Advanced Language Mechanisms

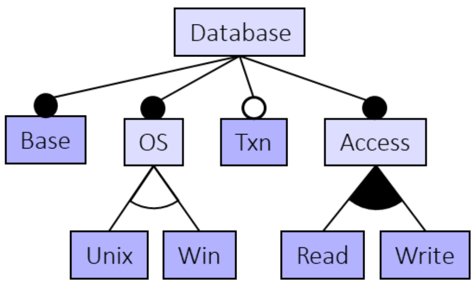
Prof. Sven Apel

Universität des Saarlandes

How to implement variability?

Domain Engineering

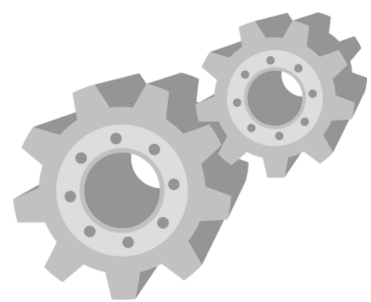
Feature model



Reusable / configurable code

Application Engineering

Feature selection



Variant generator

Software variant

Goals

Solving problems of state-of-the-art techniques

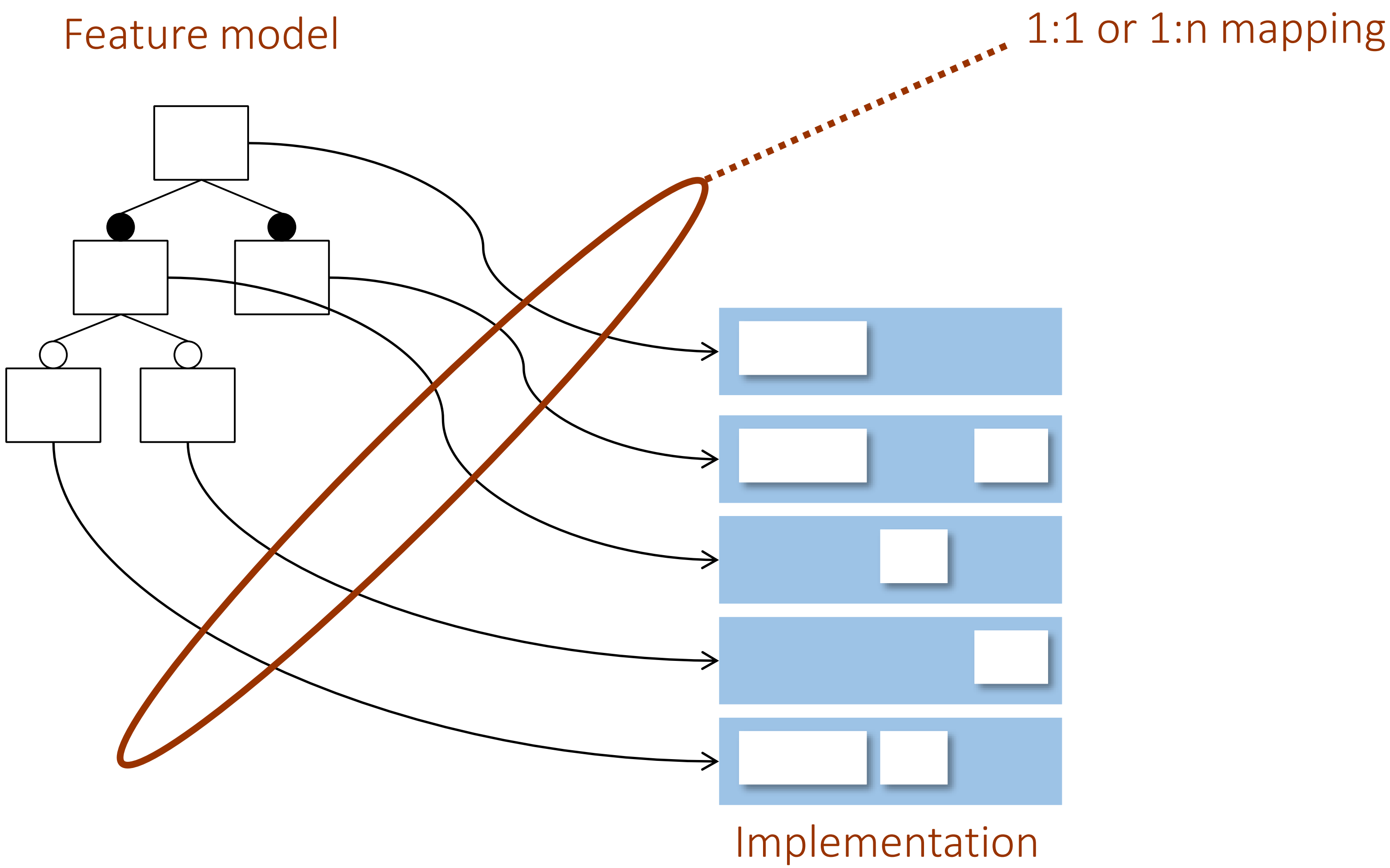
- Feature traceability
- Crosscutting concerns
- Preplanning problem
- Inflexible class extension

→ Modular feature implementation

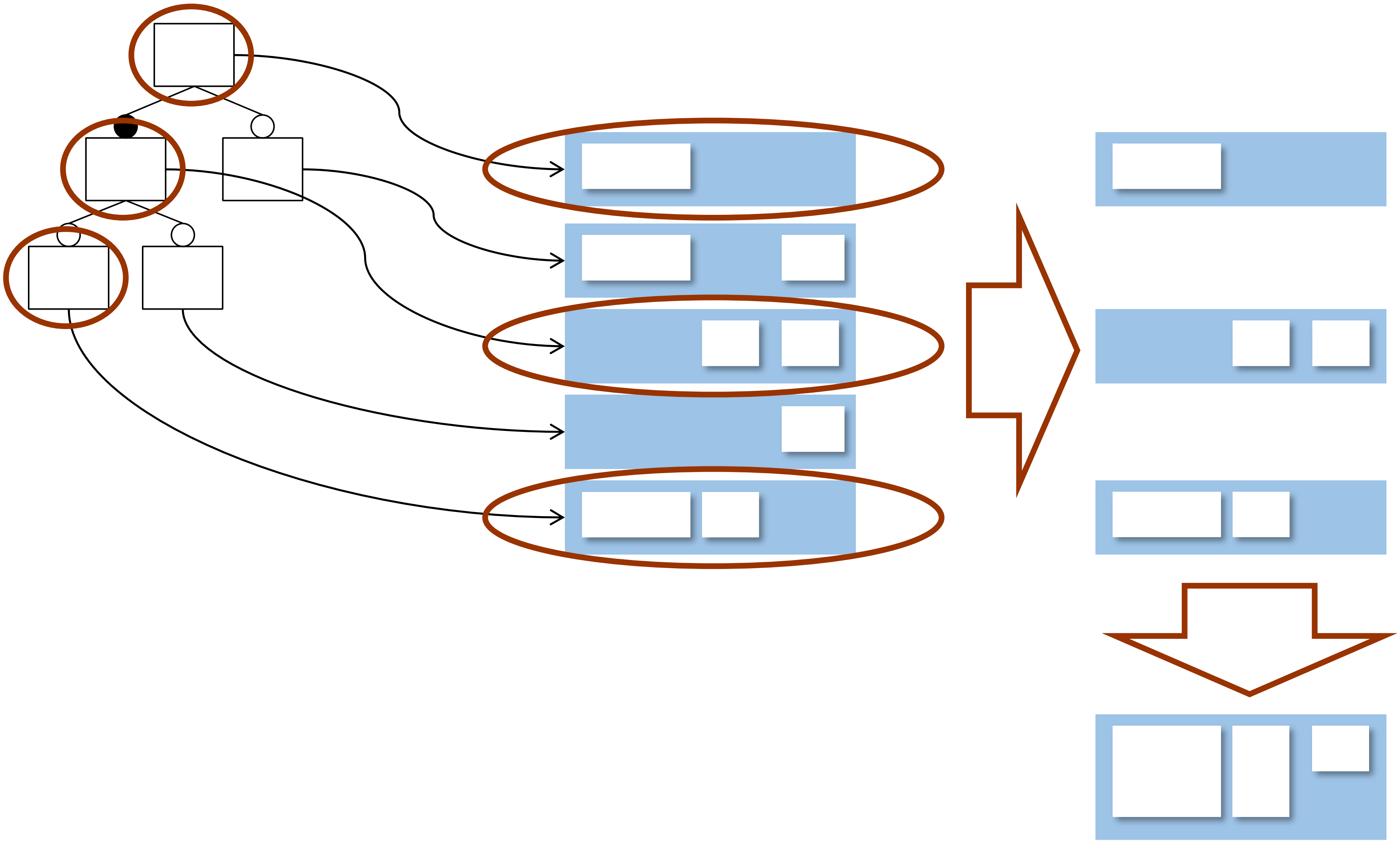
Part I

Collaborations and Roles

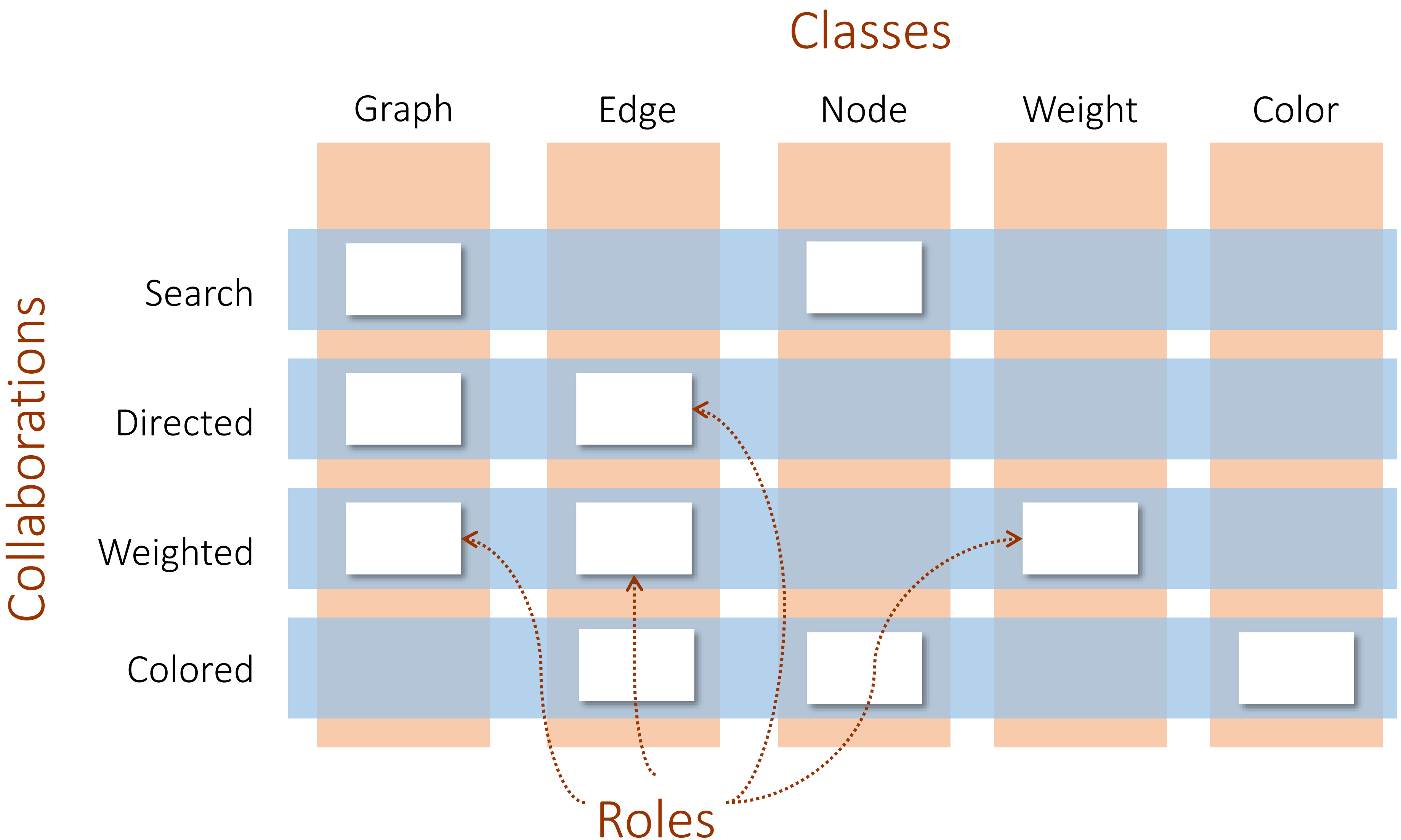
Holy Grail: Feature Modularity



Feature Composition



Collaborations and Roles



Graph Example

```java
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```java
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```java
class Node {
  int id = 0;
  void print() {

System.out.print(id);
  }
}
```

```java
@Role class Graph {
  Edge add(Node n, Node m) {
    Edge e = Super.add(n, m);
    e.weight = new Weight();
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
}
```

```java
@Role class Edge {
  Weight weight = new Weight();
  void print() {
    Super.print(); weight.print();
  }
}
```

```java
class Weight {
  void print() { ... }
}
```

Graph Example

```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i+
      ((Edge)ev.get(i)).print();
  }
}
```

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
```

```
class Node {
  int id = 0;
  void print() {
System.out.print(id);
  }
```

Scala: traits

C++: mixin templates

Ruby: modules

C#: partial classes

…

```
@Role class Graph {
  Edge add(Node n, Node m) {
    Edge e = Super.add(n, m);
    e.weight = new Weight();
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
}
```

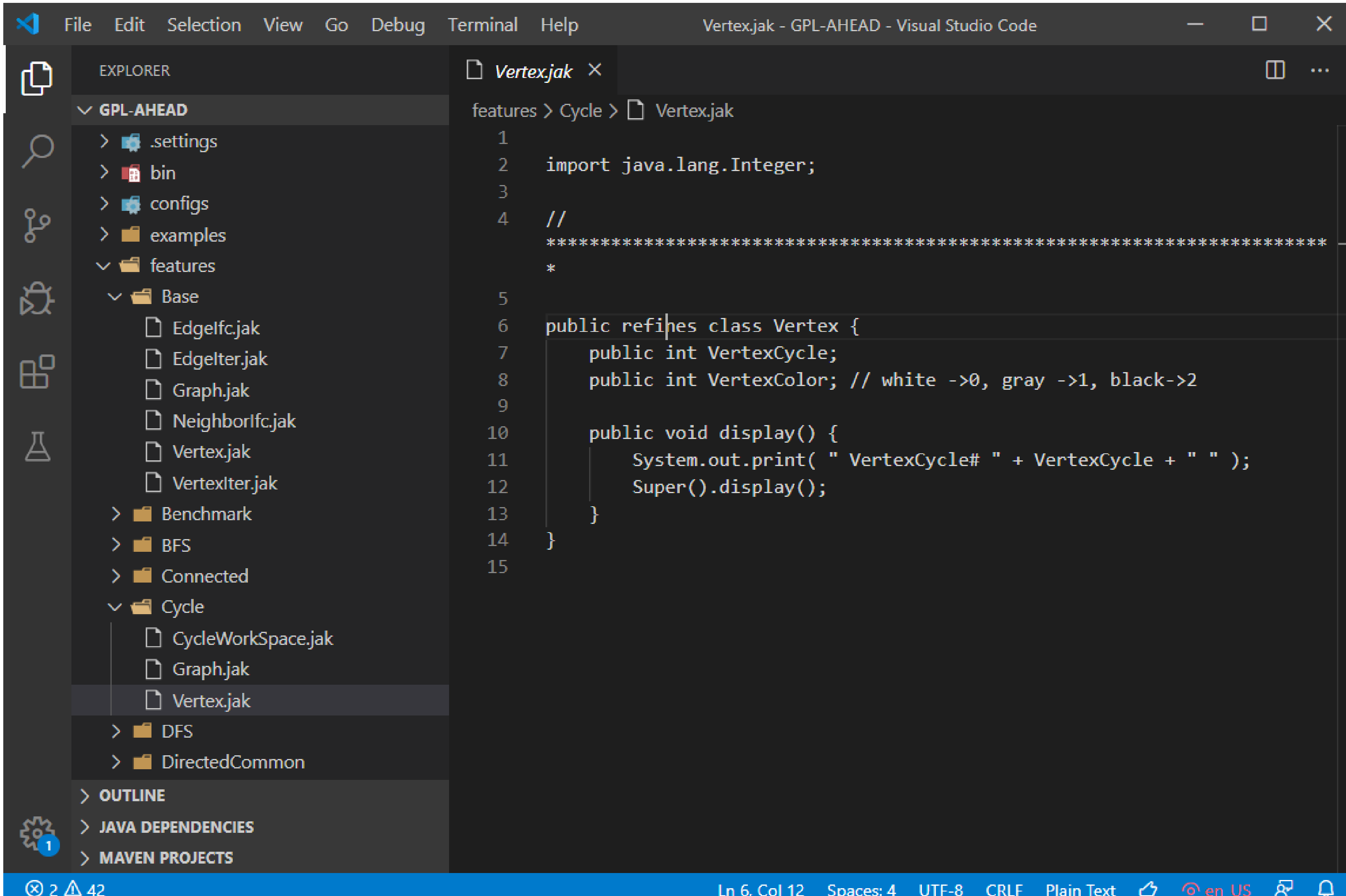
```
@Role class Edge {
  Weight weight = new Weight();
  void print() {
    Super.print(); weight.print();
  }
}
```

```
class Weight {
  void print() { ... }
}
```

Part II

AHEAD

Collaboration = Directory



Role = Class Refinement

Edge.jak

```
class Edge {
    ...
}
```

Edge.jak

```
refines class Edge {
    private Node start;
    ...
}
```

Edge.jak

```
refines class Edge {
    private int weight;
    ...
}
```

Stepwise Refinement

“Imprecise” definition of super class

Method Refinement = Overriding

Like method overriding in subclasses

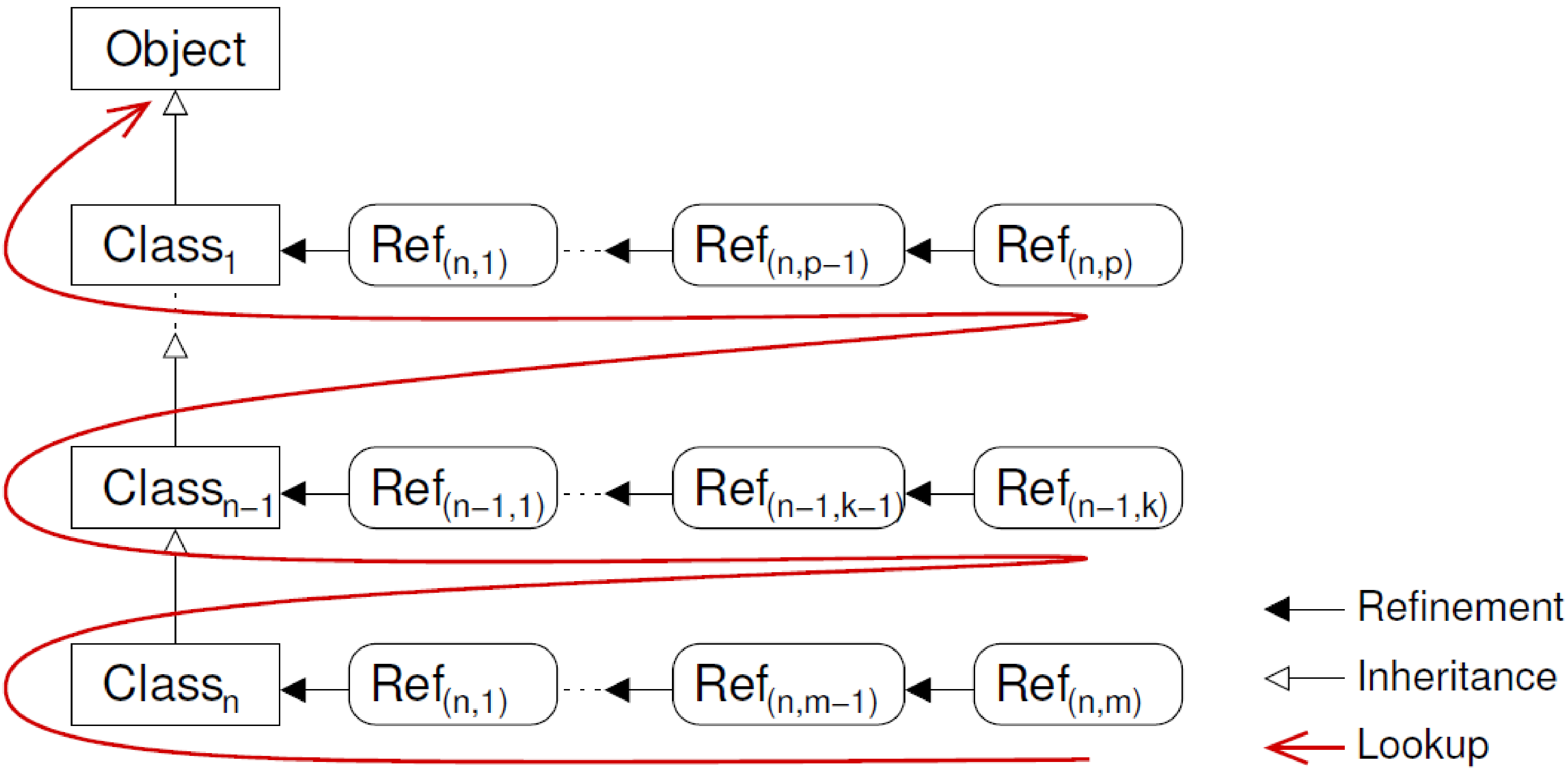
Calling previous method refinement via Super

```
class Edge {
    void print() {
        System.out.print(
          " Edge between " + node1 +
          " and " + node2);
    }
}
```

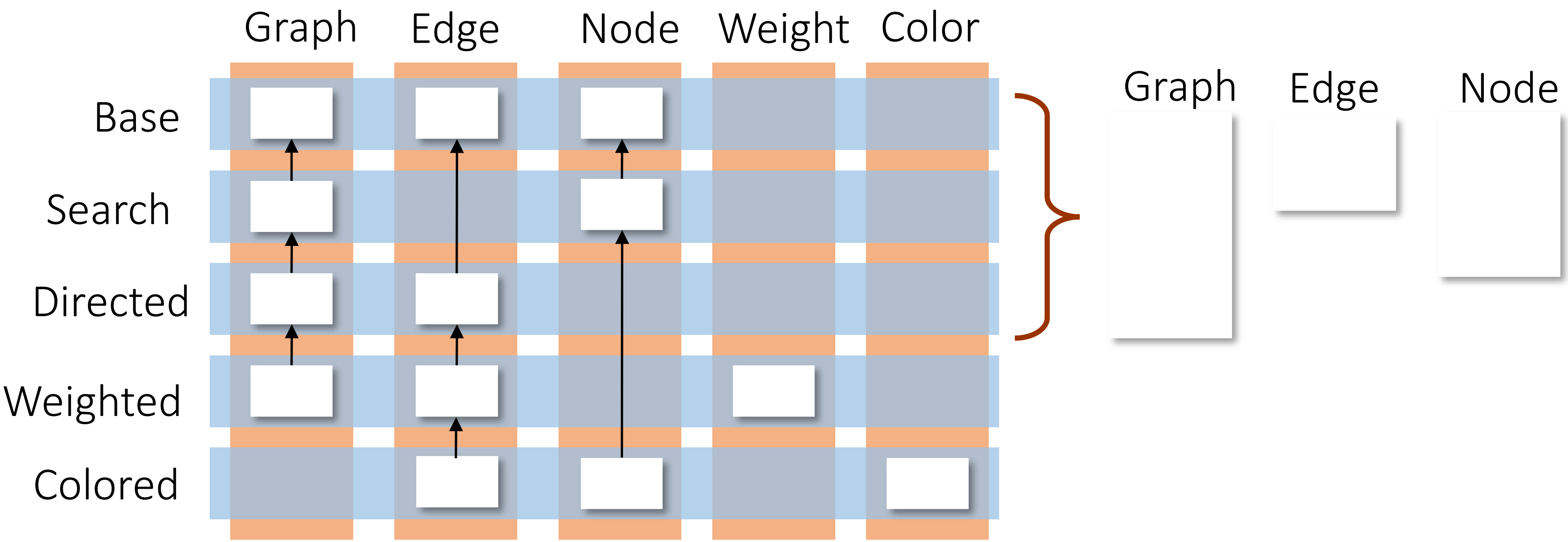
```
refines class Edge {
    private Node start;
    void print() {
        Super.print();
        System.out.print(
            " directed from " + start);
    }
}
```

```
refines class Edge {
    private int weight;
    void print() {
        Super.print();
        System.out.print(
            " weighted with " + weigth);
    }
}
```

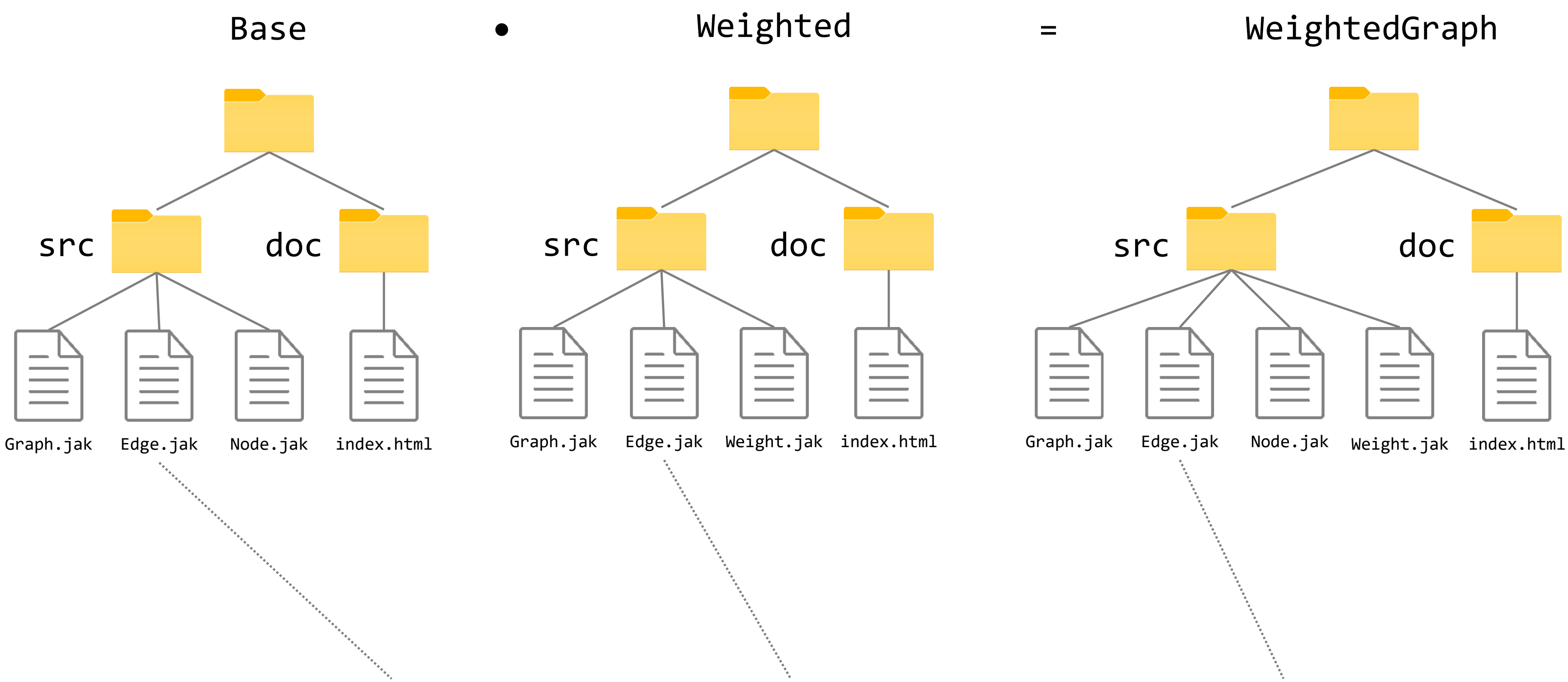
Method Lookup



Graph Example: Composition



Composition = Superimposition



`Base/src/Edge.jak • Weighted/src/Edge.jak = WeightedGraph/src/Edge.jak`

Composition = Superimposition

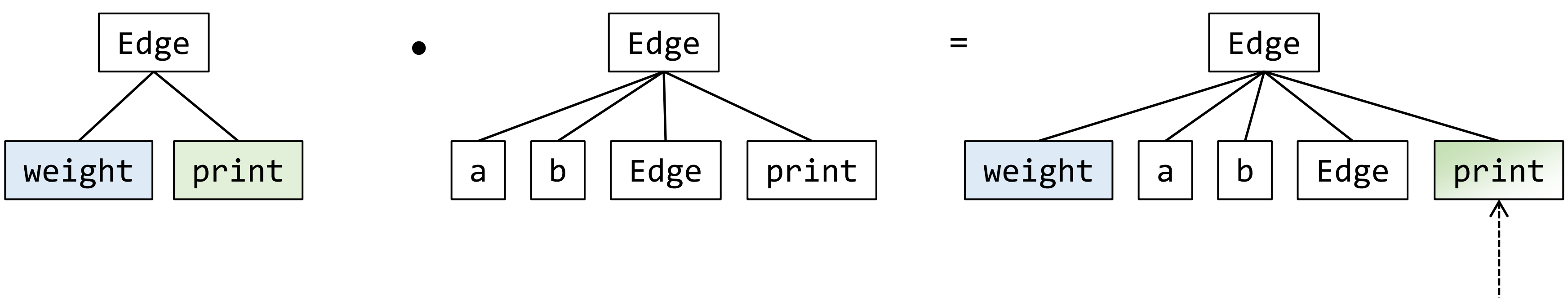
```
refines class Edge {
  Weight weight;
  void print() {
    Super.print();
    weight.print();
  }
}
```

•

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

=

```
class Edge {
  Weight weight;
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
    weight.print();
  }
}
```



Method composition
preserves overriding
semantics!

Tools

AHEAD Tool Suite + Documentation

Command line tools for Jak (Java 1.4 extension)

http://www.cs.utexas.edu/users/schwartz/ATS.html

FeatureHouse

Command-line tools for Java, C#, C, Haskell, UML, …

http://www.fosd.de/fh

FeatureIDE

Eclipse plugin for AHEAD, FeatureHouse and FeatureC++

Automates compilation; syntax highlighting; etc

http://www.fosd.de/featureide

Example in Scala

```
abstract class BasicGraph {
  abstract class Edge {
    def start: Node
    def end: Node
    override def toString(): String =
      return "(" + start + ", " + end + ")"
  }
  abstract class Node {
    def id: Int = 0
    override def toString(): String = return "" + id
  }
}
```

Example in Scala

```
abstract class BasicGraph {
  abstract class Edge {
    def start: Node
    def end: Node
    override def toString(): String =
      return "(" + start + ", " + end + ")"
  }
  abstrac
    def i
    overr
  }
}
```

```
trait WeightedGraph extends BasicGraph {
  trait WeightedEdge extends Edge {
    def weight: Int = 1
    override def toString(): String =
      return super.toString() + " [" + weight + "]"
  }
}
```

Example in Scala

```
abstract class BasicGraph {
  abstract class Edge {
    def start: Node
    def end: Node
    override def toString(): String =
      return "(" + start + ", " + end + ")"
  }
  abstrac
    def i
    overr
  }
}
```

```
trait WeightedGraph extends BasicGraph {
  trait WeightedEdge extends Edge {
    def weight: Int = 1
    override def toString(): String =
      ret
  }
}
```

```
trait ColoredGraph extends BasicGraph {
  trait ColoredEdge extends Edge {
    def color: Int = 255
    override def toString(): String =
      return super.toString() + " // " + color
  }
  trait ColoredNode extends Node {
    def color: Int = 255
    override def toString(): String =
      return super.toString() + " // " + color
  }
}
```

Example in Scala

```
abstract class BasicGraph {
  abstract class Edge {
    def start: Node
    def end: Node
    override def toString(): String =
      return "(" + start + ", " + end + ")"
  }
  abstrac
    def i
    overr
  }
}
```

```
trait WeightedGraph extends BasicGraph {
  trait WeightedEdge extends Edge {
    def weight: Int = 1
    override def toString(): String =
      ret
  }
}
```

```
trait ColoredGraph extends BasicGraph {
  trait ColoredEdge extends Edge {
    def color: Int = 255
    override def toString(): String =
      return super.toString() + " // " + color
  }
  trait ColoredNode extends Node {
    def color: Int = 255
    override def toString(): String =
      return super.toString() + " // " + color
  }
}
```

```
object Graph extends WeightedGraph with ColoredGraph {
  class CEdge extends WeightedEdge with ColoredEdge {
    override def start = new CNode()
    override def end = new CNode()
  }
  class CNode extends Node with ColoredNode {}
  def main(args: Array[String]): Unit = {
    val edge1 = new Graph.CEdge()
    println(edge1)
  }
}
```

Example in Ruby

```
module BasicGraph
  module Edge
    def initialize(s, e)
      @startNode = s
      @endNode = e
    end
    def toString
      return "(" + @startNode.toString.to_s +
        ", " + @endNode.toString.to_s + ")"
    end
  end
  module Node
    def initialize(i)
      @id = i
    end
    def toString
      return @id.to_s
    end
  end
end
```

Example in Ruby

```
module BasicGraph
  module Edge
    def initialize(s, e)
      @startNode = s
      @endNode =
    end
    def toString
      return "("
        ", " + @e
    end
  end
  module Node
    def initializ
      @id = i
    end
    def toString
      return @id.to_s
    end
  end
end
```

```
module WeightedGraph
  module Edge
    def initialize(s, e)
      super(s,e)
      @weight = 1
    end
    def toString
      return super + " [" + @weight.to_s + "]"
    end
  end
end
```

Example in Ruby

```
module BasicGraph
  module Edge
    def initialize(s, e)
      @startNode = s
      @endNode =
    end
    def toString
      return "("
        ", " + @e
    end
  end
  module Node
    def initializ
      @id = i
    end
    def toString
      return @id.to_s
    end
  end
end
```

```
module WeightedGraph
  module Edge
    def initialize(s, e)
      super(s,e)
      @we
    end
    def t
      ret
    end
  end
end
```

```
module ColoredGraph
  module Edge
    def initialize(s, e)
      super(s,e)
      @color = 255
    end
    def toString
      return super + " // " + @color.to_s
    end
  end
  module Node
    def initialize(i)
      super(i)
      @color = 255
    end
    def toString
      return super + " // " + @color.to_s
    end
  end
end
```

Example in Ruby

```
module BasicGraph
  module Edge
    def initialize(s, e)
      @startNode = s
      @endNode =
    end
    def toString
      return "("
        ", " + @e
    end
  end
  module Node
    def initializ
      @id = i
    end
    def toString
      return @id.to_s
    end
  end
end
```

```
module WeightedGraph
  module Edge
    def initialize(s, e)
      super(s,e)
      @we
    end
    def t
      ret
    end
  end
end
```

```
module ColoredGraph
  module Edge
    def initialize(s, e)
      super(s,e)
      @color = 255
    end
    def toString
      return super + " // " +
    end
  end
  module Node
    def initialize(i)
      super(i)
      @color = 255
    end
    def toString
      return super + " // " +
    end
  end
end
```

```
module Graph
  class Edge
    include  BasicGraph::Edge
    include  WeightedGraph::Edge
    include  ColoredGraph::Edge
    def initialize(s, e)
      super(s,e)
    end
  end
  class Node
    include BasicGraph::Node
    include ColoredGraph::Node
    def initialize(i)
      super(i)
    end
  end
end

edge = Graph::Edge.new(Graph::Node.new(11),
Graph::Node.new(22))
node = Graph::Node.new(3)
puts edge.toString
```

Example in Rust

```
trait Node { fn to_string(&self) -> String; }
struct BaseNode { id: u32 }
impl Node for BaseNode {
  fn to_string(&self) -> String {
    String::from(self.id.to_string())
  }
}

trait Edge { fn to_string(&self) -> String; }
struct BaseEdge<NodeTy: Node> {
  start: NodeTy,
  end: NodeTy,
}
impl<NodeTy: Node> Edge for BaseEdge<NodeTy> {
  fn to_string(&self) -> String {
    format!("({}, {})", self.start.to_string(),
            self.end.to_string())
  }
}

struct Graph<NodeTy: Node, EdgeTy: Edge> {
  nodes: Vec<NodeTy>,
  edges: Vec<EdgeTy,
}
```

Example in Rust

```
trait Node { fn to_string(&self) -> String; }
struct BaseNode { id: u32 }
impl Node for BaseNode {
  fn to_string(&self) -> String {
    String::from(
  }
}

trait Edge { fn t
struct BaseEdge<N
  start: NodeTy,
  end: NodeTy,
}
impl<NodeTy: Node
  fn to_string(&s
    format!("({},
            self.
  }
}

struct Graph<Node
  nodes: Vec<Node
  edges: Vec<Edge
}
```

```
struct WeightedNode<NodeTy: Node> {
  wrapped: NodeTy,
  weight: u32,
}
impl<NodeTy: Node> Node for WeightedNode<NodeTy> {
  fn to_string(&self) -> String {
    format!("{}[{}]", self.wrapped.to_string(), self.weight)
  }
}

type WNode = WeightedNode<BaseNode>;
type WEdge = BaseEdge<WNode>;
type WeightedGraph = Graph<WNode, WEdge>;
impl ColoredGraph {
  fn node(id: u32, weight: u32) -> WNode {
    WeightedNode { wrapped: BaseNode { id }, weight }
  }
  fn edge(start: WNode, end: WNode) -> WEdge {
    BaseEdge { start, end }
  }
}
```

Example in Rust

```
trait Node { fn to_string(&self) -> String; }
struct BaseNode { id: u32 }
impl Node for BaseNode {
  fn to_string(&self) -> String {
    String::from(
  }
}

trait Edge { fn t
struct BaseEdge<N
  start: NodeTy,
  end: NodeTy,
}
impl<NodeTy: Node
  fn to_string(&s
    format!("({},
            self.
  }
}

struct Graph<Node
  nodes: Vec<Node
  edges: Vec<Edge
}
```

```
struct WeightedNode<NodeTy: Node> {
  wrapped: NodeTy,
  weight: u32,
}
impl<Node
  fn to_s
    forma
  }
}

type WNod
type WEdg
type Weig
impl Colo
  fn node
    Weigh
  }
  fn edge
    BaseE
  }
}
```

```
struct ColoredNode<NodeTy: Node> { wrapped: NodeTy, color: u8 }
impl<NodeTy: Node> Node for ColoredNode<NodeTy> {
  fn to_string(&self) -> String {
    format!("{} // {}", self.wrapped.to_string(), self.color)
  }
}
struct ColoredEdge<EdgeTy: Edge> { wrapped: EdgeTy, color: u8 }
impl<EdgeTy: Edge> Edge for ColoredEdge<EdgeTy> {
  fn to_string(&self) -> String {
    format!("{} // {}", self.wrapped.to_string(), self.color)
  }
}

type CNode = ColoredNode<BaseNode>;
type CEdge = ColoredEdge<BaseEdge<CNode>>;
type ColoredGraph = Graph<CNode, CEdge>;
impl ColoredGraph {
  fn node(id: u32, color: u8) -> CNode {
    ColoredNode { wrapped: BaseNode { id }, color }
  }
  fn edge(start: CNode, end: CNode, color: u8) -> CEdge {
    ColoredEdge { wrapped: BaseEdge { start, end }, color }
  }
}
```

Example in Rust

```rust
trait Node { fn to_string(&self) -> String; }
struct BaseNode { id: u32 }
impl Node for BaseNode {
  fn to_string(&self) -> String {
    String::from(
  }
}

trait Edge { fn t
struct BaseEdge<N
  start: NodeTy,
  end: NodeTy,
}
impl<NodeTy: Node
  fn to_string(&s
    format!("({},
            self.
  }
}

struct Graph<Node
  nodes: Vec<Node
  edges: Vec<Edge
}
```

```rust
struct WeightedNode<NodeTy: Node> {
  wrapped: NodeTy,
  weight: u32,
}
impl<Node
  fn to_s
    forma
  }
}

type WNod
type WEdg
type Weig
impl Colo
  fn node
    Weigh
  }
  fn edge
    BaseE
  }
}
```

```rust
struct ColoredNode<NodeTy: Node> { wrapped: NodeTy, color: u8 }
impl<NodeTy: Node> Node for ColoredNode<NodeTy> {
  fn to_string(&self) -> String {
    format!("{} // {}", self.wrapped.to_string(), self.color)
  }
}
struct ColoredE
impl<EdgeTy: Ec
  fn to_string(
    format!("{}
  }
}

type CNode = Co
type CEdge = Co
type ColoredGra
impl ColoredGra
  fn node(id: u
    ColoredNode
  }
  fn edge(start
    ColoredEdge
  }
}
```

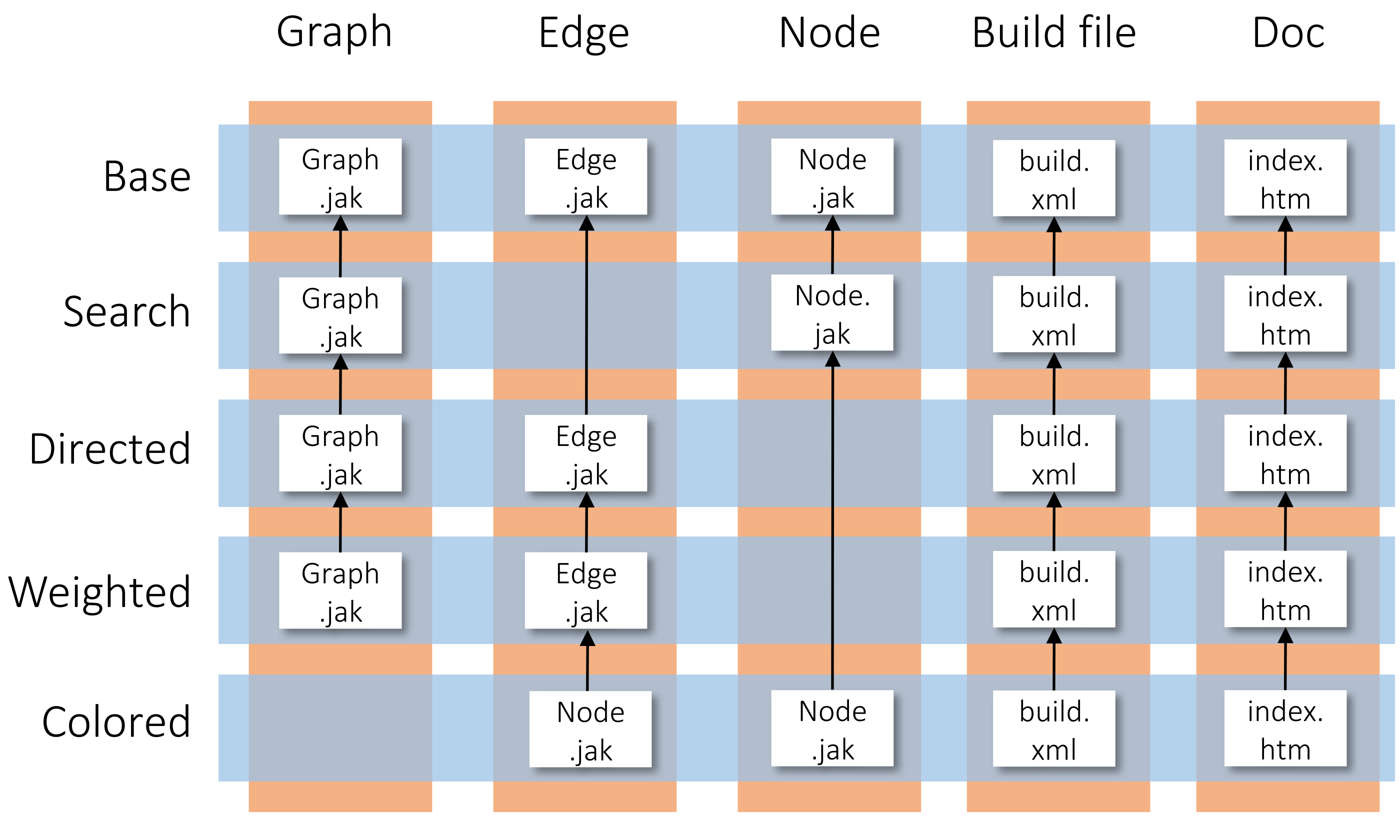
```rust
type CWNode = ColoredNode<WeightedNode<BaseNode>>;
type CWEdge = ColoredEdge<BaseEdge<CWNode>>;
type ColoredWeightedGraph = Graph<CWNode, CWEdge>;
impl ColoredWeightedGraph {
  fn node(id: u32, weight: u32, color: u8) -> CWNode {
    ColoredNode { wrapped: WeightedNode {
      wrapped: BaseNode { id }, weight }, color
    }
  }
  fn edge(start: CWNode, end: CWNode, color: u8) -> CWEdge {
    ColoredEdge { wrapped: BaseEdge { start, end }, color }
  }
}

fn main() {
  let node1 = ColoredWeightedGraph::node(42, 3, 255);
  let node2 = ColoredWeightedGraph::node(1337, 7, 0);
  let edge = ColoredWeightedGraph::edge(node1, node2, 128);
  println!("{}", edge.to_string());
}
```

Principle of Uniformity

Features are implemented by a diverse selection of software artifacts and any kind of software artifact can be subject of subsequent refinement.

– Don Batory

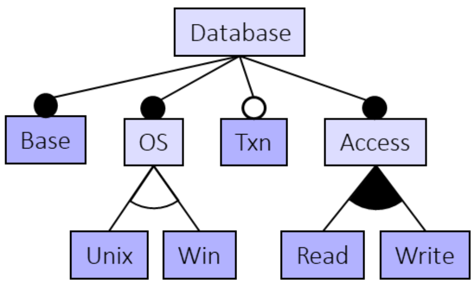
Graph Example: Uniformity



How to implement variability?

Domain Engineering

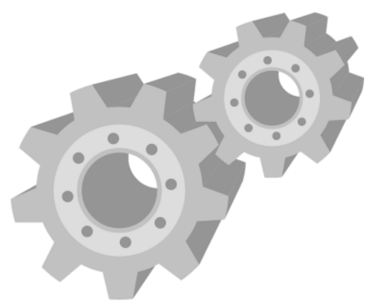
List of features & mapping to collaborations



Set of collaborations with classes and roles

Application Engineering

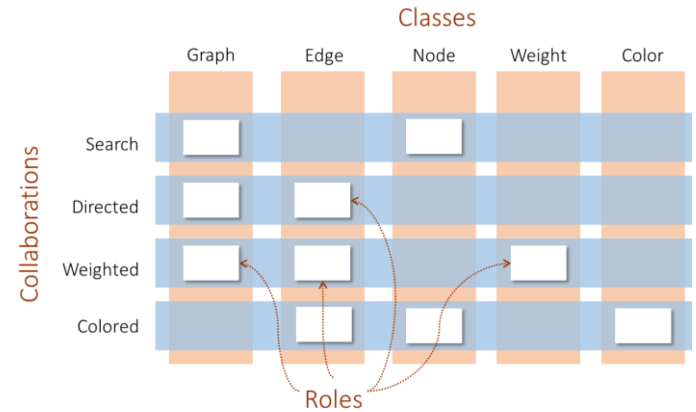
Feature selection
(collaboration selection)



Superimposition

Software variant

Discussion



Feature traceability ✓

Crosscutting concerns ✓

Preplanning problem ✓

Inflexible class extension ✓

Literature

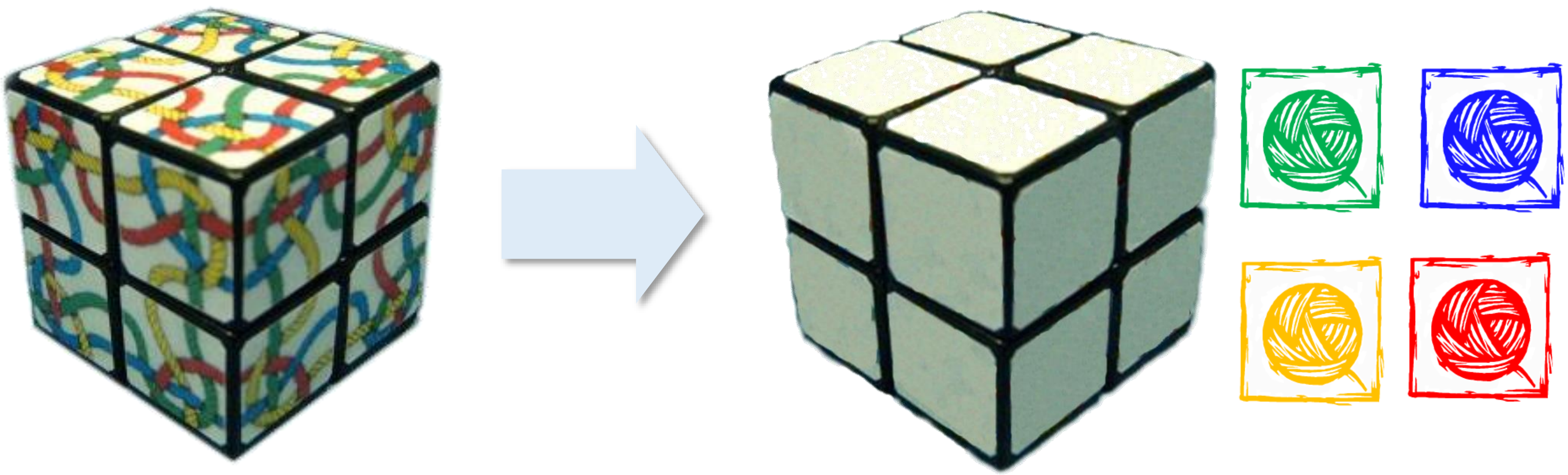
D. Batory, et al.: *Scaling step-wise refinement*. IEEE TSE 30(6): 355-371, 2004

S. Apel, et al.: *Language-Independent and automated software composition: The FeatureHouse experience*. IEEE TSE 39(1): 63-79, 2013

Part II

Aspects, Pointcuts, and Advice

Idea: Modularizing Crosscutting Concerns

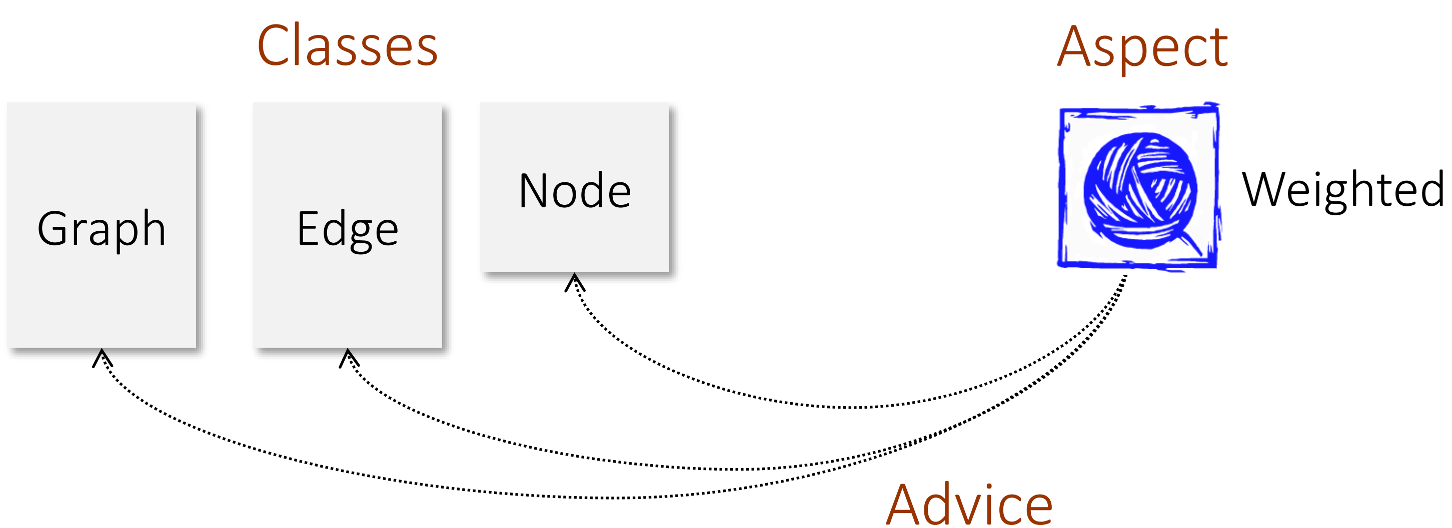


Idea

One aspect implements *one feature*

This aspect describes the changes necessary to add the feature to the base system

Technically: source code or binary transformation, runtime observer & dispatcher, meta-object protocol, ...



Pointcuts and Advice

Join point: event during program execution (method call, field access, class init, ...)

Pointcut: predicate to select a set of join points

```
aspect Weighted {
    ...
    pointcut printExecution(Edge edge) :
        execution(void Edge.print()) && this(edge);

    after(Edge edge) : printExecution(edge) {
        System.out.print(' weight ' + edge.weight);
    }
}
```

Advice: code to be executed once a selected join point occurs at runtime

Join Points: Example

```
class Test {
  MathUtil u;
  public void main() {
    u = new MathUtil();
    int i = 2;
    i = u.twice(i);
    System.out.println(i);
  }
}
class MathUtil {
  public int twice(int i) {
    return i * 2;
  }
}
```

method execution

field access (set)

constructor call

field access (get)

method call

method call

method execution

Pointcut example: execution

Selects the execution of a method

```
aspect A1 {
  after() : execution(int MathUtil.twice(int)) {
    System.out.println("MathUtil.twice executed");
  }
}
```

```
class Test {
  public static void main(String[] args) {
    MathUtil u = new MathUtil();
    int i = 2;
    i = u.twice(i);
    System.out.println(i);
  }
}
class MathUtil {
  public int twice(int i) {
    return i * 2;
  }
}
```

Execution

Syntax:

```
execution(ReturnType ClassName.Methodname(ParameterTypes))
```

Patterns

“Imprecise“ definition of target join points

```
aspect Execution {
  pointcut P1() : execution(int MathUtil.twice(int));

  pointcut P2() : execution(* MathUtil.twice(int));

  pointcut P3() : execution(int MathUtil.twice(*));

  pointcut P4() : execution(int MathUtil.twice(..));

  pointcut P5() : execution(int MathUtil.*(int, ..));

  pointcut P6() : execution(int *Util.tw*(int));

  pointcut P7() : execution(int *.twice(int));

  pointcut P8() : execution(int MathUtil+.twice(int));

  pointcut P9() : execution(public int package.MathUtil.twice(int)
  throws ValueNotSupportedException);

  pointcut Ptypisch() : execution(* MathUtil.twice(..));
}
```

* as wildcard for value or type

.. as wildcard for multiple values or types

+ for subclasses

Graph Example

Basic Graph

```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    System.out.print(id);
  }
}
```

Color

```
aspect ColorAspect {
  Color Node.color = new Color();
  Color Edge.color = new Color();
  before(Node c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  before(Edge c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  static class Color { ... }
}
```

Typical Aspects I

Logging, Tracing, Profiling

Adds similar code to many methods

```
aspect Profiler {
    /** record time to execute my public methods */
    Object around() : execution(public * com.company..*.* (..)) {
        long start = System.currentTimeMillis();
        try {
            return proceed();
        } finally {
            long end = System.currentTimeMillis();
            printDuration(start, end,
                thisJoinPoint.getSignature());
        }
    }
    // implement recordTime...
}
```

Typical Aspects II

Caching, Pooling

Central implementation of cache or resource pool, which is then used in many places of the program

```
aspect ConnectionPooling {
    ...
    Connection around() : call(Connection.new()) {
        if (enablePooling)
            if (!connectionPool.isEmpty())
                return connectionPool.remove(0);
        return proceed();
    }
    void around(Connection conn) :
          call(void Connection.close()) && target(conn) {
        if (enablePooling) {
            connectionPool.put(conn);
        } else {
            proceed();
        }
    }
}
```

Typical Aspects III

Observer

- Collect possibly nested events
- React to nested events only once (`cflowbelow`)

```
abstract class Shape {
    abstract void moveBy(int x, int y);
}
class Point extends Shape { ... }
class Line extends Shape {
    Point start, end;
    void moveBy(int x, int y) { start.moveBy(x,y); end.moveBy(x,y); }
}

aspect DisplayUpdate {
    pointcut shapeChanged() : execution(void Shape+.moveBy(..));

    after() : shapeChanged() && !cflowbelow(shapeChanged()) {
        Display.update();
    }
}
```

Typical Aspects IV

Policy enforcement

- Policy is implemented globally and externally
- Example: autosave every five commands

```
aspect Autosave {
    int count = 0;
    after(): call(* Command+.execute(..)) {
        count++;
    }
    after(): call(* Application.save())
          || call(* Application.autosave()) {
        count = 0;
    }
    before(): call (* Command+.execute(..)) {
        if (count > 4) Application.autosave();
    }
}
```

Tools

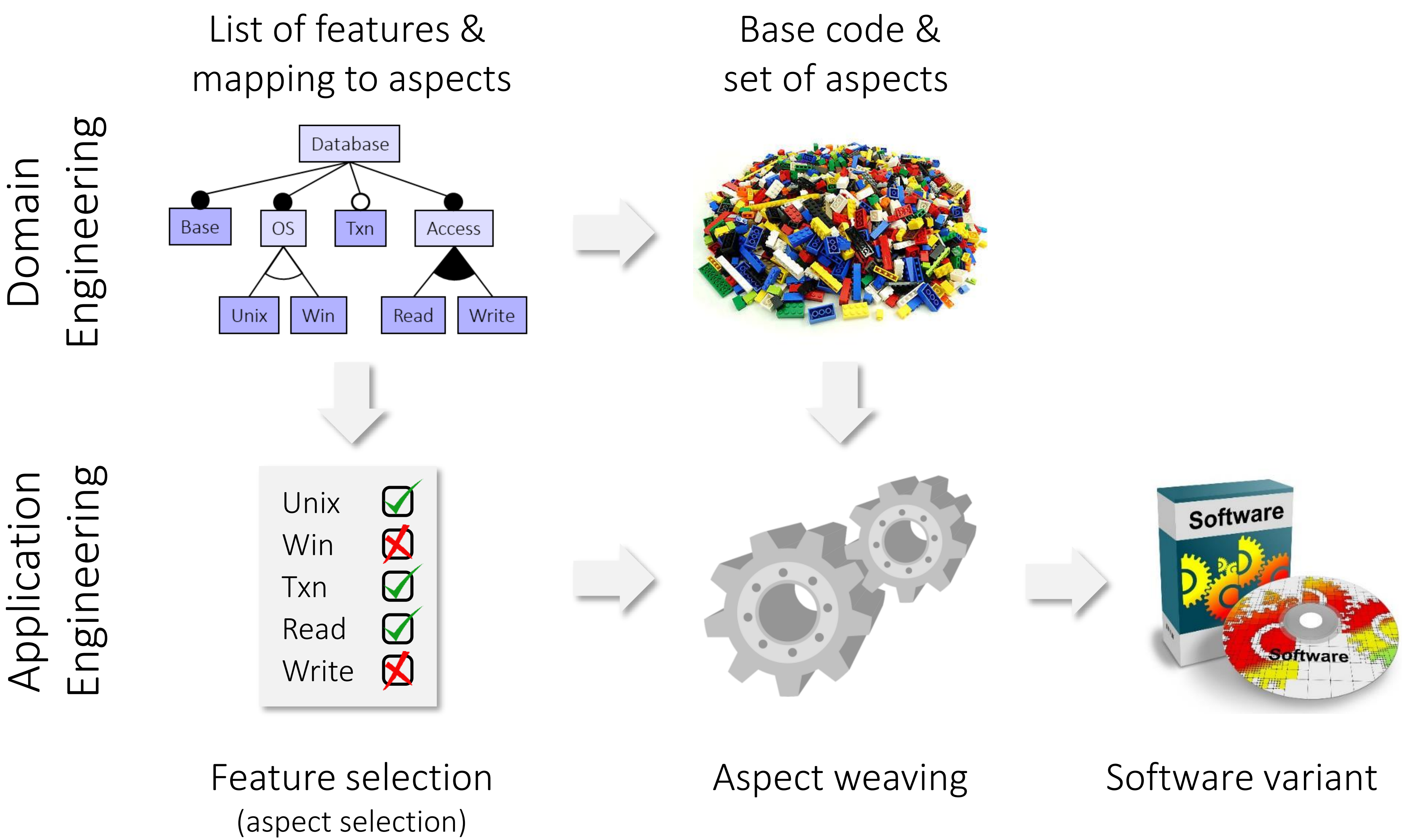
aspectj

spring Framework

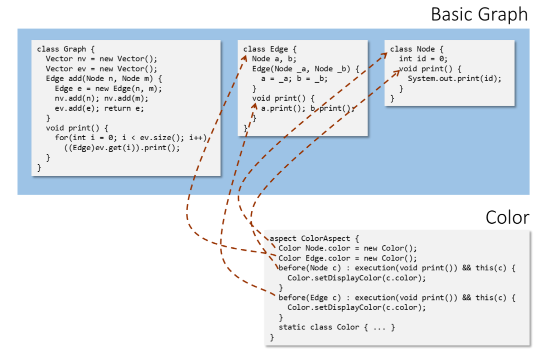
```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.AfterReturning;

@Aspect
public class AfterReturningExample {
  @AfterReturning(
    pointcut="com.xyz.myapp.dataAccessOperation()",
    returning="retVal")
  public void doAccessCheck(Object retVal) {
    // ...
  }
}
```

JProfiler

How to implement variability?



Discussion I



Feature traceability ✓

Crosscutting concerns ✓

Preplanning problem ✓

Inflexible class extension ✓

Discussion II

Obliviousness

Quantification

Separation of Concerns

Information Hiding

Obliviousness

The base program does not (need to) know about aspects

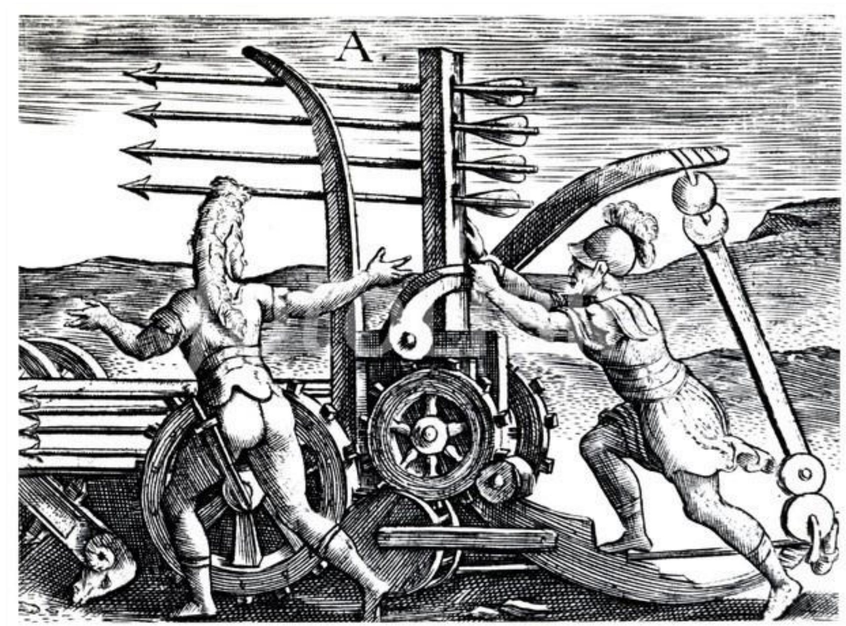
```
public class Connection {
    public Connection() { ... }
    ...
}
...
```

```
Connection around() : call(Connection.new()) {
    if (enablePooling)
        if (!connectionPool.isEmpty())
            return connectionPool.remove(0);
    return proceed();
}
```

Quantification

A piece of advice can extend a program at multiple (possibly very many) join points

```
Object around() : execution(public * com.company..*.* (..)) { ... }
```



Fragile Pointcuts

Base program is *not aware* of aspects

Definition of target joint points is "*imprecise*"

```
class Chess {
    void drawStalemate() { ... }
    void drawDeadPosition() { ... }
    void draw50Moves() { ... }
}

aspect UpdateDisplay {
    pointcut drawn: execution(* draw*(..));
    ...
}
```

Separation of Concerns

Implement each concern in a *distinct* unit of code

Basic Graph

```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    System.out.print(id);
  }
}
```

Concerns separated!

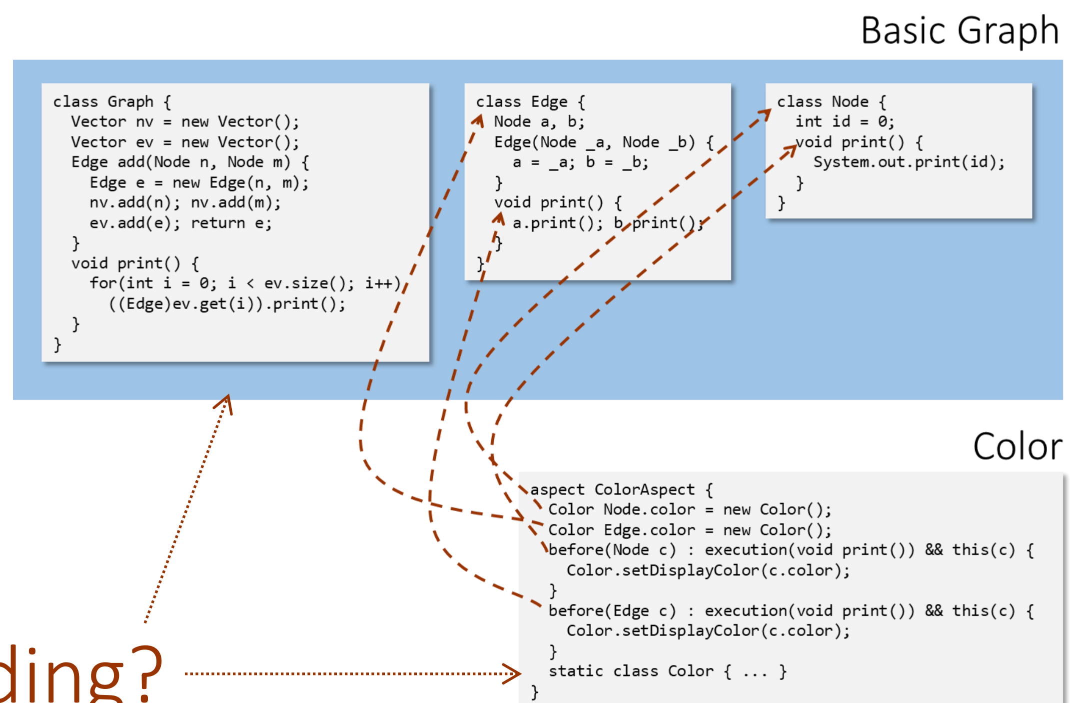
Color

```
aspect ColorAspect {
  Color Node.color = new Color();
  Color Edge.color = new Color();
  before(Node c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  before(Edge c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  static class Color { ... }
}
```

Information Hiding

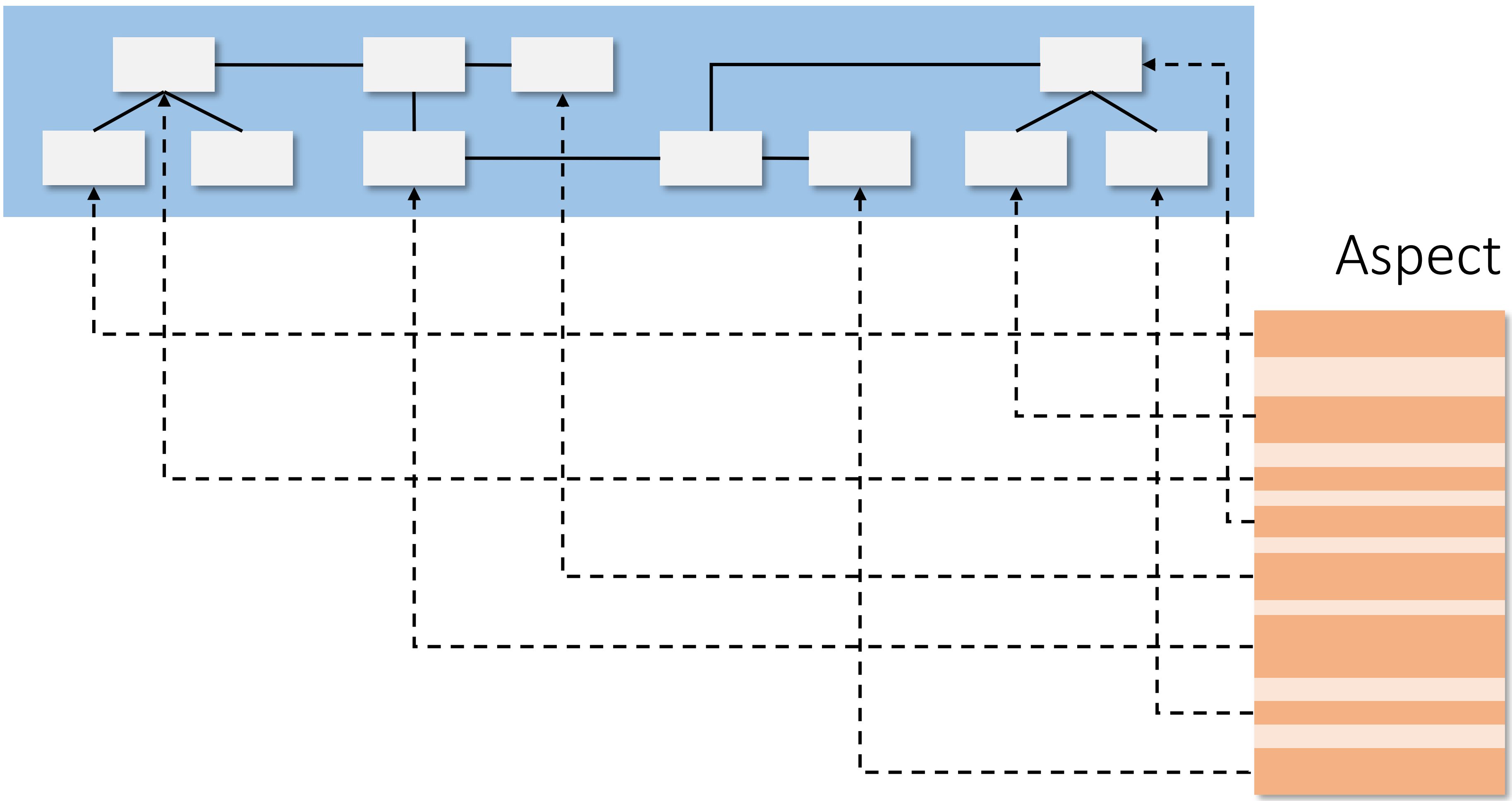
Expose a *public interface* to external clients

Hide internal details from external clients



Information hiding?

Collaborations vs. Aspects



One aspect can extend *multiple* classes

Advanced control-flow-dependent extensions (e.g., `cflow`)

Literature

R. Laddad. *AspectJ in Action*. Manning Publications, 2009

F. Steimann, et al.: *Types and modularity for implicit invocation with implicit announcement*. ACM TOSEM 20(1): 1:1-1:43, 2010

Part III

Structural Feature Interactions

Feature Modularity

Jo Atlee @ ESEC/FSE'19

https://vimeo.com/356373889/dcb19424f9

So far: focus on feature modularity

...but features often *interact*

...intentionally or inadvertently!

Boeing 737 Max8

"The anti-stall system [...] has been pinpointed by investigators as a possible cause in a fatal Lion Air crash in Indonesia and the one in Ethiopia."

Example #1

Phones support typically

- *Call Waiting*
- *Call Forwarding*

What happens if both features are activated?

- Free line → no problem
- Busy line → wait or forward?

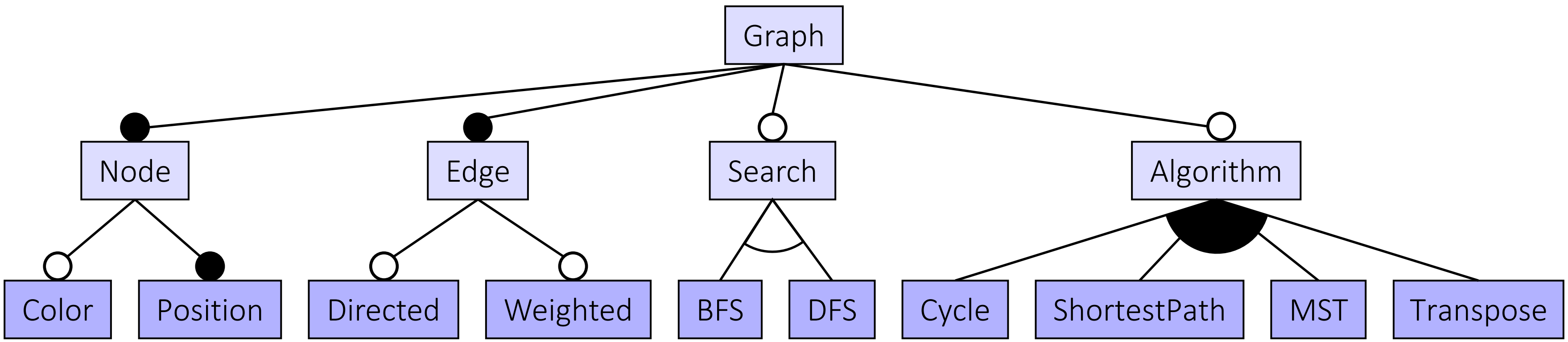
Detection? Coordination?

Example #2

Flood control vs. *fire control* in building automation

Is there a problem?

Example #3



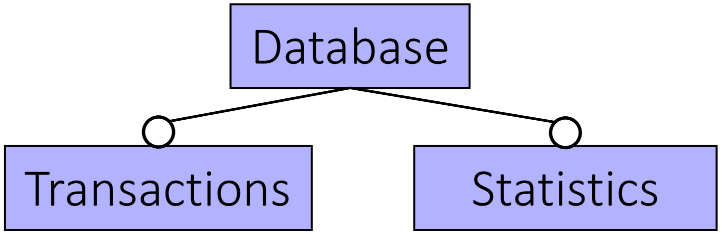
Cycle requires DFS

ShortestPath requires Weighted

MST precludes Directed

...

Example #4



Database with two features

Statistics: collects basic statistics such as buffer hit ratio, table size, transactions per second, etc.

Transactions: ensures ACID properties

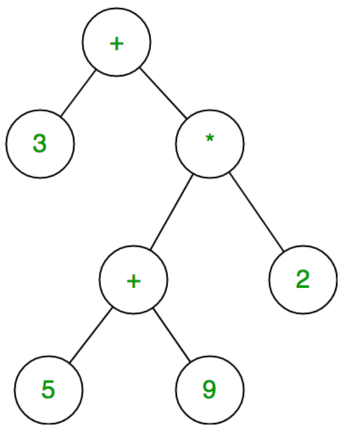
Both features shall be *optional!*

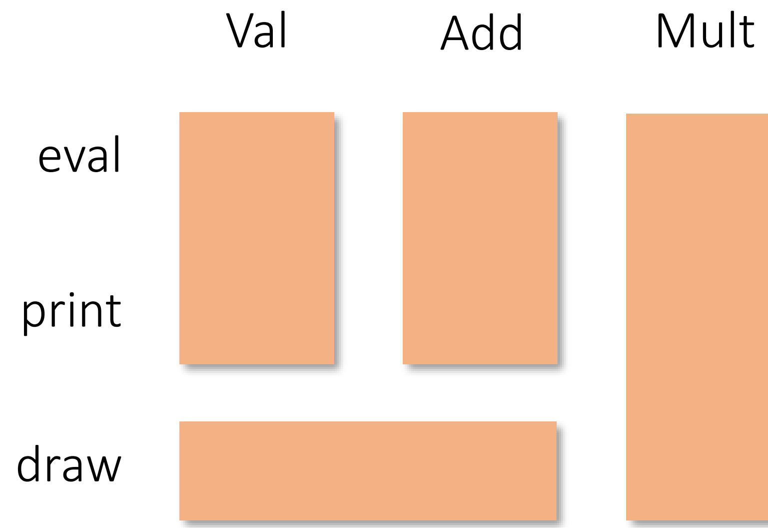
But: *Statistics* collects information about *Transactions*

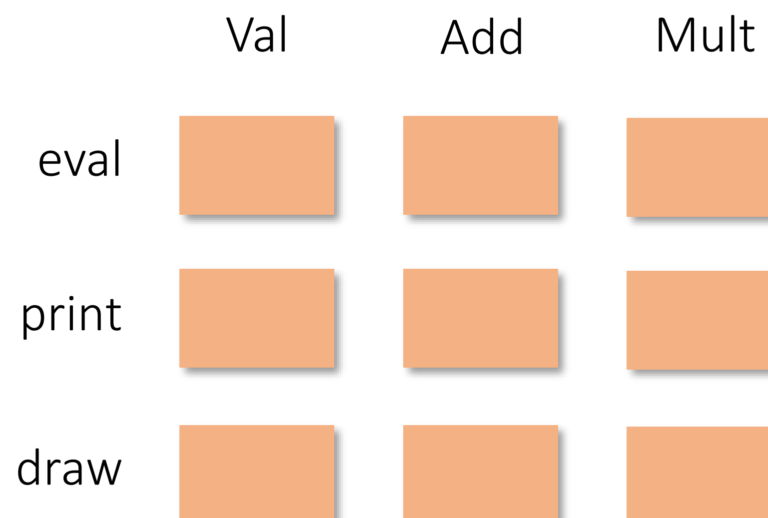
... and *Transactions* uses information from *Statistics*

Example #5

$e = 3 + ((5 + 9) * 2)$

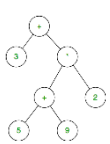






$\text{eval}(e) = 31$

$\text{print}(e) =$ ”3+((5+9)·2)“

$\text{draw}(e) =$ 

Observation

Features *use* other features

Cycle calls method `Graph.search` of *DFS*

Features *extend* other features

Weighted overrides and extends methods of class `Edge`

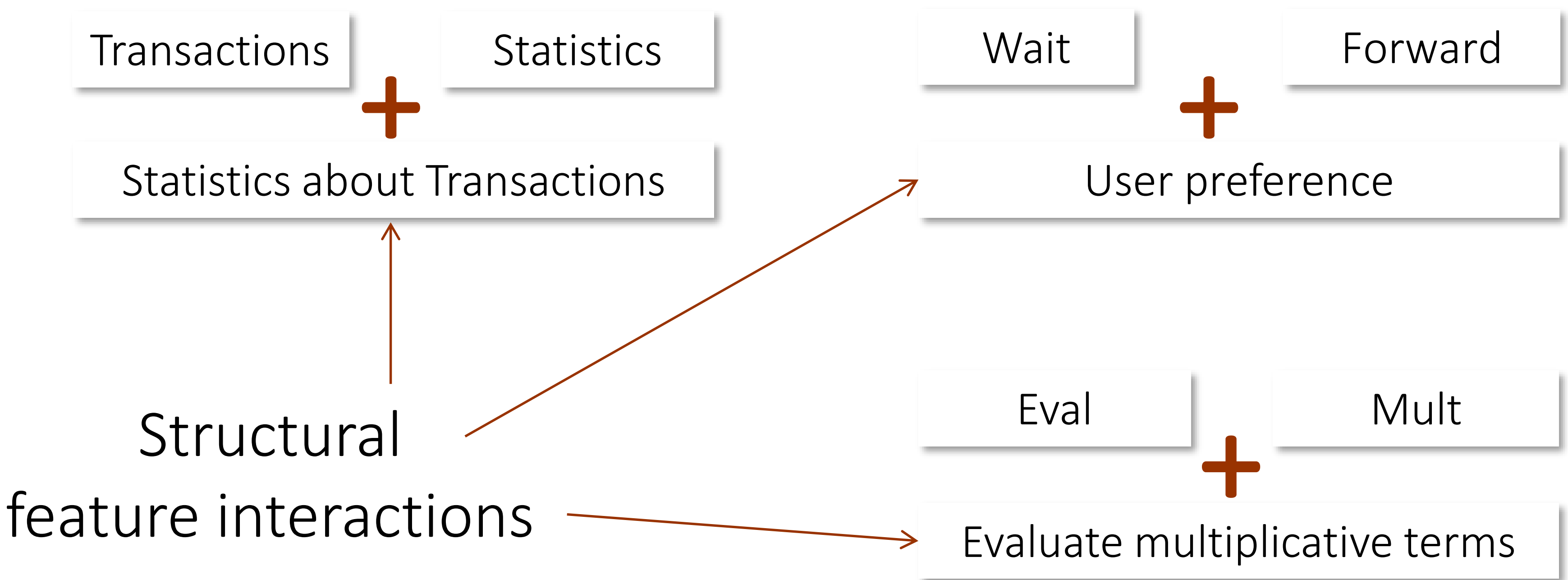
Features *rely on behavior* provided by other features

Some graph features may require that a given graph instance has no cycles

Optional and Interacting Features

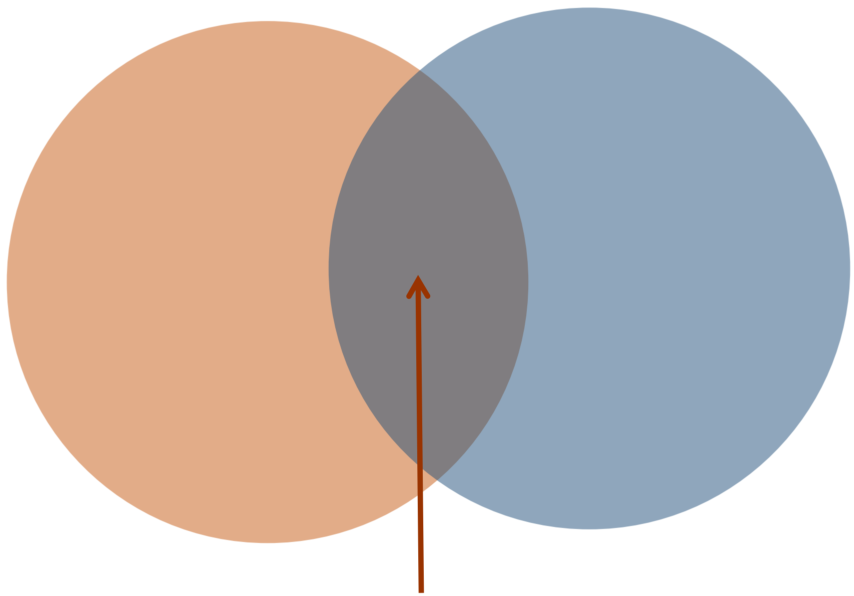
Features behave *individually as expected*

When combined, their *joint behavior* needs to be *coordinated*!



Coordination Code ⬄ Structural Feature Interaction

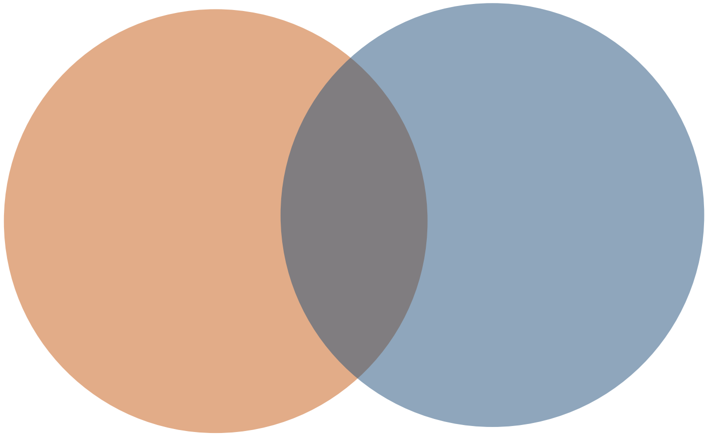
Statistics
(buffer hit ratio,
table size, …)



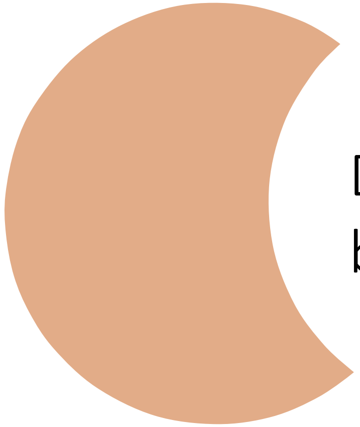
Transactions
(locks, commits,
rollback, …)

Throughput measurement
(“transactions per second”)

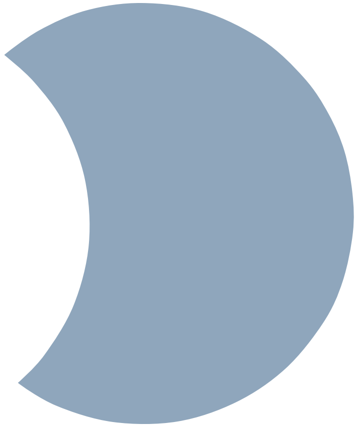
Desired Configurations



Database with *Statistics*
and *Transactions*



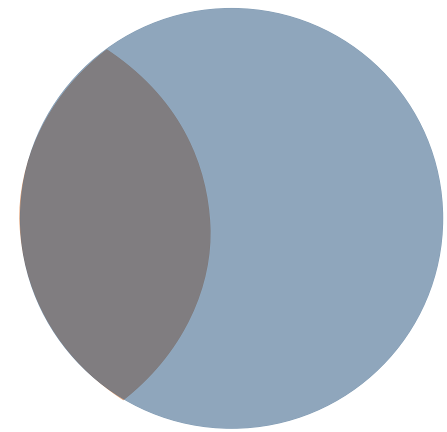
Database with *Statistics*
but without *Transactions*

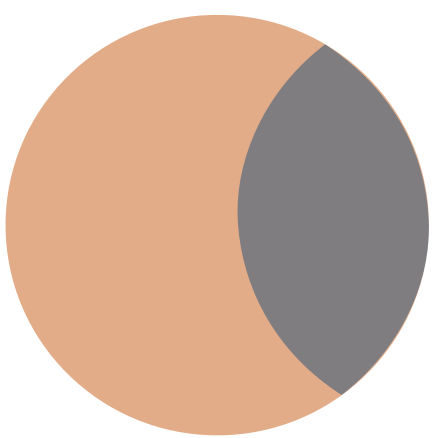


Database with *Transactions*
but without *Statistics*

Undesired or Impossible Configurations

Database with *Transactions* and without *Statistics* that still measures throughput
(larger and slower than necessary)





Database with *Statistics* and without *Transactions* that measures throughput??

Code Example

```
class Database {
  List locks;
  void lock() { ... }
  void unlock() { ... }
  void put(Object key, Object data) {
    lock();
    ...
    unlock();
  }
  Object get(Object key) {
    lock();
    ...
    unlock();
  }
  int getOpenLocks() {
    return locks.size();
  }
  int getDbSize() {
    return calculateDbSize();
  }
  static int calculateDbSize() {
    lock();
    ...
    unlock();
  }
}
```

Sync (blue)

Code Example

```
class Database {
  List locks;
  void lock() { ... }
  void unlock() { ... }
  void put(Object key, Object data) {
    lock();
    ...
    unlock();
  }
  Object get(Object key) {
    lock();
    ...
    unlock();
  }
  int getOpenLocks() {
    return locks.size();
  }
  int getDbSize() {
    return calculateDbSize();
  }
  static int calculateDbSize() {
    lock();
    ...
    unlock();
  }
}
```

Statistics (red)

Code Example

```
class Database {
  List locks;
  void lock() { ... }
  void unlock() { ... }
  void put(Object key, Object data) {
    lock();
    ...
    unlock();
  }
  Object get(Object key) {
    lock();
    ...
    unlock();
  }
  int getOpenLocks() {
    return locks.size();
  }
  int getDbSize() {
    return calculateDbSize();
  }
  static int calculateDbSize() {
    lock();
    ...
    unlock();
  }
}
```

Sync (blue)

+

Statistics (red)

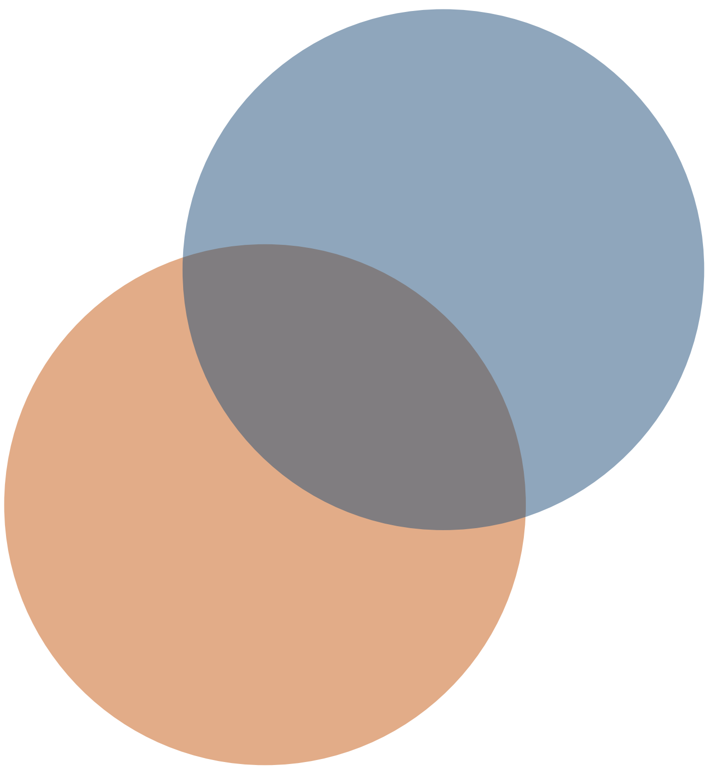
They overlap in two positions (purple)

Statistics about *Locking*

Synchronization of *Statistics* methods

Central Question

Where shall we implement the coordination code?

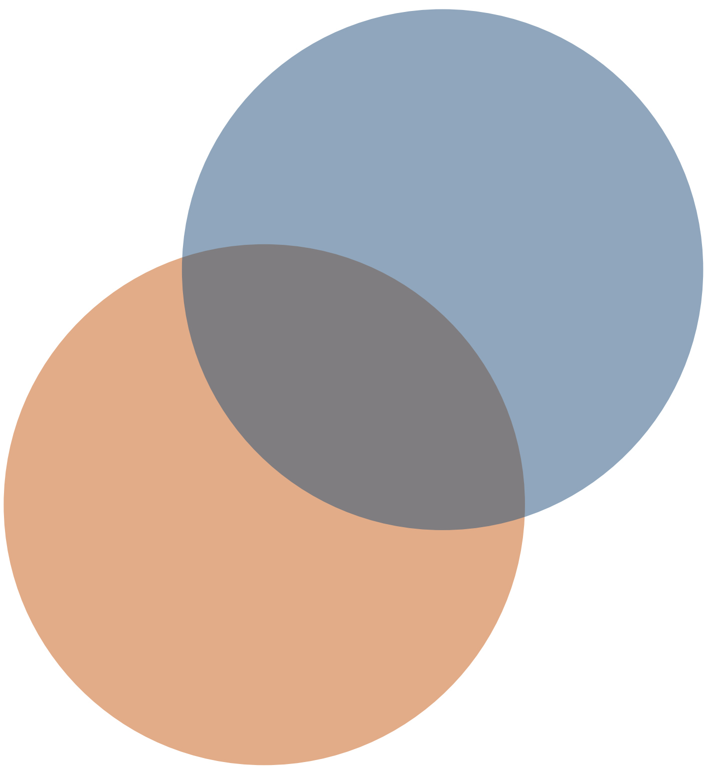


In either of the feature modules?

- → Reduced variability
- → Suboptimal code

Central Question

Where shall we implement the coordination code?



In either of the feature modules?

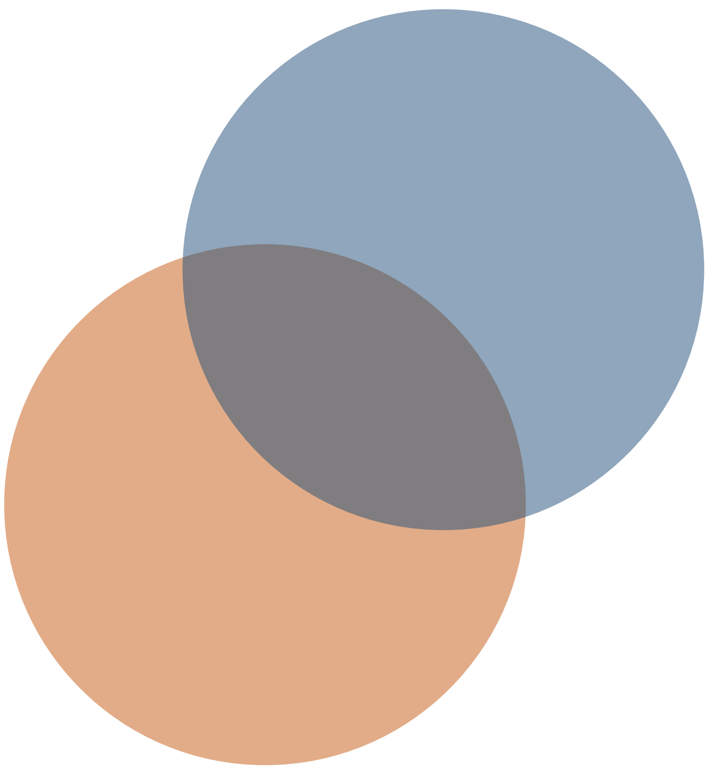
- → Reduced variability
- → Suboptimal code

In an extra module?

- → Many extra modules

Central Question

Where shall we implement the coordination code?



In either of the feature modules?

→ Reduced variability

→ Suboptimal code

In an extra module?

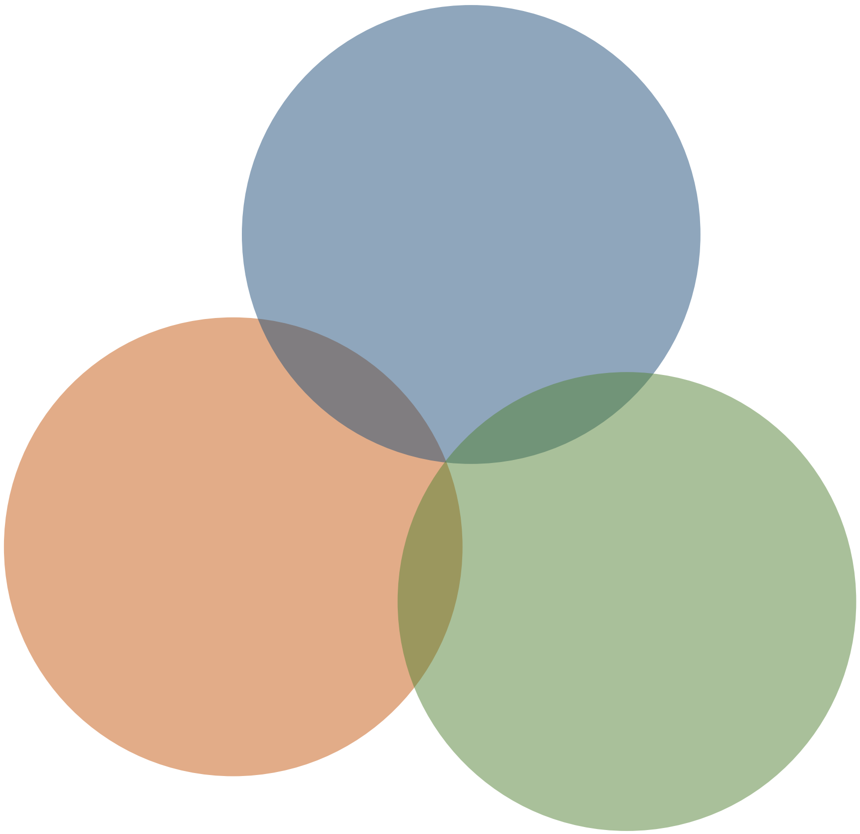
→ Many extra modules

Preprocessor?

→ Anti-modular

Key Problem

Many potential interactions that need to be coordinated!

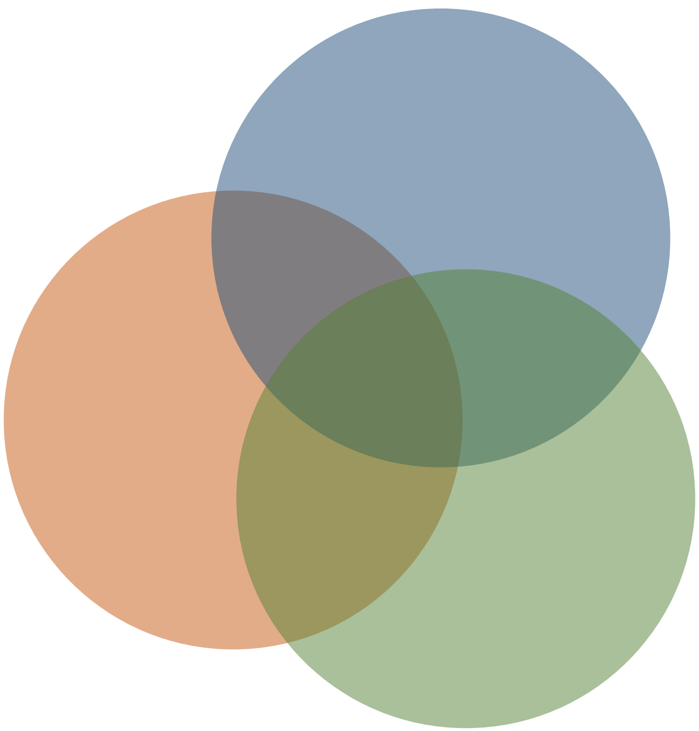


Pair-wise interactions:

$$I^{(2)} = \binom{|F|}{2} = \frac{|F|^2 - |F|}{2}$$

Key Problem

Interactions may be of higher order!



Interactions of all orders:

$$I^* = \sum_{o=1}^{|F|-1} \binom{|F|}{o-1} = 2^{|F|} - |F| - 1$$

Code Example

```java
class Stack {
    boolean push(Object o) {
        Lock lock = lock();
        if (lock == null) {
            log(“lock failed for: ” + o);
            return false;
        }
        rememberValue ();
        elementData[size ++] = o;
        ...
    }

    void log(String msg) { ... }

    boolean undo () {
        Lock lock = lock();
        if (lock == null) {
            log(“undo-lock failed!”);
            return false;
        }
        restoreValue ();
        ...
        log(“undone!”);
    }
}
```

Sync

Sync + Logging

Undo

Logging

Undo

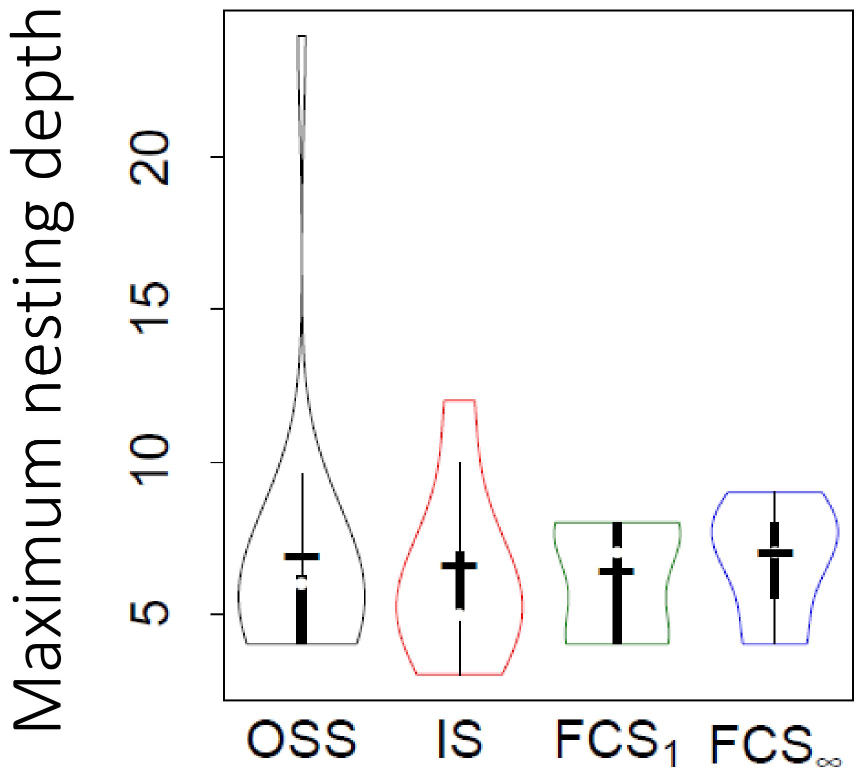
Undo + Sync

Undo + Sync + Logging

Undo

Undo + Logging

Case Study: Nesting of Preprocessor Directives



Software system	Version	Dev. Time	Domain	C [%]
Open-source systems (OSS)				
Apache	2.4.6	(2013; 18)	Web server	95
BerkeleyDB	6.0.20	(2013; 27)	Database system	69
BusyBox	1.21.1	(2013; 19)	Unix tool collection	98
Cherokee	1.2.101	(2013; 12)	Web server	64
FreeBSD	9.1.0	(2013; 20)	Operating system	87
GIMP	2.8.6	(2013; 15)	Graphics editor	91
Gnumeric	1.10.15	(2011; 10)	Spreadsheet application	97
gnuplot	4.6.3	(2013; 27)	Plotting tool	81
libxml2	2.9.0	(2012; 13)	XML library	85
Linux	3.9	(2013; 22)	Operating system	97
OpenVPN	2.3.2	(2013; 11)	Security application	90
Parrot	5.0.0	(2013; 4)	High-level virtual machine	39
PostgreSQL	9.3.0	(2013; 18)	Database system	97
QEMU	1.6.1	(2013; 10)	System-level virtual machine	97
Sendmail	8.14.7	(2013; 30)	Mail transfer agent	96
SQLite	3.8.0.2	(2013; 13)	Database system	90
Subversion	1.8.1	(2013; 13)	Revision control system	86
Vim	7.3	(2010; 19)	Text editor	65
xfig	3.2.5b	(2013; 28)	Vector graphics editor	100
xterm	296	(2013; 29)	Terminal emulator	95
Formerly closed-source systems: first version (FCS_1) and current version (FCS_∞)				
Android System Core	1.0	(2007; 2)	Mobile operating system	88
	4.4_r1.2	(2013; 6)		80
Blender	2.26	(2003; 5)	3D graphics editor	99
	2.69	(2013; 10)		57
KornShell (ksh93)	12-02-29	(2000; 17)	Terminal emulator	85
	12-08-01	(2012; 12)		85
mDNSResponder	22	(2002; 4)	mDNS networking service	93
	541	(2013; 11)		85
Netscape/SeaMonkey	98-3-31	(1998; 4)	Internet suite	77
	2.23	(2013; 15)		25
(Open-)Solaris	1.0	(2005; 14)	Operating system	88
	13-10-28	(2013; 8)		95
VirtualBox	1.6.0	(2007; 3)	System-level virtual machine	61
	4.3.2	(2013; 6)		66
Industrial systems (IS)				
A	–	(2013; 8)	Combustion-engine control	–
B	–	(2013; 7)	Frequency converter	–
C	–	(2009; 5)	Embedded automation controller	–
D	–	(2011; 6)	Inertial sensor controller	–
E	–	(2012; 7)	Frequency converter rail domain	–
F	–	(2013; 10)	Audio processing solutions	–
G	–	(2013; 7)	Inertial sensor controller	–

Literature

C. Kästner, et al.: *On the impact of the optional feature problem: Analysis and case studies*. Proceedings SPLC, 181-190, SEI, 2009

C. Hunsen, et al.: *Preprocessor-based variability in open-source and industrial software systems: An empirical study*. EMSE 21(2): 449-482, 2016

Quiz

Mark all joint points:

```
class Test {
  MathUtil u;
  public void main() {
    u = new MathUtil();
    int i = 2;
    i = u.twice(i);
  }
}
```

Are there extensions that cannot be expressed by AspectJ? Give an example!

Sketch the implementation of the Display Update example with collaborations and roles

Quiz

Consider a program with 10 optional and independent features...

Calculate $I^{(2)}$ and I^*

How many configurations are possible (see feature model right)?

Without implementation dependencies?

With *Statistics* depending on *Index*

With *Transactions* depending on *Statistics* and vice versa