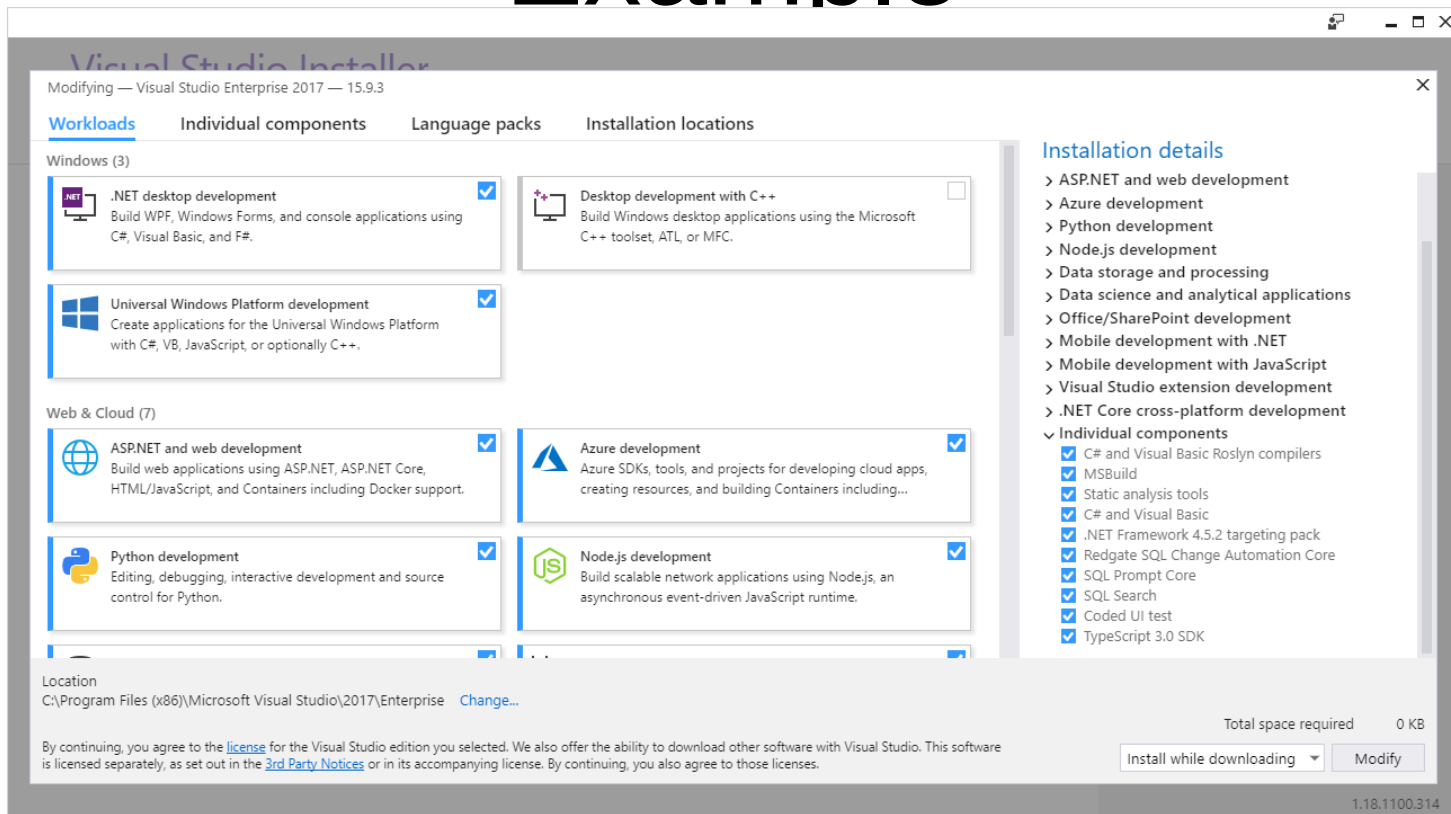
The Open/Closed Principle

- Why is it a “good” thing to adhere to?
 - Change of requirements is immanent through the life cycle of most software
 - Decreases change impact, i.e. number of modules to alter
 - This number should tend to 0 since you are just add new modules (replacing others)
 - Allows new features without the need to re-engineer / re-arrange exiting ones.
 - Decreases risk of regression because of human error, i.e. developer faults
 - Since change impact decreases
 - Increases a software designs capability for adaption to change
 - ... *uhm*, “agile”?

Example

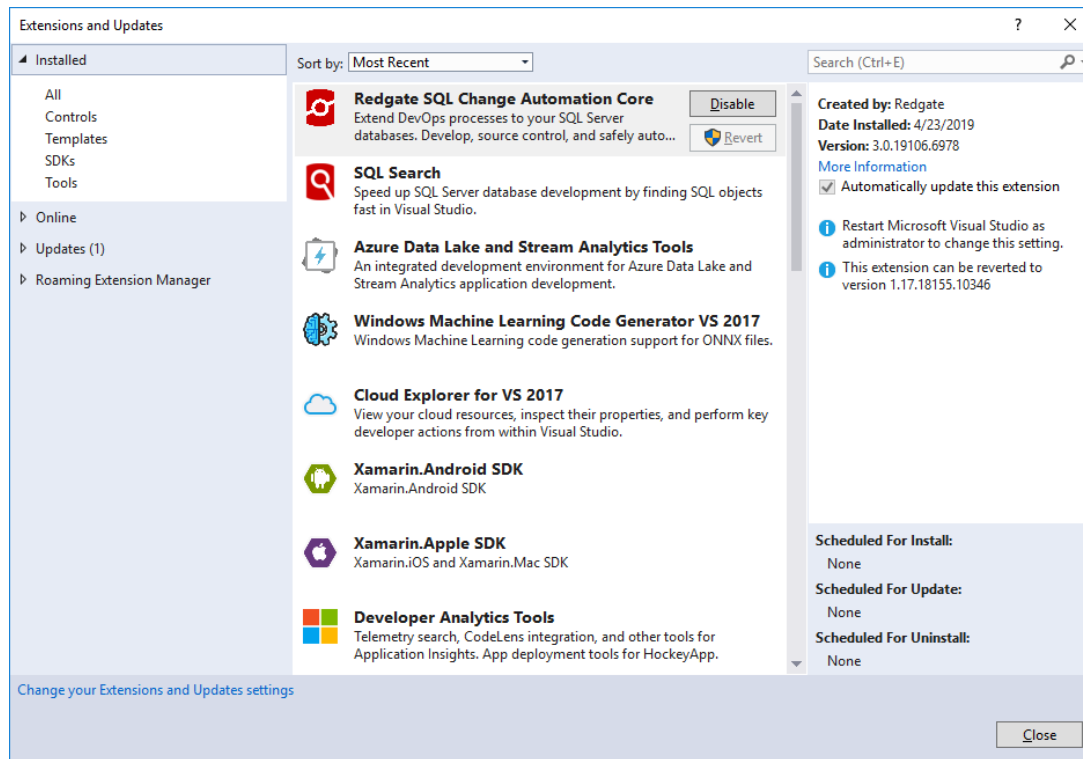
Plugin Architectures

Example



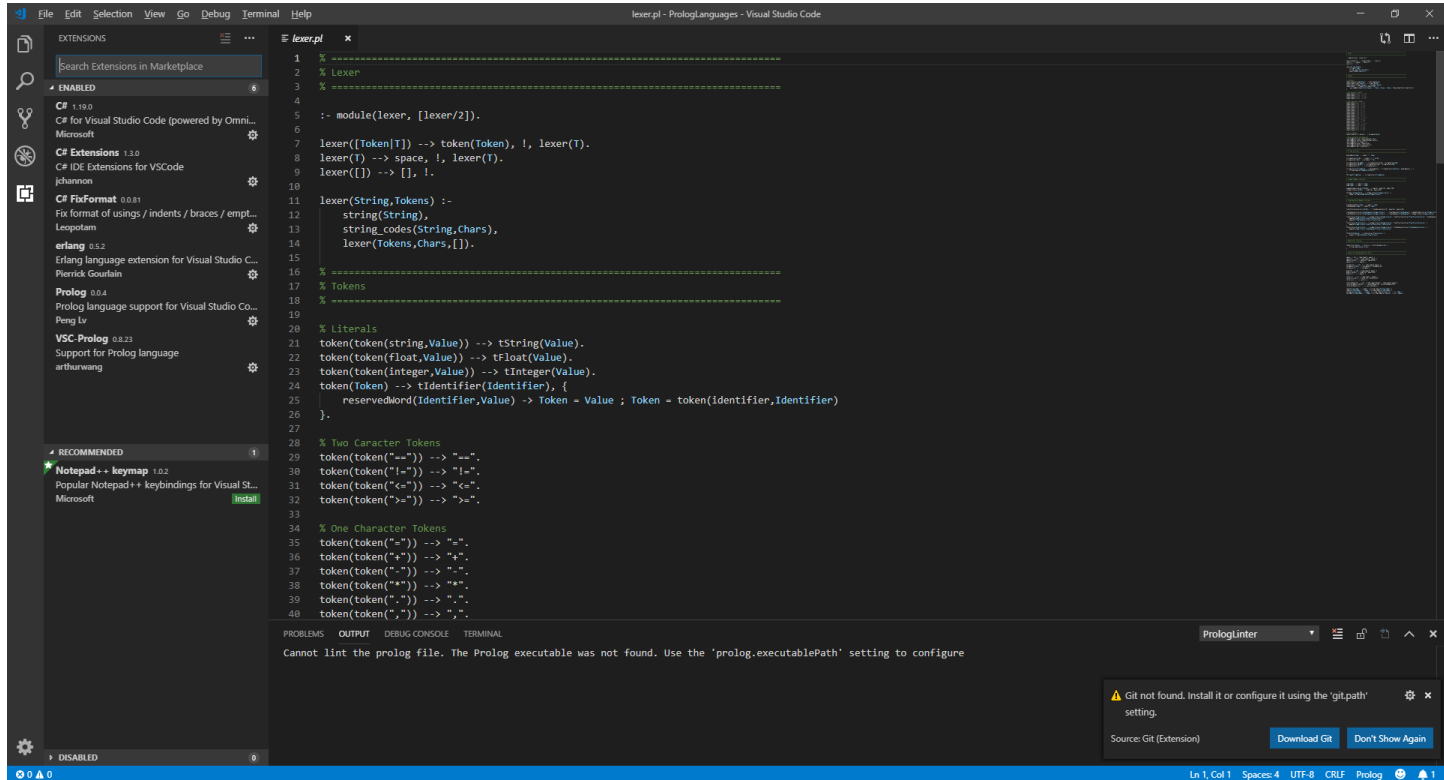
Visual Studio

Example



Visual Studio (Marketplace)

Example

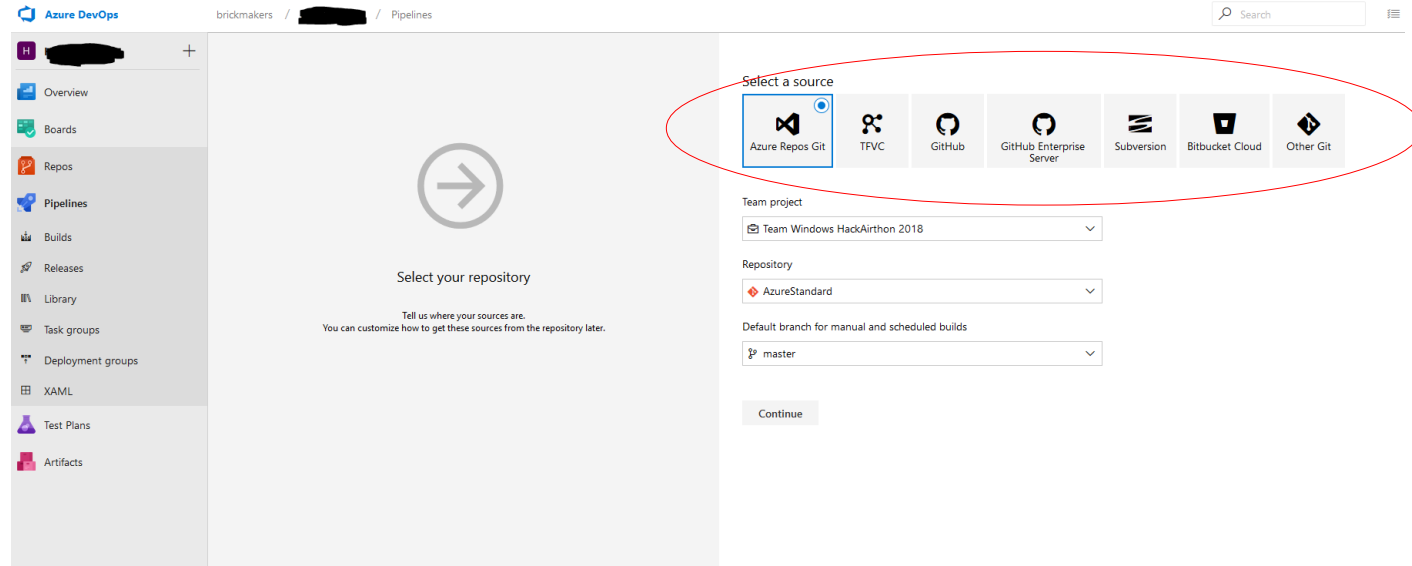


Visual Studio Code

Example

- Other (modern) IDEs also make heavy use of plugin architectures
 - IntelliJ
 - eclipse
 - Rider
 - ...

Example



Azure DevOps Build Pipelines: VCS

Example

The screenshot displays the Azure DevOps Build Pipeline configuration page. The left sidebar contains navigation links: Overview, Boards, Repos, Pipelines (selected), Builds, Releases, Library, Task groups, Deployment groups, XAML, Test Plans, and Artifacts. The main area shows the 'Build pipeline' configuration for a pipeline named 'Build pipeline'. The 'Tasks' tab is active, showing a list of tasks for 'Agent job 1'. The tasks are: 'Get sources' (AzureStandard, master), 'Agent job 1' (Run on agent), 'Use NuGet 4.4.1' (NuGet Tool Installer), 'NuGet restore' (NuGet), 'Build solution ***.sln' (Visual Studio Build), 'VsTest - testAssemblies' (Visual Studio Test), 'Publish symbols path' (Index Sources & Publish Symbols), 'Copy Files to: \$(build.artifactstagingdirectory)' (Copy Files), and 'Publish Artifact: drop' (Publish Build Artifacts). The right sidebar shows a list of build definitions.

Overview
Boards
Repos
Pipelines
Builds
Releases
Library
Task groups
Deployment groups
XAML
Test Plans
Artifacts

Tasks Variables Triggers Options Retention History | Save & queue Discard

Pipeline
Build pipeline

Get sources
AzureStandard master

Agent job 1
Run on agent

Use NuGet 4.4.1
NuGet Tool Installer

NuGet restore
NuGet

Build solution ***.sln
Visual Studio Build

VsTest - testAssemblies
Visual Studio Test

Publish symbols path
Index Sources & Publish Symbols

Copy Files to: \$(build.artifactstagingdirectory)
Copy Files

Publish Artifact: drop
Publish Build Artifacts

Azure DevOps Build Pipelines: Agent Jos

Example

Higher-order functions

Example

```
const twice = (f, v) => f(f(v));  
const add3 = v => v + 3;  
  
twice(add3, 7); // 13
```

```
Func<Func<int,int>,Func<int,int>> twice = f => x => f(f(x));  
Func<int,int> plusThree = x => x + 3;  
  
Console.WriteLine(twice(plusThree)(7)); // 13
```

Higher-order functions

Example

Gang of 4 Patterns

Template Method

Strategy

Command

Template Method

Visitor

Thanks!

References

- Robert C. Martin. 2003. Agile Software Development, Principles, Patterns, and Practices, Prentice Hall.
-