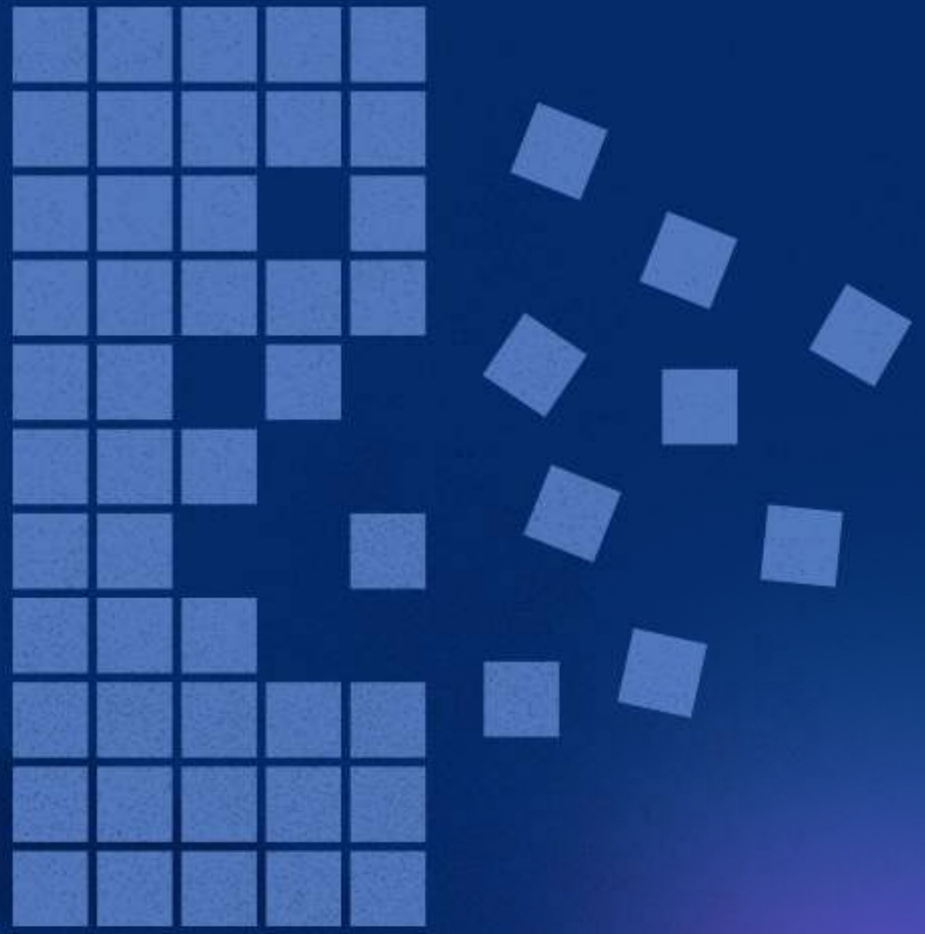
Breaking down the (dependency) monolith



Marco Fogliatto

January, 2024

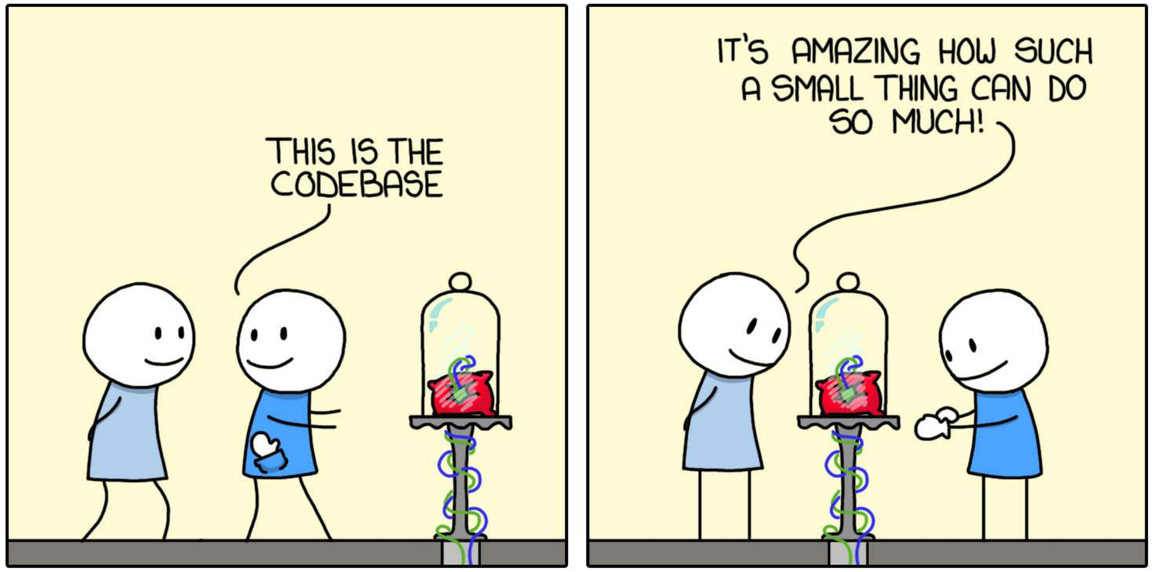
Hi, I’m Marco

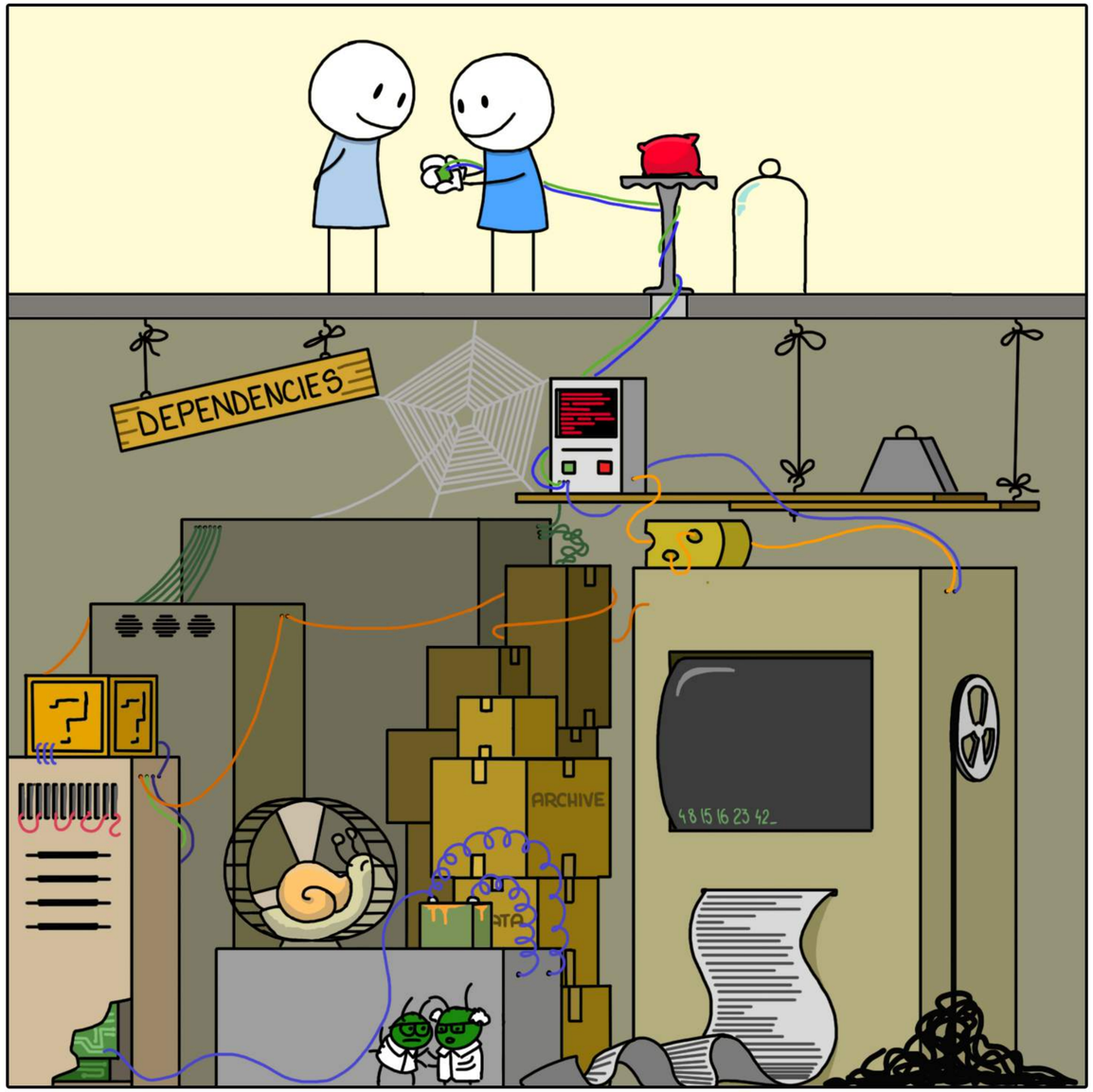
marcofogliatto

Software Engineer
@ Microsoft

From Rosario,
Argentina

IMPLEMENTATION





Source: https://www.monkeyuser.com/2018/implementation/

What is a dependency monolith?

What is a dependency monolith?

A software project that has accumulated **a large and complex network of dependencies** relative to its own size **that reduces its change agility and increases maintenance efforts** beyond acceptable limits.

Dependency monolith == a bad dependency management strategy

Why is this a problem?

“Dependency management, and therefore the SOLID principles are at the foundation of the -ilities that software developers desire.”

“Poor dependency management leads to code that is hard to change, fragile, and non-reusable.”

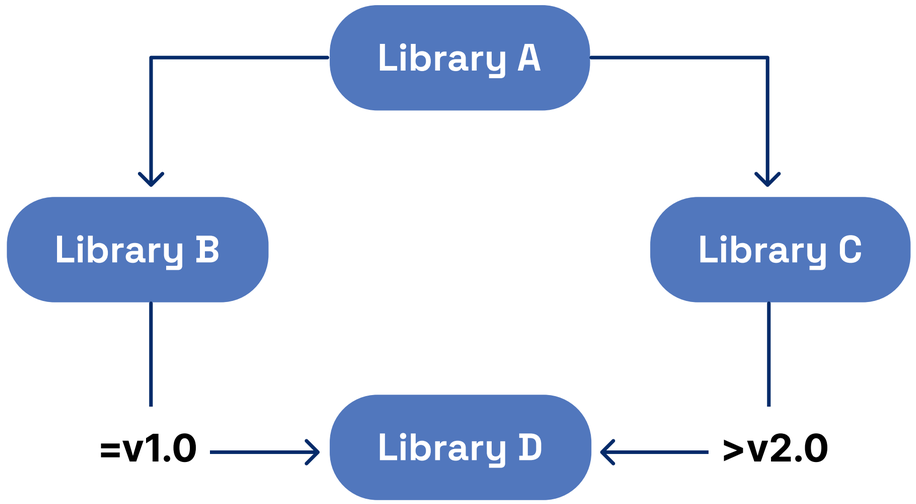
Uncle Bob
(Robert Martin)

Source: ArticleS.UncleBob.PrinciplesOfOod (butunclebob.com)

What are the symptoms?

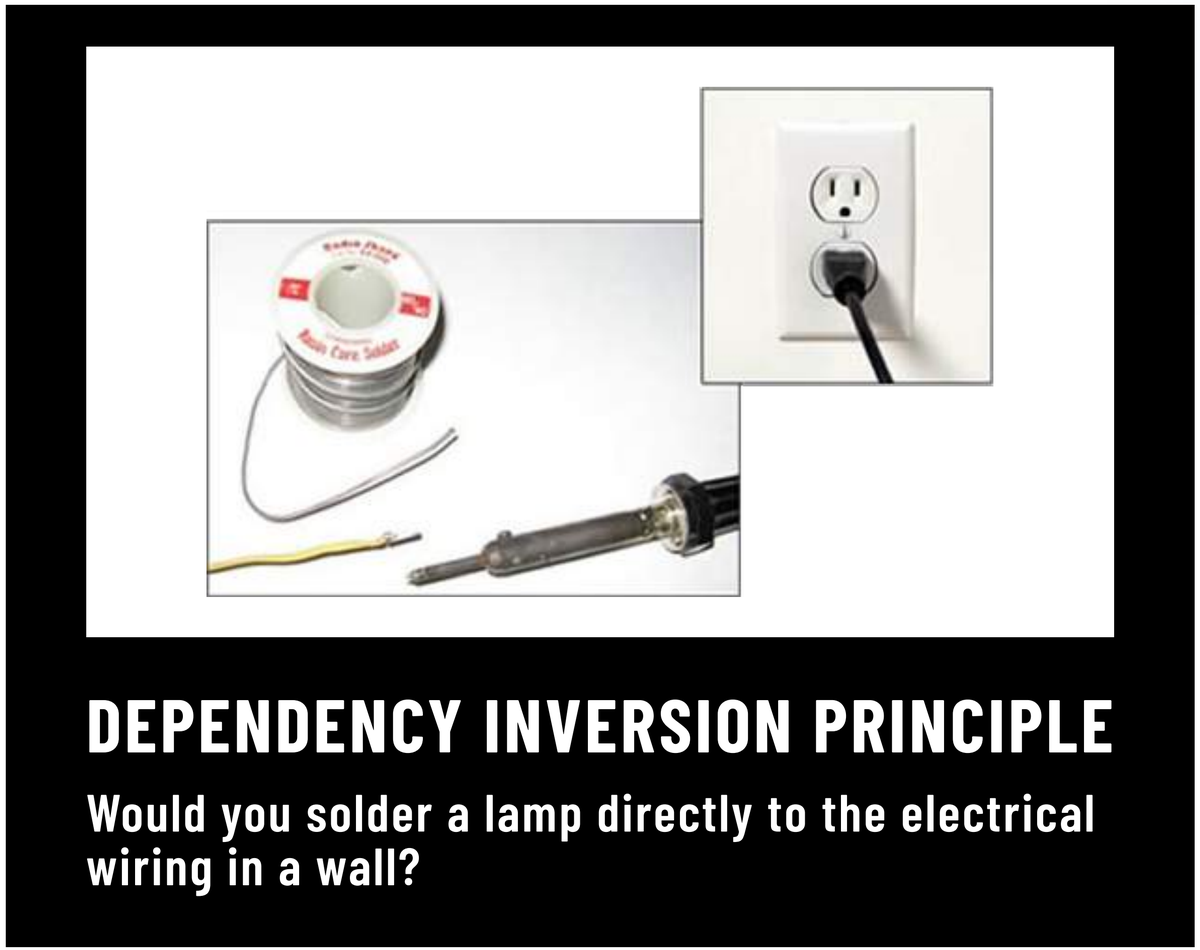
Dependency Hell: The Diamond Problem

Dependency Hell: The Diamond Problem



- Version constraint as higher or equal to (>= 1.2.3.4).
- Version constraint as exactly equal to (= 1.2.3.4).
- Version constraint as a range or as floating (1.0 ≤ x < 2.0).

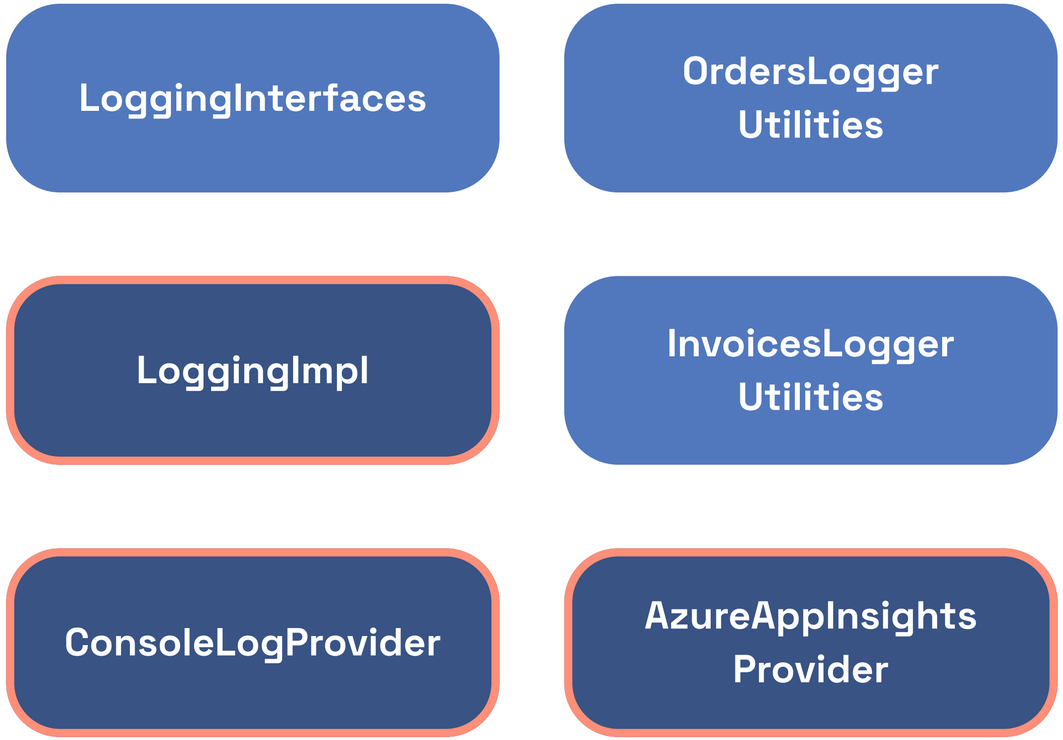
Libraries Impose “Implementation Specific” Constraints



Source: 7 Software Development Principles Boiled Down to Memes | by Jose Granja | Better Programming

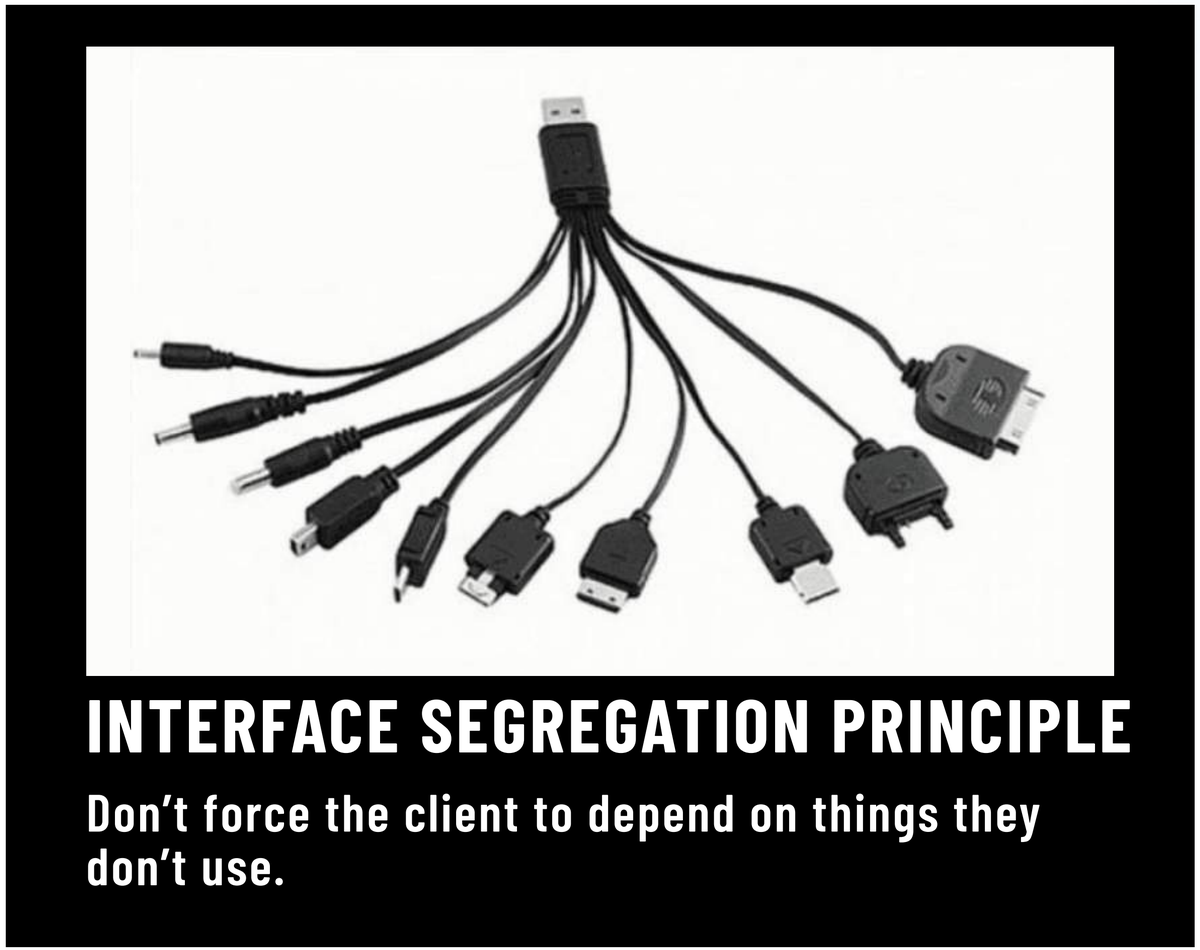
Libraries Impose “Implementation Specific” Constraints

LoggingLibrary



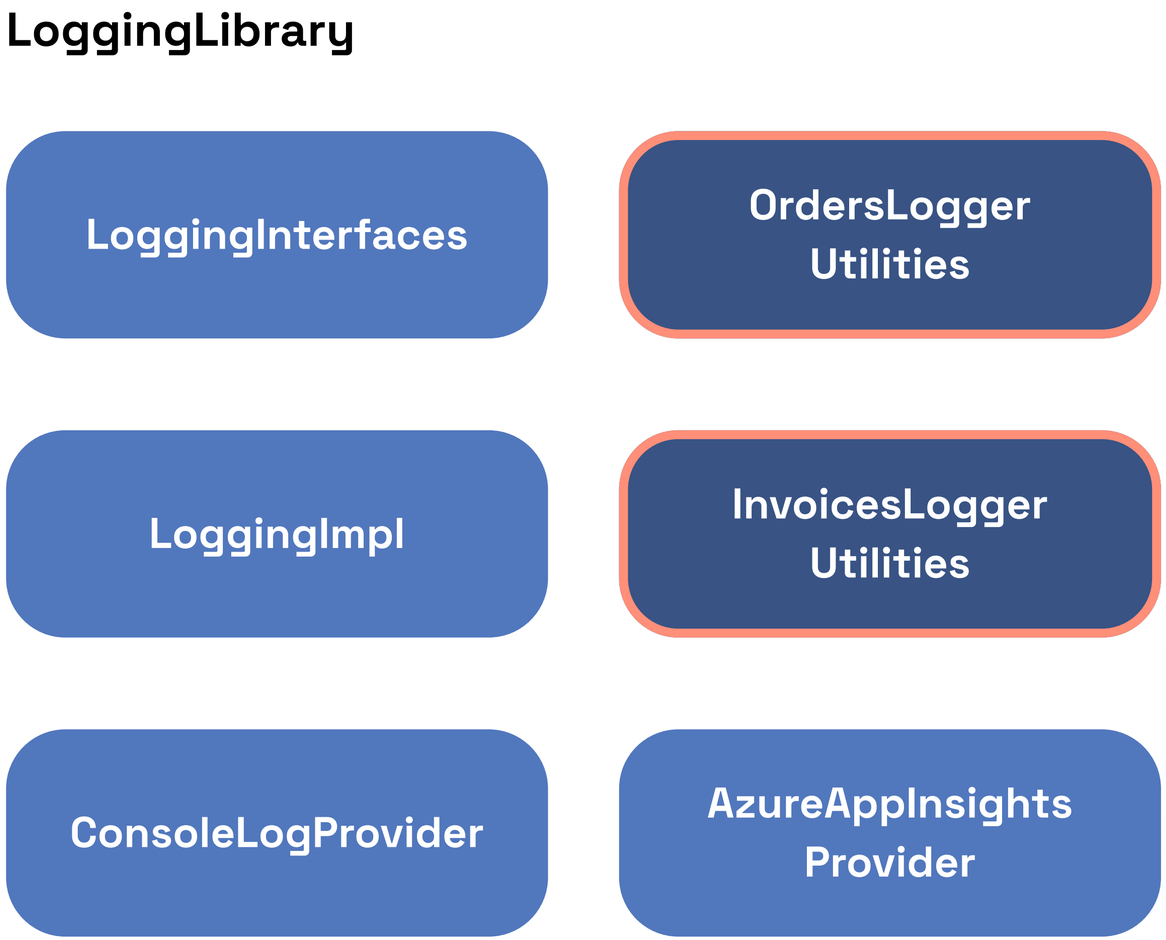
- Technology Constraints (too much opinion)
- Version Constraints

Libraries are Multi-Purposed and Include Unrelated Aspects

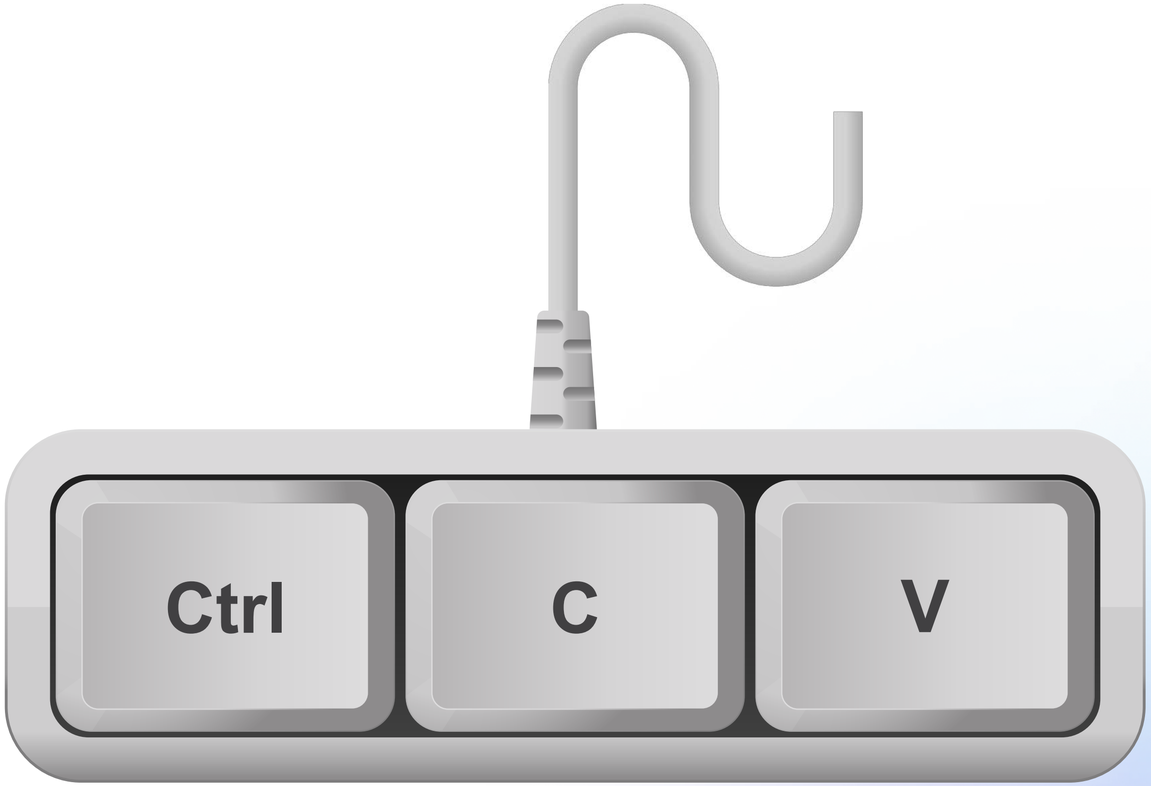


Source: 7 Software Development Principles Boiled Down to Memes | by Jose Granja | Better Programming

Libraries are Multi-Purposed and Include Unrelated Aspects

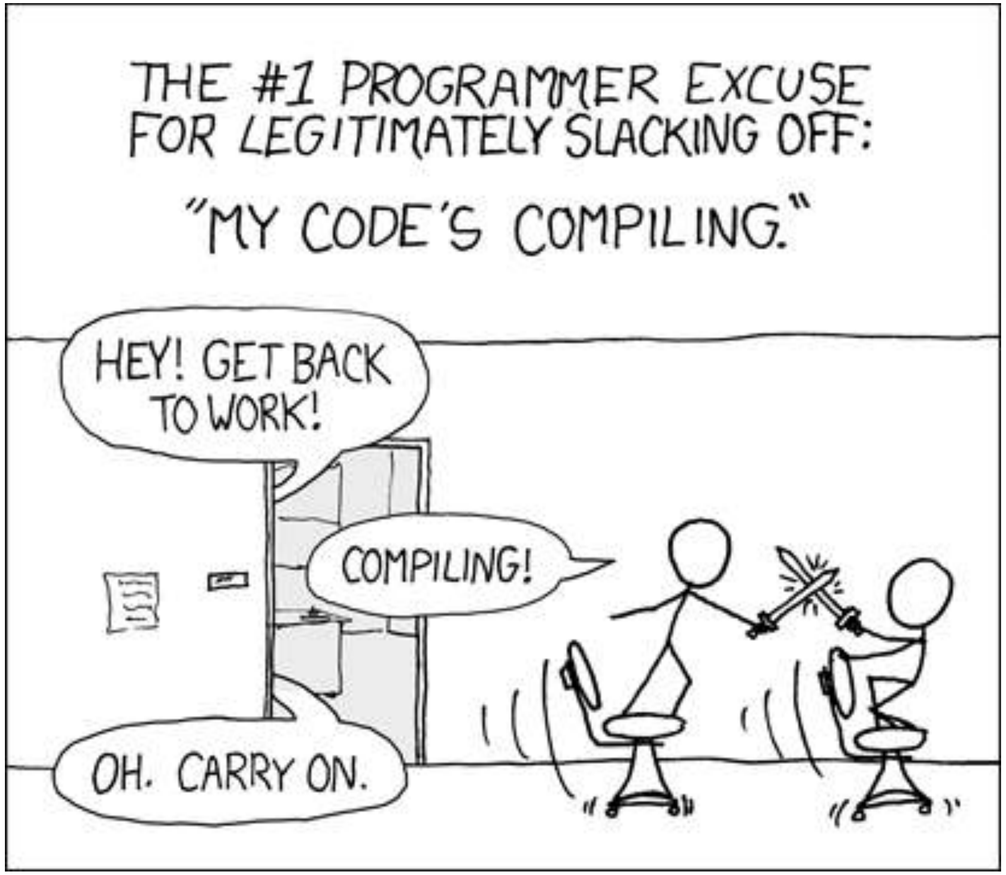


Code is Duplicated and Scattered



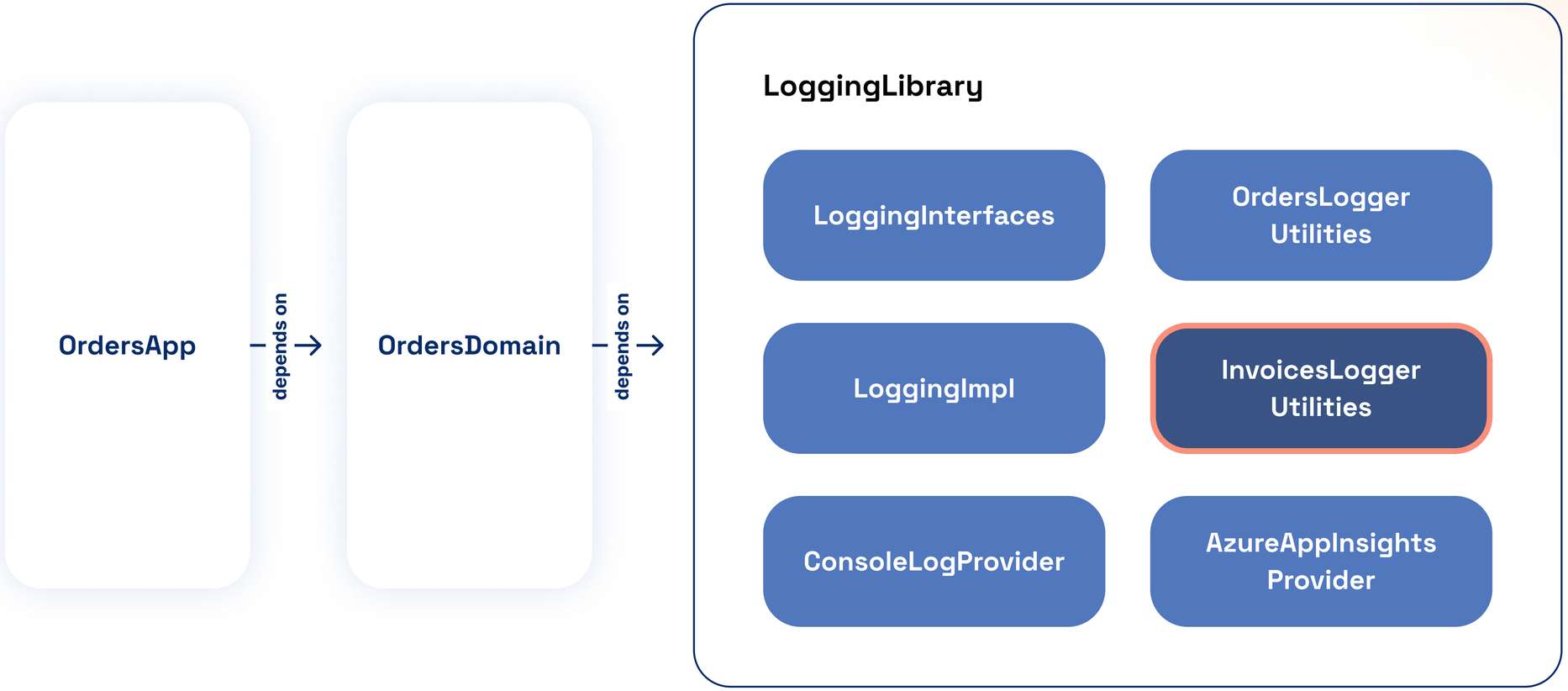
Source: Copy and paste - A programming horror story - James Croft

Long Restore and Build Times



Source: xkcd: Compiling

Long Restore and Build Times

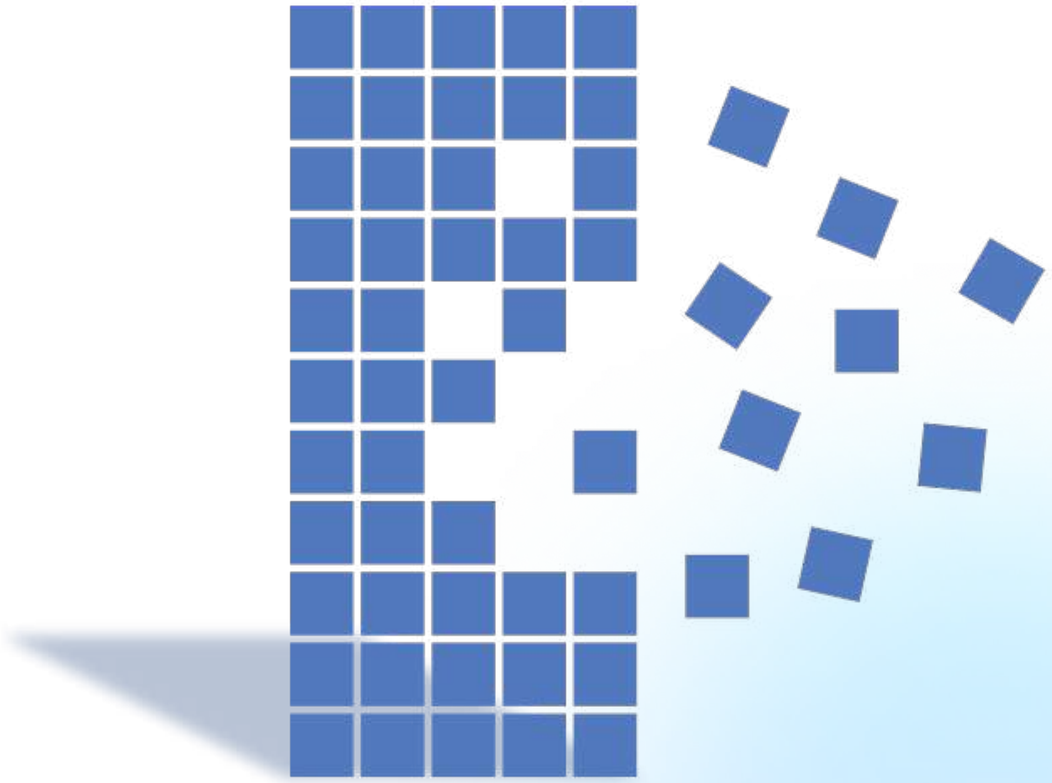


We have a dependency monolith,

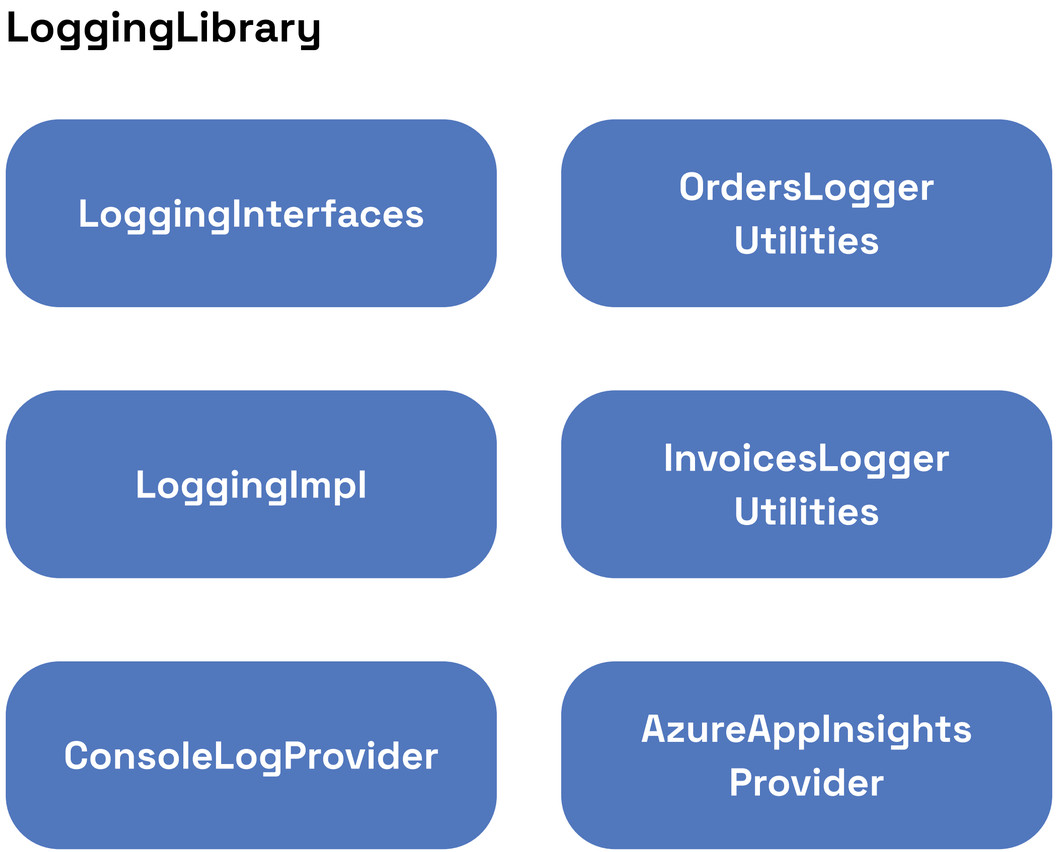
what can we do to break it down?

Refactor, refactor, refactor!

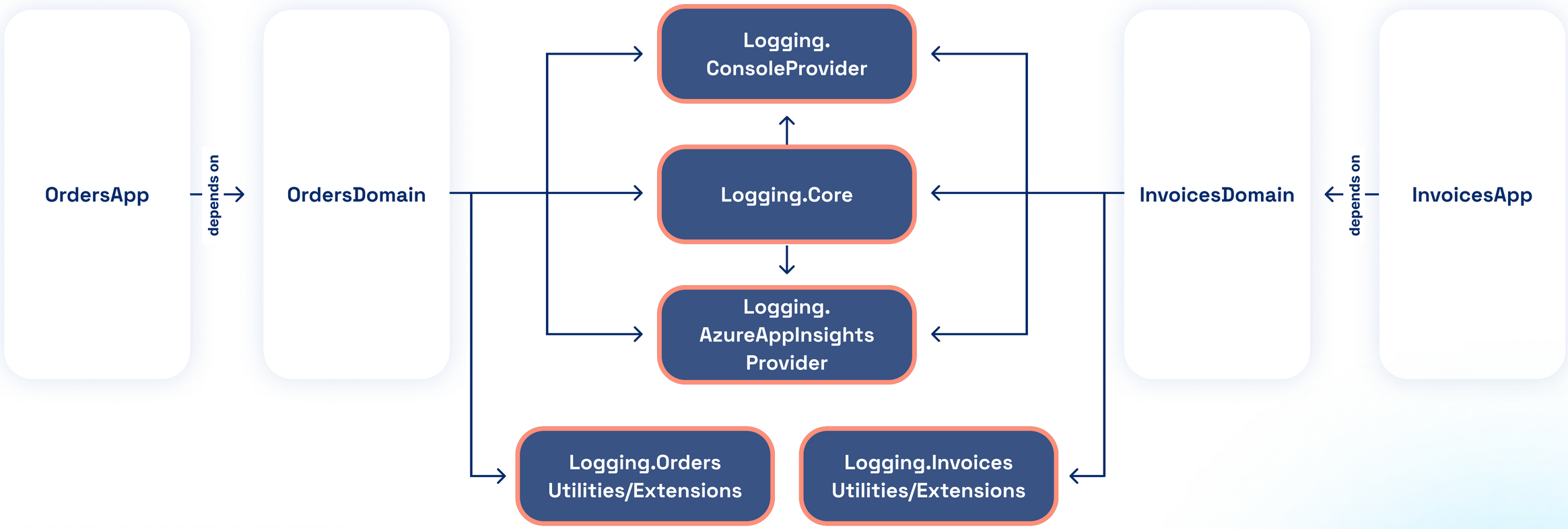
- Split Libraries into Aspects or “Slices”
- Decouple Abstractions/APIs from Implementations
- Review Dependencies and Remove Unnecessary Ones



Split Libraries into Aspects or “Slices”

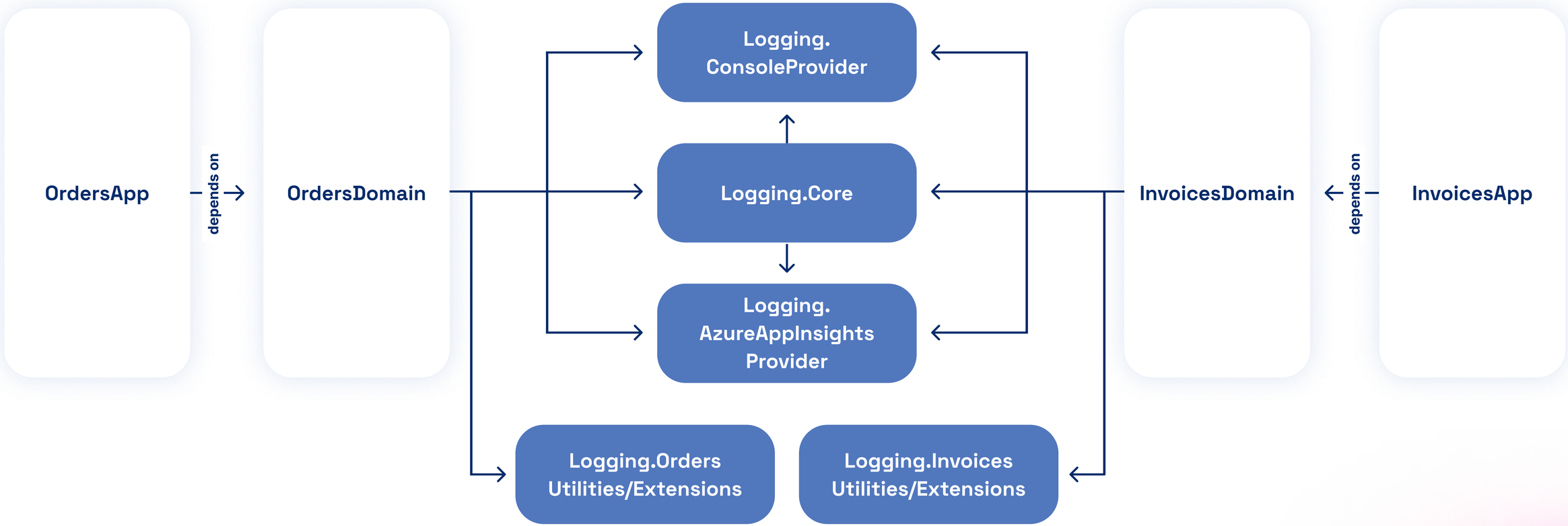


Split Libraries into Aspects or “Slices”

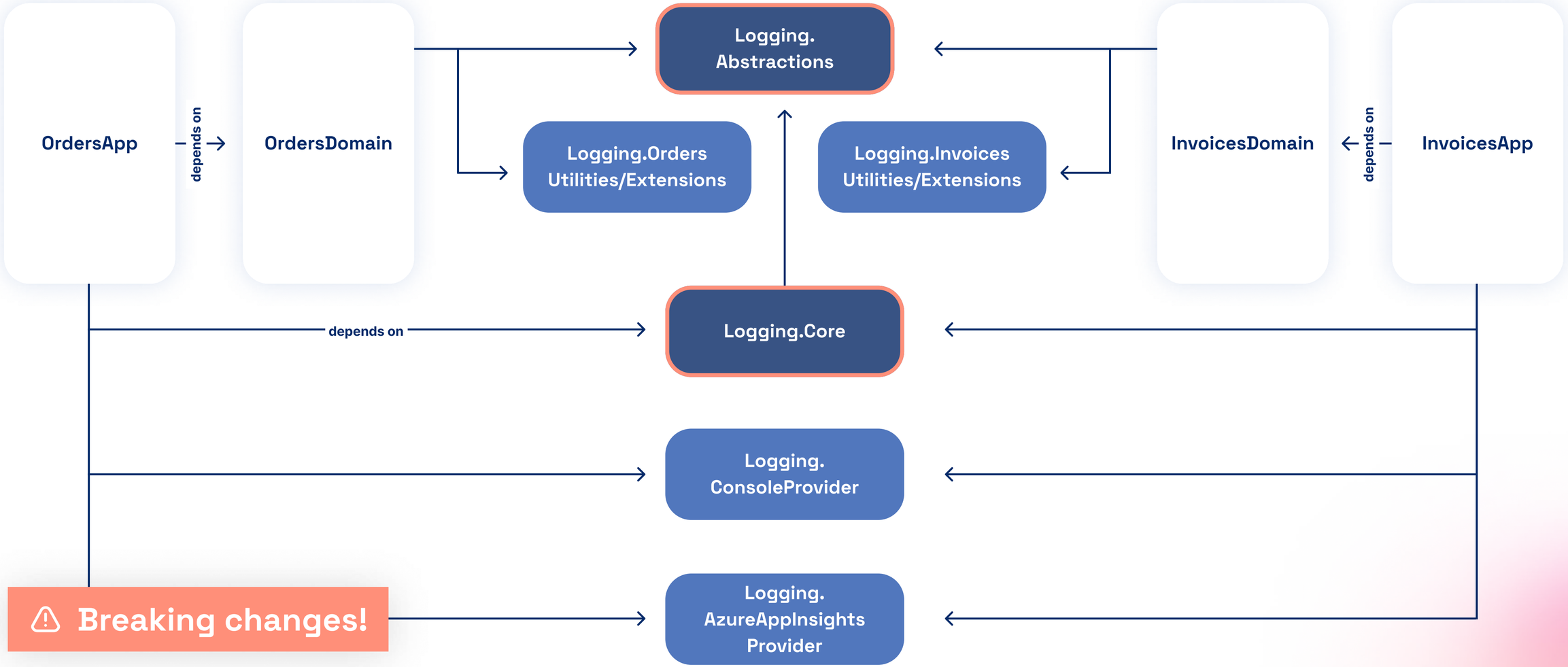


⚠ Breaking changes!

Decouple Abstractions/APIs from Implementations



Decouple Abstractions/APIs from Implementations



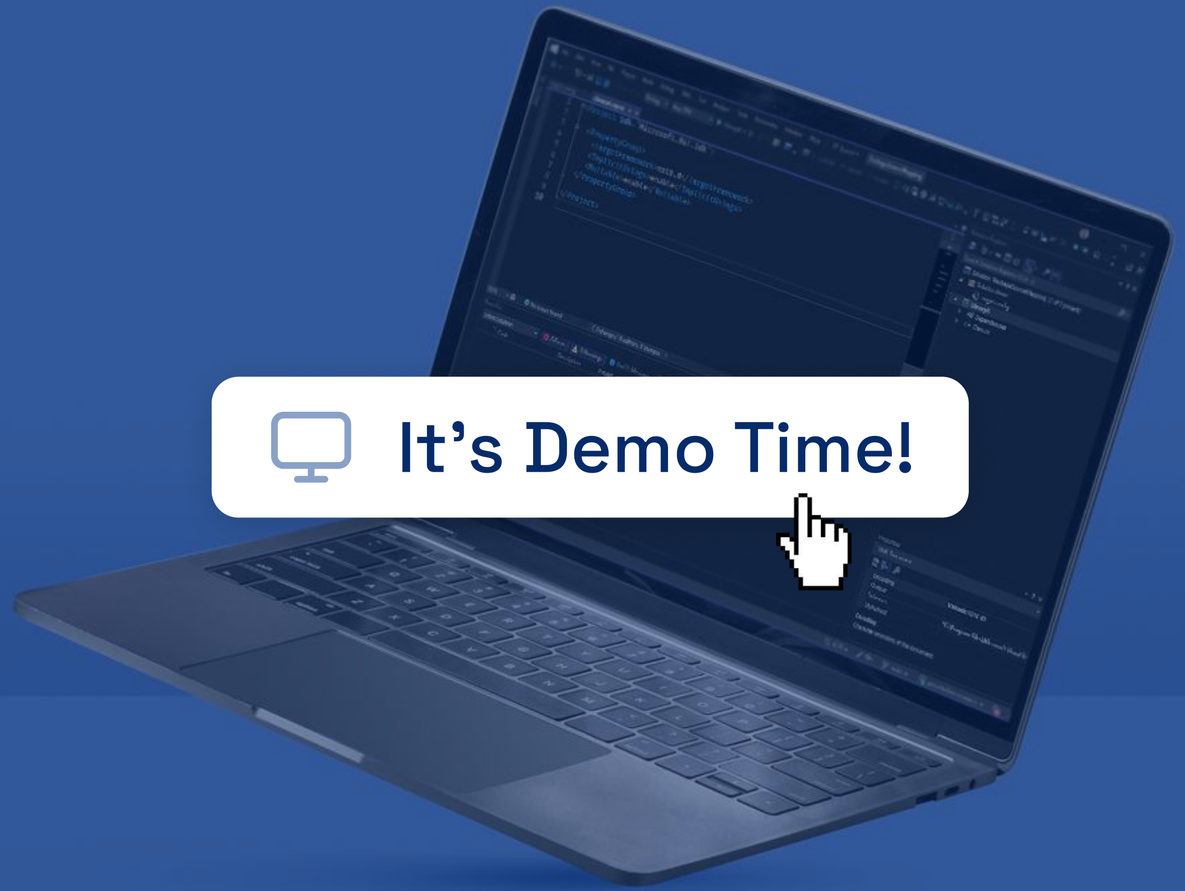
Review Dependencies and Remove Unnecessary Ones

Analyze and remove unused dependencies using:

Visual Studio

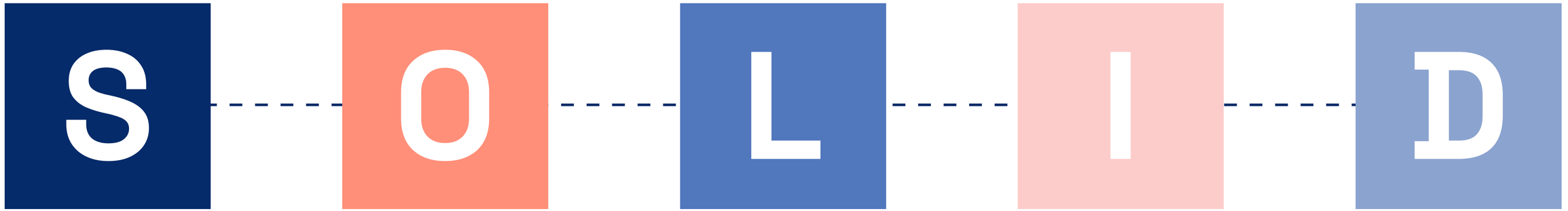
ReSharper

Review Dependencies and Remove Unnecessary Ones



What can we do to avoid building a dependency monolith?

Embrace the Design Principles from the Beginning

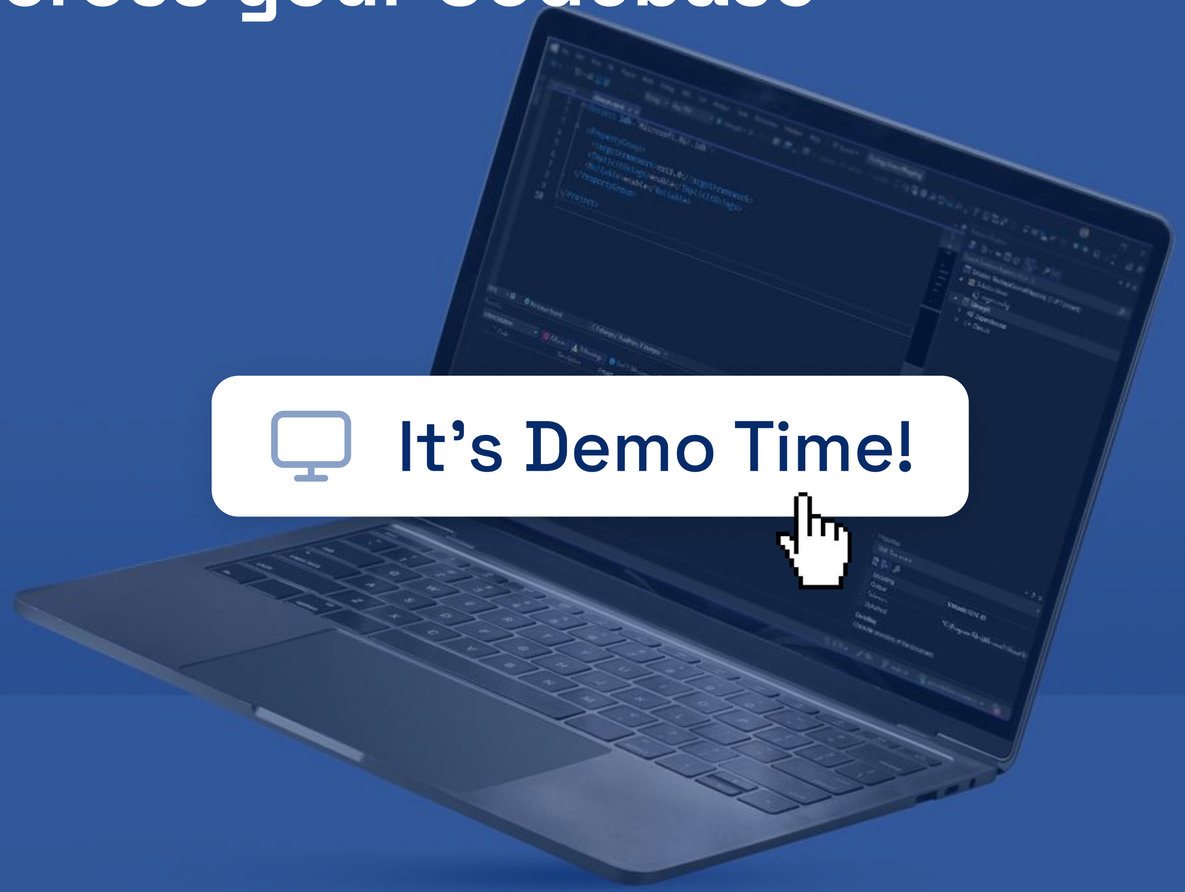


Source: SOLID Principles | Thoughtworks

Enforce your Dependency Management Strategy Consistently Across your Codebase

- Leaving unused references -> ReferenceTrimmer
- Introducing illegal dependencies -> MSBuild Tasks

Enforce your Dependency Management Strategy Consistently Across your Codebase

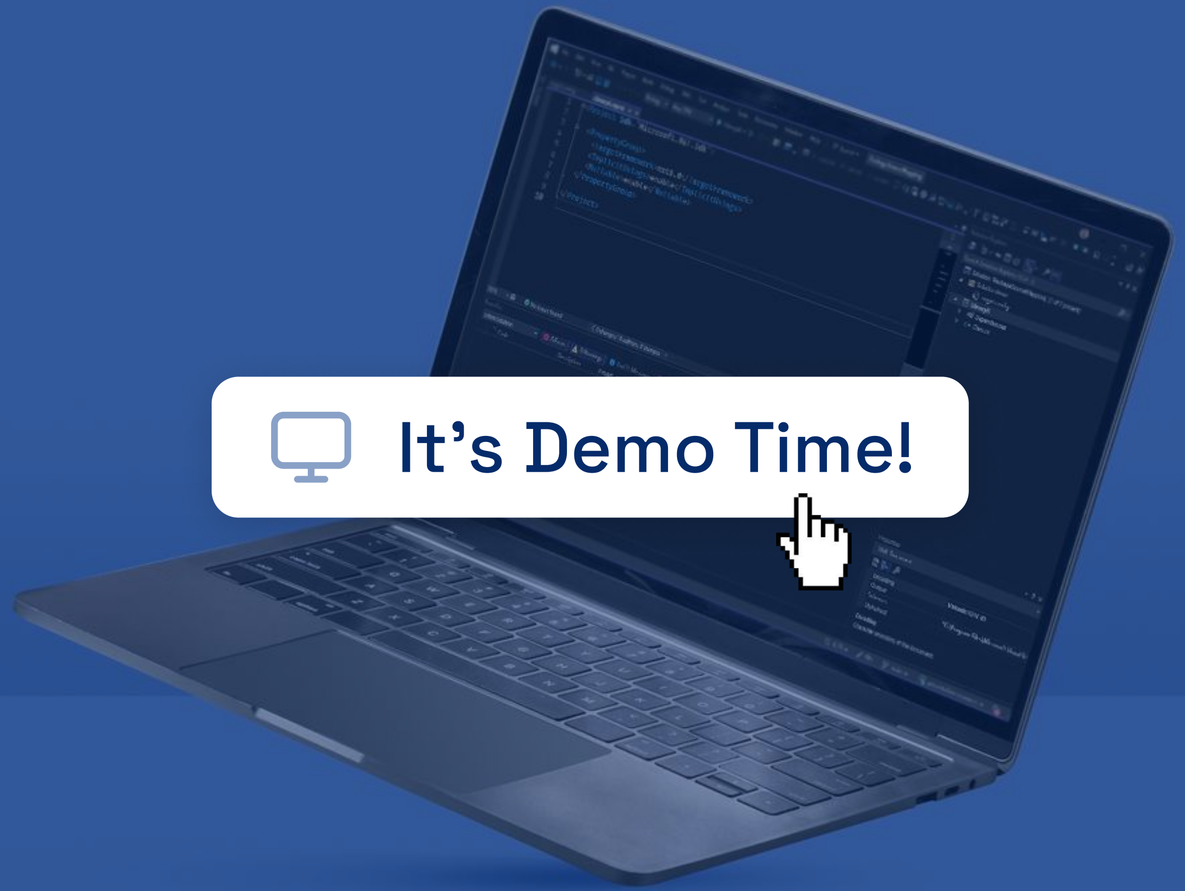


Unify Package Versions Used Across your Codebase

Central Package Management

- All PackageReferences will use the centralized version
- PackageReferences can still override the centralized version
- Transitive packages can also use a centralized version (transitive pinning)

Unify Package Versions Used Across your Codebase

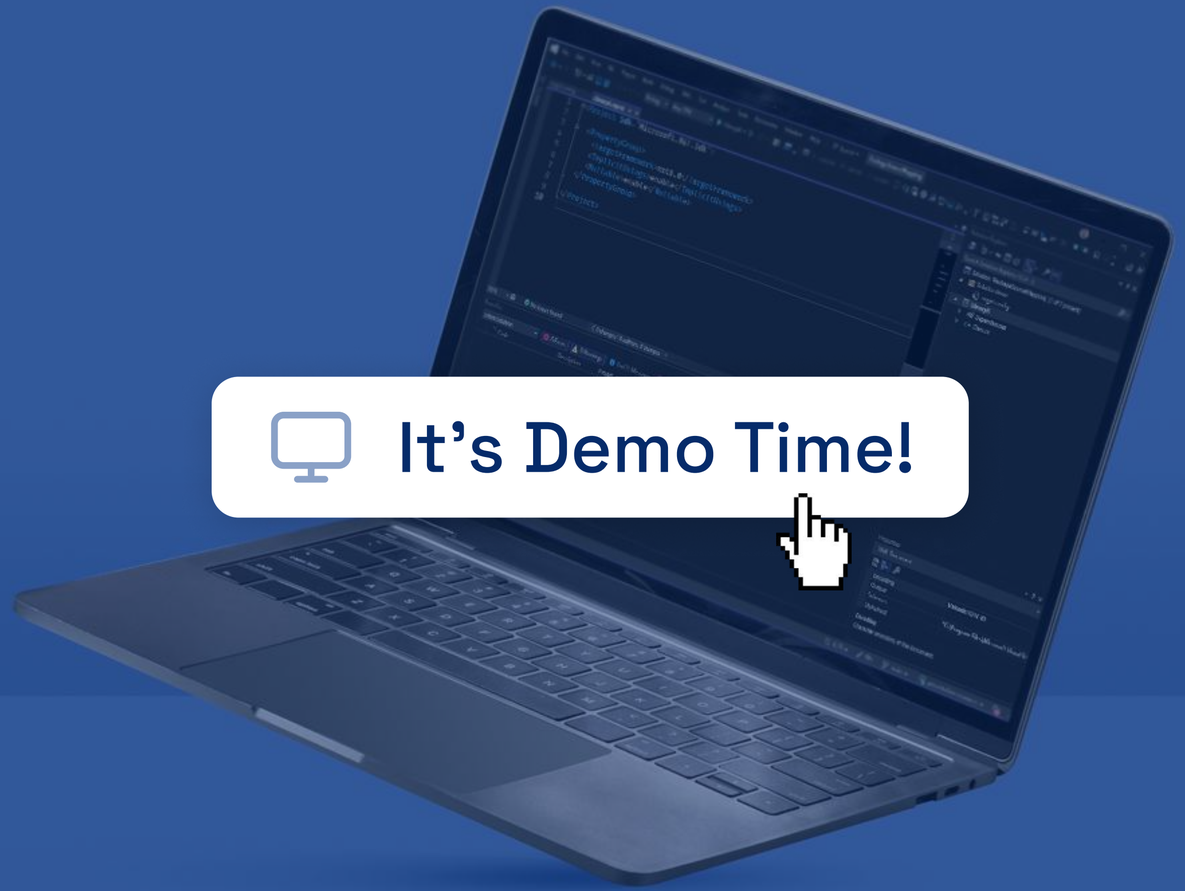


Enable Package Source Mapping

Package Source Mapping

- Maps a package ID or pattern to a specific source
- Adds extra layer of security to the software supply chain
- Allows restore and build reproducibility

Enable Package Source Mapping



Avoid Breaking Changes in Minor/Patch Versions

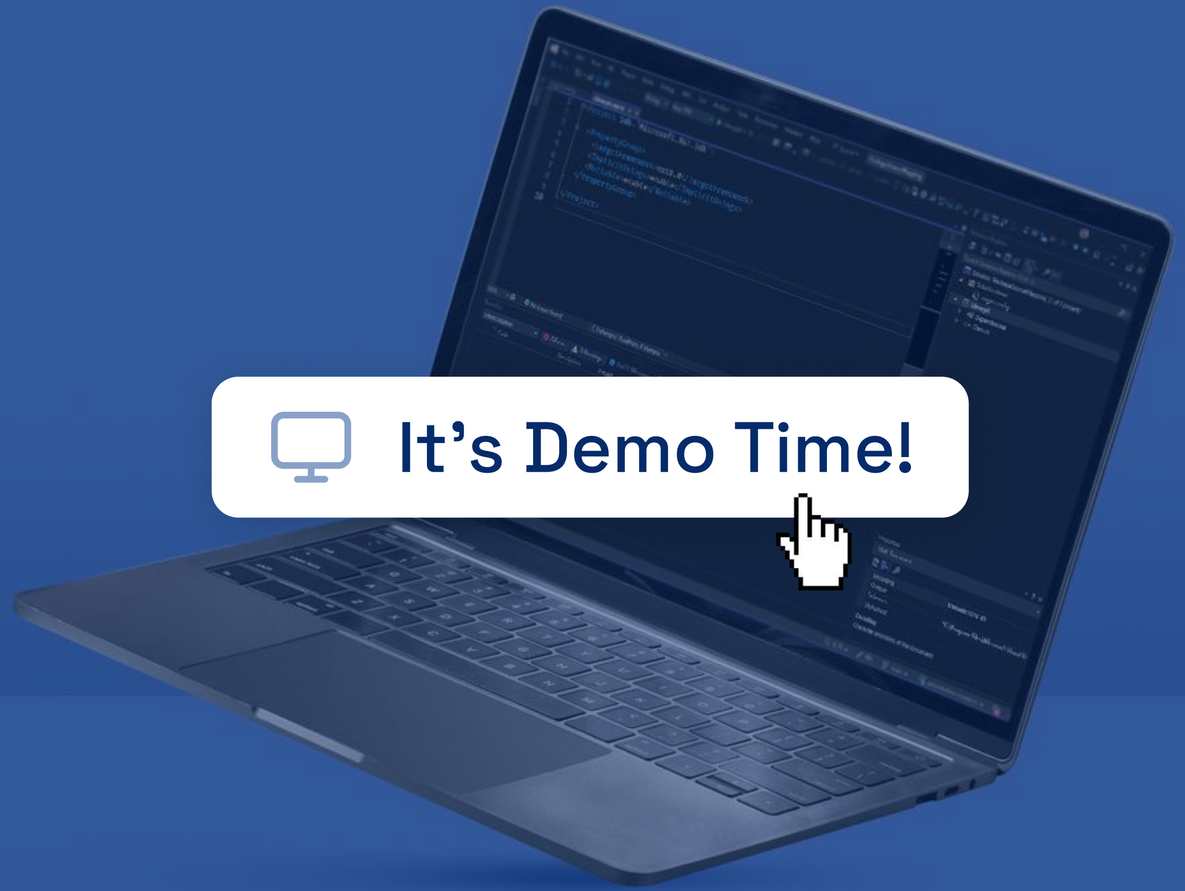
Package Validation

- Validates packages during “dotnet pack”
- Can also be used to validate non-packable assemblies
- Can be used either as an MSBuild task or via the standalone tool

Available validations:

1. Baseline version validator
2. Compatible runtime validator
3. Compatible framework validator

Avoid Breaking Changes in Minor/Patch Versions



Few Last Tips

- (Strongly) Consider using SemVer for your published packages.
- Do not impose exact or upper version constraints for your dependencies.
- Try stay away from packages that are not actively maintained.

Closing

Thank you!

Let’s connect
and keep in touch!