Design Principles IV

Ergude Bao
Beijing Jiaotong University

Contents

Introduction to Package Principles

Introduction to Package Principles

- Classes are not sufficient to group code
  - Some classes collaborate with dependencies
  - Some do not know each other
- Grouping related classes together seems natural

Six Principles

- Three principles of package cohesion
  - Reuse-release equivalence principle
  - Common-reuse principle
  - Common-closure principle
- Govern partitioning of classes into packages

Six Principles

- Three principles of package coupling
  - Acyclic-dependencies principle
  - Stable-dependencies principle
  - Stable-abstractions principle
- Govern interrelationships between packages

Three Principles of Package Cohesion

Reuse-Release Equivalence Principle

- Either all of the classes inside the package are reusable, or none of them are

Common-Reuse Principle

- The classes in a package are reused together
- If one reuses one of the classes in a package, one may reuse all the classes in the same package but not any of the classes in other packages

Common-Closure Principle

- The classes in a package are closed together against the same kinds of changes
- A change that affects a class may affect all the classes in the same package but not any of the classes in other packages
- A package should not have multiple reasons to change

Three Principles of Package Coupling

Acyclic-Dependencies Principle

Morning-After Syndrome Problem

- “Have you ever worked all day, gotten some stuff working, and then gone home, only to arrive the next morning to find that your stuff no longer works? Why doesn’t it work? Because somebody stayed later than you and changed something you depended on! I call this ‘the morning-after syndrome’.”
  - Robert C. Martin, Agile Principles, Patterns, and Practices in C#

Two Solutions

- Weekly Build
- Acyclic-Dependencies Principle

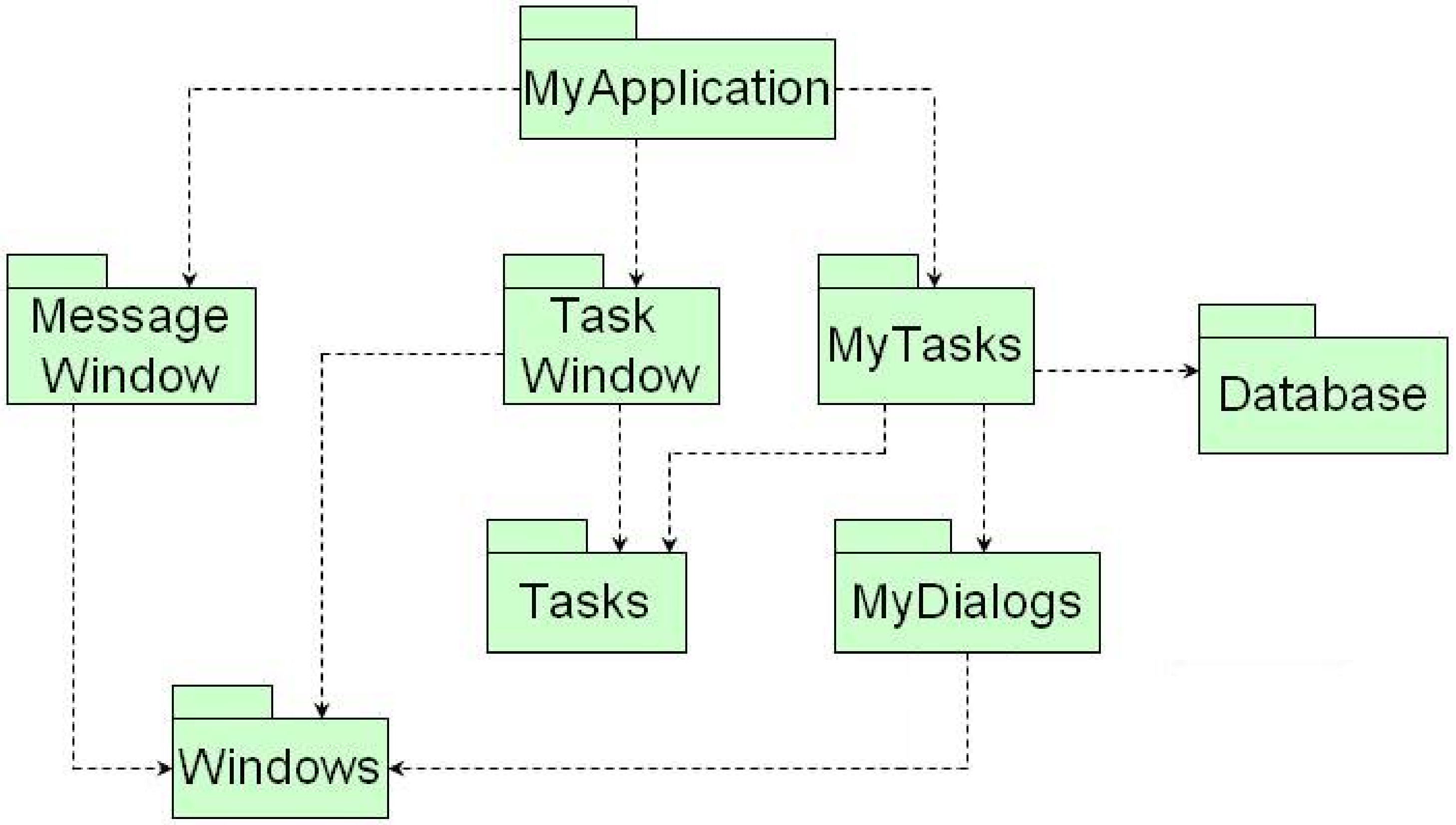
Weekly Build

- In the first four days of a week, all the developers work on private copies of the code and do not worry about integrating with each other
- On Friday, they integrate all their changes and build the system
- It is common in medium-sized projects

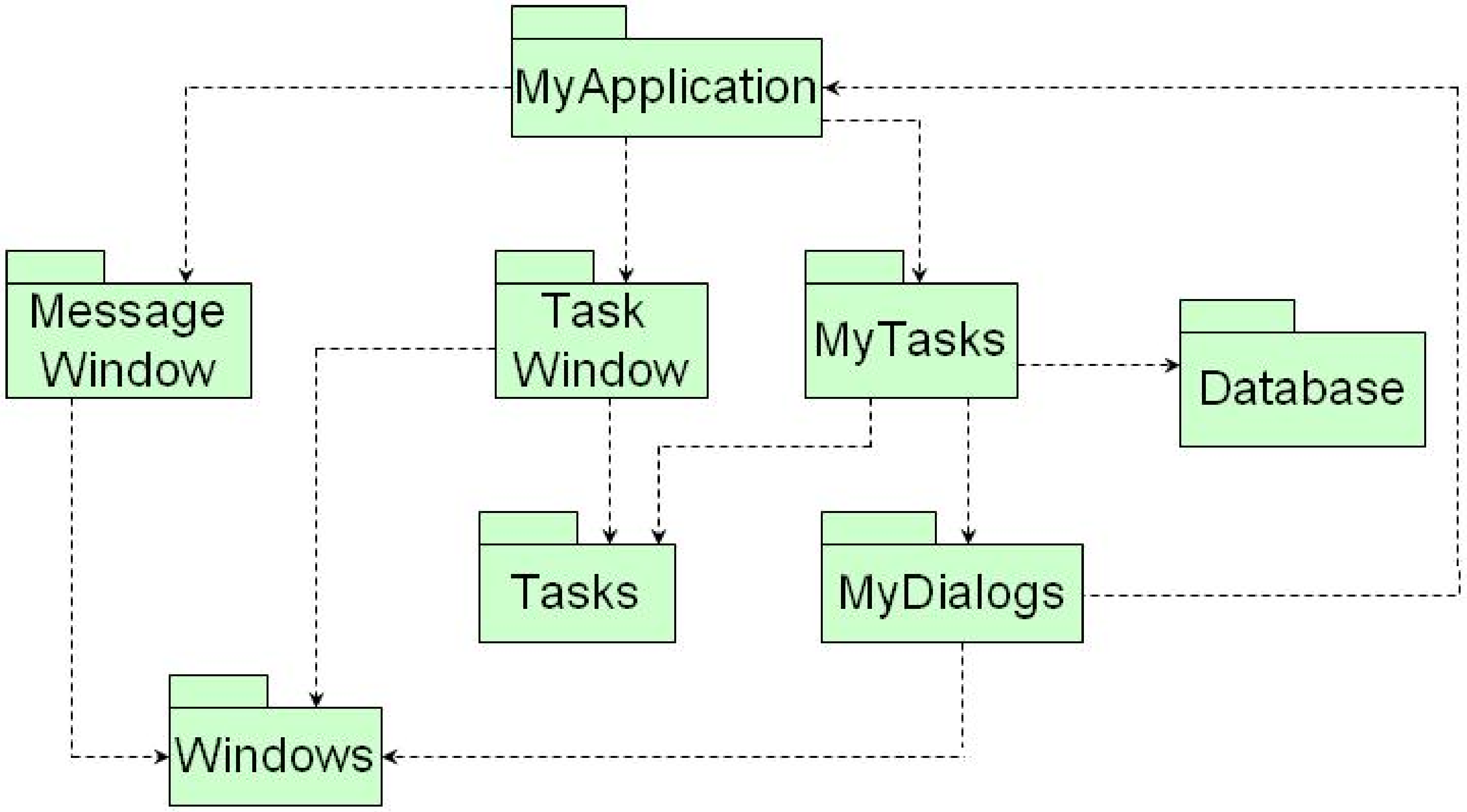
Acyclic-Dependencies Principle

- Package dependency graph
  - Directed graph with each vertex per package and each edge per dependency between two packages
- There should be no cycle in the package-dependency graph

Cycle Breaking Problem



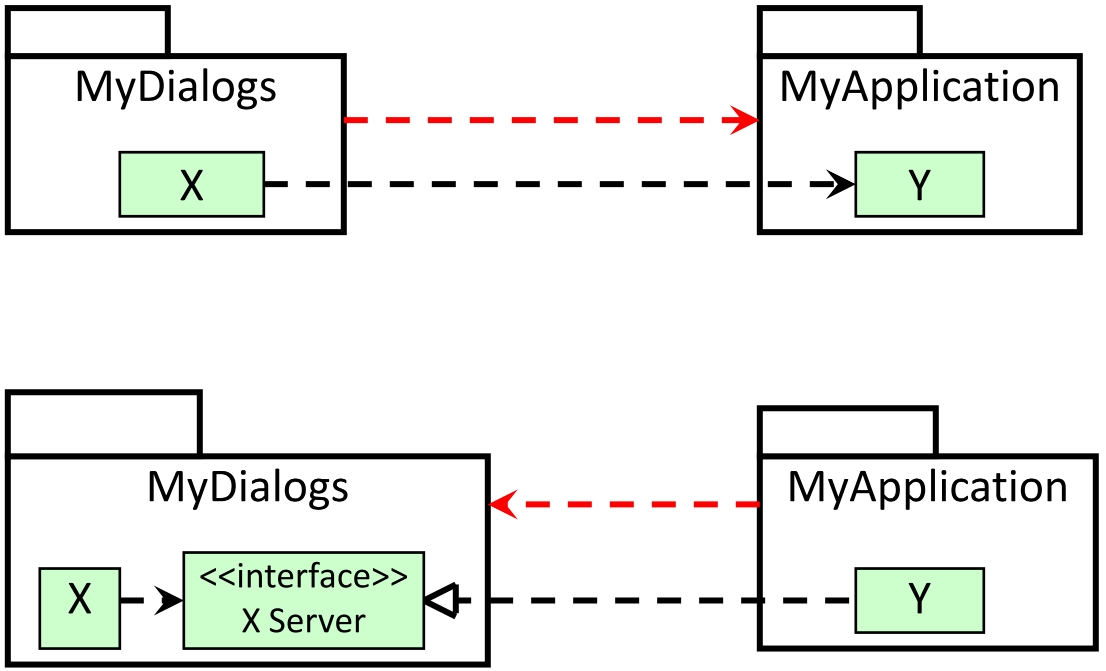
Cycle Breaking Problem



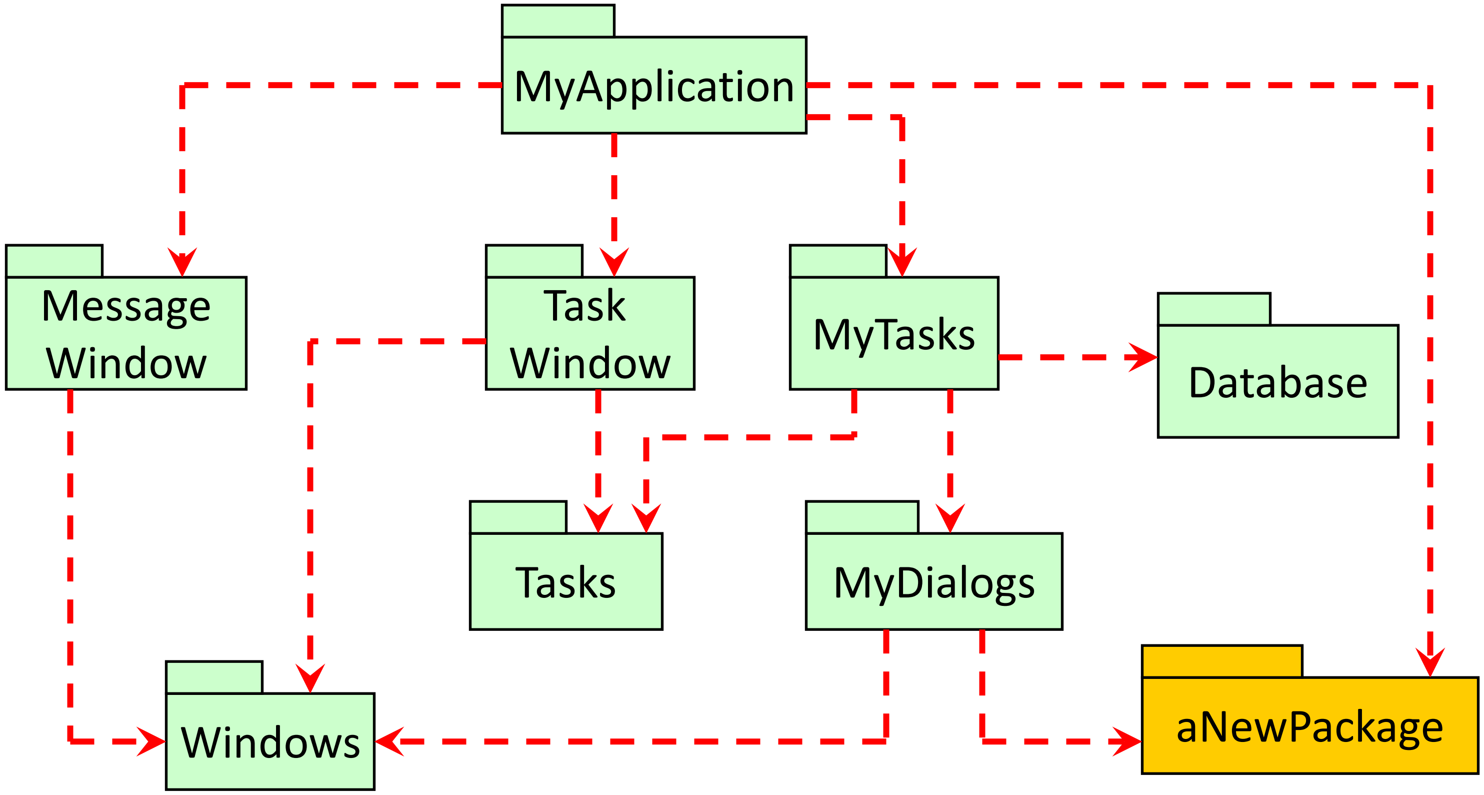
Two Solutions

- Two mechanisms to break a cycle of packages and reinstate the dependency graph as a DAG
  - Apply the Dependency-Inversion Principle (DIP)
  - Create a new package both MyDialog and MyApplication depend on

Applying the DIP



Creating a New Package



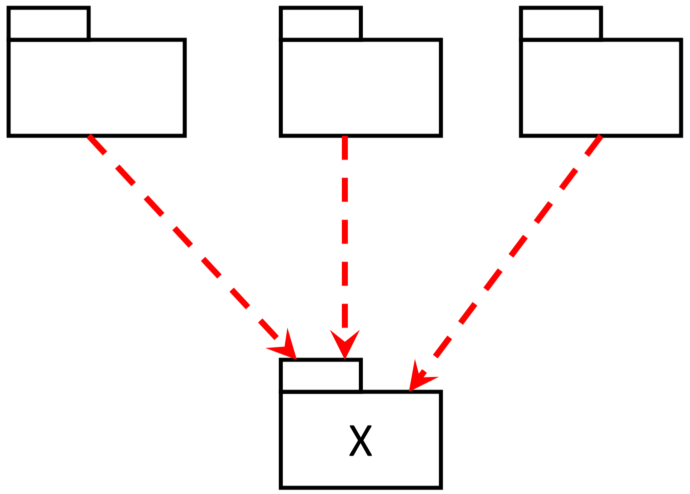
Stable-Dependencies Principle

Stable-Dependencies Principle

- Dependency should go from instable packages to stable packages
  - It ensure that packages hard to change do not depend on packages easy to change

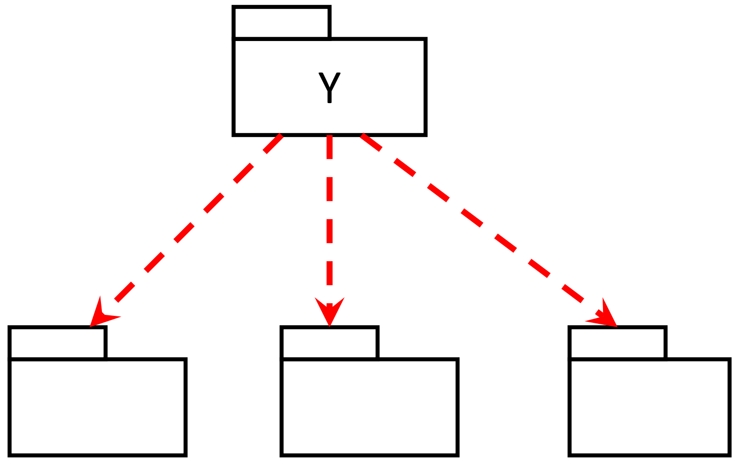
Stable package

- Three packages depend on X
  - X should not change much
- X depends on nothing
  - Nothing can incur X's changes



Instable package

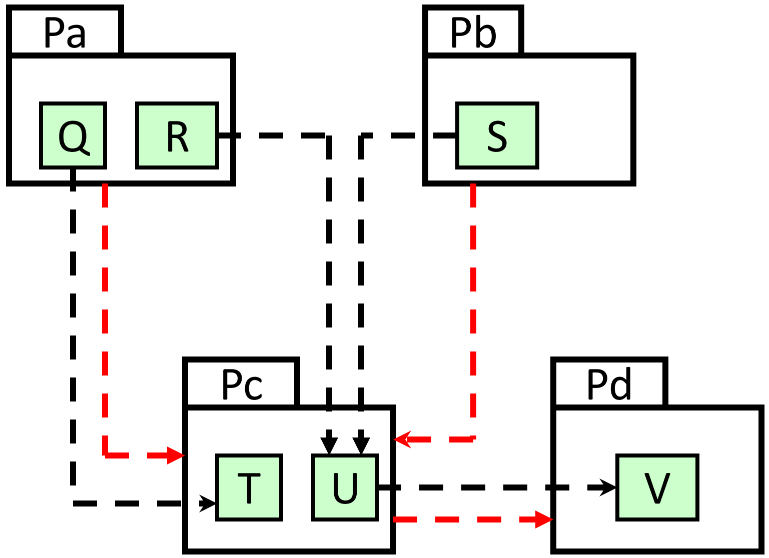
- Nothing depends on Y
  - Y could change much
- Y depends on three packages
  - They can incur Y’s changes



Stability Metrics

- C_e: Efferent Couplings
  - The number of classes inside this package that depend on classes outside this package
- C_a: Afferent Couplings
  - The number of classes outside this package that depend on classes within this package
- $I = C_e / (C_a + C_e)$: Instability
  - $I = 0$ indicates a maximally stable package
  - $I = 1$ indicates a maximally instable package

Stability Metrics



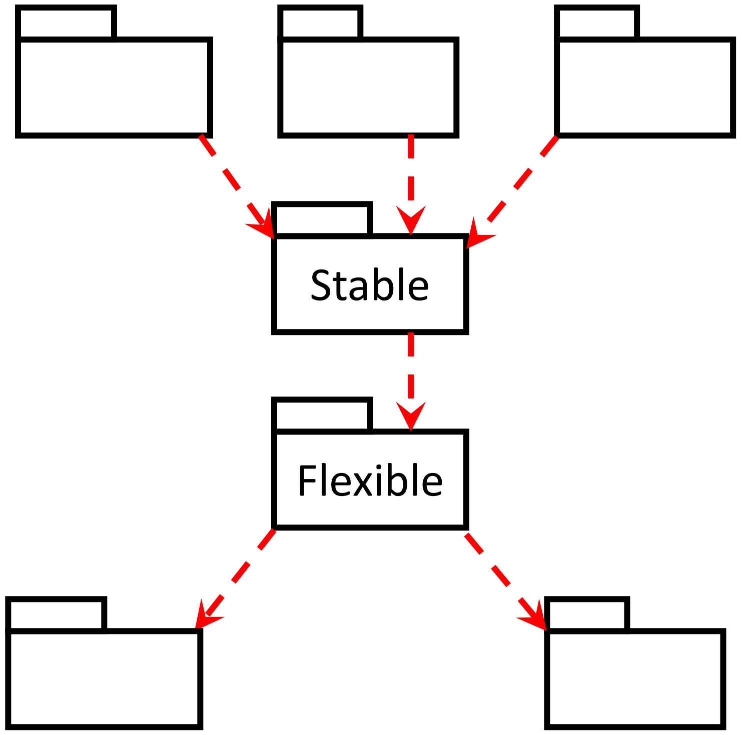
- $C_a = 3$
- $C_e = 1$
- $I = 1/4$

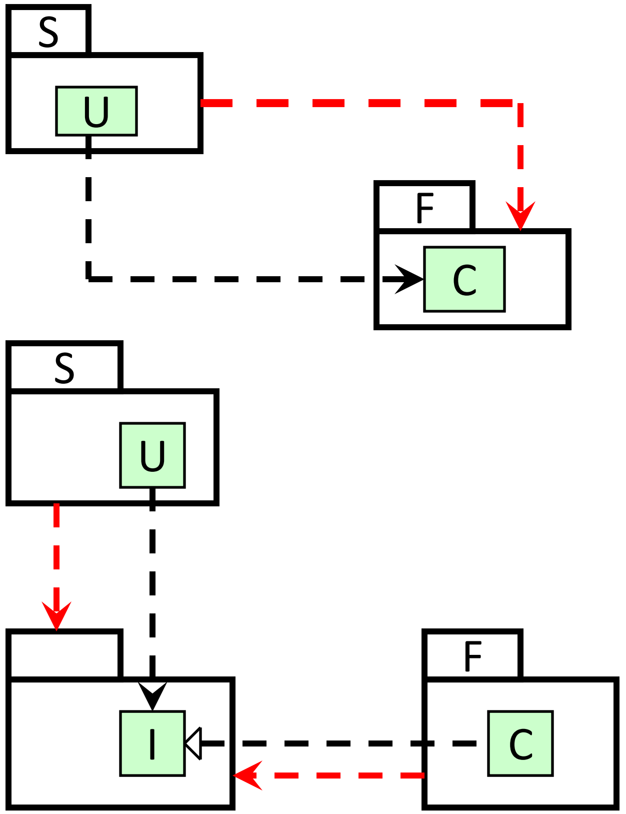
Stability Metrics

- Dependency should go from packages of large I metric to packages of small I metric

Problem and Solution

- Still the DIP





Stable-Abstractions Principle

Stable-Abstractions Principle

- A package should be as abstract as it is stable
  - Abstract package is depended by subpackages, so it should be stable

Abstraction Metrics

- N_a: Number of abstract classes in the package
- N_c: Number of concrete classes in the package
- $A = N_a / (N_a + N_c)$: Abstractness
  - A = 1 indicates a maximally abstract package
  - A = 0 indicates a maximally concrete package