Dependency Cycles

Master Project
Bledar Aga
University of Bern

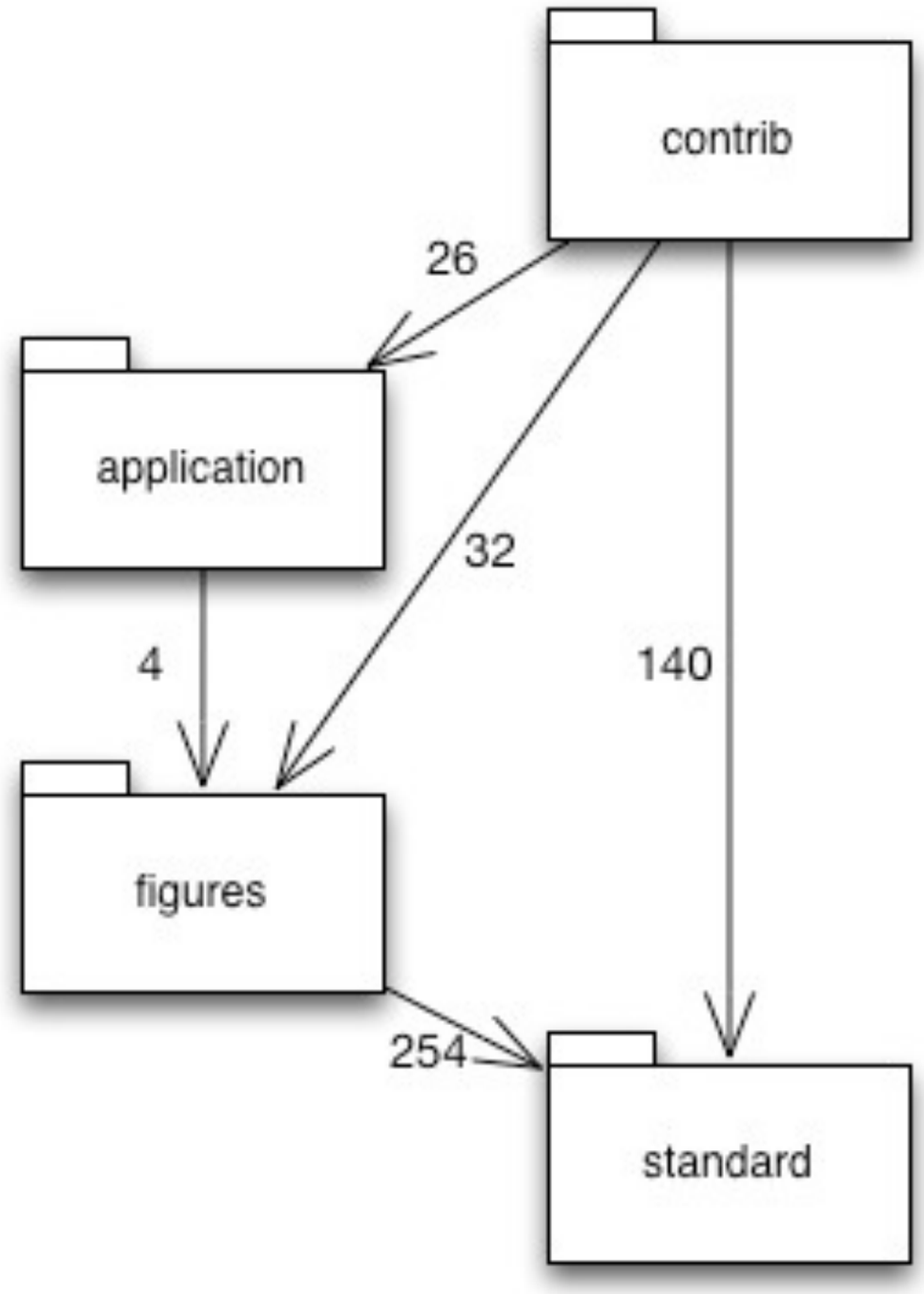
Software Composition Group
Spring 2013

Content

- Introduction
- Problem
- Dependency types
- Approaches
- Concrete example
- Existing approaches
- Existing solutions
- Our solution
- Refactoring
- Best refactoring
- Profit
- Best profit example
- Next steps
- Summary

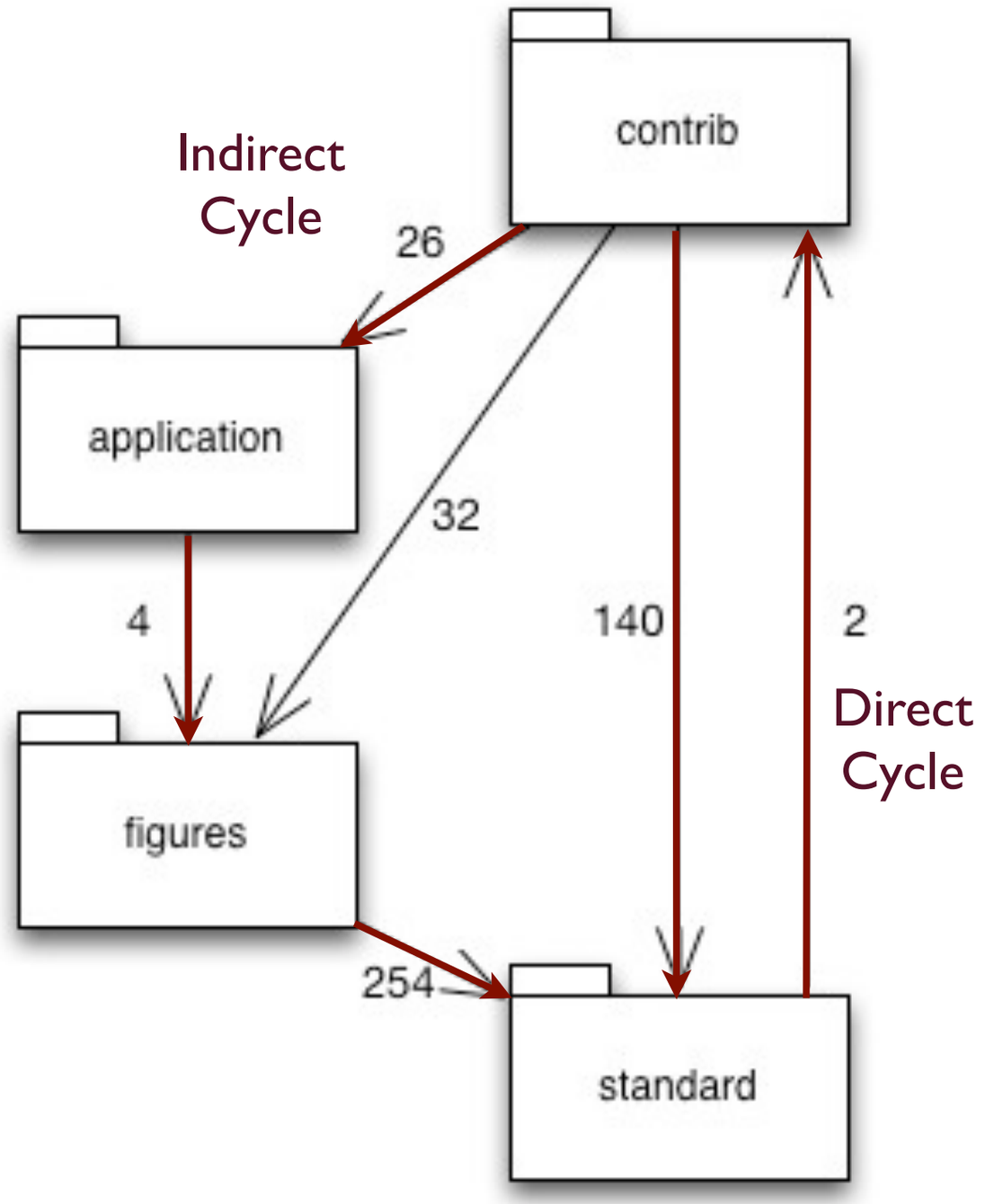
Introduction

Test and release with minimal amount of work



Problem

- Compromises the modularity of the system
- Increase the development time
- Prevents proper reuse



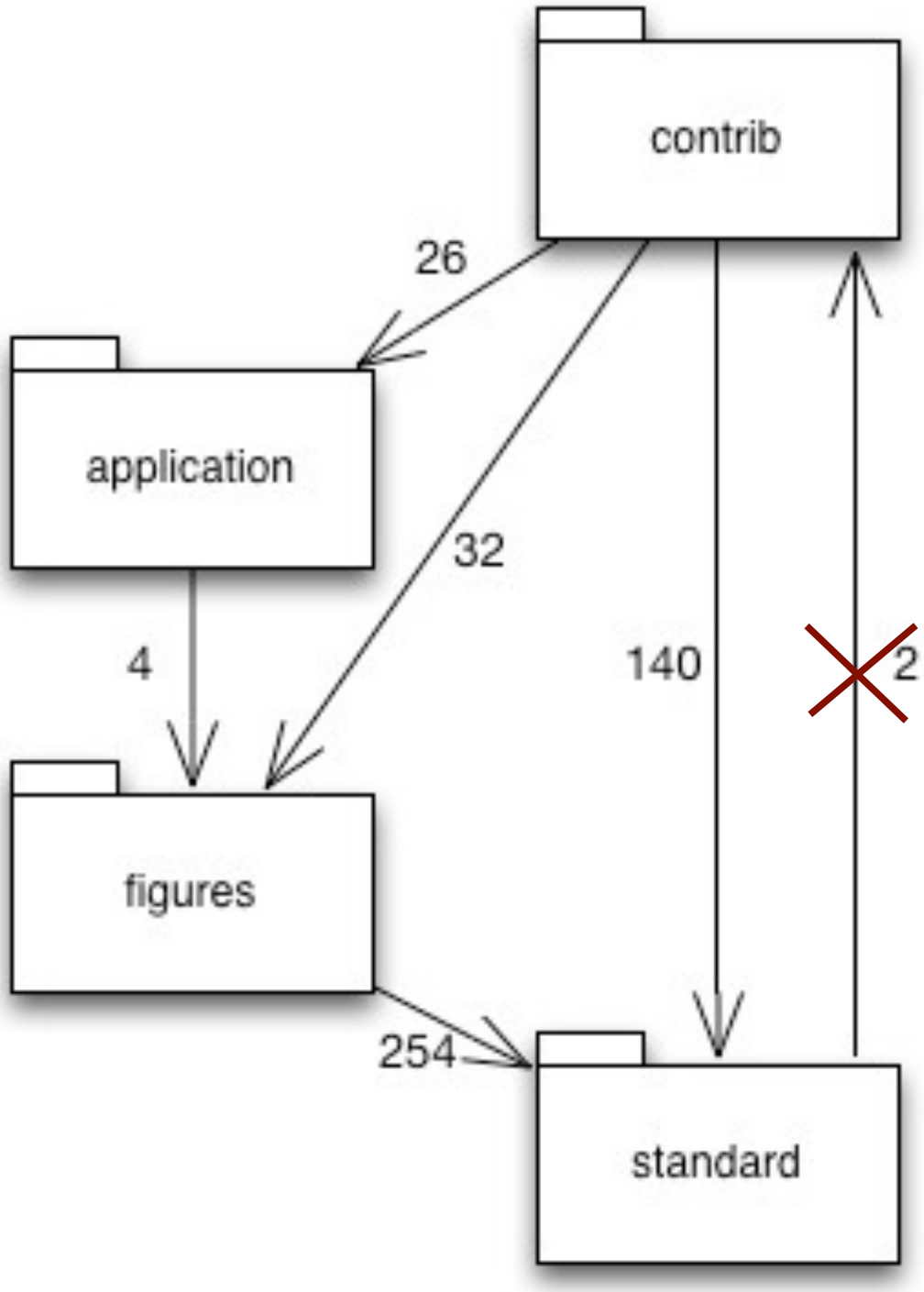
Dependency types

- **Inheritance** : Extends or implement
- **Reference** : `new Object();`
- **Invocation** : `object.equals();`
- **Extension** : Method defined in a package, class is defined in different package

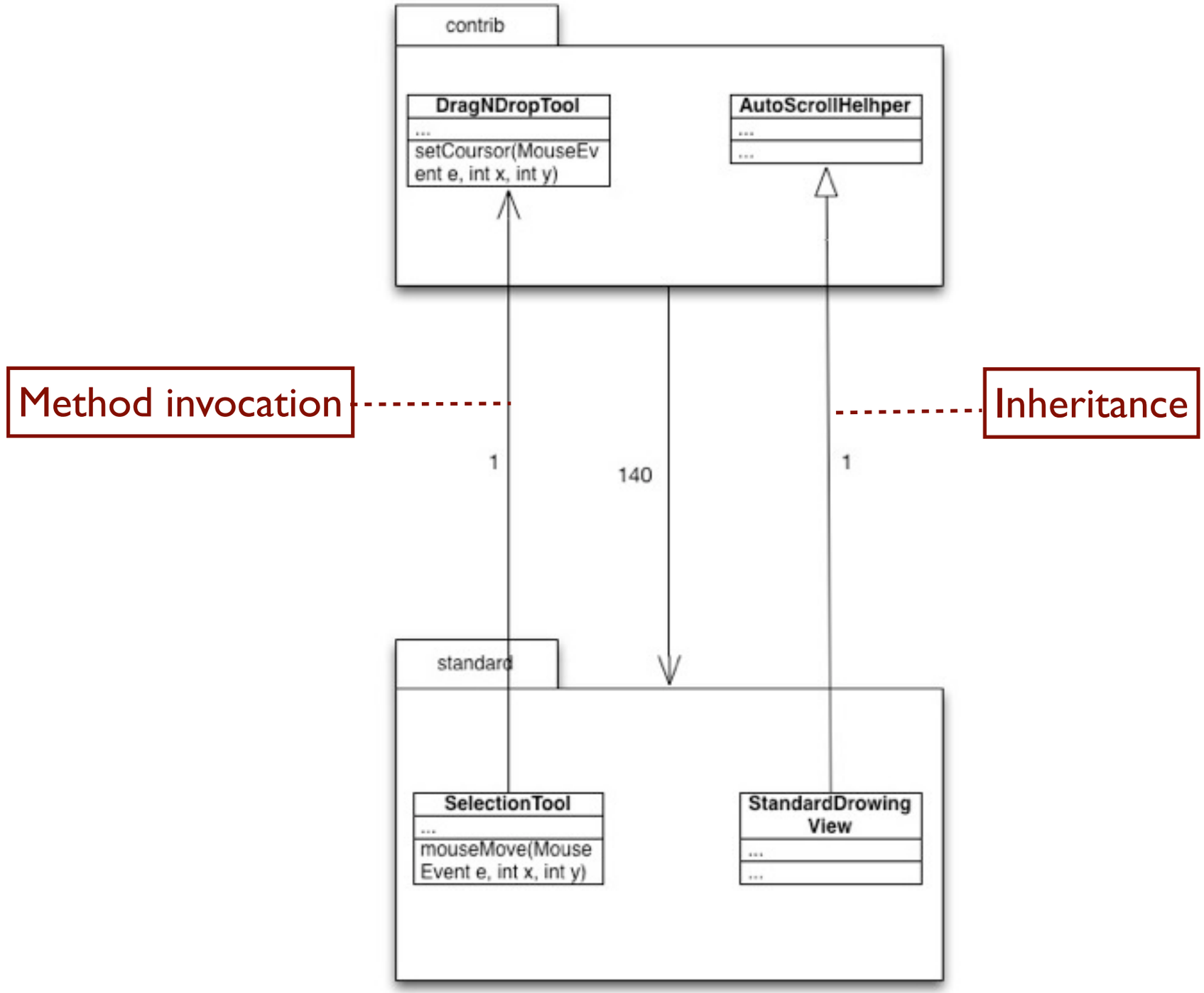
Approaches

- **Move class**
- **Move method**
- The use of **design principles** and **patterns**
  - Dependency Inversion Principle, DIP
  - Dependency Injection Pattern, DI
  - Combination of both DIP + DI

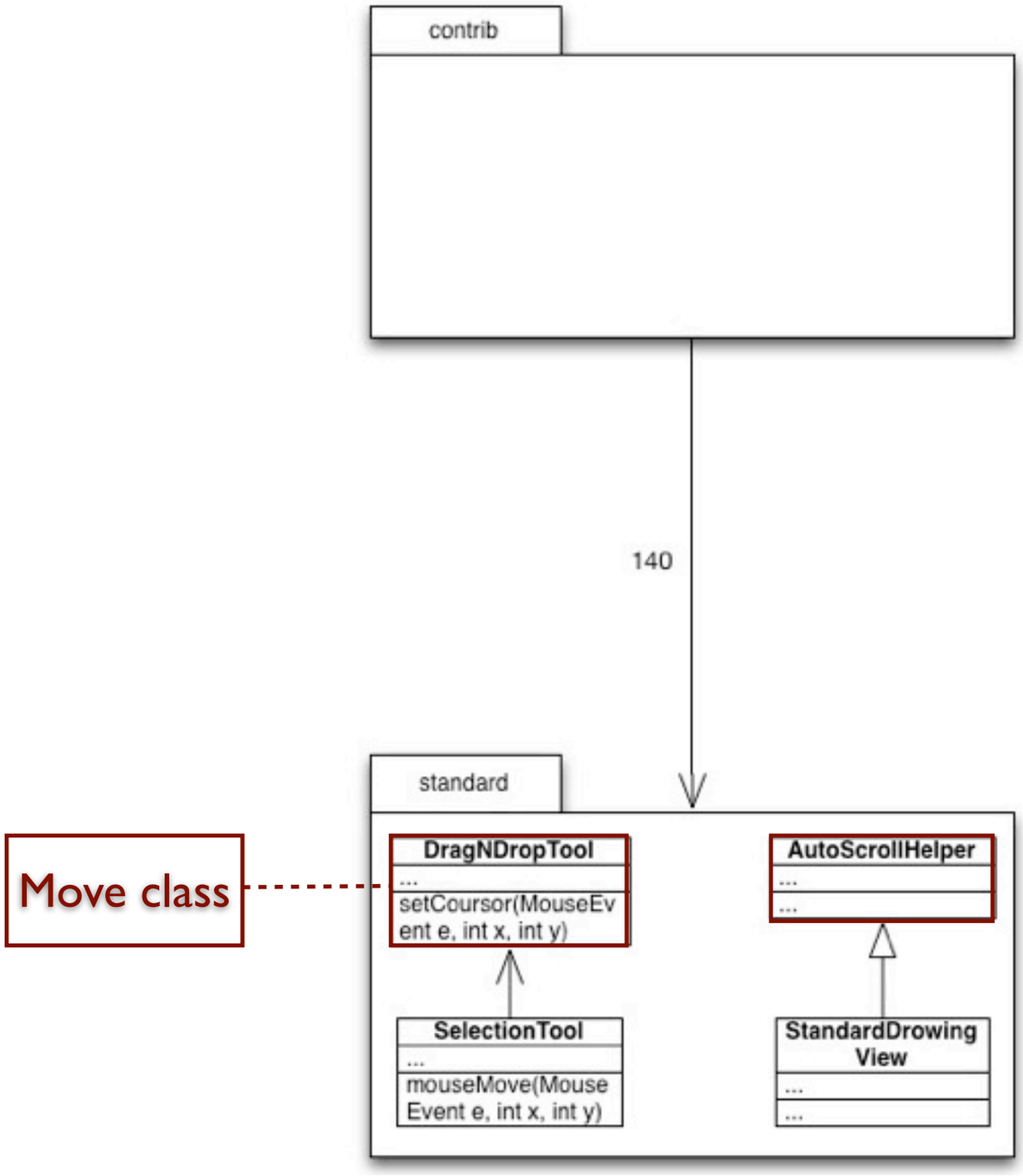
Concrete example



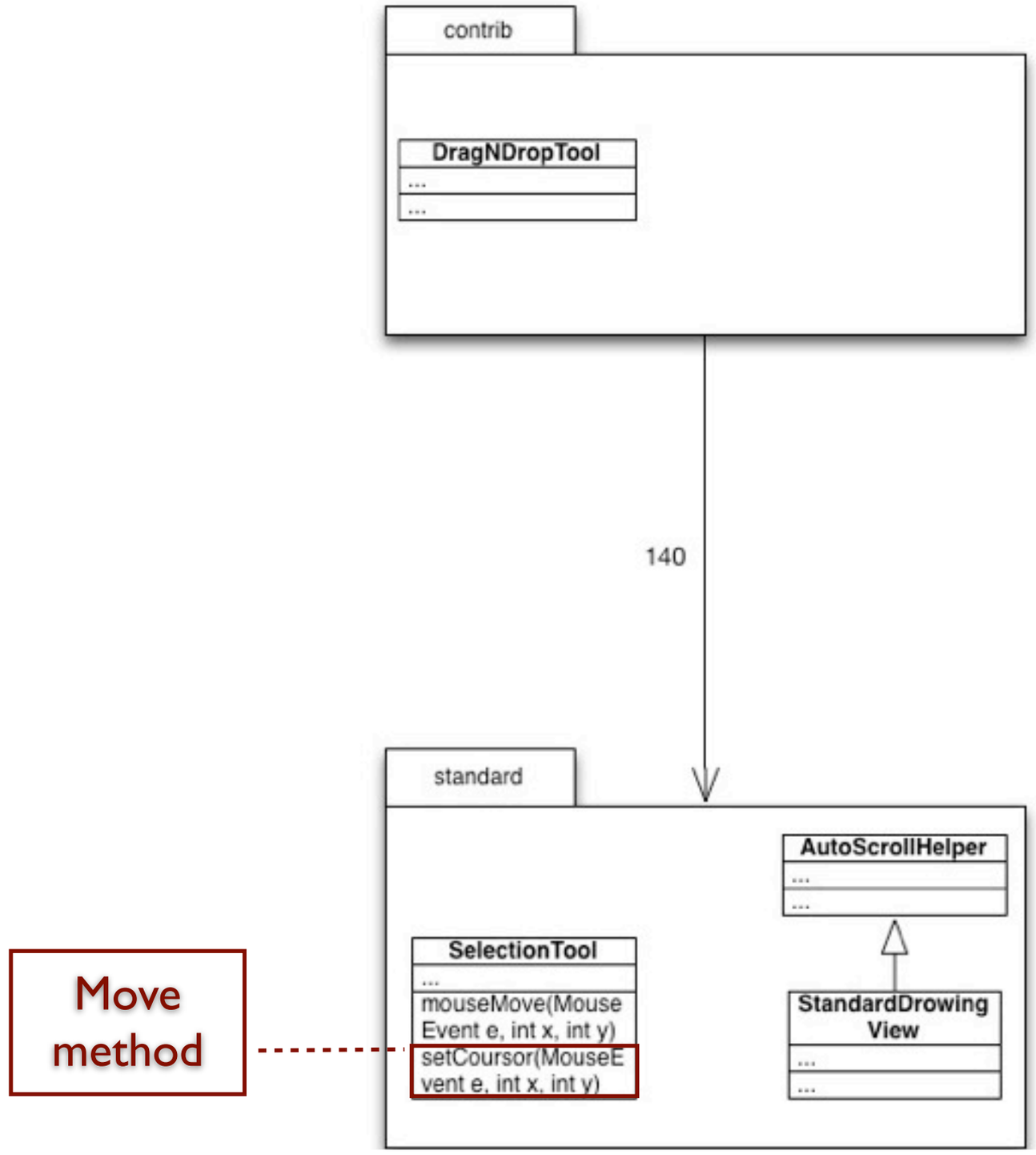
Concrete example



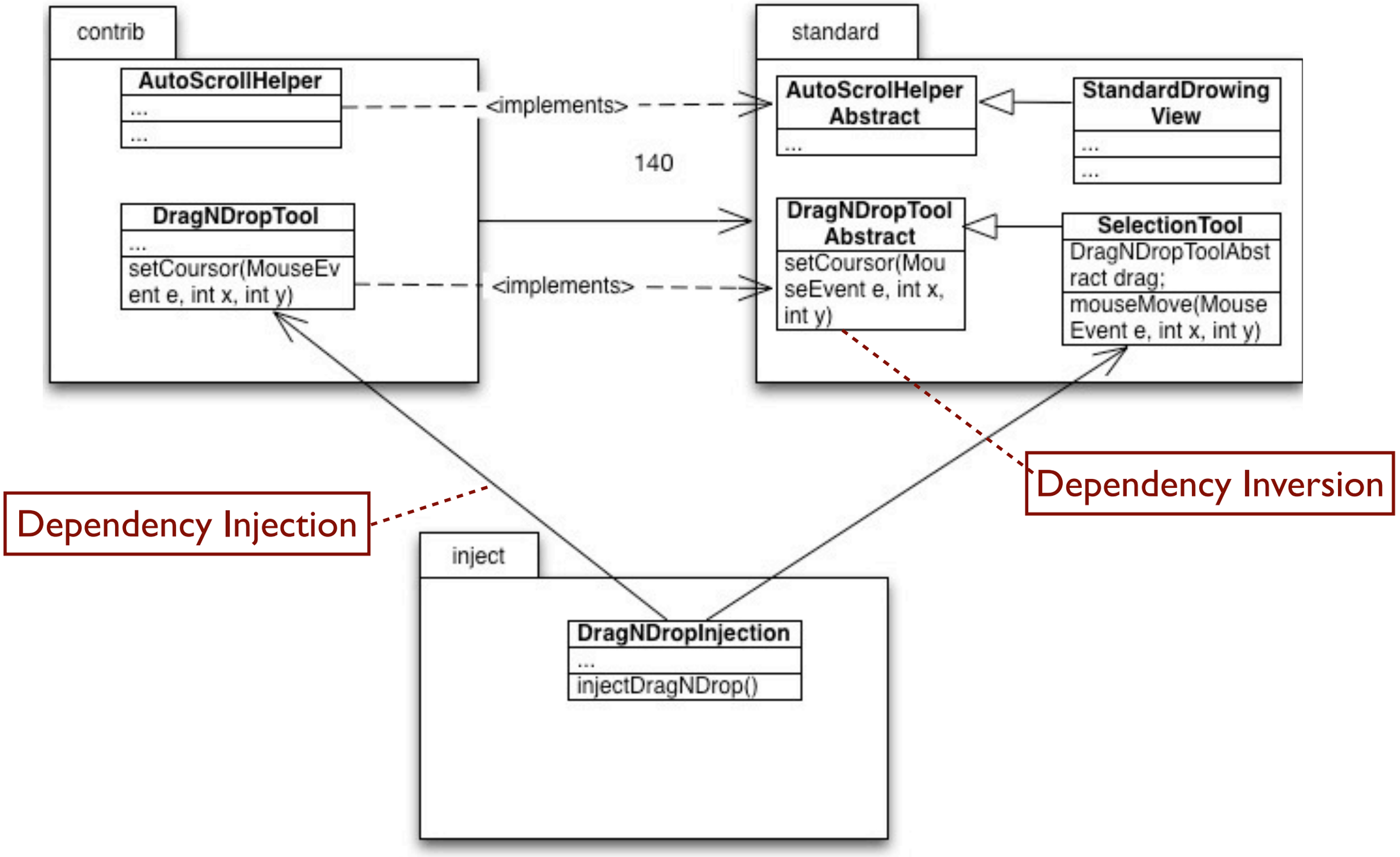
Move class example



Move method example



Design pattern example



Existing approaches

Approach \ Type	Inheritance	Reference	Invocation	Extension
Move class [1]	Yes	Yes	Yes	Yes
Move method [1]	No	No	Yes	Yes
DIP [2]	Yes	No	Yes	Yes
DI [3]	No	Yes	No	No
DIP + DI	Yes	Yes	Yes	Yes

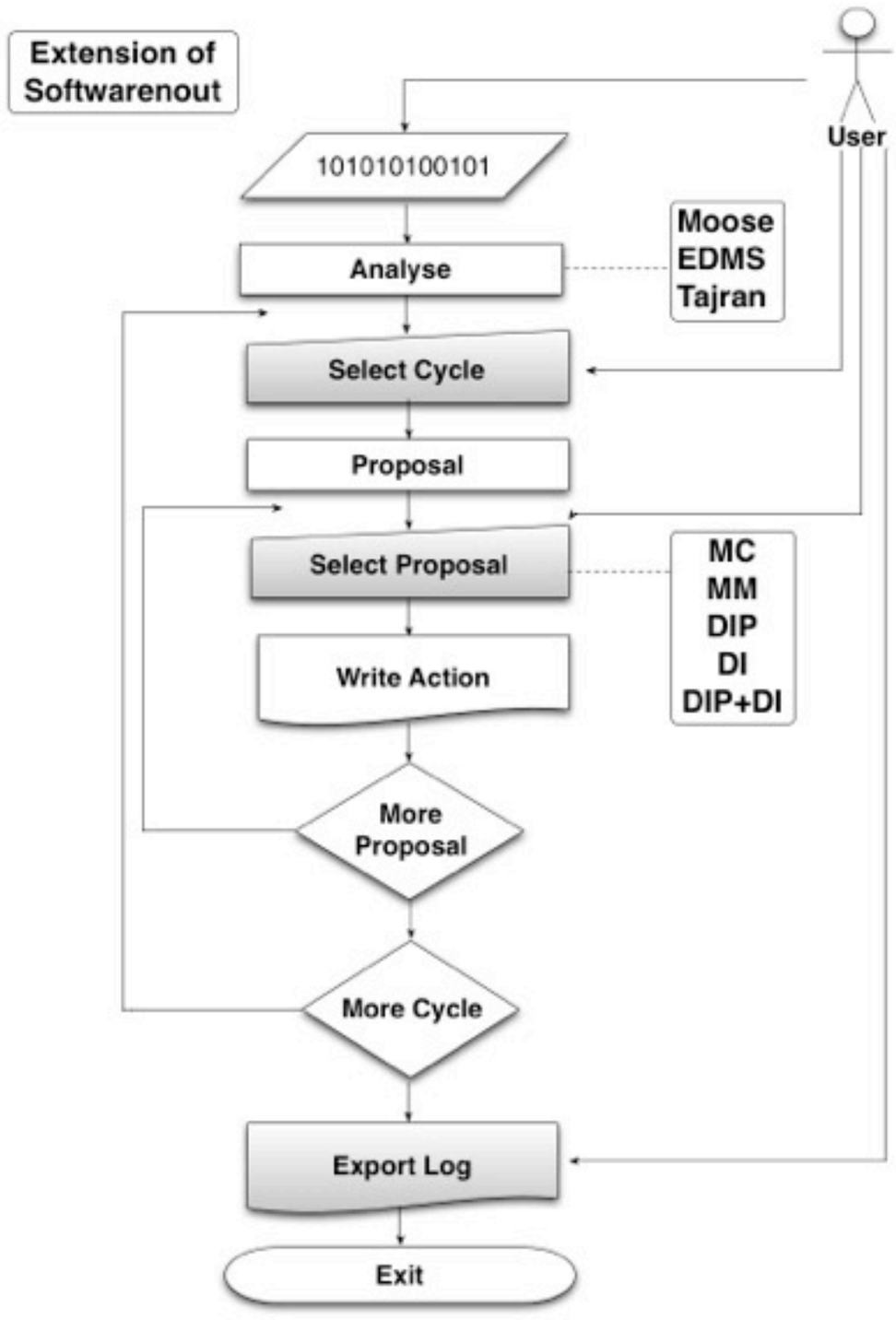
[1] Janik Laval - Package Dependencies Analysis and Remediation in Object-Oriented Systems
[2] Robert C. Martin - Design Principles and Design Patterns
[3] Martin Fowler - Inversion of Control Containers and the Dependency Injection pattern

Existing solutions

Tool \ Type	Analysis	Refactoring	Multi platform
JooJ [1]	Yes	No	No
ByCycle [2]	Yes	No	No
Lattix LDM[3]	Yes	No	No
Pasta [4]	Yes	MC	No
Stan [5]	Yes	No	No
Structure101 [6]	Yes	MC, MM	Yes
EDSM [7]	Yes	MC, MM	Yes

[1] JooJ: Real-time Support for Avoiding Cyclic Dependencies
[2] ByCycle: http://byecycle.sourceforge.net/
[3] Lattix LDM: Using de- pendency models to manage complex software architecture
[4] Pasta: Improving Java software through package structure analysis
[5] Stan: Structure Analysis for Java http://stan4j.com
[6] Structure101: http://structure101.com
[7] Enriched DSM: Package Dependencies Analysis and Remediation in Object-Oriented Systems

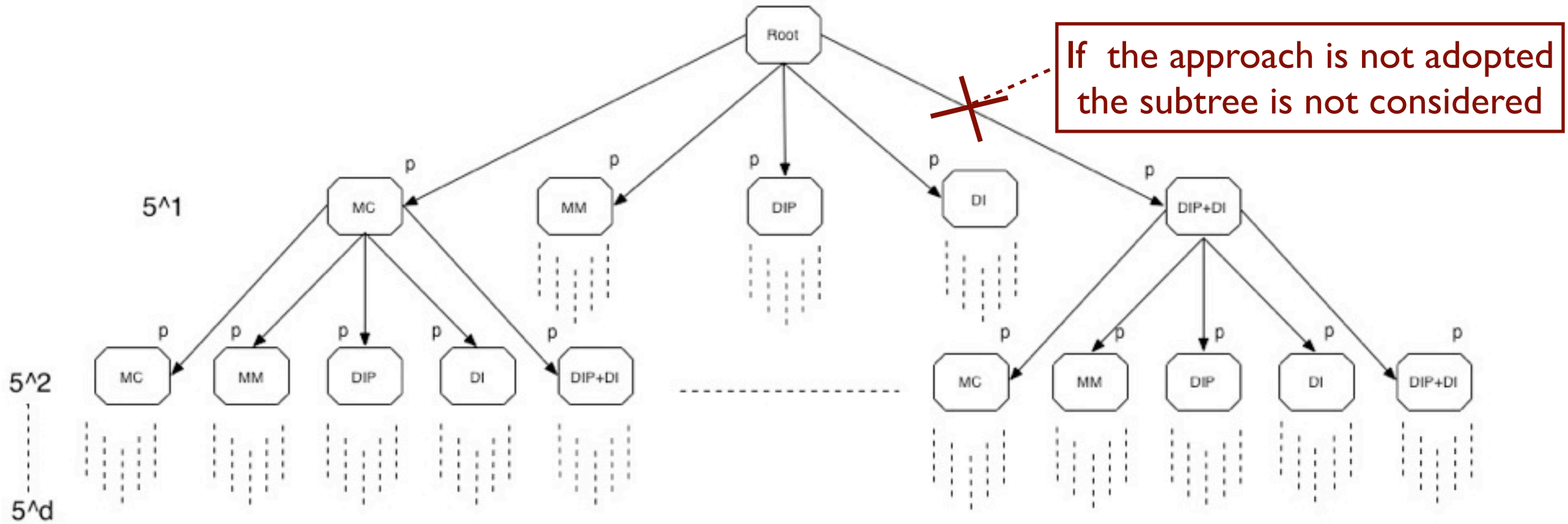
Our solution



Refactoring

```
refactor(model){
    sccs = findSccs(model) // use Tarjan's algorithm
    orderScc(sccs) // min # of cycle
    WHILE sccs NOT EMPTY {
        scc = sccs.getFirstElement()
        WHILE scc NOT EMPTY {
            orderCycle(scc) // min # of arcs
            cycle = scc.getFirstElement()
            WHILE cycle NOT EMPTY {
                orderDependency(cycle) // min # of dependency
                dependency =cycle.getFirstElement()
                propose(dependency) // using approaches
            }
        }
    }
}
```

Best refactoring



Root = starting point

p = profit

d = # of dependences

t = # of types

All possibilities = t^d

MC = Move class

MM = Move method

DIP = Dependency Inversion Principle

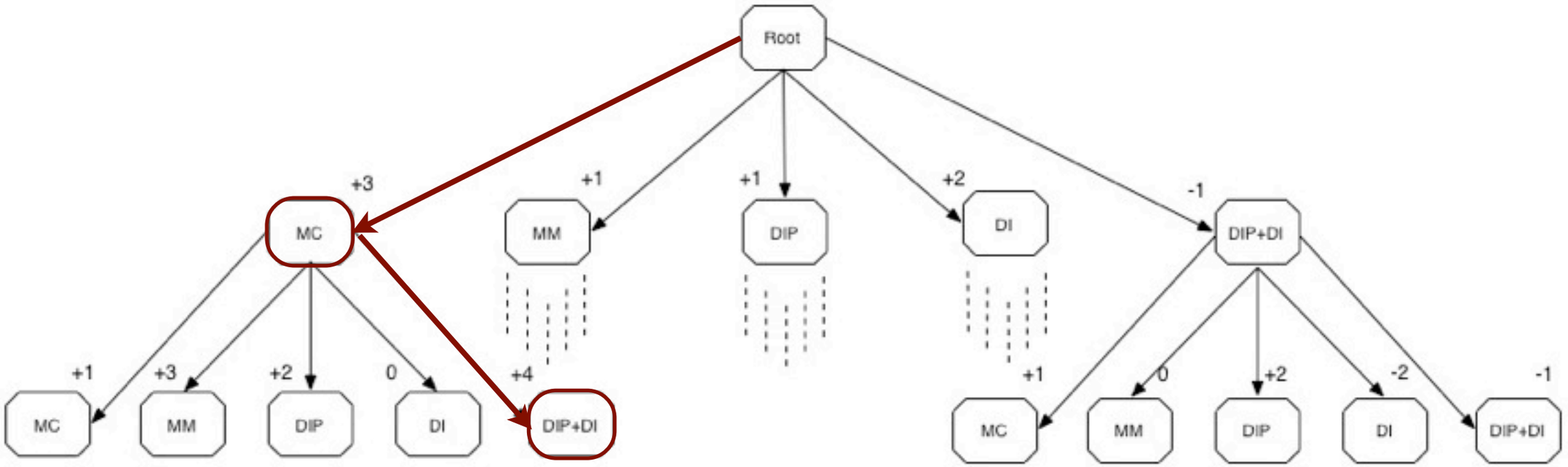
DI = Dependency Injection

Profit

- Profit is equal to the number of deleted dependencies
- We are evaluating other metrics
  - Stable Dependencies Principle, SDP [1]
  - Stable Abstraction Principle SAP [1]

[1] Robert C. Martin - Design Principles and Design Patterns

Best profit example



Root = starting point

MC = Move class

MM = Move method

DIP = Dependency Inversion Principle

DI = Dependency Injection

Best profit = MC + (DIP+DI)
= 3 + 4
= 7

Next steps

- Understanding Moose, Softwarenaut
- Experimenting hypothesis
- Finding new alternatives and evaluate
- Implementation of solution

Summary

- Problem caused by dependency cycles
- Different type of dependencies
- Several approaches
- Semiautomatic solution
- Refactoring with best profit