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
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
```
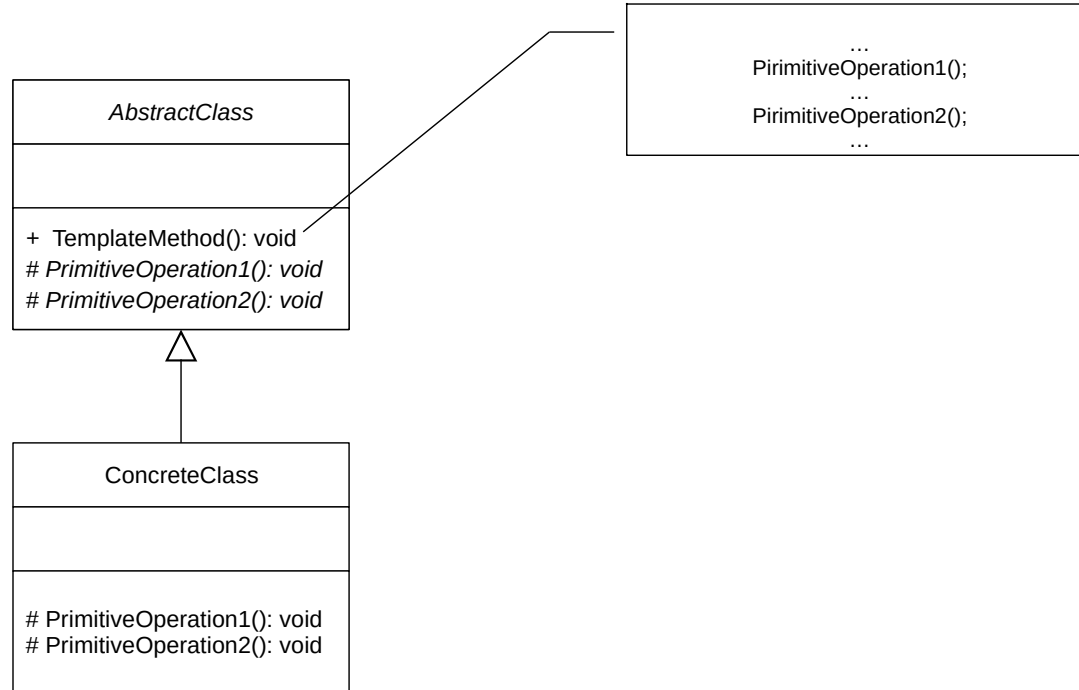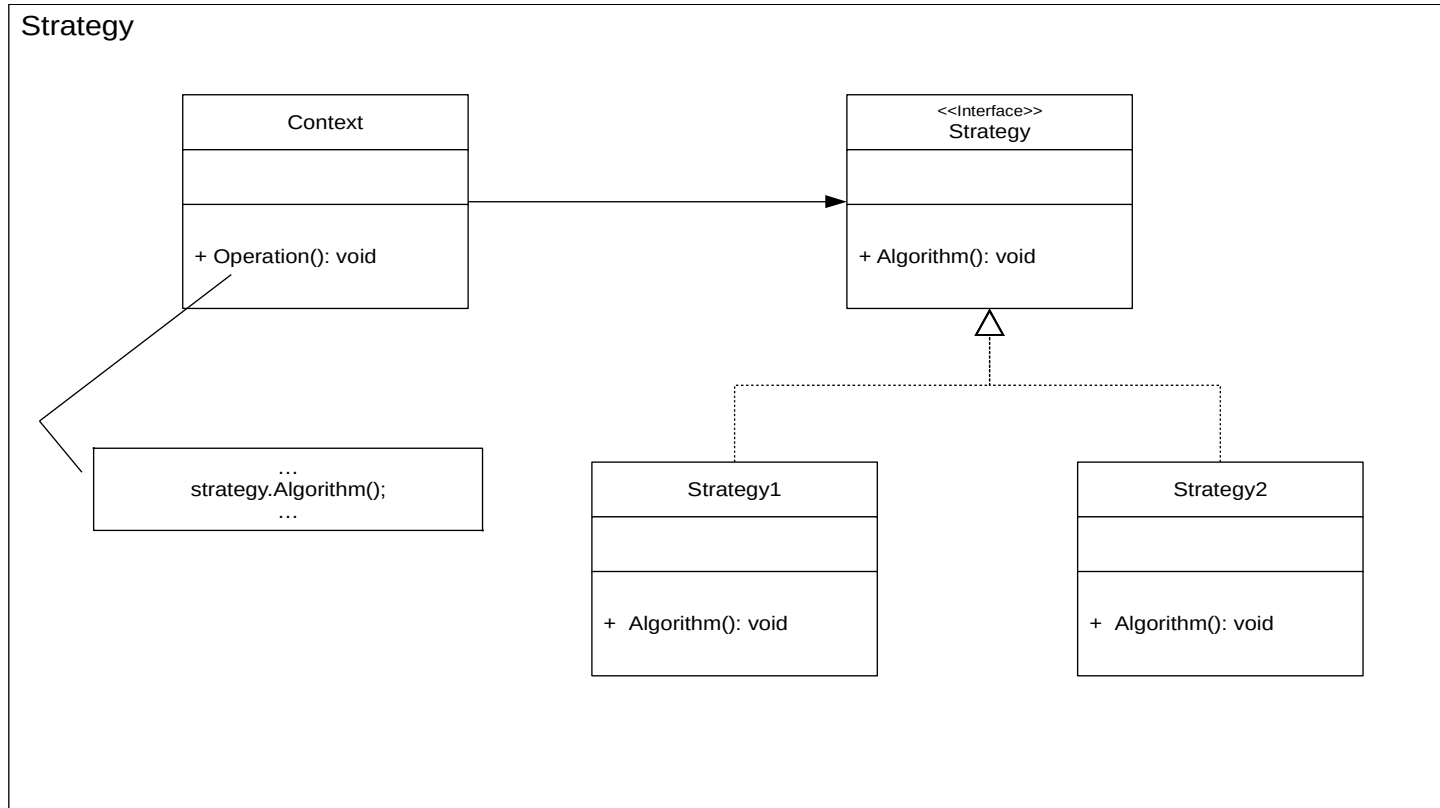
Higher-order functions

Example

*A non-exclusive list of Open/Closed
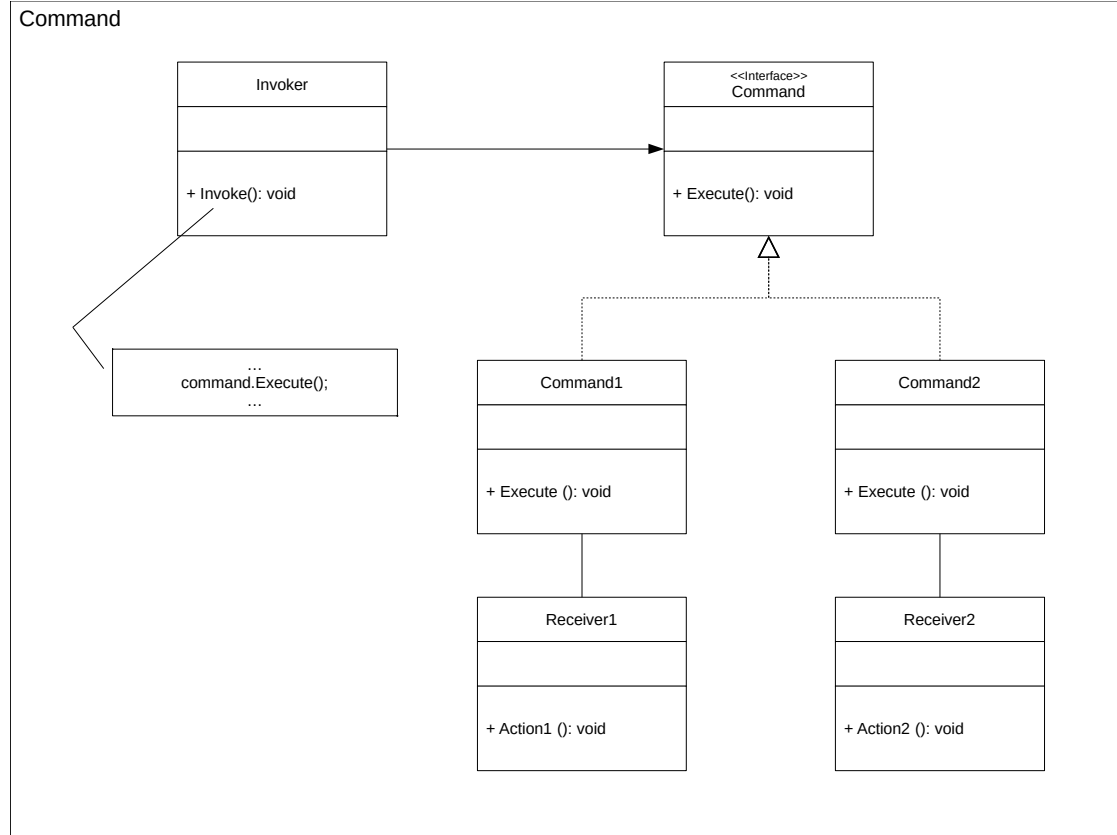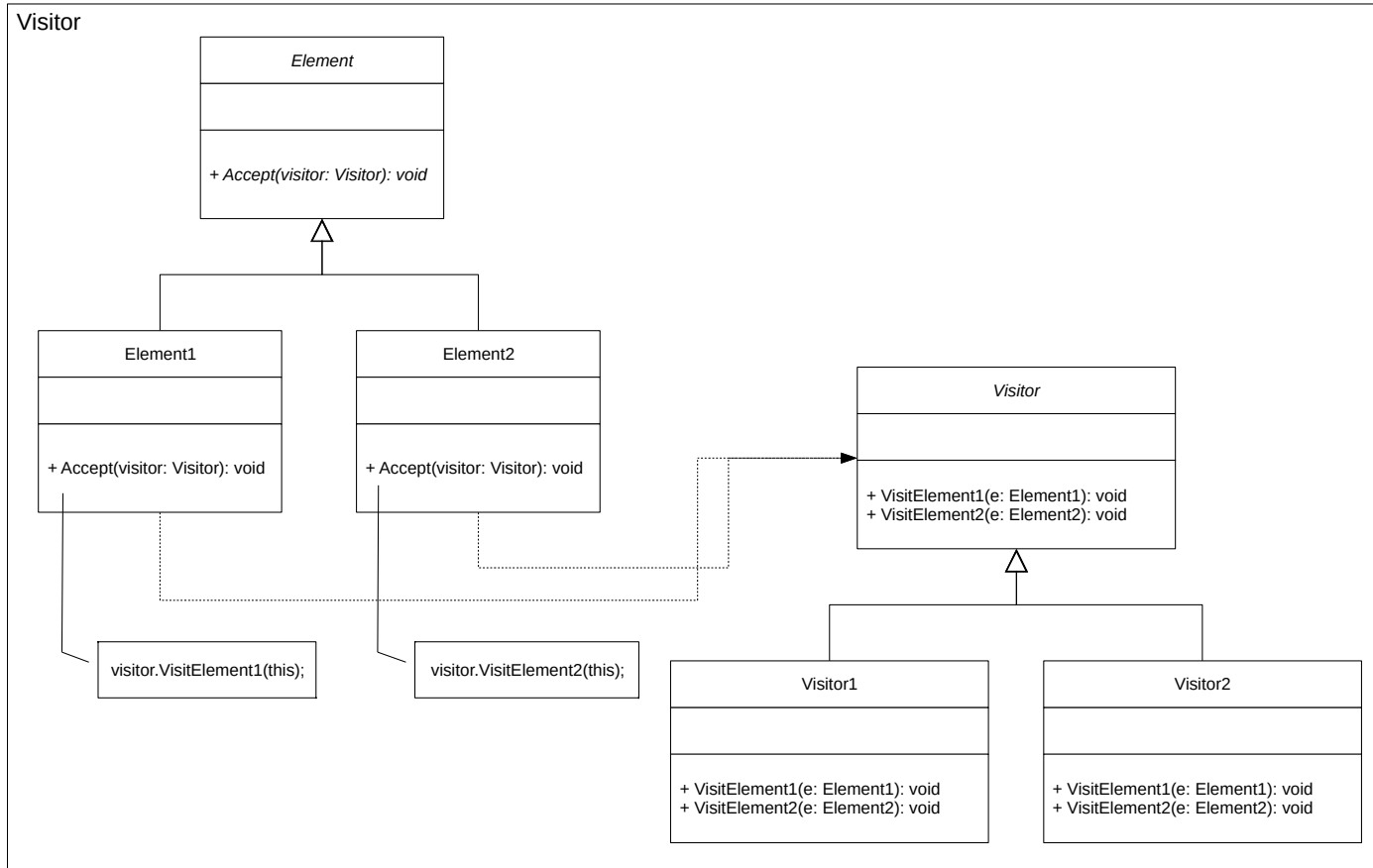Gang of 4 Patterns*

Example

Template Method

Example

Strategy

Context
+ Operation(): void

<<Interface>> Strategy
+ Algorithm(): void

… strategy.Algorithm(); …

Strategy1
+ Algorithm(): void

Strategy2
+ Algorithm(): void

Example

Command

Invoker
+ Invoke(): void

<<Interface>> Command
+ Execute(): void

... command.Execute(); ...

Command1
+ Execute (): void

Command2
+ Execute (): void

Receiver1
+ Action1 (): void

Receiver2
+ Action2 (): void

Example

Example
Plugin Architectures

Example
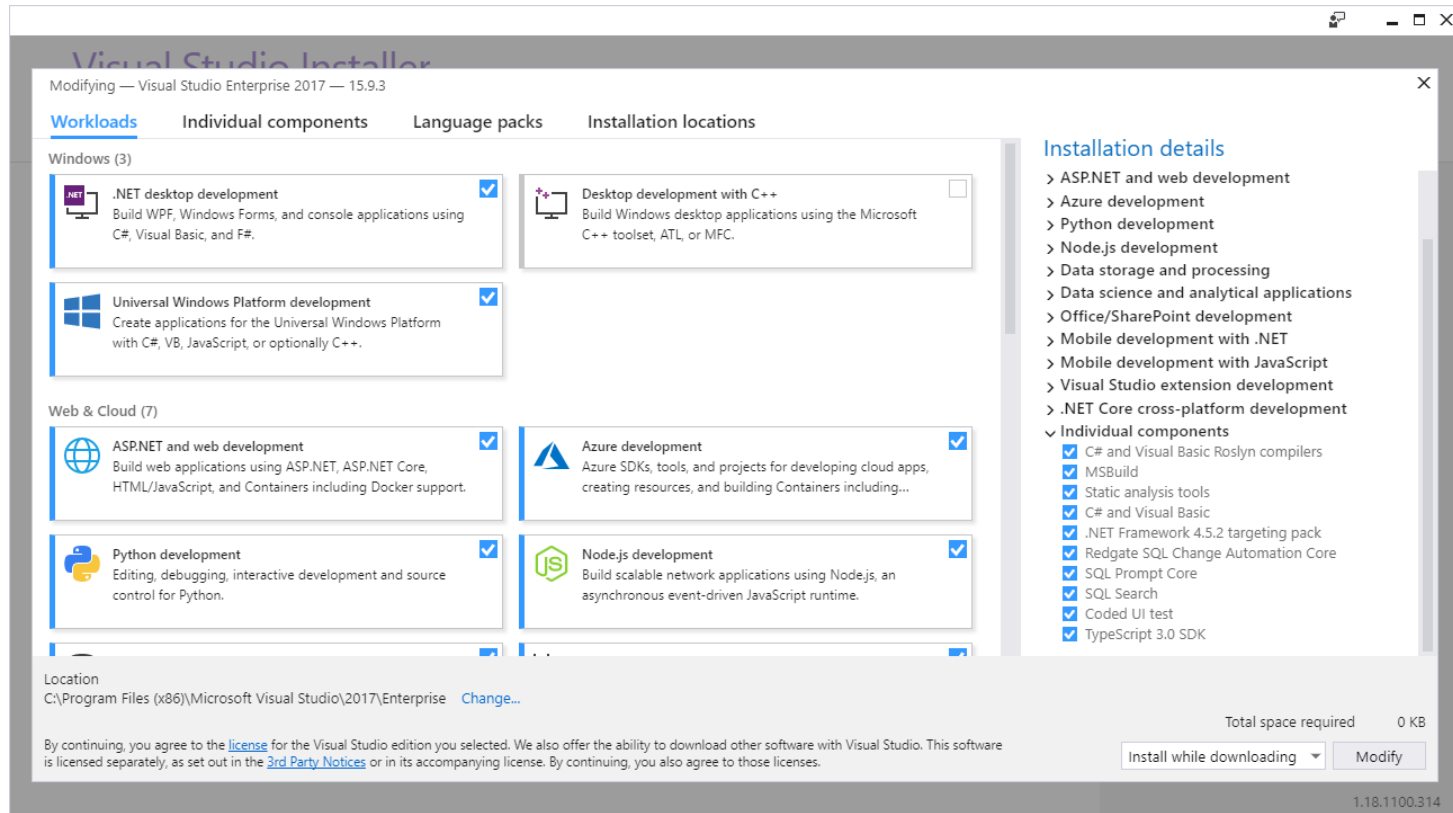
Visual Studio

Example
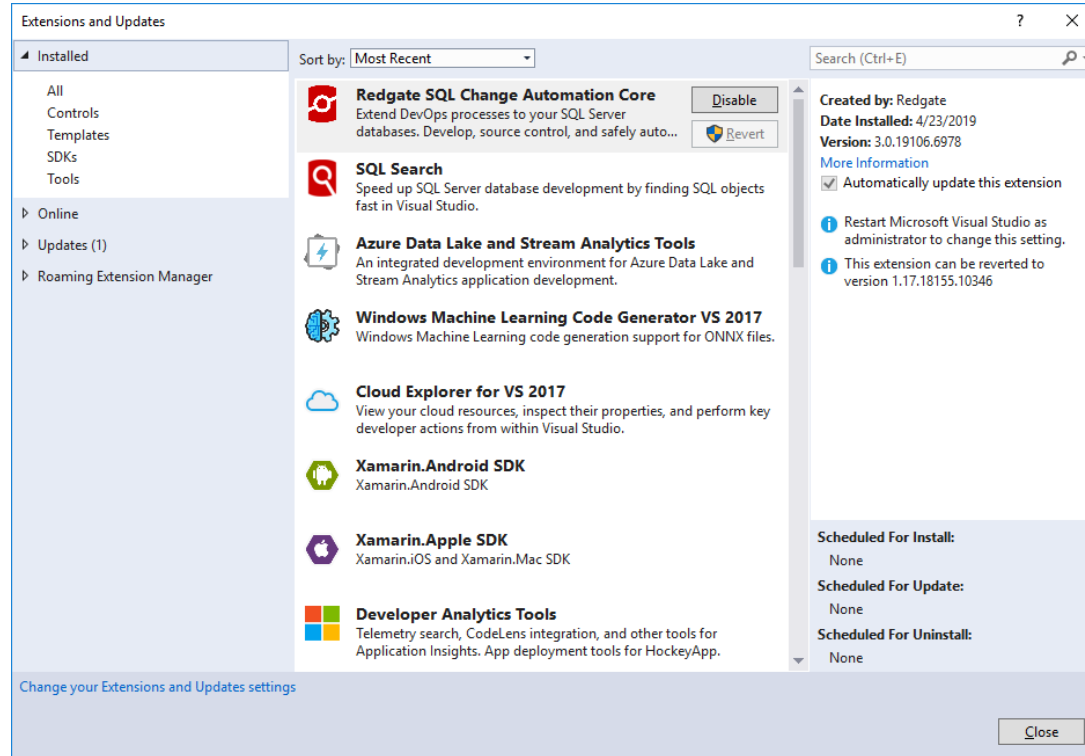
Visual Studio (Marketplace)

Example
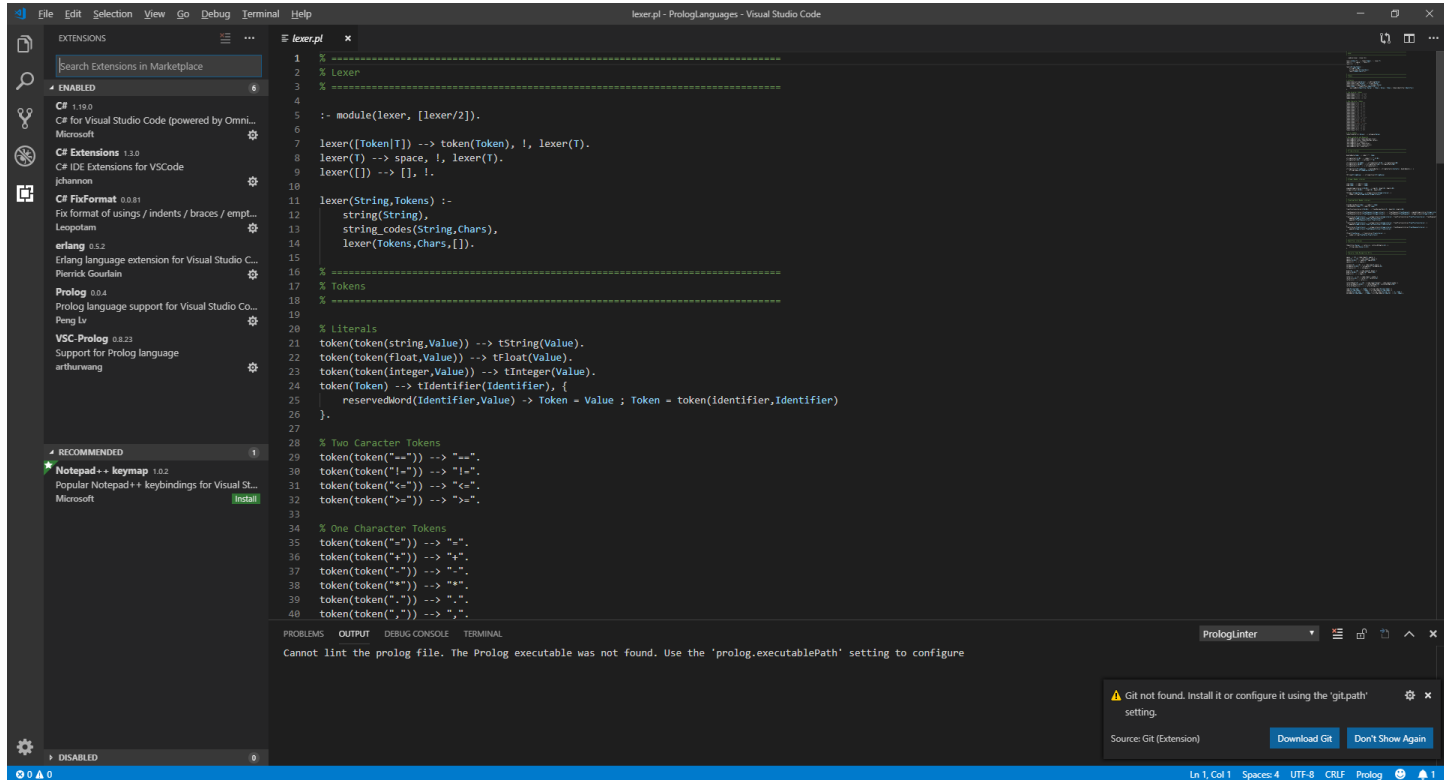
Visual Studio Code

Example

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

Example
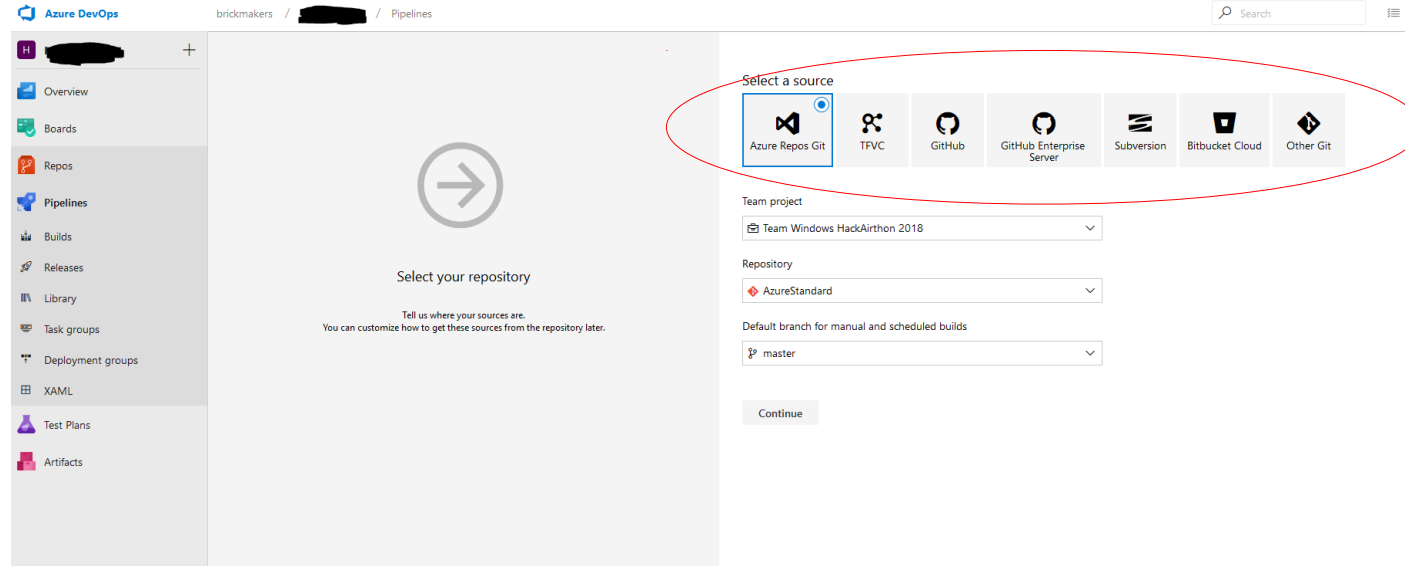
Azure DevOps Build Pipelines: VCS

Example

Azure DevOps Build Pipelines: Agent Jos

Thanks!

References

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

-