Architektur II
Principles

TU-Wien, Sommersemester 2003
Rudolf Lewandowski

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

Principles of Object Oriented Class Design

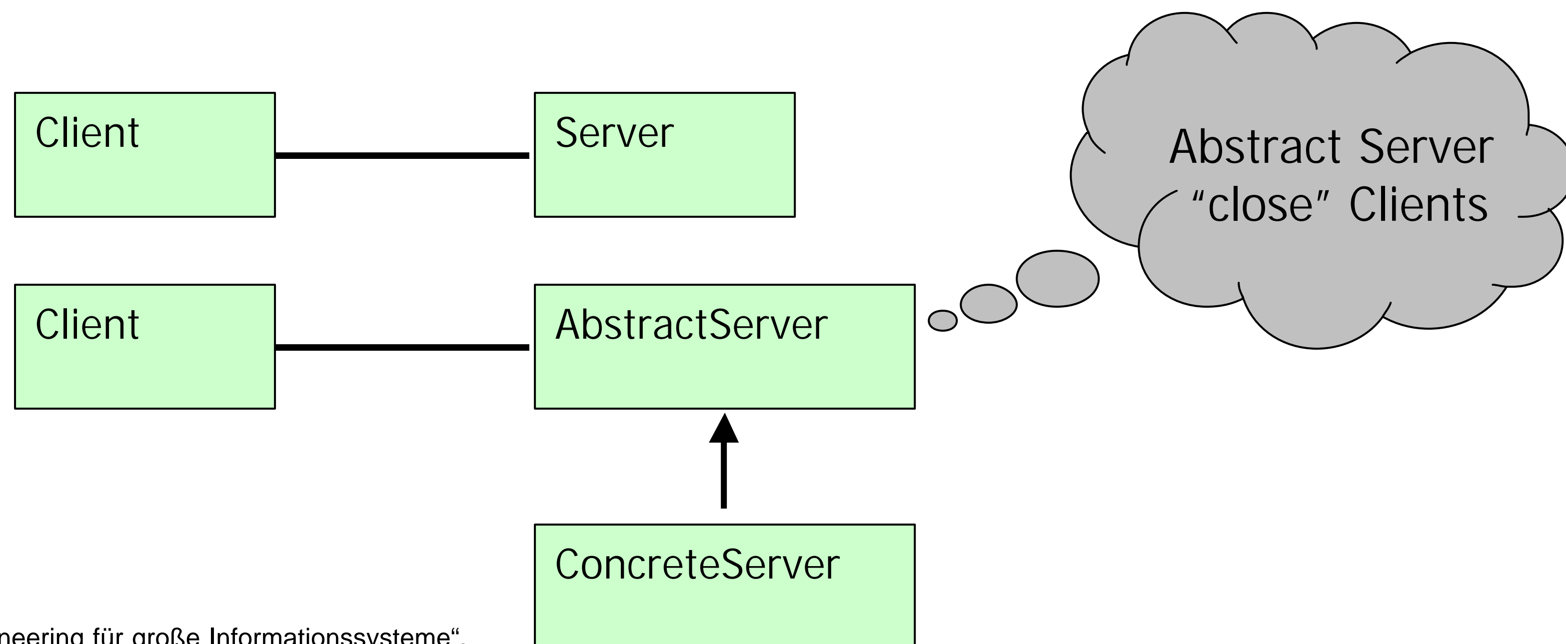
- (OCP) Open Closed Principle
- (LSP) Liskov Substitution Principle
- (DIP) The Dependency Inversion Principle
- (ISP) The Interface Segregation Principle
- (LOD) Law of Demeter

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(OCP) Open Closed Principle

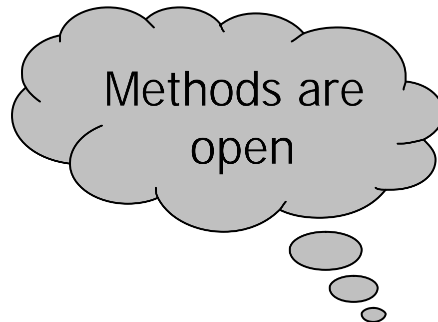
Bertrand Meyer

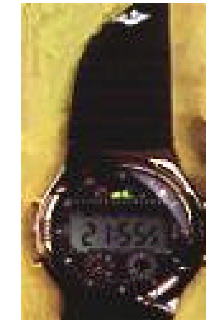
- A module is said to be open if it is still available for extensions.
- A module is said to be closed if it is available for use by other modules.



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(OCP)
Open Closed Principle

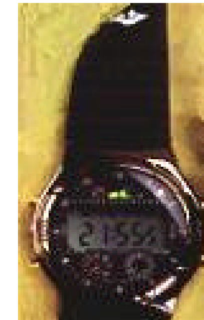




```
Watch>>upperButtonClicked
        self state = #stopwatch ifTrue: [self startStopwatch].
        self state = #time ifTrue: [self showDate].
        self state = #set ifTrue: [self inrementTime]

Watch>>lowerButtonClicked
        self state = #stopwatch ifTrue: [self stopStopwatch].
        self state = #time ifTrue: [self showTime].
        self state = #set ifTrue: [self decrementTime]
```


© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

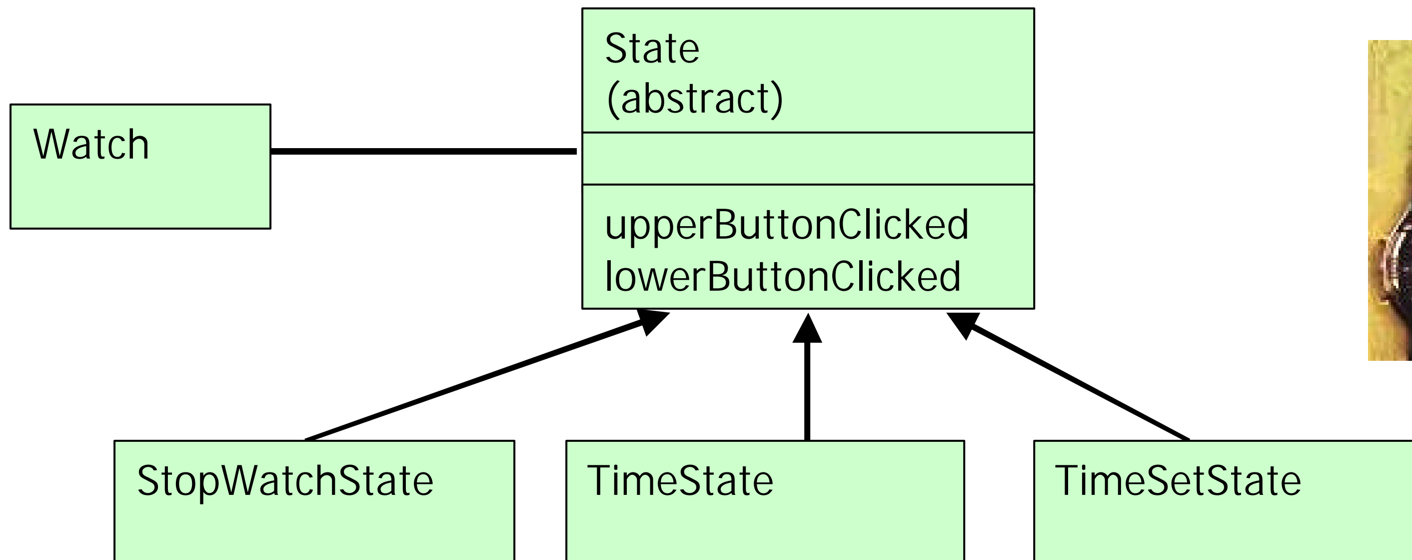
(OCP)
Open Closed Principle

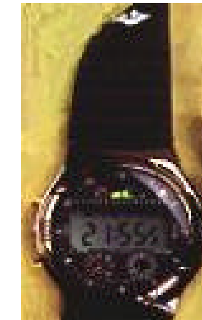


- This Implementation cannot be closed.
  In an complex Application this "switch/case" Statement is repeated over and over.
- These statements are difficult to find, and it is easy to make mistakes when modifying them.
- Worse, every module that contains such a "switch/case" Statement retains a dependency upon every possible state the watch can be.
- The if statements are usually seldom so clean and nice as those shown.
- All methods have to be updated for each new state added.

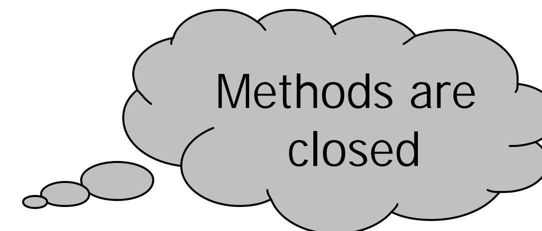
© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(OCP)
Open Closed Principle





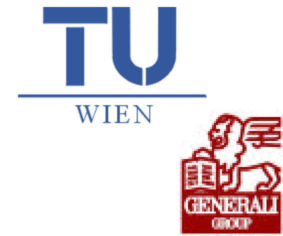
```
Watch>>upperButtonClicked
        self state upperButtonClicked

Watch>>lowerButtonClicked
        self state lowerButtonClicked
```



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LSP) Liskov Substitution Principle

Barbara Liskov



"What we want here is something like the following substitution property: If for each object o1 of type S there is an object o2 of type T such that all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T."

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LSP)
Liskov Substitution Principle

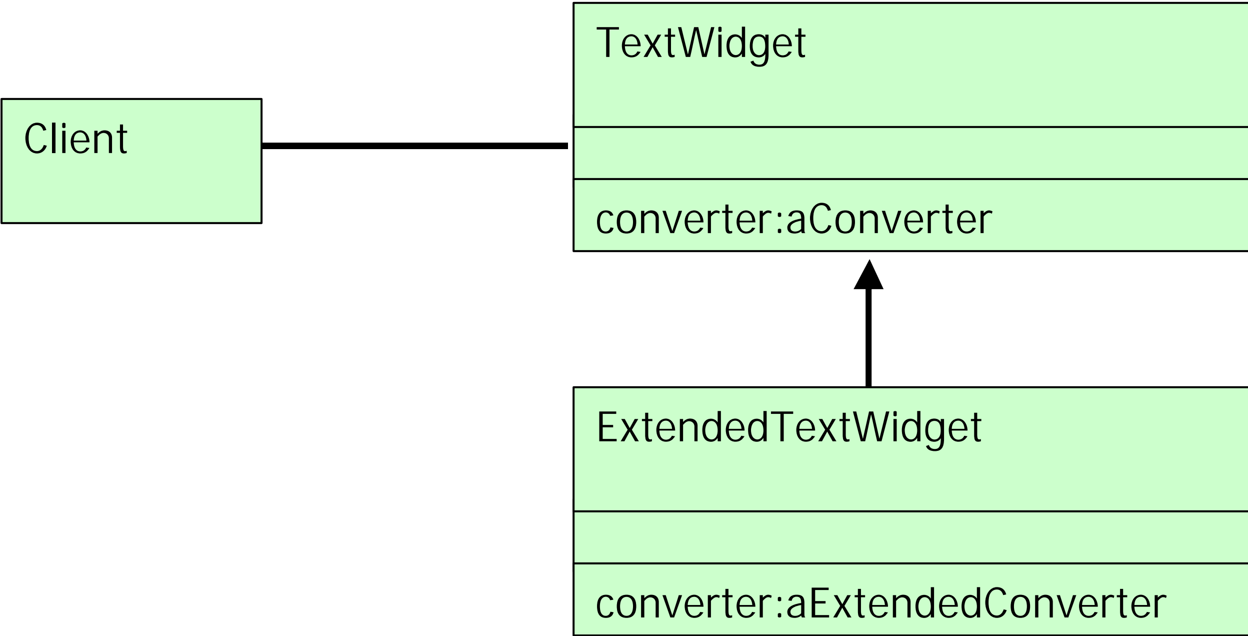
- Derived Classes must be usable through the base class interface without the need for the client to know the difference.
  - All derived classes must be suitable for their base classes
  - This principle guides us in the creation of abstractions

- If derived Classes are not suitable for their base classes then implementation within clients is often open.

```
...
aParameter class = BaseClass
    ifTrue: [...]
    ifFalse: [...]
```


© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LSP)
Liskov Substitution Principle



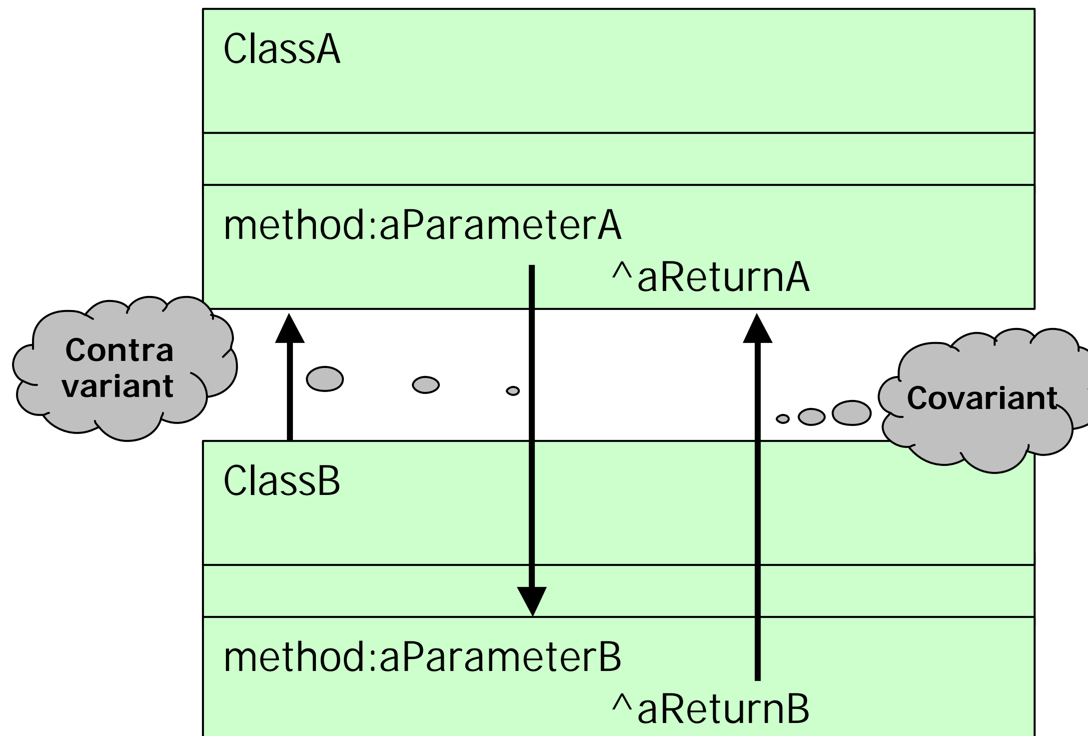
© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LSP)
Liskov Substitution Principle

- Design by Contracts (Assertions)
  - Precondition can be weakened in Subtypes
  - Postconditions can be strengthened in Subtypes
- Parametertypes of methods of Subtypes can only be of same type or one of its base types (Contravariant)

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LSP)
Liskov Substitution Principle



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LSP) Liskov Substitution Principle

	When S is substitutable for T (polymorphism)			
	Must S subclass from T?	For S>>m to be consistent with T>>m		
		Return types	Argument Types	Behavioral Conditions
C++	Yes	Covariant	Must agree	n/a
Eiffel	Yes	Covariant	Covariant	May weaken preconditions or strengthen postconditions
Emerald	No	Covariant	Contravariant	n/a
Java	No	Covariant	Must agree	n/a
Modula-3	Yes	Must agree	Must agree	May raise fewer exceptions
POOL-I	No	Covariant	Contravariant	May have more "properties"
Smalltalk	No	No restriction	No restriction	n/a
Theoretical ideal	No	Covariant	Contravariant	May weaken preconditions or strengthen postconditions

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(DIP) The Dependency Inversion Principle

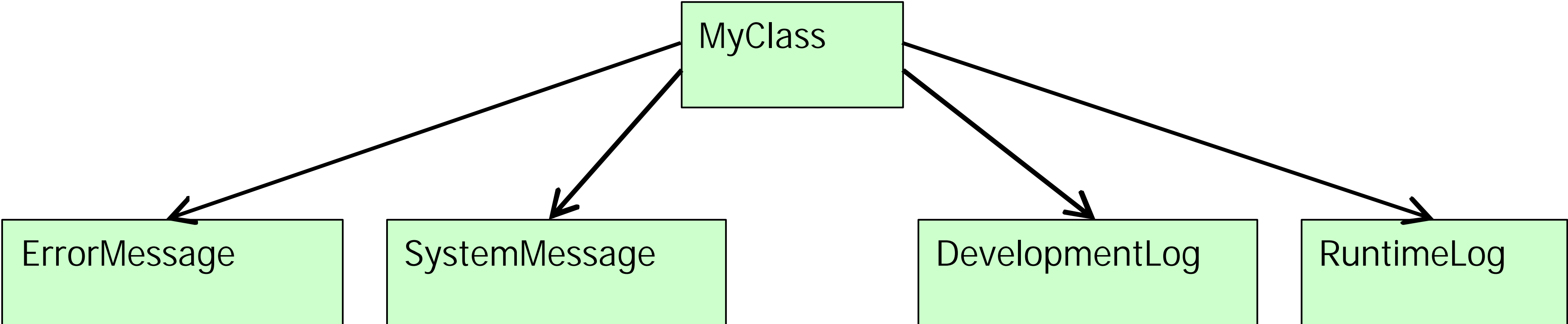
Robert Martin

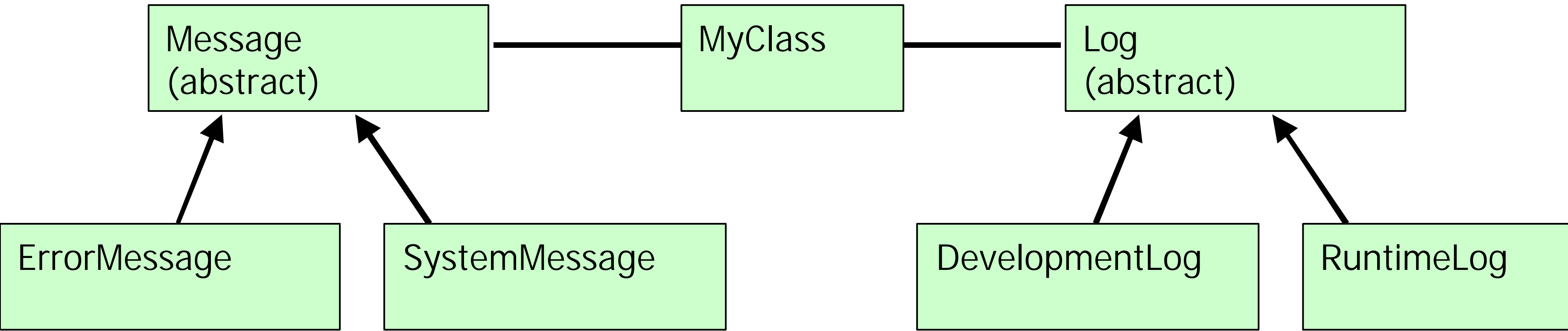
- Details should depend upon abstractions. Abstractions should not depend upon details.

But: In Structured Analysis Structured Design High level Structures are decomposed into smaller ones. So these Structures depend on lower level ones.

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(DIP)
The Dependency Inversion Principle



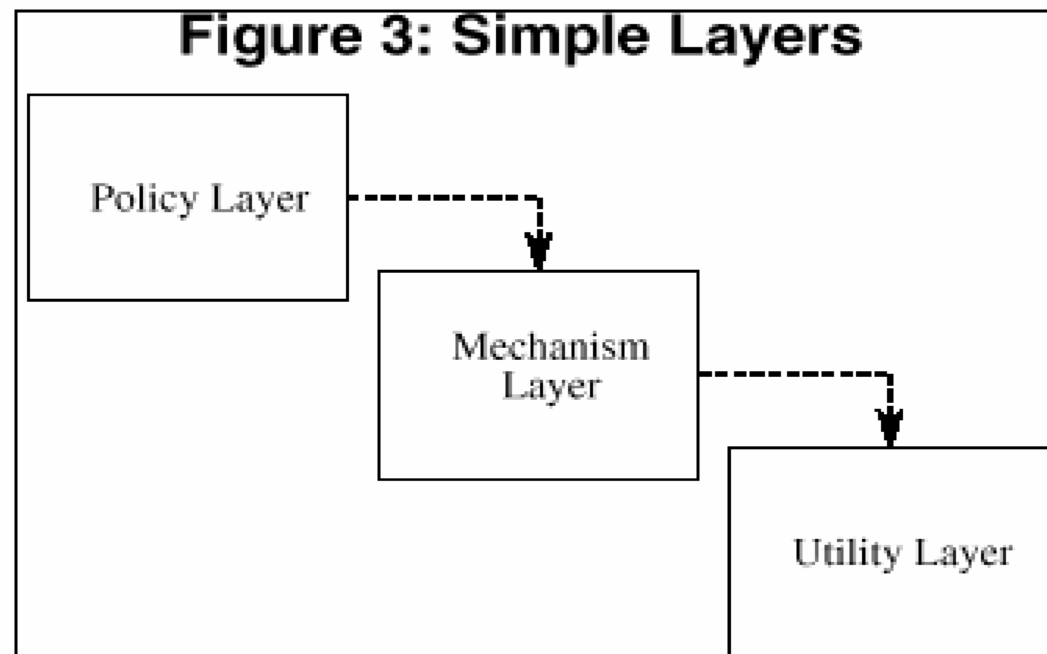


© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(DIP)
The Dependency Inversion Principle

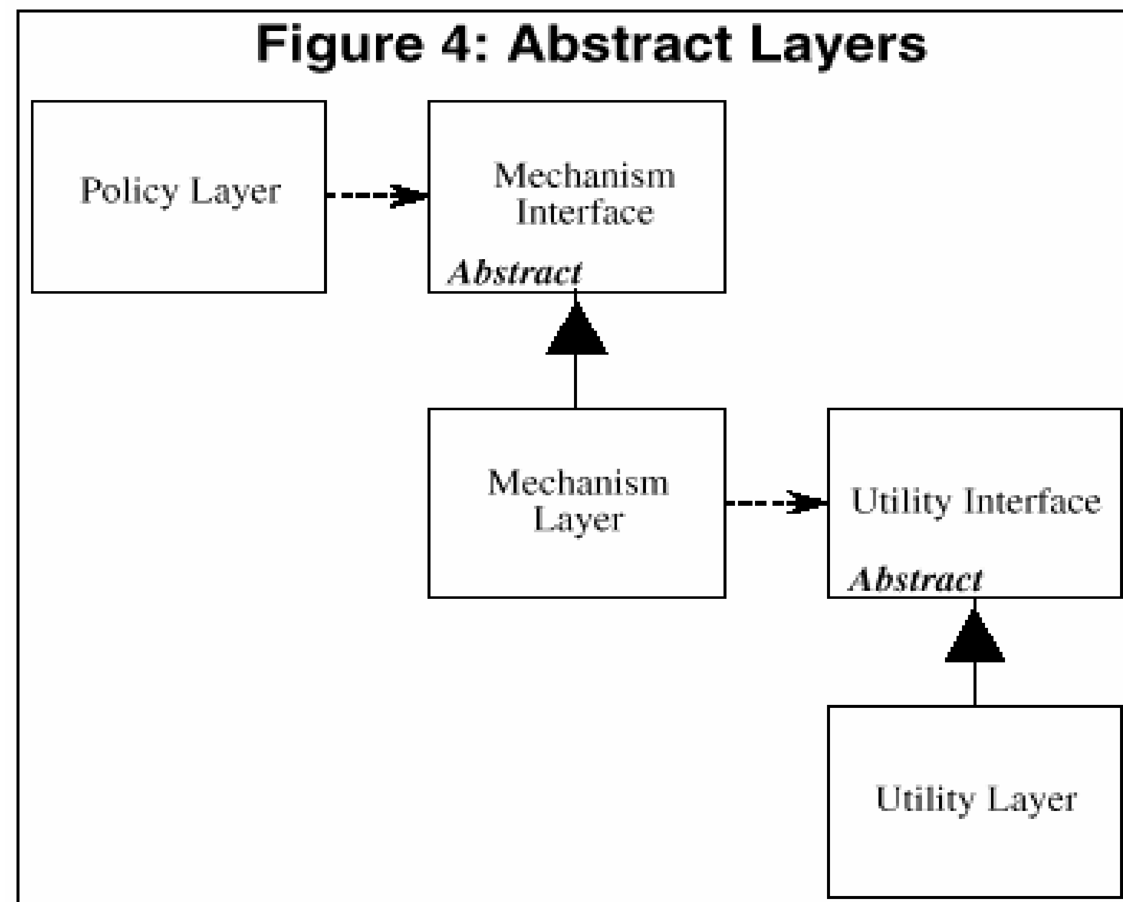
- High level modules should not depend on low level modules

Grady Booch



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

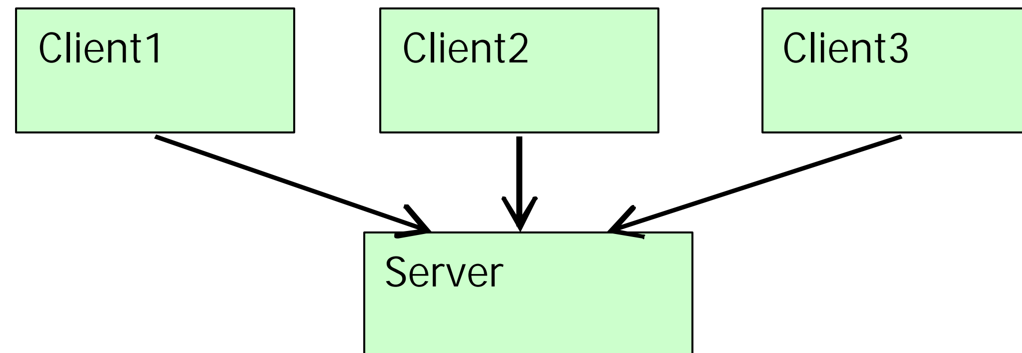
(DIP)
The Dependency Inversion Principle



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

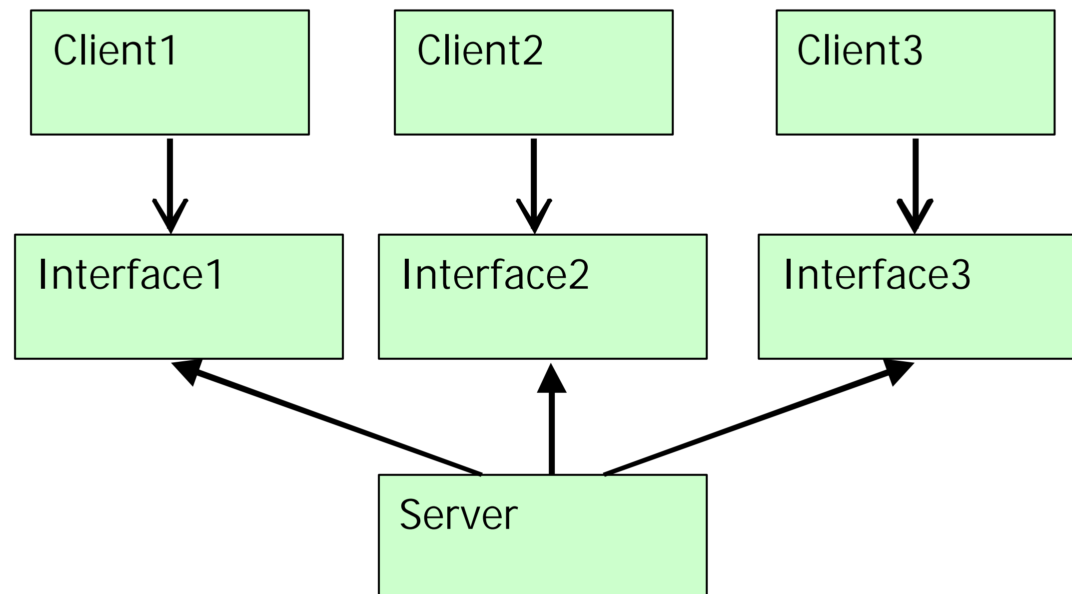
(ISP)
The interface Segregation Principle

- Clients should not depend on interfaces they do not use



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

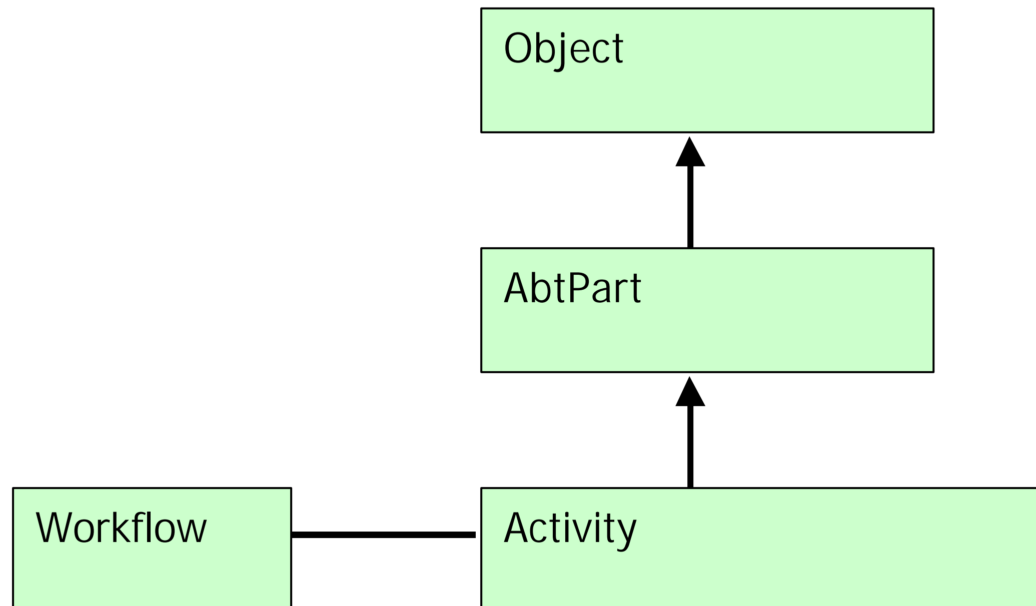
(ISP)
The interface Segregation Principle



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(ISP)
The interface Segregation Principle

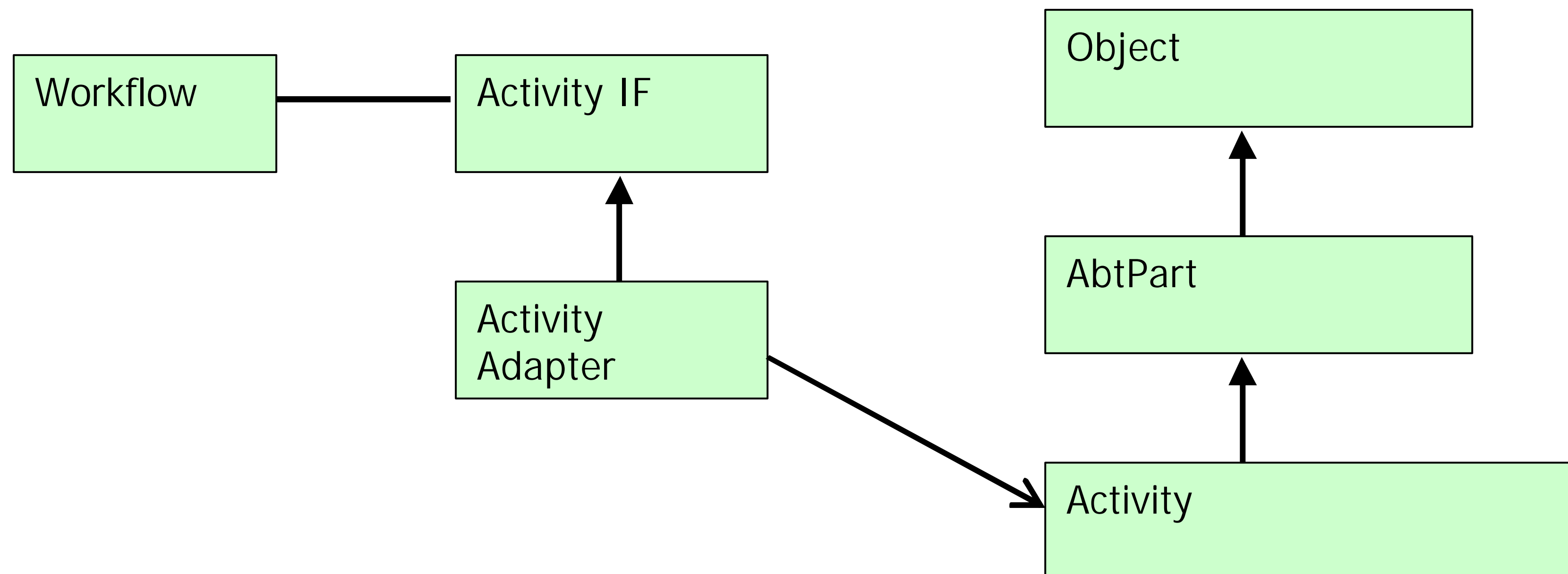
- Smalltalk: Clients should not depend on interfaces they do not use



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(ISP)
The interface Segregation Principle

- Use Adapters



Vorlesung „Software Engineering für große Informationssysteme“,
© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

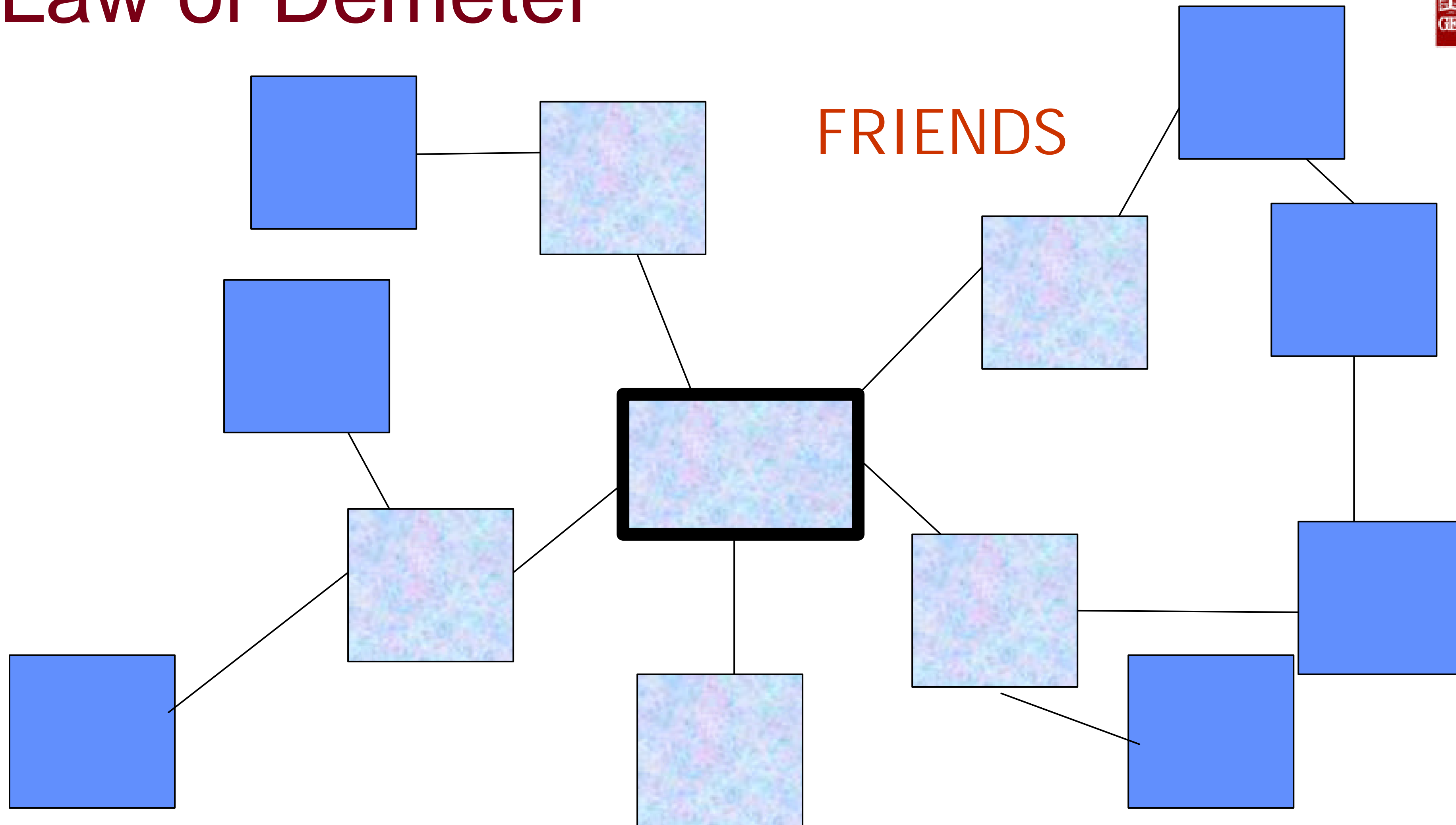
(LOD) Law of Demeter

Karl J. Lieberherr

- Each unit should only use a limited set of other units: only units “closely” related to the current unit.
- “Each unit should only talk to its friends.” “Don’t talk to strangers.”
- Main Motivation: Control information overload. We can only keep a limited set of items in short-term memory.

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LOD)
Law of Demeter



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(LOD)
Law of Demeter

- A Method of a Class may use
  - its Parameters
  - instance Variables
  - local Variables
  - Global Variables
- A method should have limited knowledge of an object model.

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

Principles of Object Oriented Package Design

- (REP) The Reuse Release Principle
- (CCP) Common Closure Principle
- (CRP) Common Reuse Principle

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(REP)
The Reuse Release Principle

- Main Unit of reuse is the Module
- Expectations
  - Documentation
  - Accuracy
  - Maintenance
  - Predictability
- Reusers expect Release Control

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(REP)
The Reuse Release Principle

- Single Classes are seldom reusable
- Unreleased modules cannot be reused
- So the granularity of reuse is the granularity of release

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(CCP) Common Closure Principle

- The classes in a package should be closed together against the same kinds of changes. A change that affects a package affects all the classes in that package
- If two classes are so tightly bound together, either physically or conceptually, such that the almost always change together; then they should belong to the same package

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(CCP)
Common Closure Principle

- Recompilation issues (e.g. C++)
- A package is cohesive, if the classes within it are all closed to the same kind of modification.
- When this principle can be achieved, changes do not propagate through the system.
  - reduces frequency with which packages must be released

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(CRP)
Common Reuse Principle

- Classes within a package should be reused together
- Reuse creates a dependency upon the whole reused component
  - When a new release is created the package must be reintroduced
  - Users want each component as focused as possible

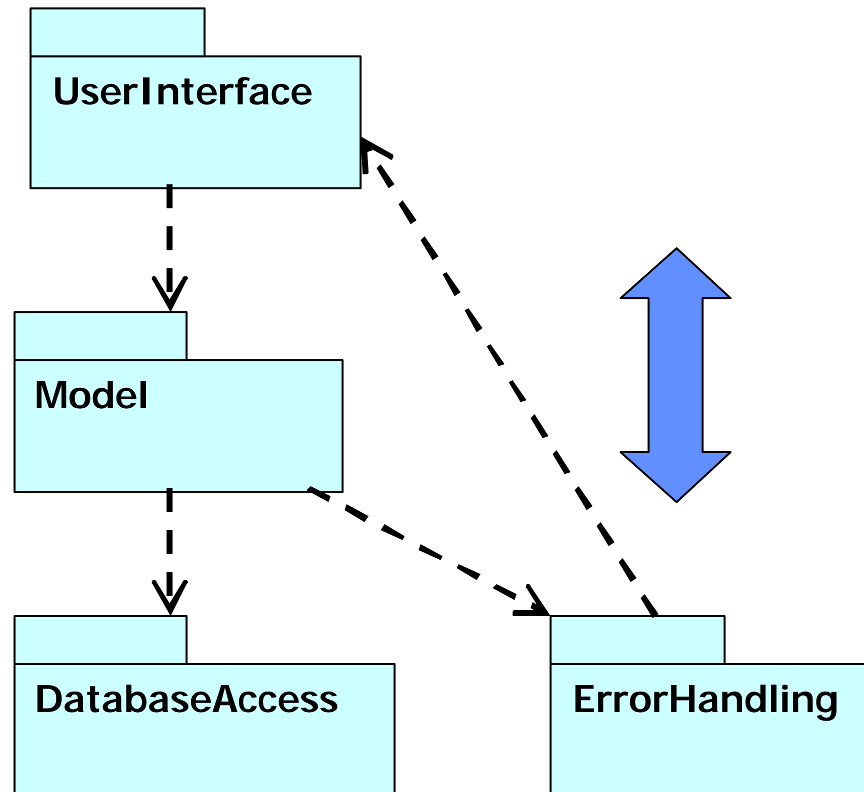
© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

Managing the development Environment

- (ADP) The Acyclic Dependencies Principle
- (SDP) The Stable Dependencies Principle
- (SAP) The Stable Abstractions Principle

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(ADP)
The Acyclic Dependencies Principle



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

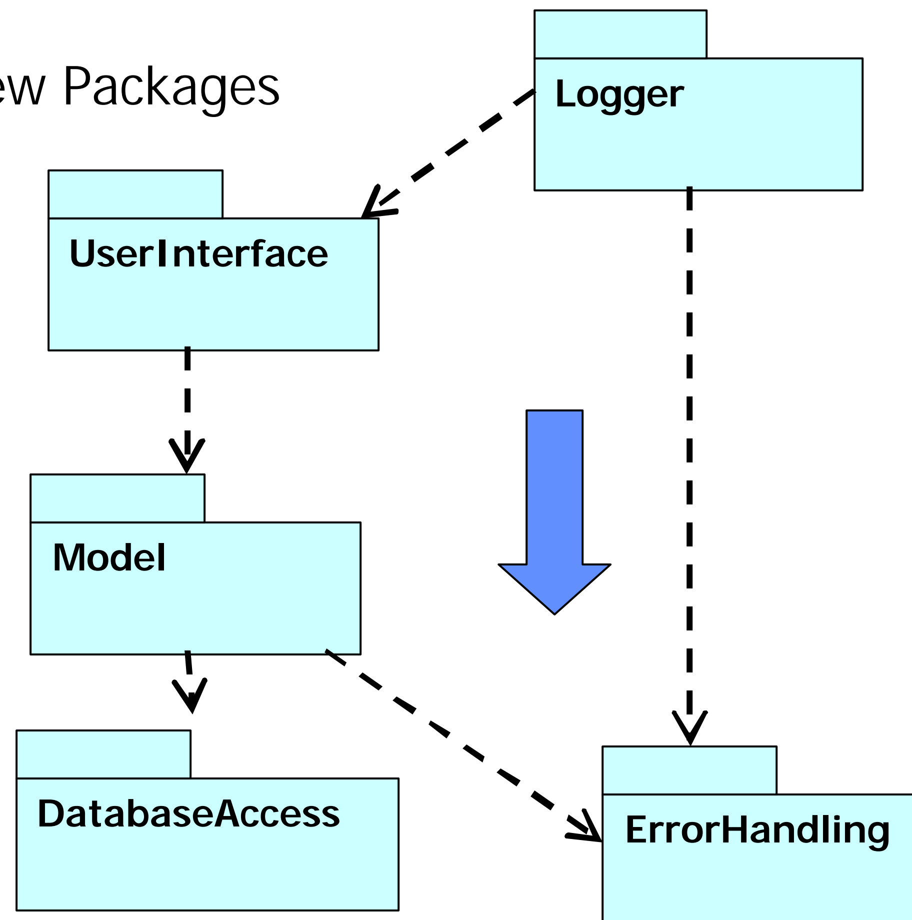
(ADP)
The Acyclic Dependencies Principle

- Smalltalk hides Cyclic Dependencies
  - unexpected Polymorphism
  - Smalltalk at:
  - perform:
  - Unallowed Class References
- All Applications of a cycle can only released together
- Inverse Dependencies
  - Callbacks
  - Events
  - Registry
  - Blocks

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

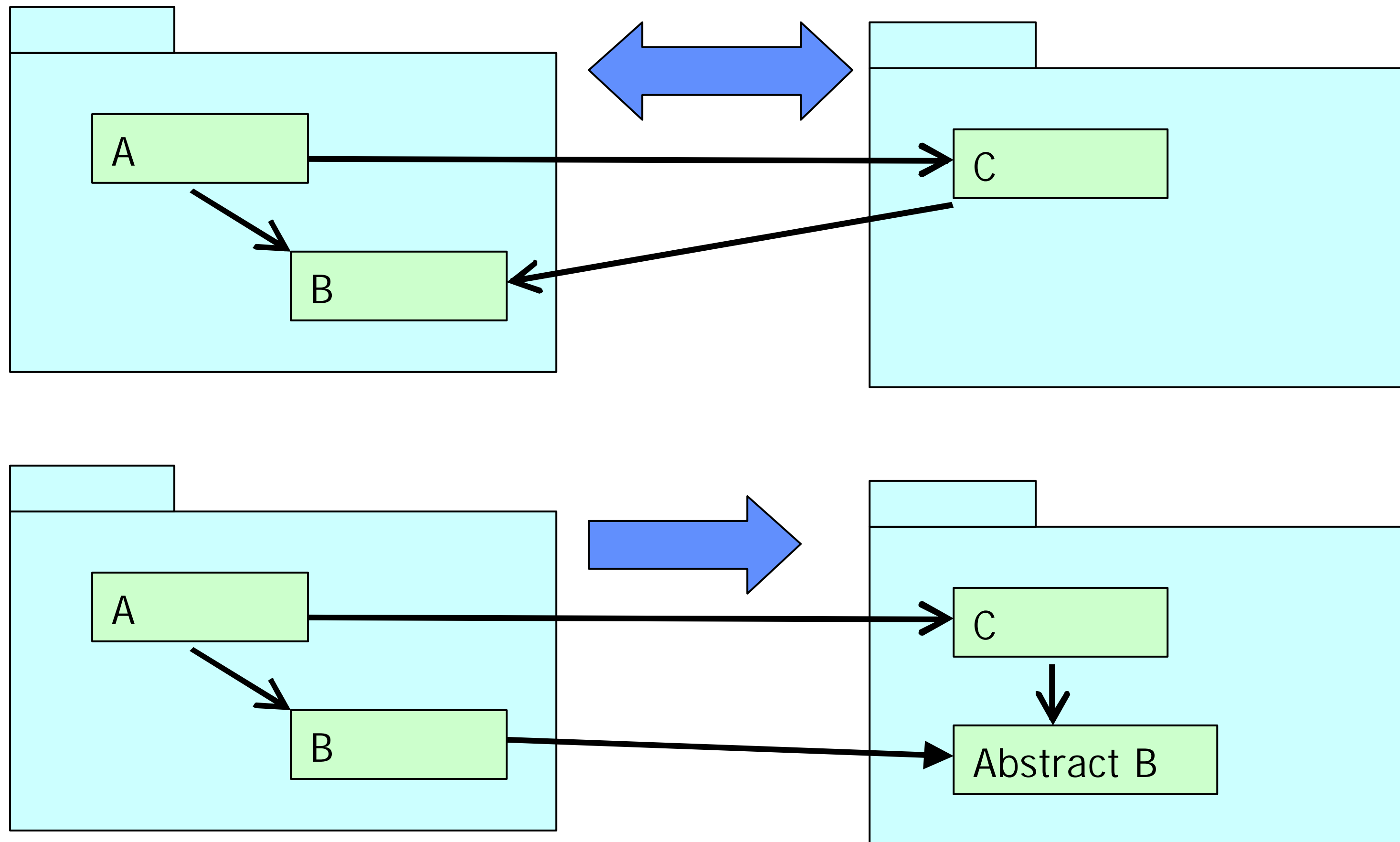
(ADP)
The Acyclic Dependencies Principle

Introduce new Packages



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(ADP)
The Acyclic Dependencies Principle



Introduce new Abstract Classes

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(SDP)
The Stable Dependencies Principle

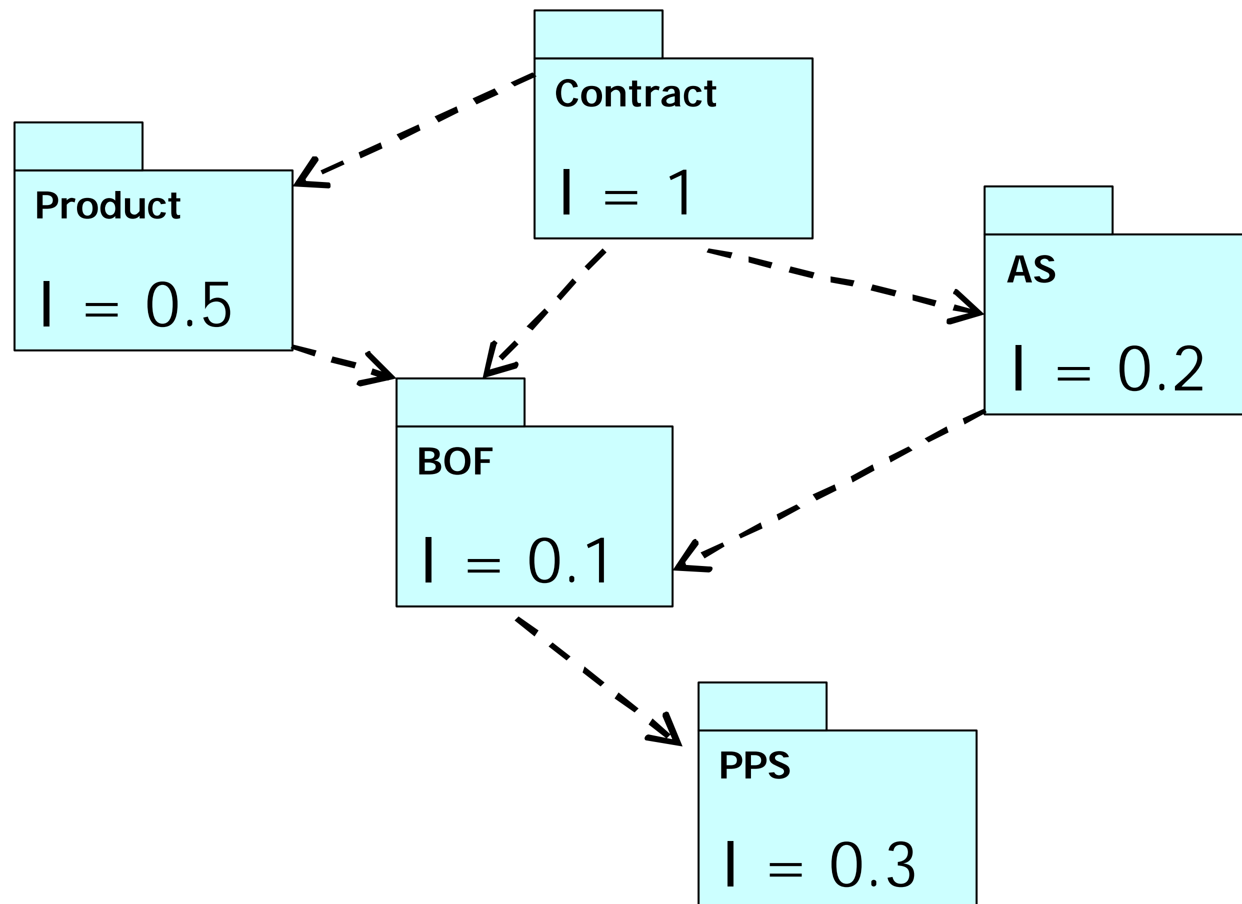
- Dependencies between released packages must run in the direction of stability. The dependee must be more stable than the depender
- The *Positional Instability* of a Package is
  - Ca (Affect Couplings) the number of classes outside the package depending on classes within the package.
  - Ce (Effect Coupling) the number of classes outside the package that classes within the package depend on.
  - I (Instability) = Ce / (Ce+Ca)

    0... ultimately stable

    1... Ultimately instable

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(SDP)
The Stable Dependencies Principle



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(SAP)
The Stable Abstractions Principle

- The more stable a package is , the more it must consist of abstract classes.
- A completely stable package should consist of nothing but abstract classes
- The A metric is The number of abstract classes divided by all classes of a package

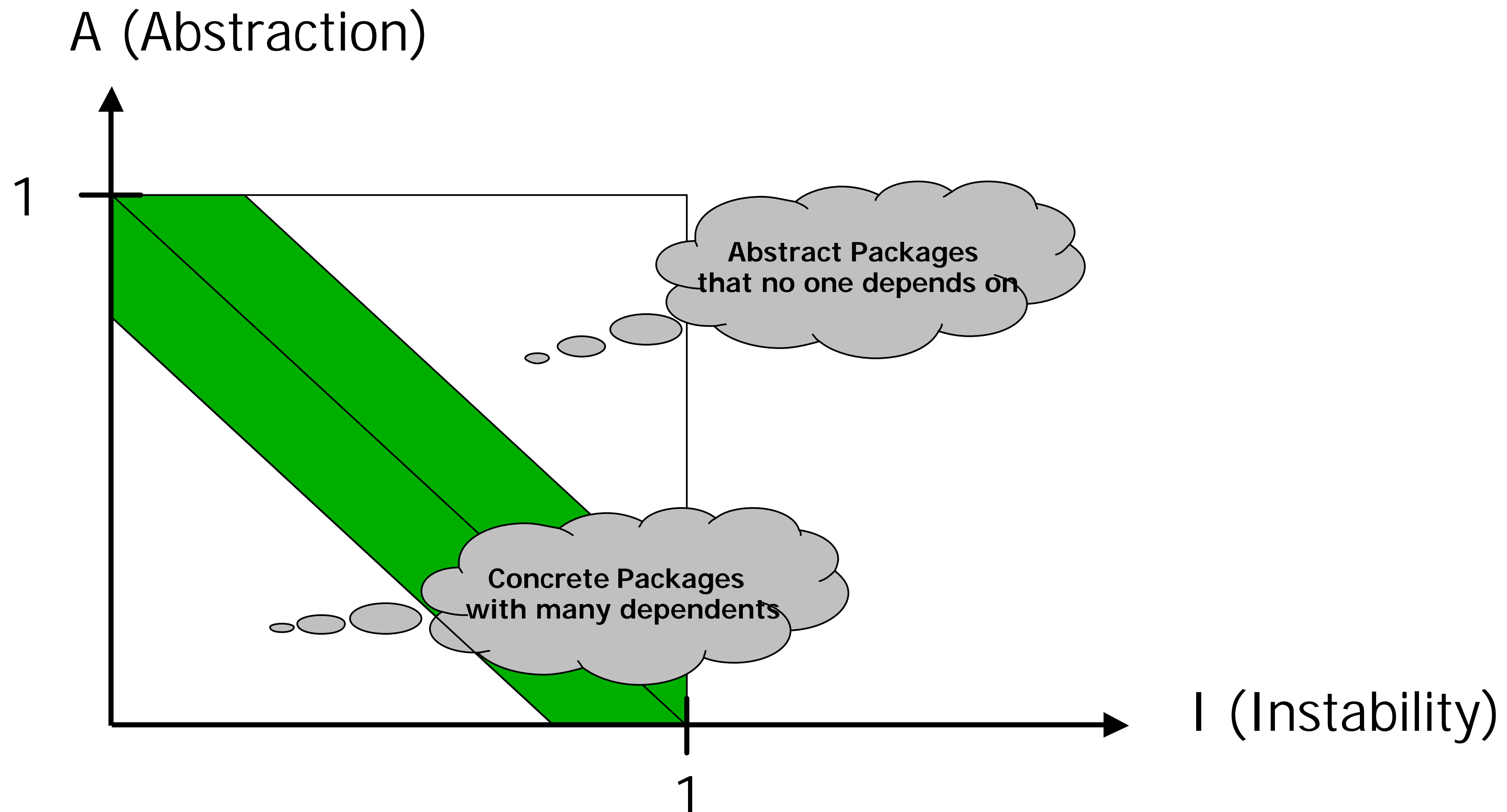
© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(SAP)
The Stable Abstractions Principle

- Stable high level designs
  - High level design decisions must be stable
  - We want high level design be flexible too
- Open Closed Principle provides a way
  - Abstractions can be stable (closed for modifications) and flexible (open for extensions)

© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003

(SDP) vs.(SAP)
Stability vs. Abstraction



© 2003, Bernhard Anzeletti, Wolfgang Keller, Rudolf Lewandowski, Klaudius Messner, All rights reserved, Technische Universität Wien, SS2003