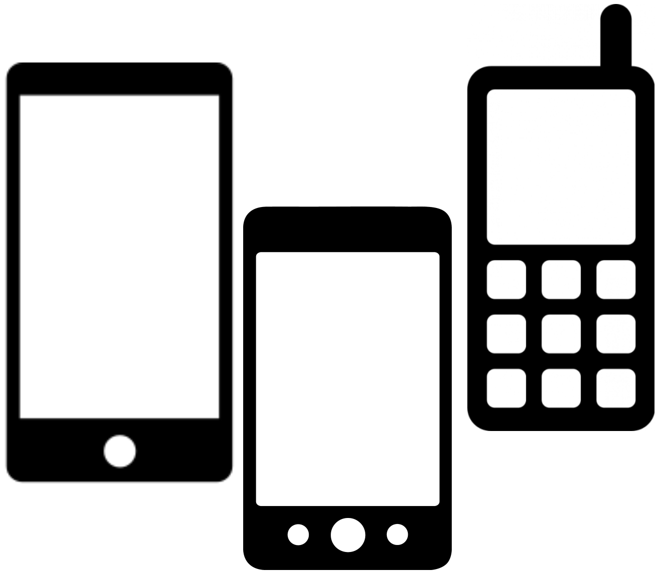
Software Product Lines

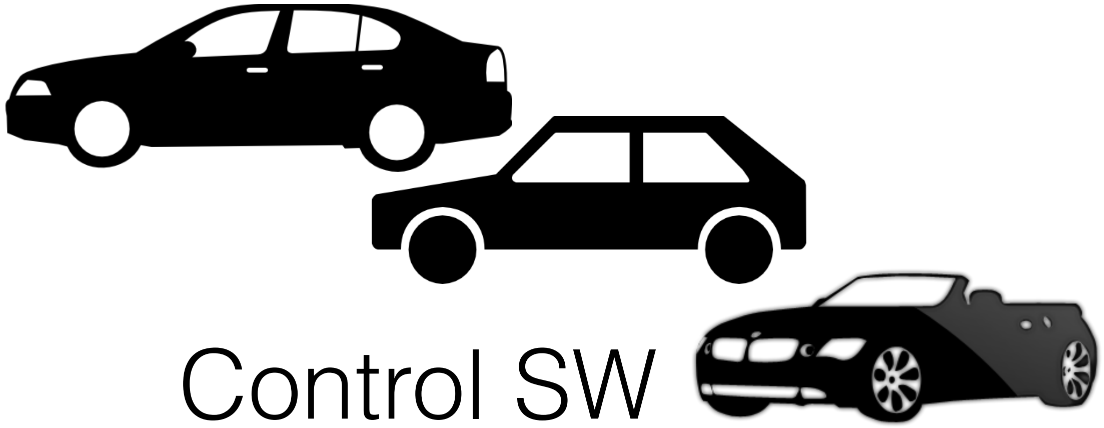
Sarah Nadi
Software Technology Group

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Examples of Software Product Lines



Mobile OS



Control SW

Printer Firmware

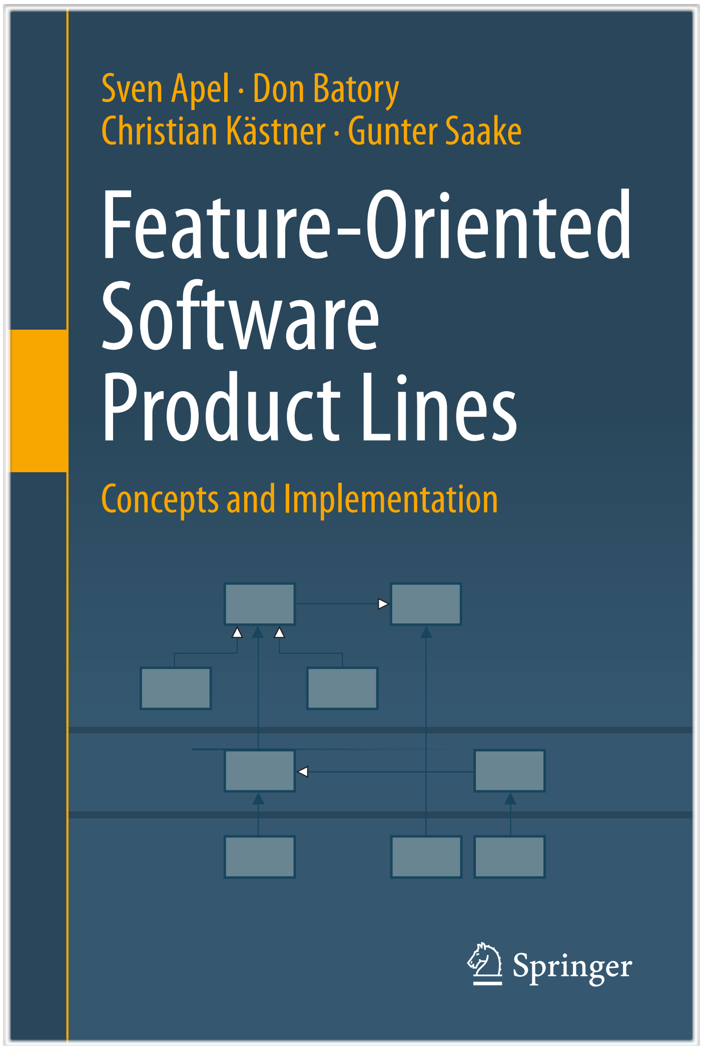
Linux Kernel

What You Will Learn Today

- What a software product line (SPL) is
- Challenges of SPLs
- What are the phases of SPL engineering (SPLE)
- Feature modeling (part of domain engineering)
- Different domain implementation techniques
- Some (advanced) research topics

Resources

- Slides largely based on:



Sven Apel · Don Batory
Christian Kästner · Gunter Saake

Feature-Oriented Software Product Lines

Concepts and Implementation

Springer

Software Product Lines

“A software product line (SPL) is a set of software-intensive systems that **share a common, managed set of features** satisfying the specific needs of a particular market segment or mission and that are **developed from a common set of core assets** in a prescribed way.”

— Software Engineering Institute
Carnegie Mellon University

Advantages of SPLs

- Tailor-made software
- Reduced cost
- Improved quality
- Reduced time to market

Success Stories

Danfoss

BOEING

hp

GM

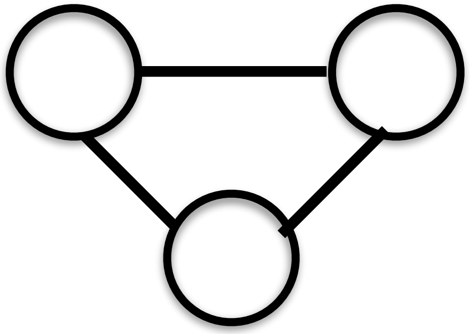
Challenges of SPLs

- Upfront cost for preparing reusable parts
- Deciding which products you can produce early on
- Thinking about multiple products at the same time
- Managing/testing/analyzing multiple products

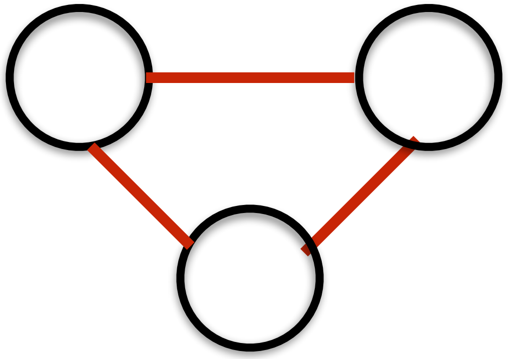
Feature-oriented SPLs

- Thinking of your product line in terms of the features offered

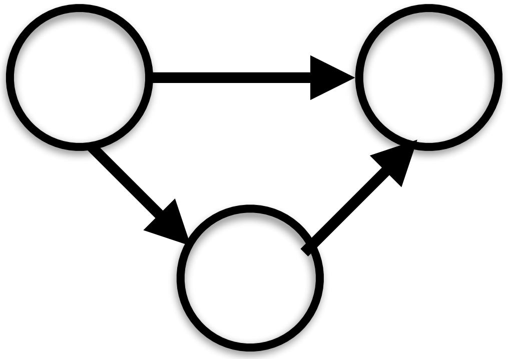
Examples of a Feature



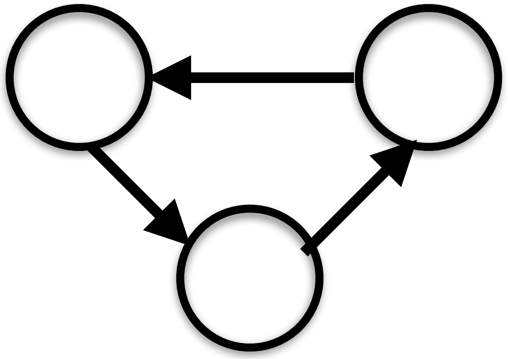
Graph product-line



feature:
edge color



feature:
edge type
(Directed vs Undirected)

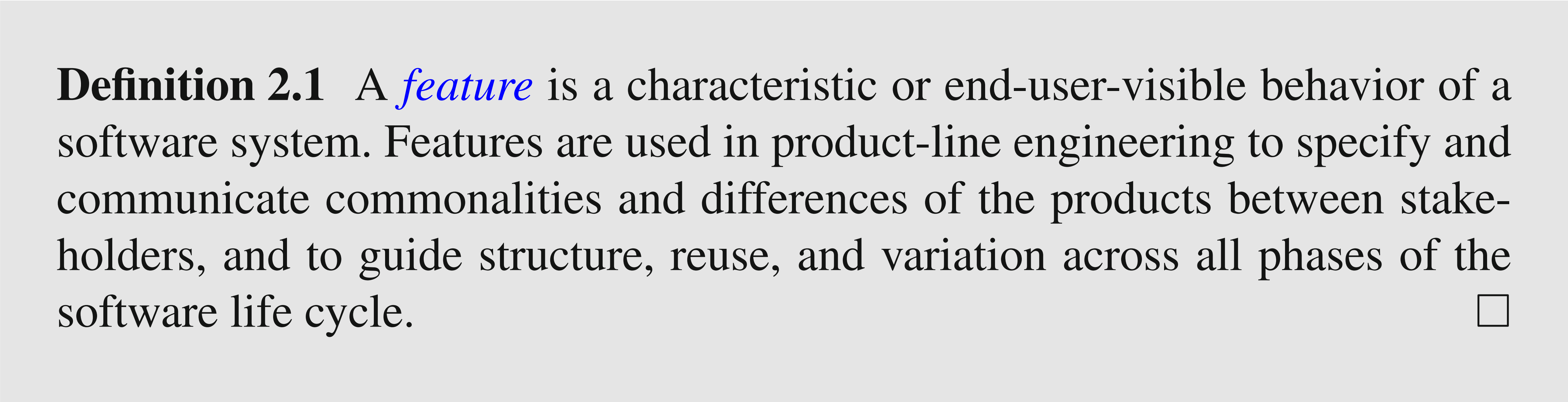


feature:
cycle detection

Examples of a Feature

- Database SPL Features:
  - Transactions
  - In-memory
  - Concurrency
  - Logging
  - Write access
  - …

Feature

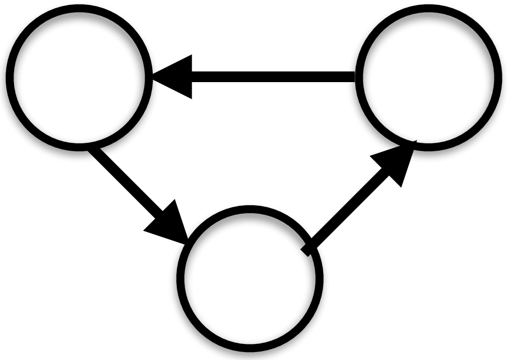


Definition 2.1 A *feature* is a characteristic or end-user-visible behavior of a software system. Features are used in product-line engineering to specify and communicate commonalities and differences of the products between stakeholders, and to guide structure, reuse, and variation across all phases of the software life cycle. □

Exercise:
What features would a car SPL contain?

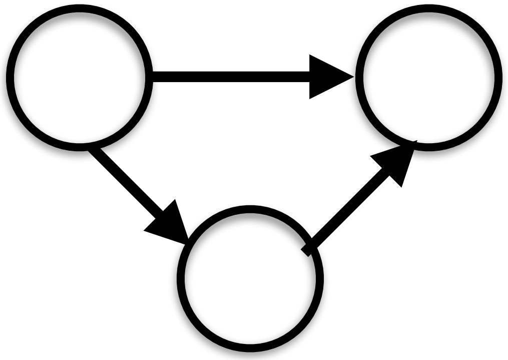
Feature Dependencies

- Constraints on the possible feature selections



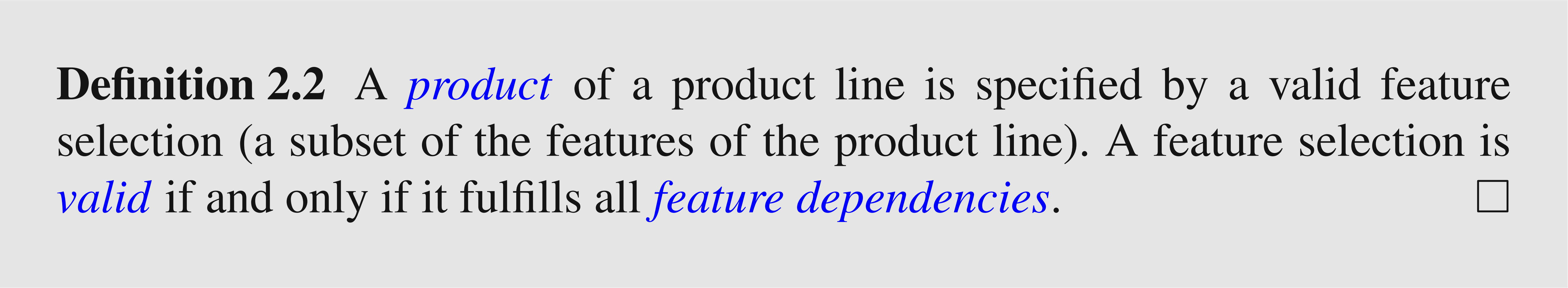
feature:
cycle detection

depends on



feature:
directed

Product



Definition 2.2 A *product* of a product line is specified by a valid feature selection (a subset of the features of the product line). A feature selection is *valid* if and only if it fulfills all *feature dependencies*. □

Exercise:
Which Product(s) are Invalid?

	Edge Color	Directed Edge	Cycle Detection
Product 1	✓	✓	✓
Product 2	✓		✓
Product 3		✓	✓

Exercise:
Which Product(s) are Invalid?

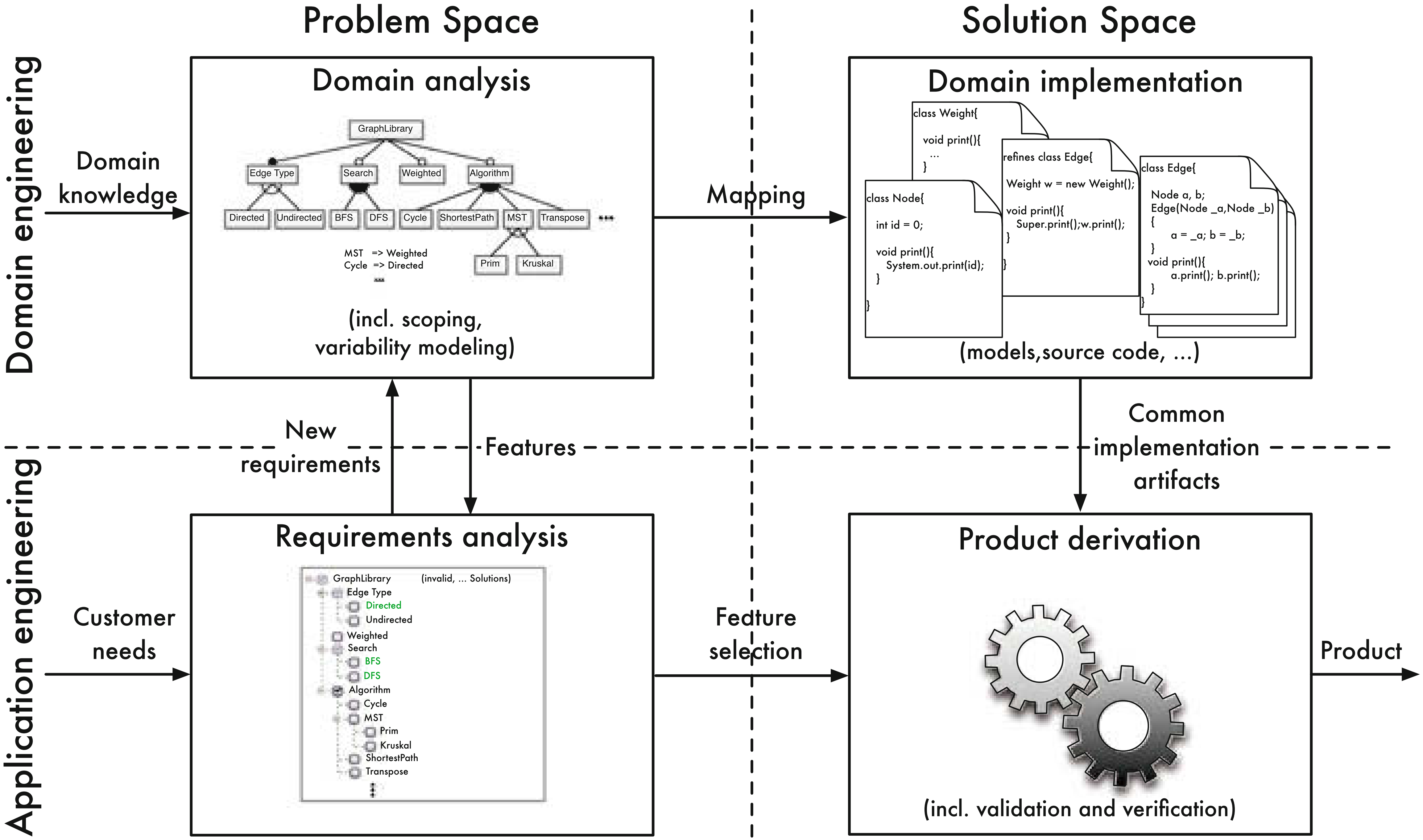
	Edge Color	Directed Edge	Cycle Detection
Product 1	✓	✓	✓
~~**Product 2**~~	~~✓~~		~~✓~~
Product 3		✓	✓

invalid
product

Cycle detection depends on Directed Edge

Exercise:
What dependencies might exist between features in a car SPL?

Software Product Line Engineering



Problem Space

Solution Space

Domain engineering

Application engineering

Domain knowledge

Domain analysis

(incl. scoping, variability modeling)

Mapping

Domain implementation

```
class Weight{

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

```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
refines class Edge{

 Weight w = new Weight();

 void print(){
   Super.print();w.print();
 }

}
```

```
class Edge{

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

(models,source code, ...)

New requirements

Features

Common implementation artifacts

Requirements analysis

- GraphLibrary (invalid, ... Solutions)
  - Edge Type
    - Directed
    - Undirected
  - Weighted
  - Search
    - BFS
    - DFS
  - Algorithm
    - Cycle
    - MST
      - Prim
      - Kruskal
    - ShortestPath
    - Transpose
    - ⋮

Customer needs

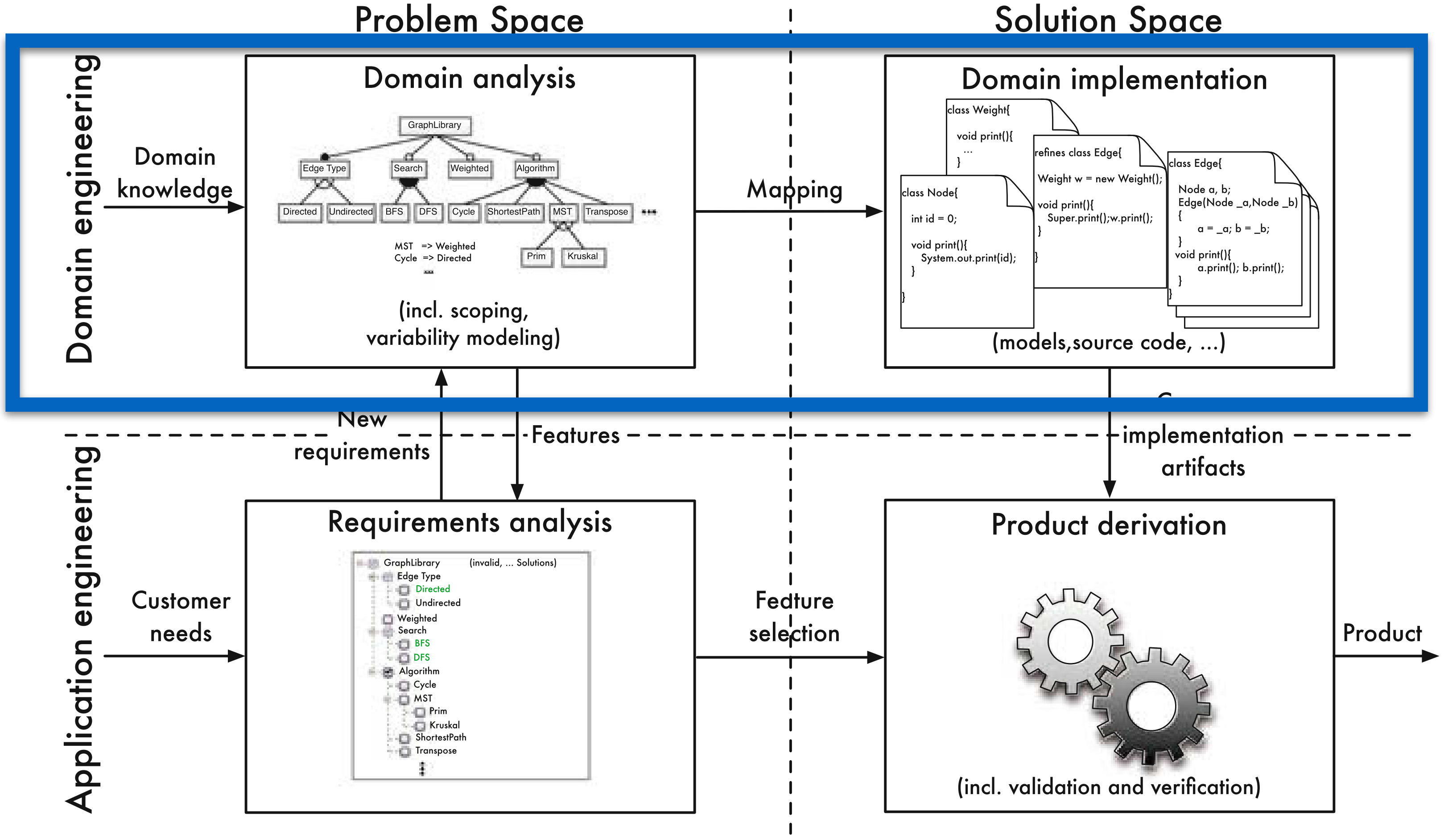
Feature selection

Product derivation

(incl. validation and verification)

Product

Software Product Line Engineering



```
class Weight{

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

```
refines class Edge{

 Weight w = new Weight();

 void print(){
   Super.print();w.print();
 }

}
```

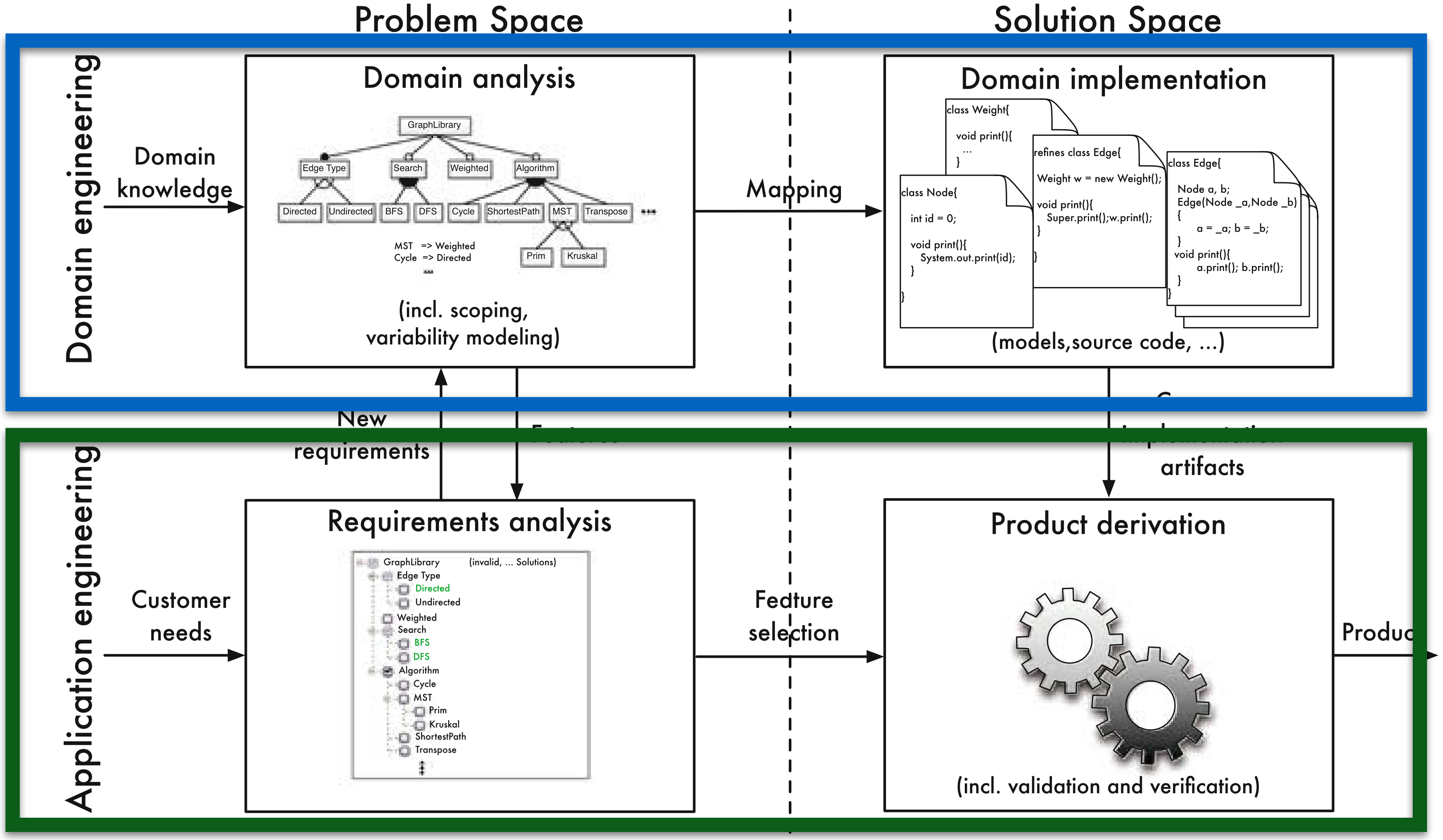
```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
class Edge{

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

Development for reuse

- Analyze domain & develop reusable artifacts
- Does not result in a specific product
- Prepares artifacts to be used in various products

Software Product Line Engineering



```
class Weight{

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

```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
refines class Edge{

 Weight w = new Weight();

 void print(){
   Super.print();w.print();
 }

}
```

```
class Edge{

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

Development for reuse

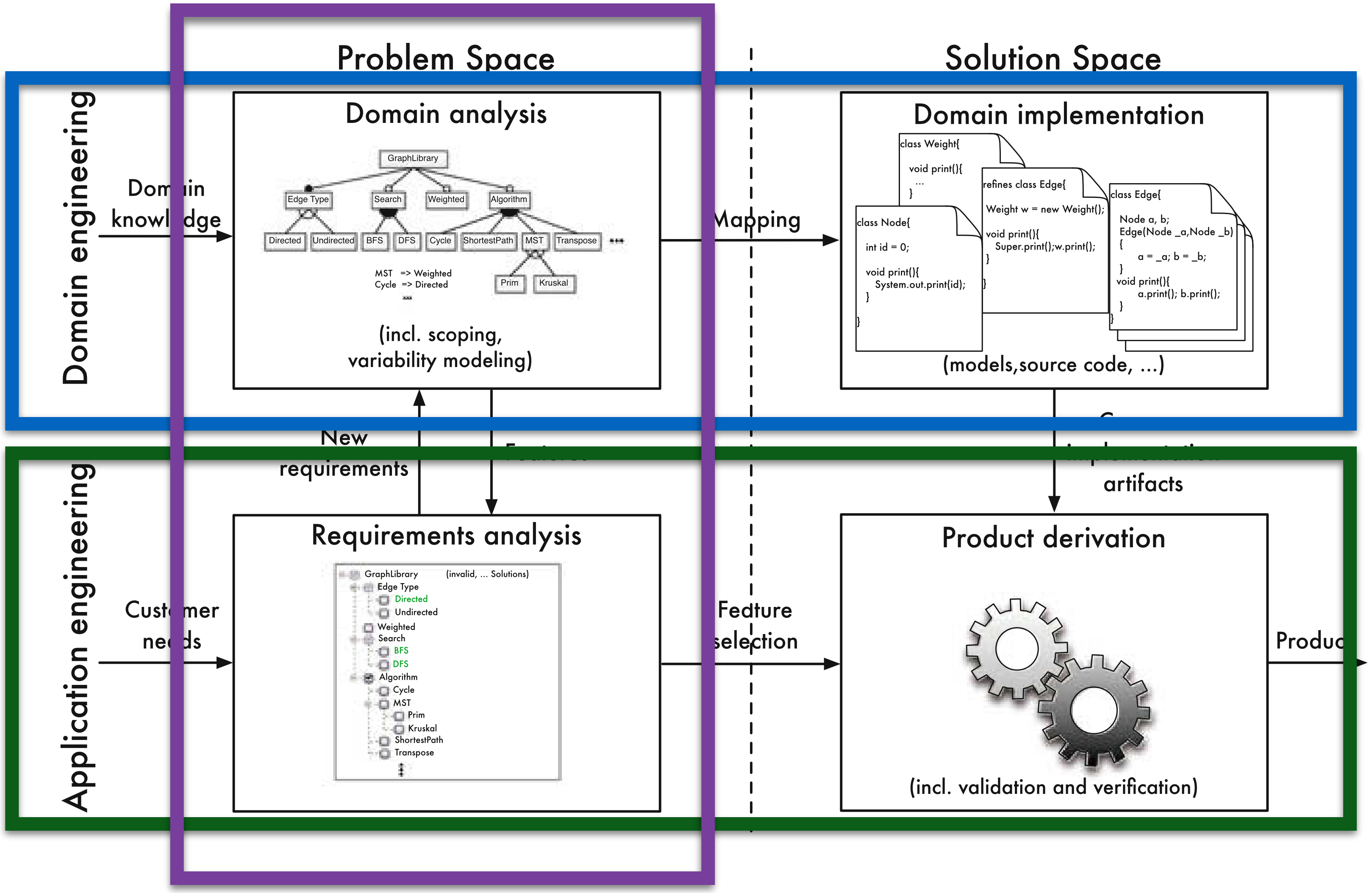
- Analyze domain & develop reusable artifacts
- Does not result in a specific product
- Prepares artifacts to be used in various products

Development with reuse

- Develop specific product for needs of a particular customer
- Repeated for every derived product

Software Product Line Engineering

Perspective of stakeholders' problems, requirements, & view on entire domain



```
class Weight{

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

```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
refines class Edge{

 Weight w = new Weight();

 void print(){
   Super.print();w.print();
 }

}
```

```
class Edge{

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

Development for reuse

- Analyze domain & develop reusable artifacts
- Does not result in a specific product
- Prepares artifacts to be used in various products

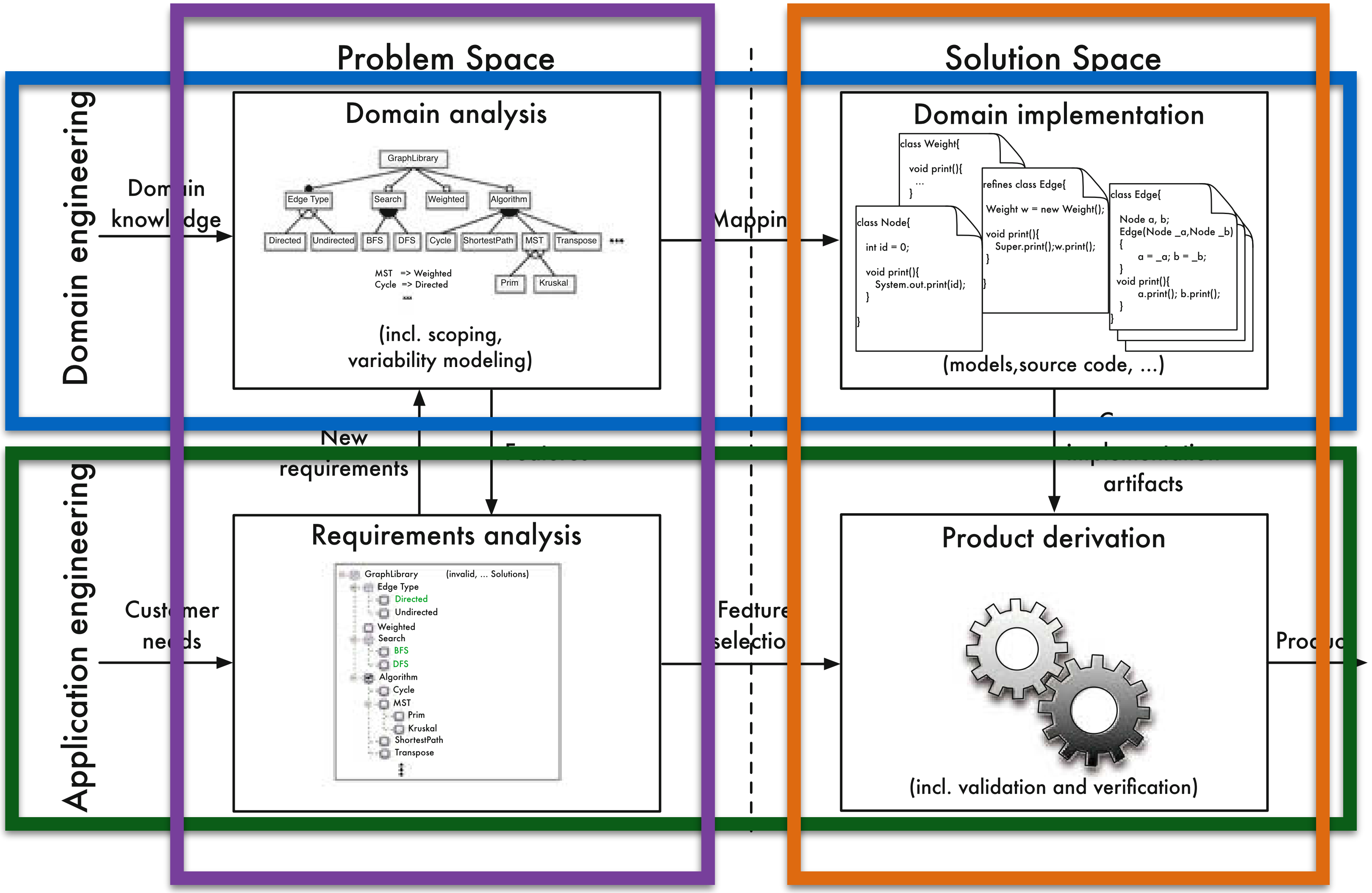
Development with reuse

- Develop specific product for needs of a particular customer
- Repeated for every derived product

Software Product Line Engineering

Perspective of stakeholders' problems, requirements, & view on entire domain

Perspective of developers & vendors



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

```
refines class Edge{
 Weight w = new Weight();
 void print(){
   Super.print();w.print();
 }
}
```

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

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

Development for reuse

- Analyze domain & develop reusable artifacts
- Does not result in a specific product
- Prepares artifacts to be used in various products

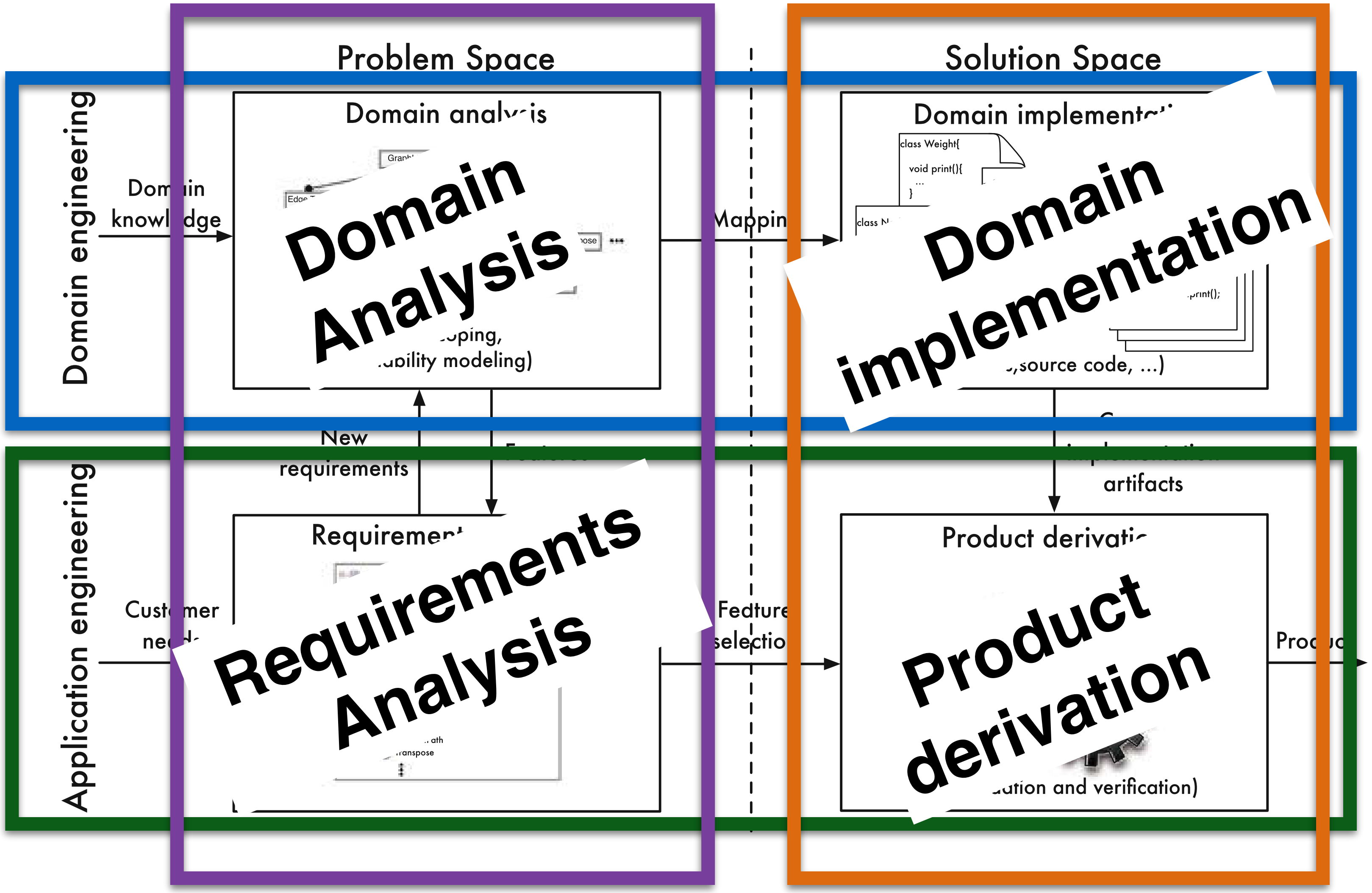
Development with reuse

- Develop specific product for needs of a particular customer
- Repeated for every derived product

Software Product Line Engineering

Perspective of stakeholders' problems, requirements, & view on entire domain

Perspective of developers & vendors



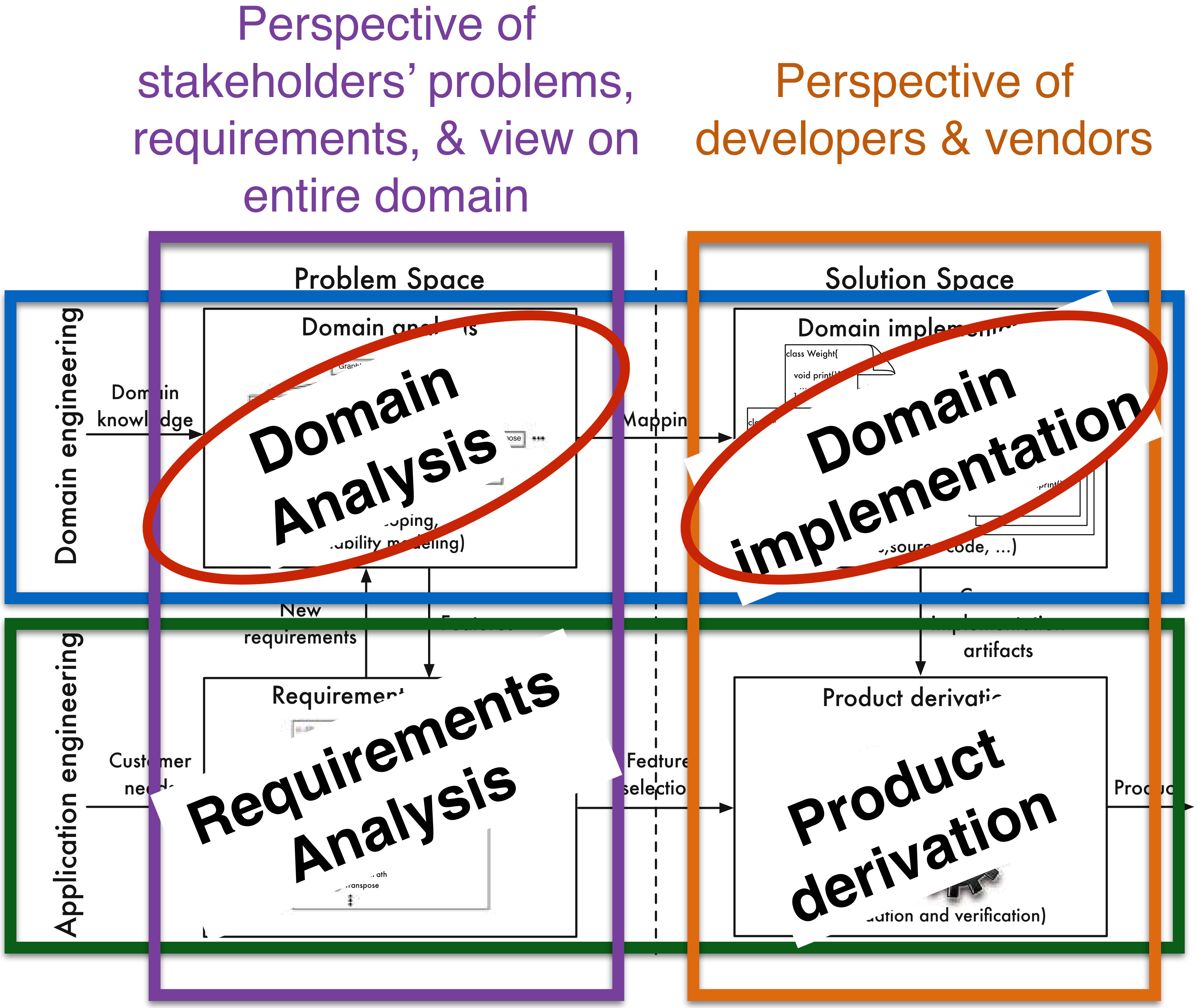
Development for reuse

- Analyze domain & develop reusable artifacts
- Does not result in a specific product
- Prepares artifacts to be used in various products

Development with reuse

- Develop specific product for needs of a particular customer
- Repeated for every derived product

Software Product Line Engineering



Development for reuse

- Analyze domain & develop reusable artifacts
- Does not result in a specific product
- Prepares artifacts to be used in various products

Development with reuse

- Develop specific product for needs of a particular customer
- Repeated for every derived product

Domain Analysis

Domain Analysis

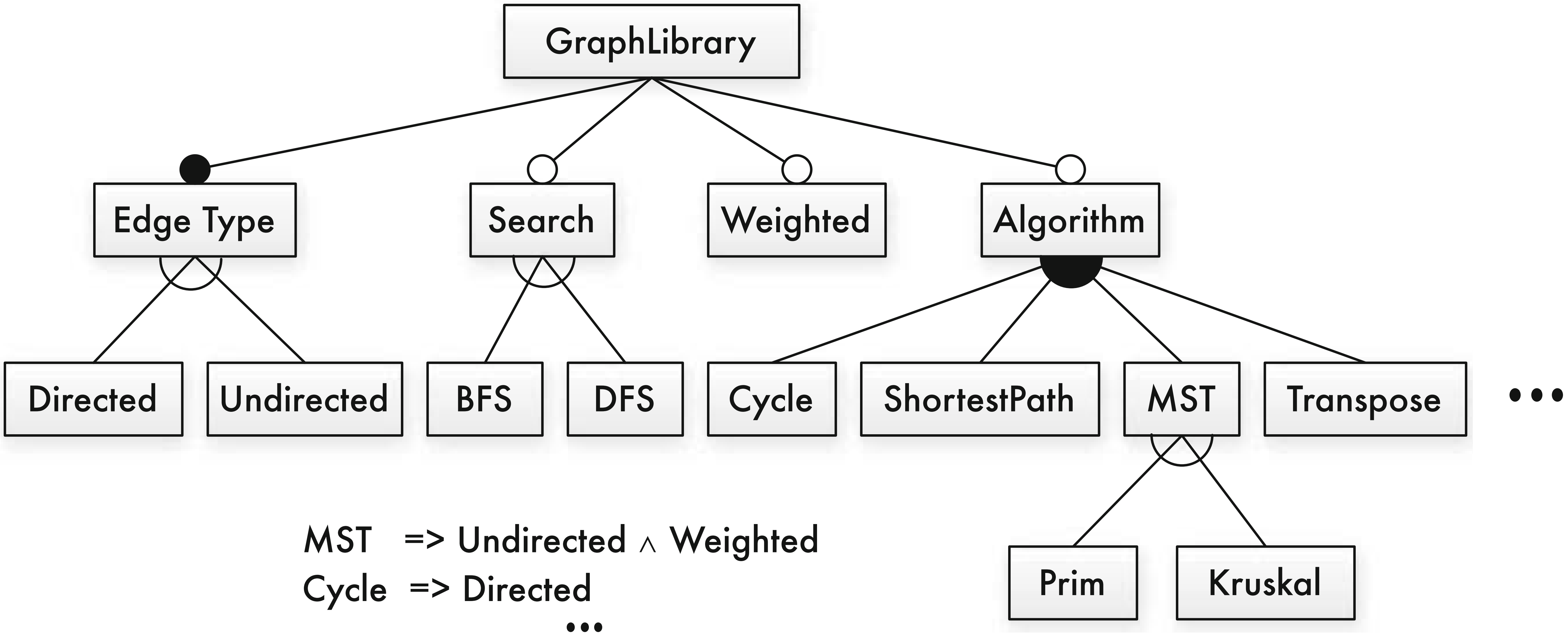
- Domain scoping
  - Deciding on product line’s extent or range
- Domain modeling
  - Captures & documents the commonalities & variabilities of the scoped domain
  - Often captured in a *feature model*

Feature Models

- Document the features of a product line & their relationships

- Can be translated into propositional logic

Graph Library Feature Model



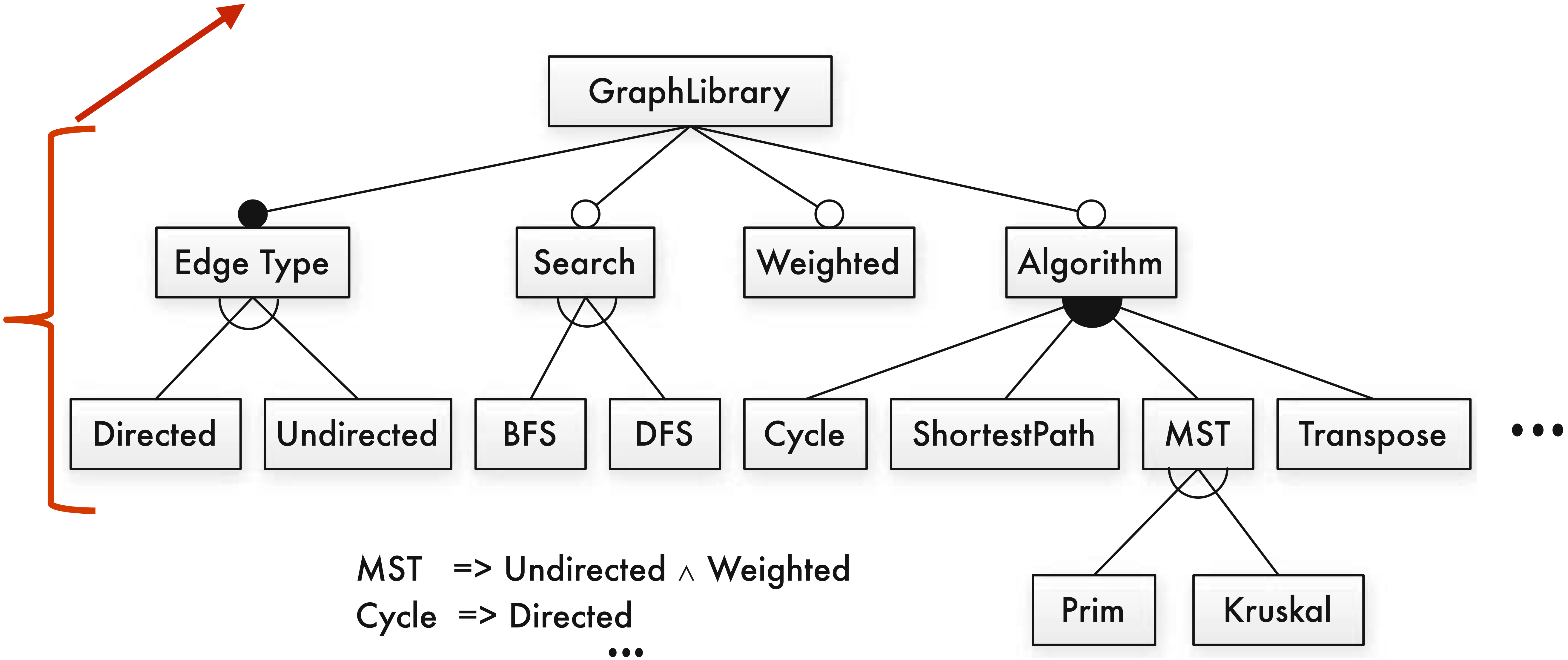
MST => Undirected $\wedge$ Weighted

Cycle => Directed

•••

Graph Library Feature Model

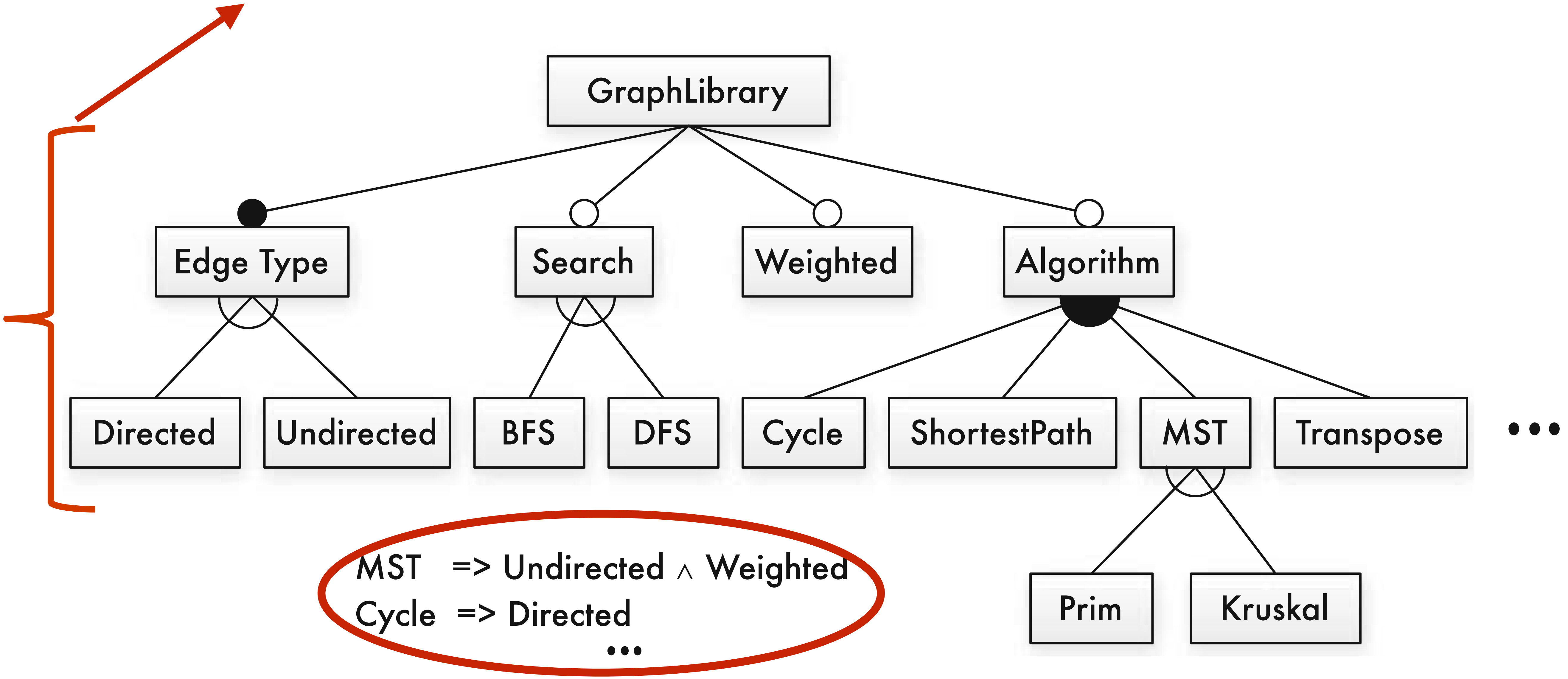
Hierarchy Constraints



MST => Undirected ∧ Weighted
Cycle => Directed
...

Graph Library Feature Model

Hierarchy Constraints



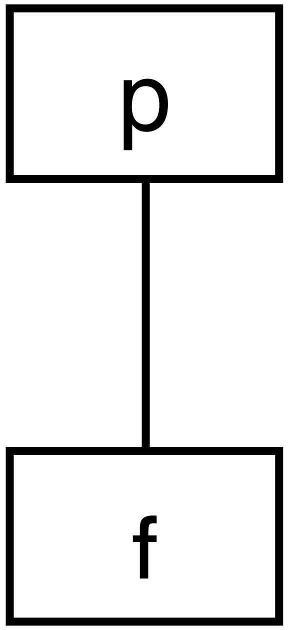
MST => Undirected ∧ Weighted

Cycle => Directed

…

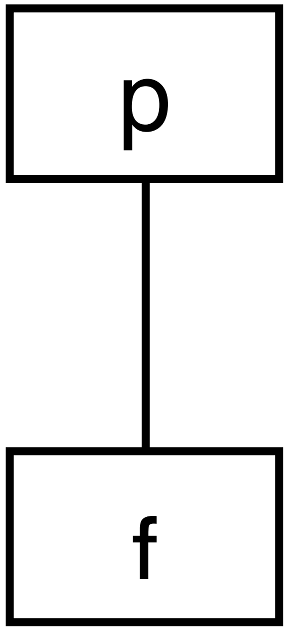
Cross-tree Constraints

Hierarchal Relationships

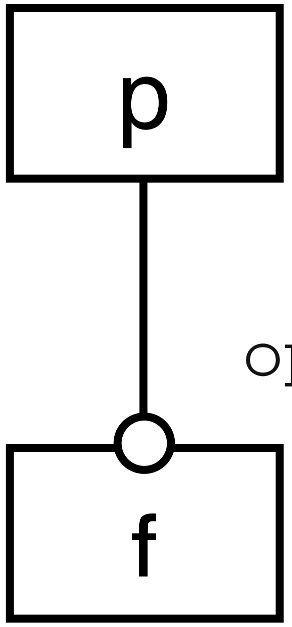


- Parent/child relationship
- Child cannot be selected unless parent is selected

Hierarchal Relationships



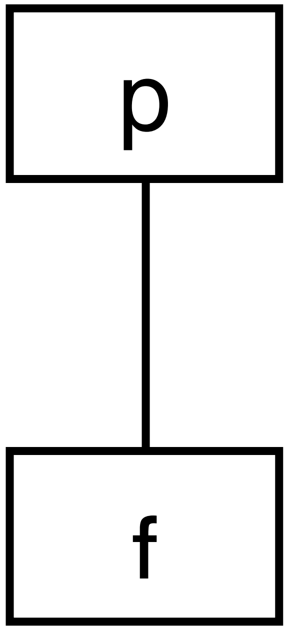
- Parent/child relationship
- Child cannot be selected unless parent is selected



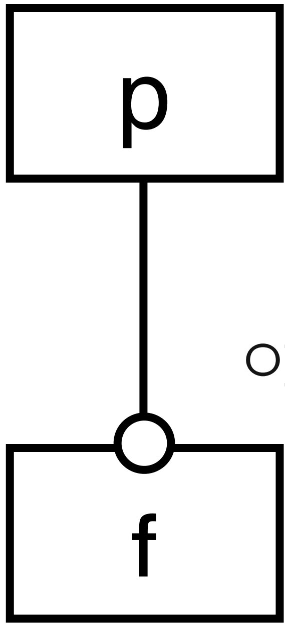
optional feature

$$\texttt{optional}(\texttt{p},\texttt{f}) \equiv \texttt{f} \Rightarrow \texttt{p}$$

Hierarchal Relationships

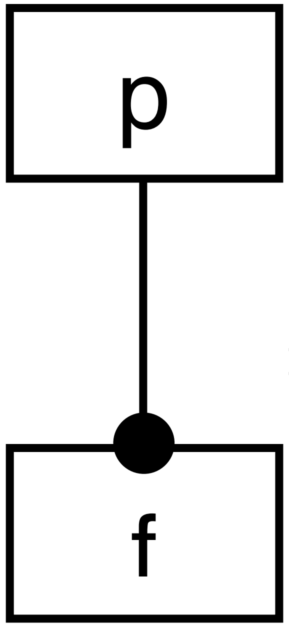


- Parent/child relationship
- Child cannot be selected unless parent is selected



optional feature

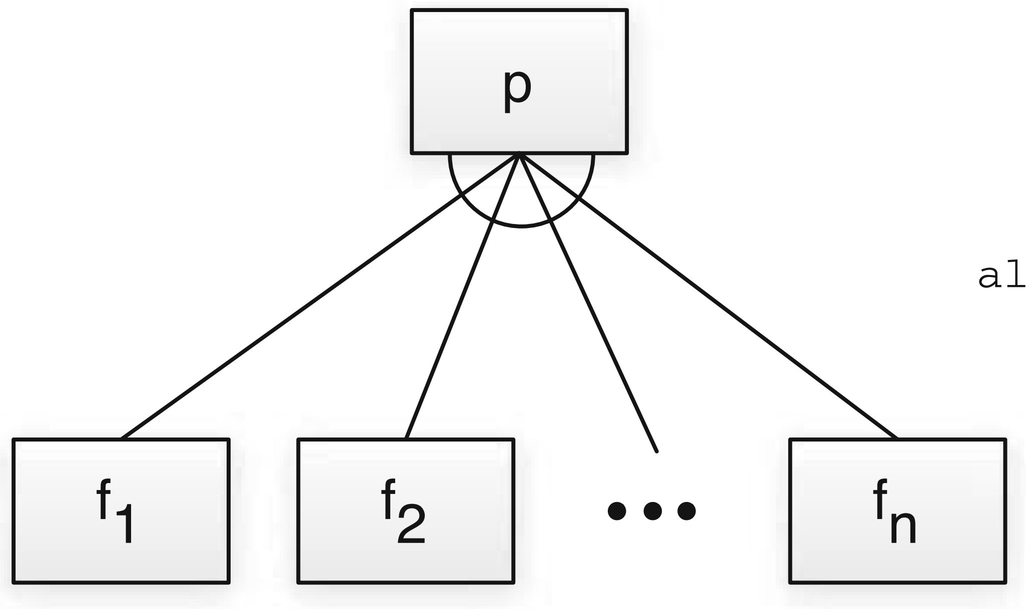
$\texttt{optional}(\texttt{p},\texttt{f}) \equiv \texttt{f} \Rightarrow \texttt{p}$



mandatory feature

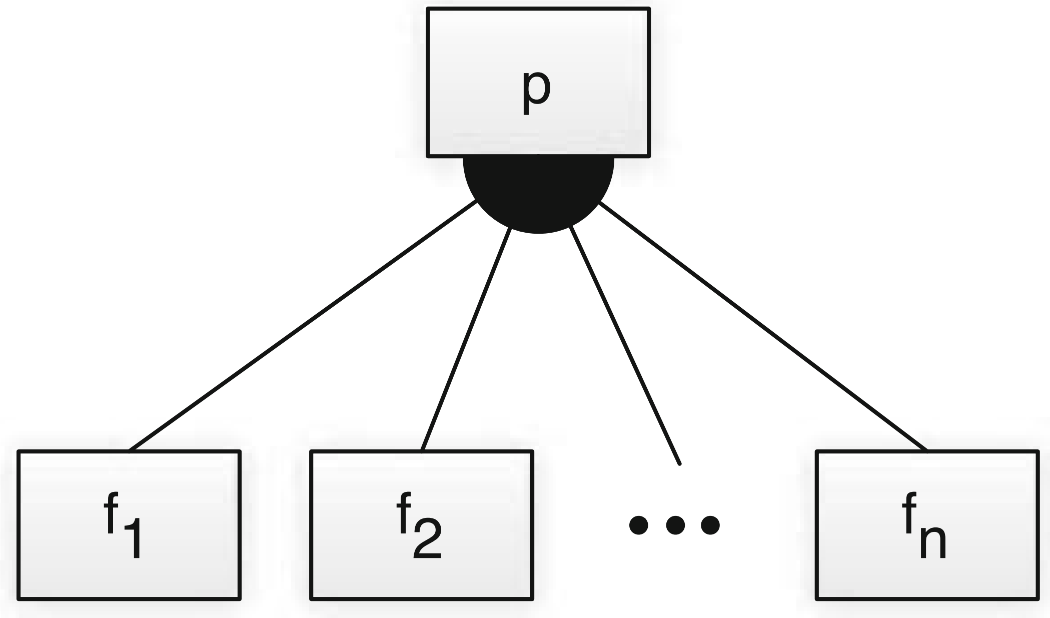
$\texttt{mandatory}(\texttt{p},\texttt{f}) \equiv \texttt{f} \Leftrightarrow \texttt{p}$

Hierarchal Relationships (Groups)



xor group

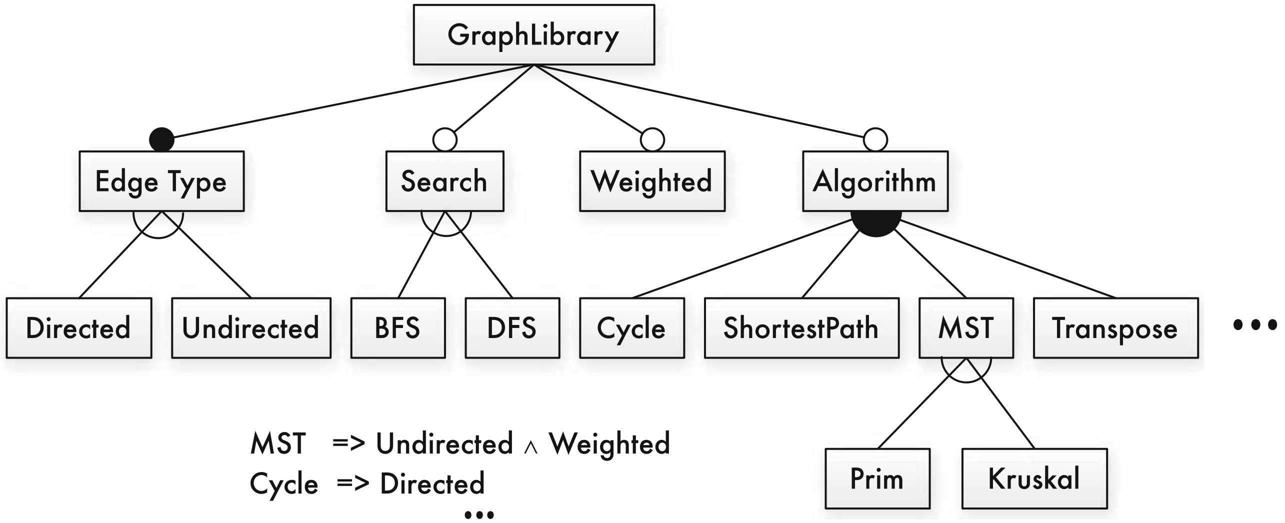
$$\texttt{alternative}(p, \{f_1, ..., f_n\}) \equiv ((f_1 \vee \ldots \vee f_n) \Leftrightarrow p) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j)$$



or group

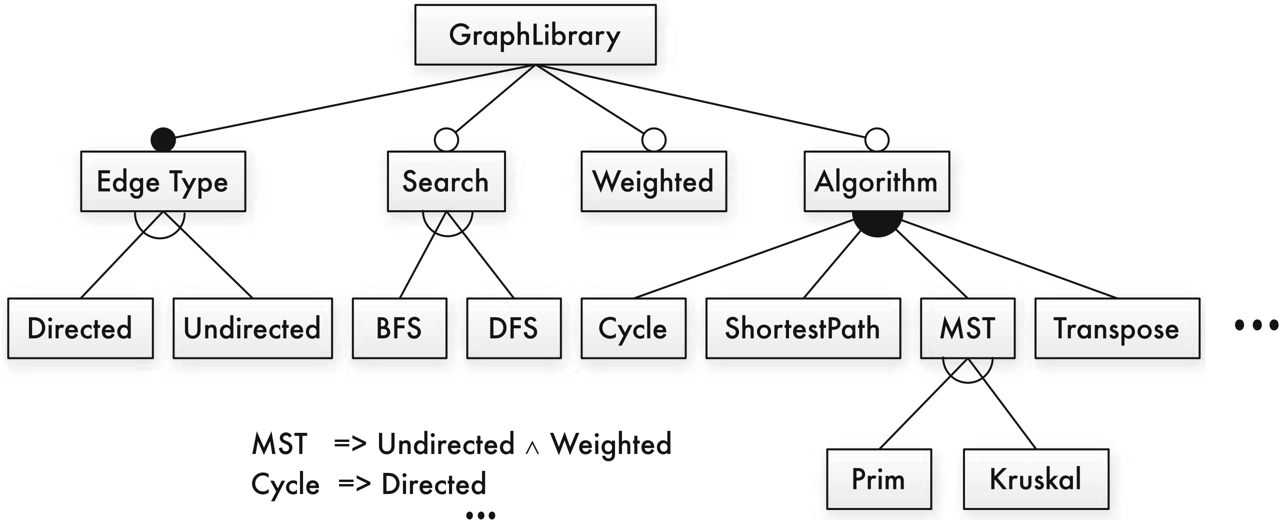
$$\texttt{or}(p, \{f_1, ...f_n\}) \equiv (f_1 \vee \ldots \vee f_n) \Leftrightarrow p$$

Feature Model in Propositional Logic



```
  root(GraphLibrary)
∧ mandatory(GraphLibrary,EdgeType)
∧ optional(GraphLibrary,Search)
∧ optional(GraphLibrary,Weighted)
∧ optional(GraphLibrary,Algorithm)
∧ alternative(EdgeType,{Directed,Undirected})
∧ or(Search,{BFS,DFS})
∧ or(Algorithm,{Cycle,ShortestPath,MST,Transpose})
∧ alternative(MST,{Prim,Kruskal})
∧ (MST ⇒ Weighted)
∧ (Cycle ⇒ Directed)
∧ (···)
```

Feature Model in Propositional Logic



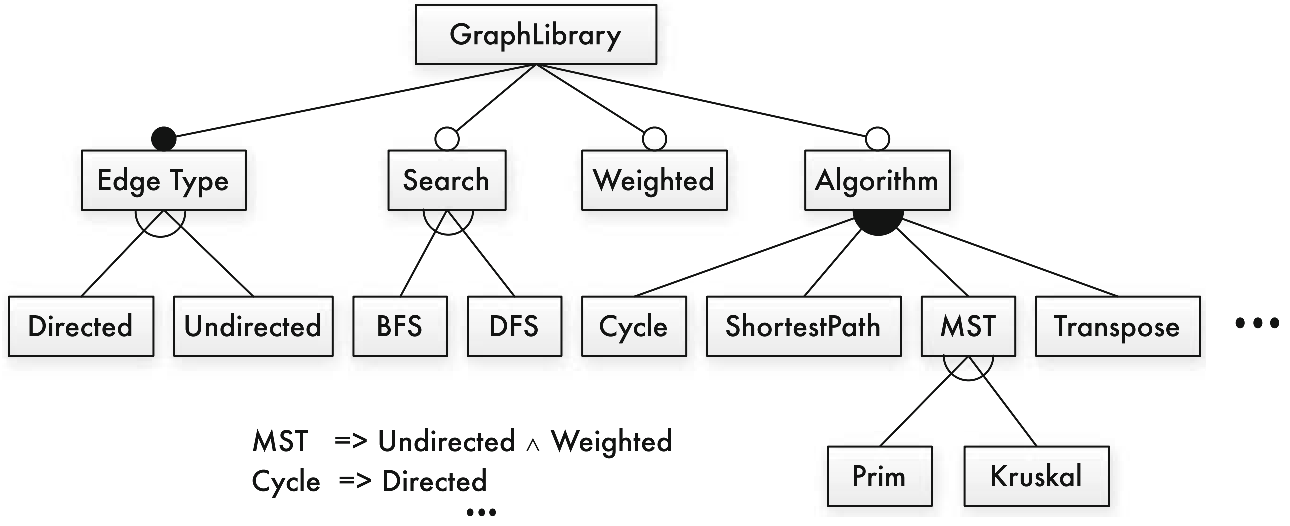
$$
\begin{aligned}
&\ \mathtt{GraphLibrary}\\
&\wedge (\mathtt{EdgeType} \Leftrightarrow \mathtt{GraphLibrary})\\
&\wedge (\mathtt{Search} \Rightarrow \mathtt{EdgeType})\\
&\wedge (\mathtt{Weighted} \Rightarrow \mathtt{EdgeType})\\
&\wedge (\mathtt{Algorithm} \Rightarrow \mathtt{EdgeType})\\
&\wedge (((\mathtt{Directed} \vee \mathtt{Undirected}) \Leftrightarrow \mathtt{EdgeType}) \wedge \neg(\mathtt{Directed} \wedge \mathtt{Undirected}))\\
&\wedge ((\mathtt{BFS} \vee \mathtt{DFS}) \Leftrightarrow \mathtt{Search})\\
&\wedge ((\mathtt{Cycle} \vee \mathtt{ShortestPath} \vee \mathtt{MST} \vee \mathtt{Transpose}) \Leftrightarrow \mathtt{Algorithm})\\
&\wedge (((\mathtt{Prim} \vee \mathtt{Kruskal}) \Leftrightarrow \mathtt{MST}) \wedge \neg(\mathtt{Prim} \wedge \mathtt{Kruskal}))\\
&\wedge (\mathtt{MST} \Rightarrow \mathtt{Weighted})\\
&\wedge (\mathtt{Cycle} \Rightarrow \mathtt{Directed})\\
&\wedge (\cdots)
\end{aligned}
$$

Feature Modeling Tools/Languages/Notations

- GuiDSL (feature models as a grammar)
- FeatureIDE (graphical and text-based)
- Clafer
- … and many more!

Graph Product Line in Clafer

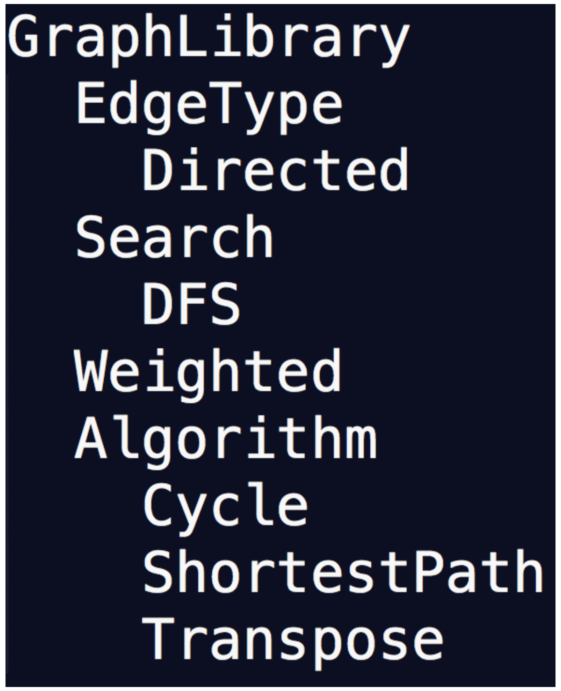
```
GraphLibrary
    xor EdgeType
        Directed
        Undirected
    xor Search ?
        BFS
        DFS
    Weighted ?
    or Algorithm ?
        Cycle
        ShortestPath
        xor MST
            Prim
            Kruskal
        Transpose

    [MST => Undirected && Weighted]
    [Cycle => Directed]
```



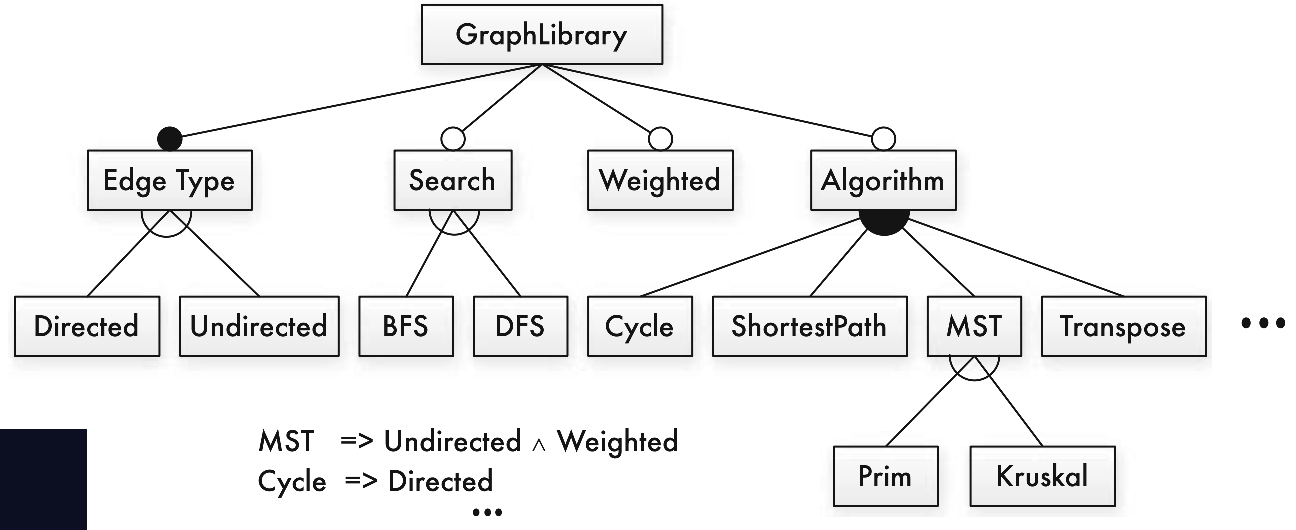
See clafer.org

Graph Product Line in Clafer

```
GraphLibrary
    xor EdgeType
        Directed
        Undirected
    xor Search ?
        BFS
        DFS
    Weighted ?
    or Algorithm ?
        Cycle
        ShortestPath
        xor MST
            Prim
            Kruskal
        Transpose

    [MST => Undirected && Weighted]
    [Cycle => Directed]
```



```
GraphLibrary
  EdgeType
    Directed
  Search
    DFS
  Weighted
  Algorithm
    Cycle
    ShortestPath
    Transpose
```

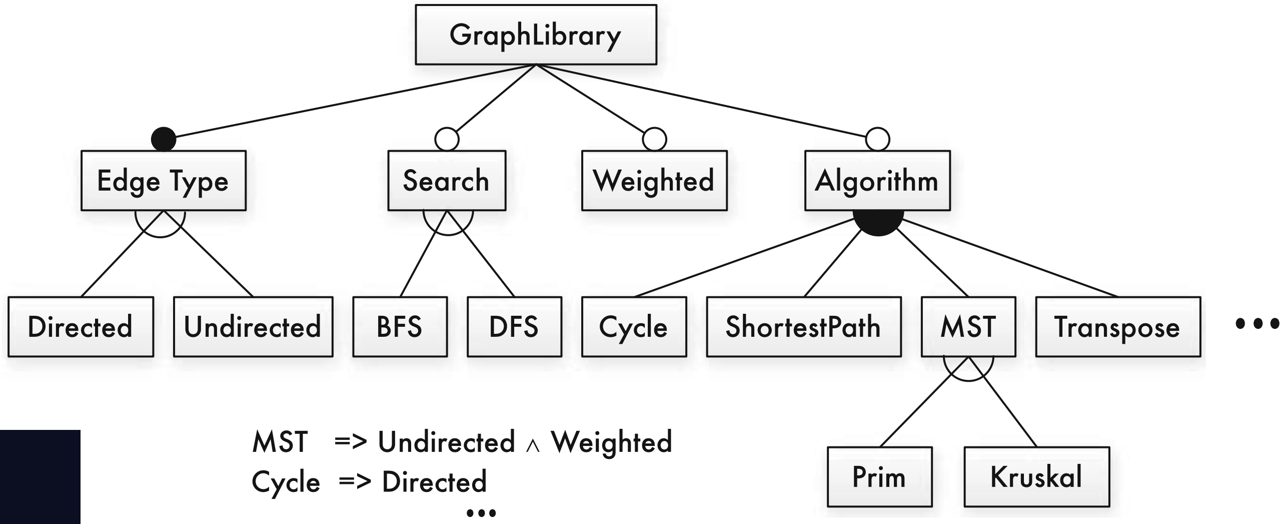


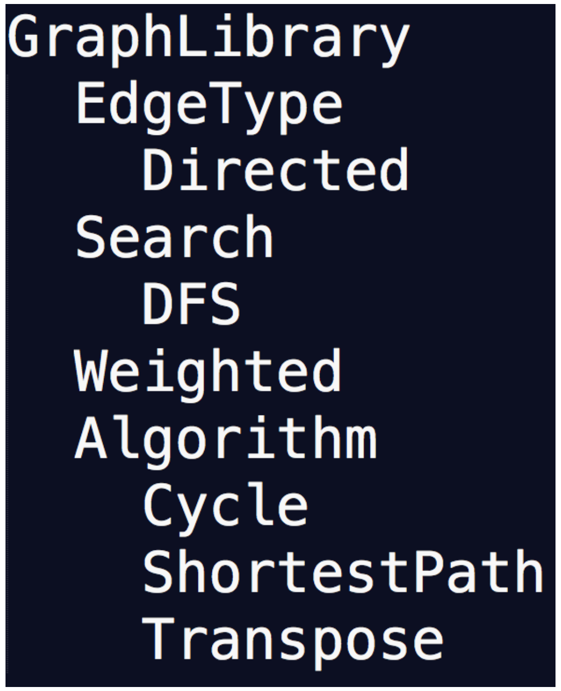
MST => Undirected ∧ Weighted
Cycle => Directed
...

See clafer.org

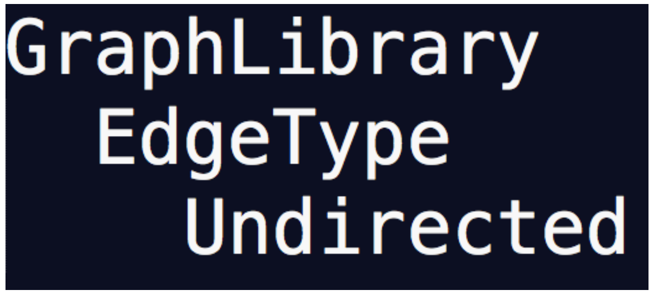
Graph Product Line in Clafer

```
GraphLibrary
    xor EdgeType
        Directed
        Undirected
    xor Search ?
        BFS
        DFS
    Weighted ?
    or Algorithm ?
        Cycle
        ShortestPath
        xor MST
            Prim
            Kruskal
        Transpose

    [MST => Undirected && Weighted]
    [Cycle => Directed]
```





```
GraphLibrary
  EdgeType
    Directed
  Search
    DFS
  Weighted
  Algorithm
    Cycle
    ShortestPath
    Transpose
```

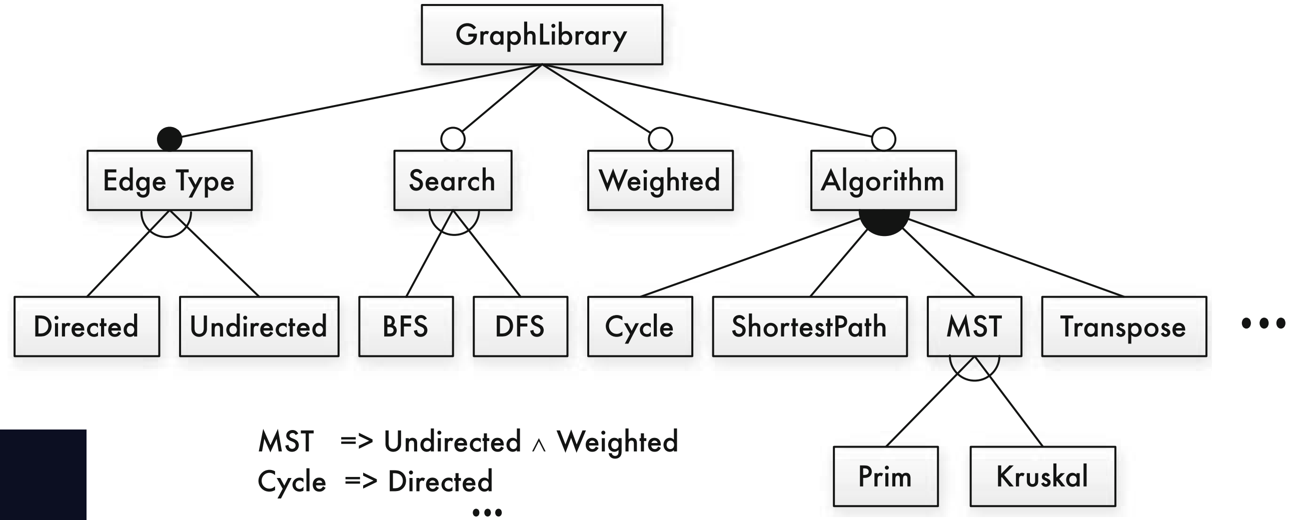


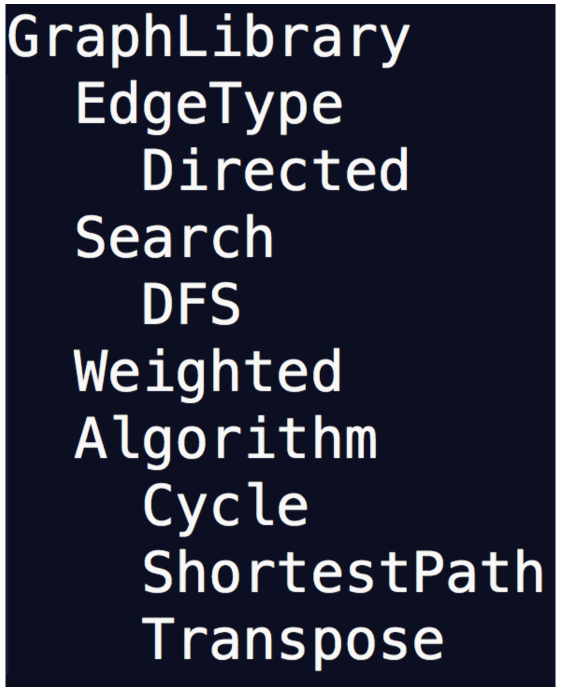
```
GraphLibrary
  EdgeType
    Undirected
```

See clafer.org

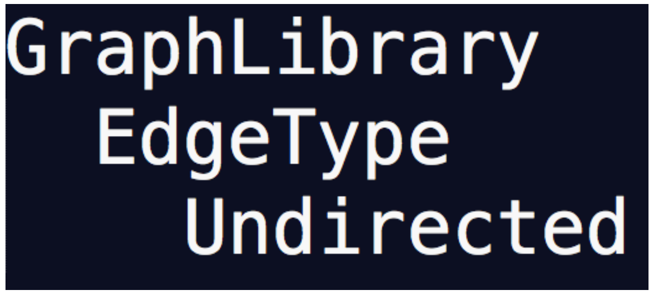
Graph Product Line in Clafer

```
GraphLibrary
    xor EdgeType
        Directed
        Undirected
    xor Search ?
        BFS
        DFS
    Weighted ?
    or Algorithm ?
        Cycle
        ShortestPath
        xor MST
            Prim
            Kruskal
        Transpose

    [MST => Undirected && Weighted]
    [Cycle => Directed]
```

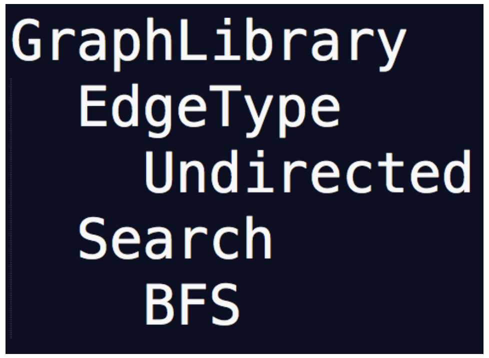




```
GraphLibrary
  EdgeType
    Directed
  Search
    DFS
  Weighted
  Algorithm
    Cycle
    ShortestPath
    Transpose
```



```
GraphLibrary
  EdgeType
    Undirected
```

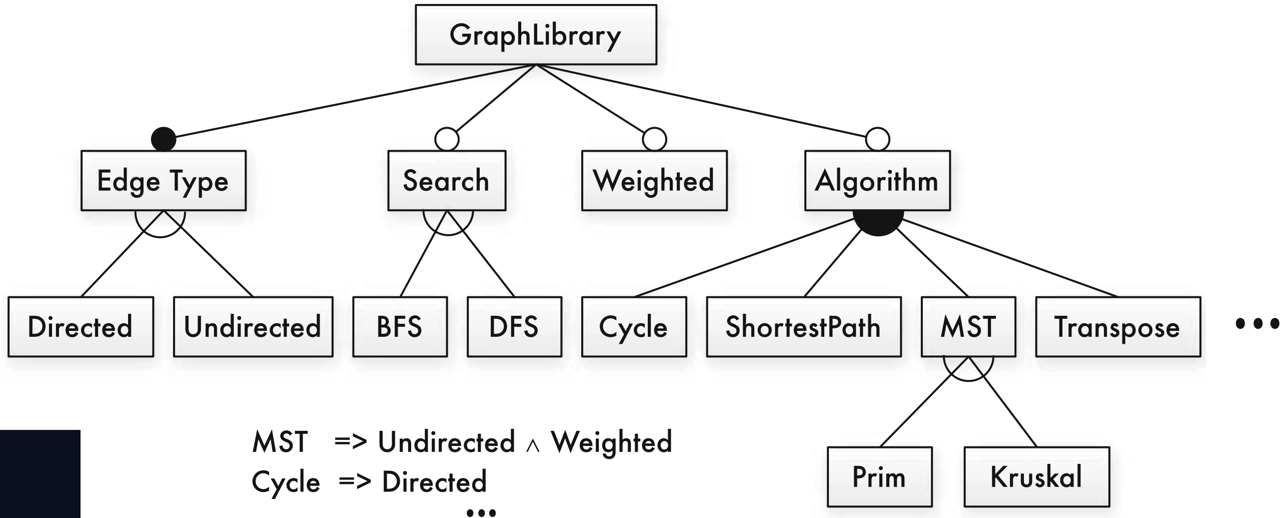


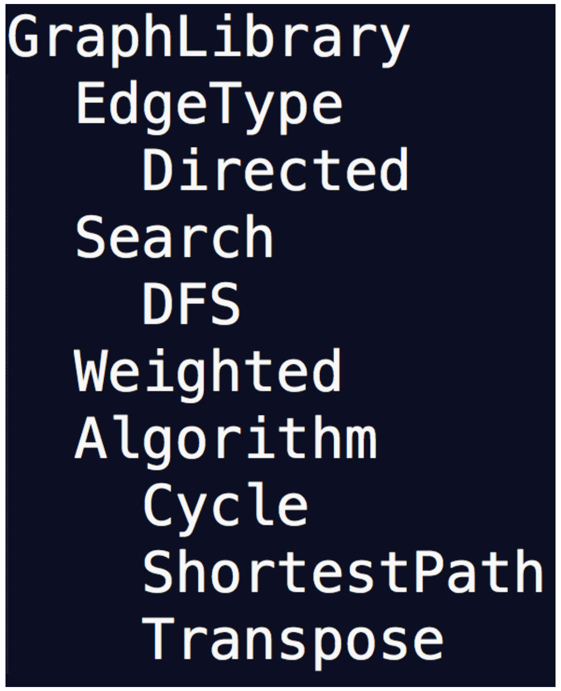
```
GraphLibrary
  EdgeType
    Undirected
  Search
    BFS
```

See clafer.org

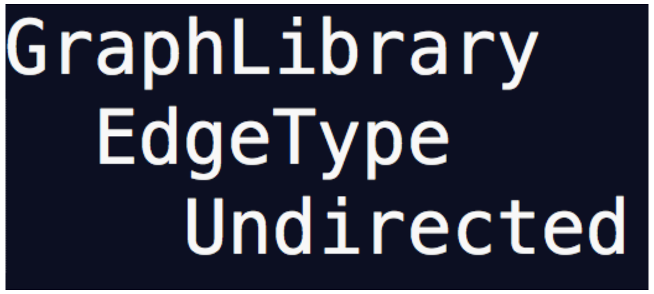
Graph Product Line in Clafer

```
GraphLibrary
    xor EdgeType
        Directed
        Undirected
    xor Search ?
        BFS
        DFS
    Weighted ?
    or Algorithm ?
        Cycle
        ShortestPath
        xor MST
            Prim
            Kruskal
        Transpose

    [MST => Undirected && Weighted]
    [Cycle => Directed]
```

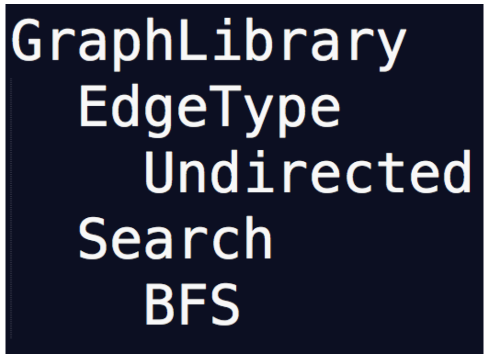




```
GraphLibrary
  EdgeType
    Directed
  Search
    DFS
  Weighted
  Algorithm
    Cycle
    ShortestPath
    Transpose
```



```
GraphLibrary
  EdgeType
    Undirected
```



```
GraphLibrary
  EdgeType
    Undirected
  Search
    BFS
```

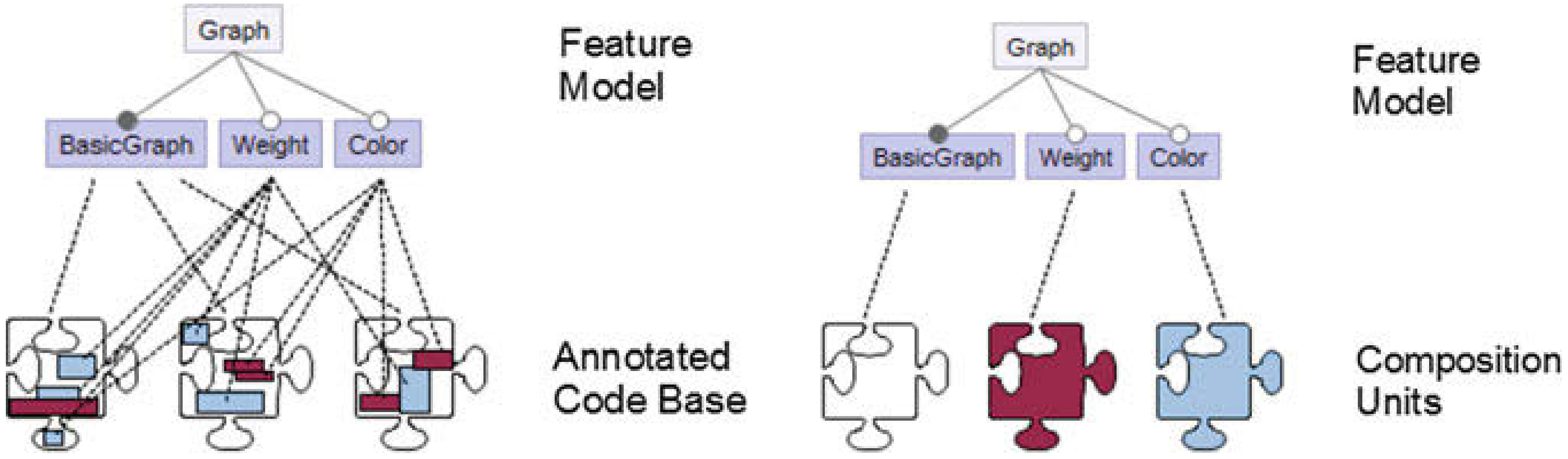
96 possible products!!

See clafer.org

Domain Implementation

Domain Implementation

- Underlying code must be *variable*
- Dimensions of implementation techniques
  - *Binding times:* compile-time binding, load-time binding, and run-time binding.
  - *Representation:* annotation vs composition

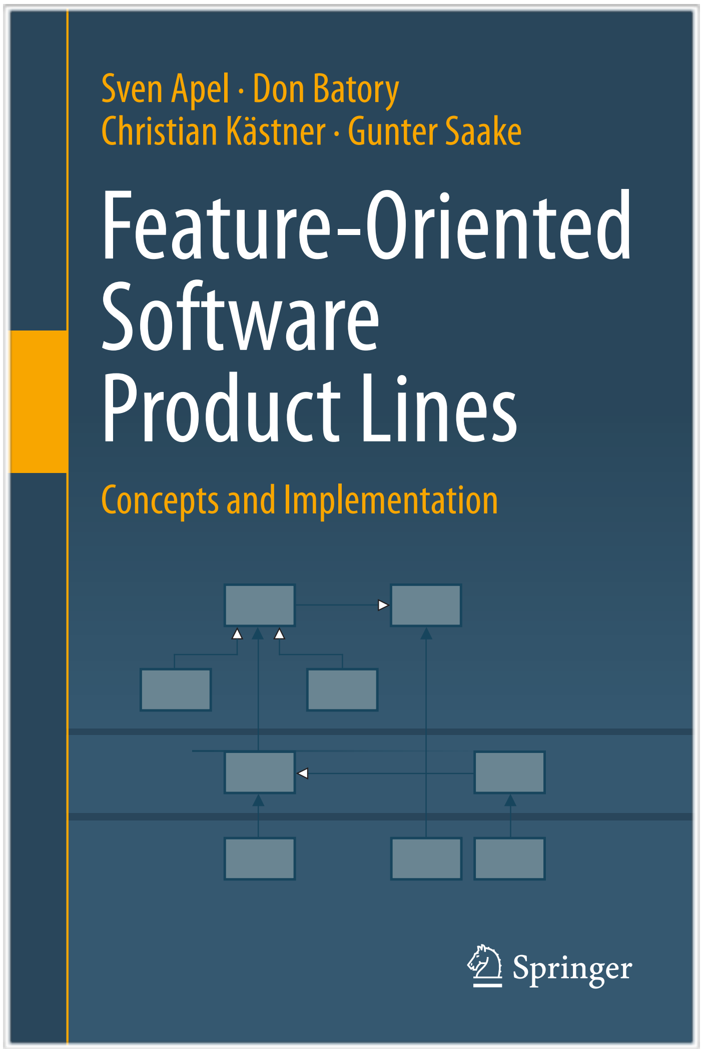


(a) Annotation-based approach **(b)** Composition-based approach

Variability Implementation

- Parameters
- Design patterns
- Build systems
- Preprocessors
- Feature-oriented programming

More inside



Sven Apel · Don Batory
Christian Kästner · Gunter Saake

Feature-Oriented Software Product Lines

Concepts and Implementation

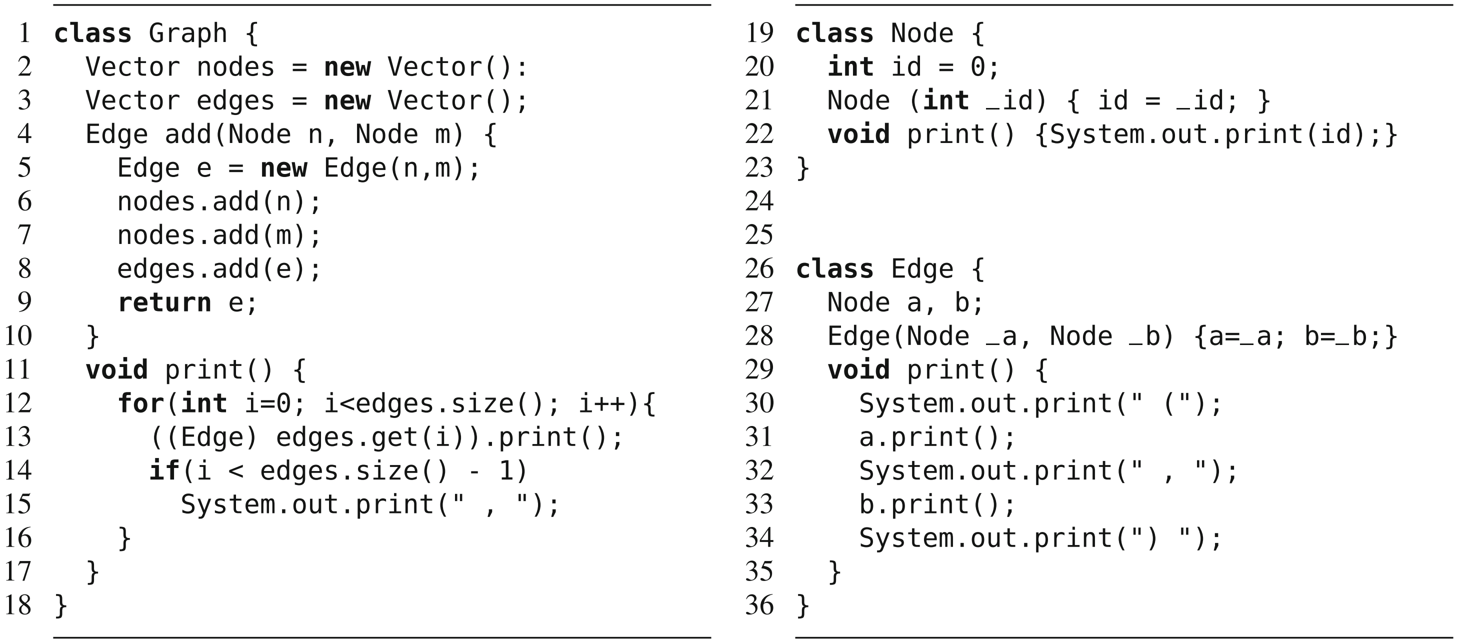
Springer

Working Example:
Basic Graph Library (Java)

```java
class Graph {
  Vector nodes = new Vector():
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n,m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i=0; i<edges.size(); i++){
      ((Edge) edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```

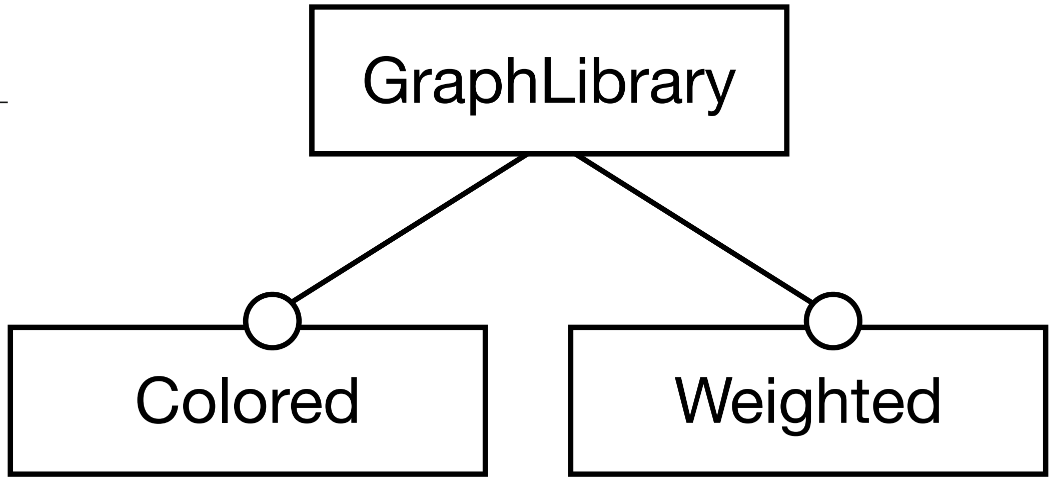
```java
class Node {
  int id = 0;
  Node (int _id) { id = _id; }
  void print() {System.out.print(id);}
}


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

Working Example:
Basic Graph Library (Java)



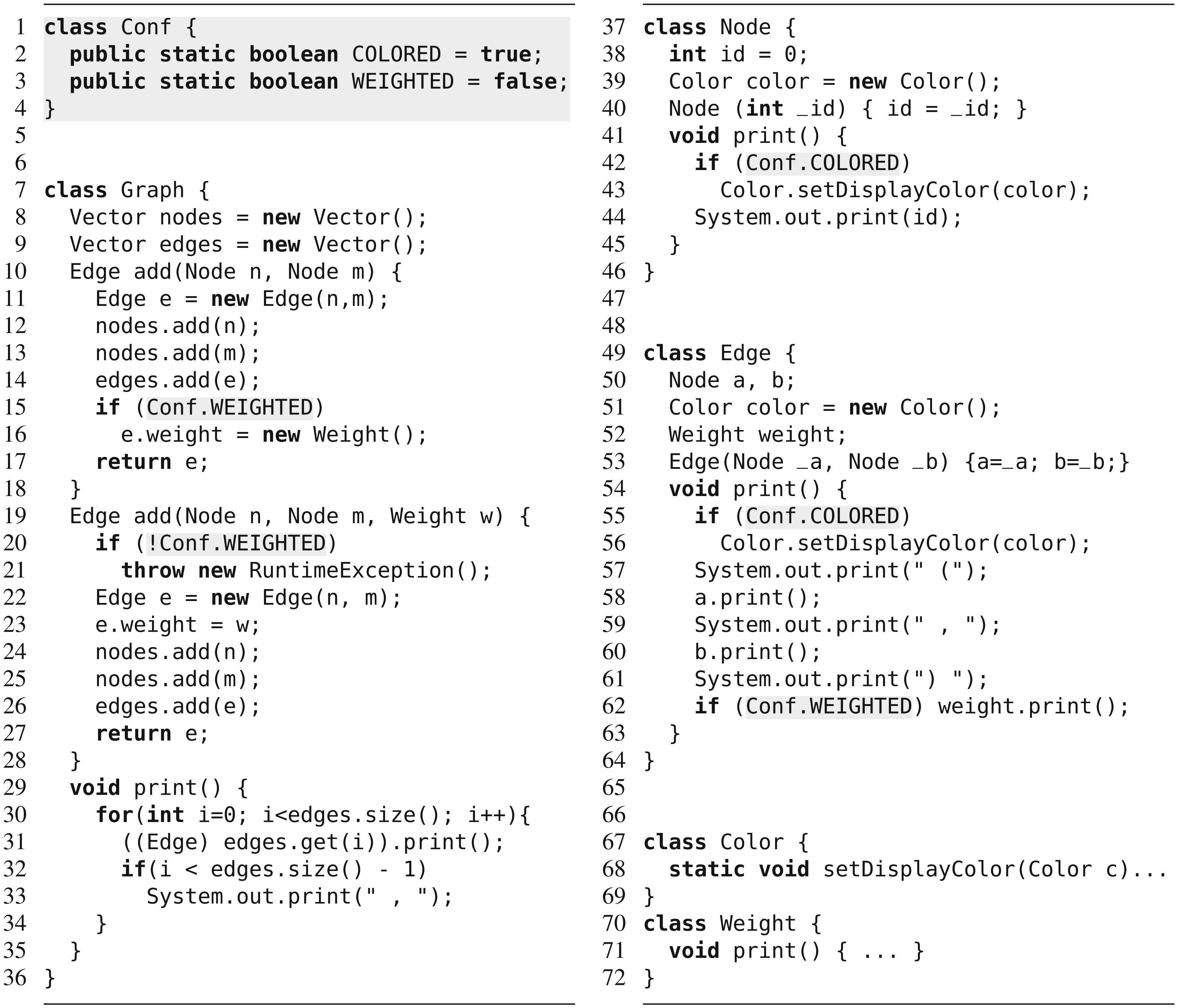
```java
class Graph {
  Vector nodes = new Vector():
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n,m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i=0; i<edges.size(); i++){
      ((Edge) edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```

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


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



Parameters

Variability using Parameters



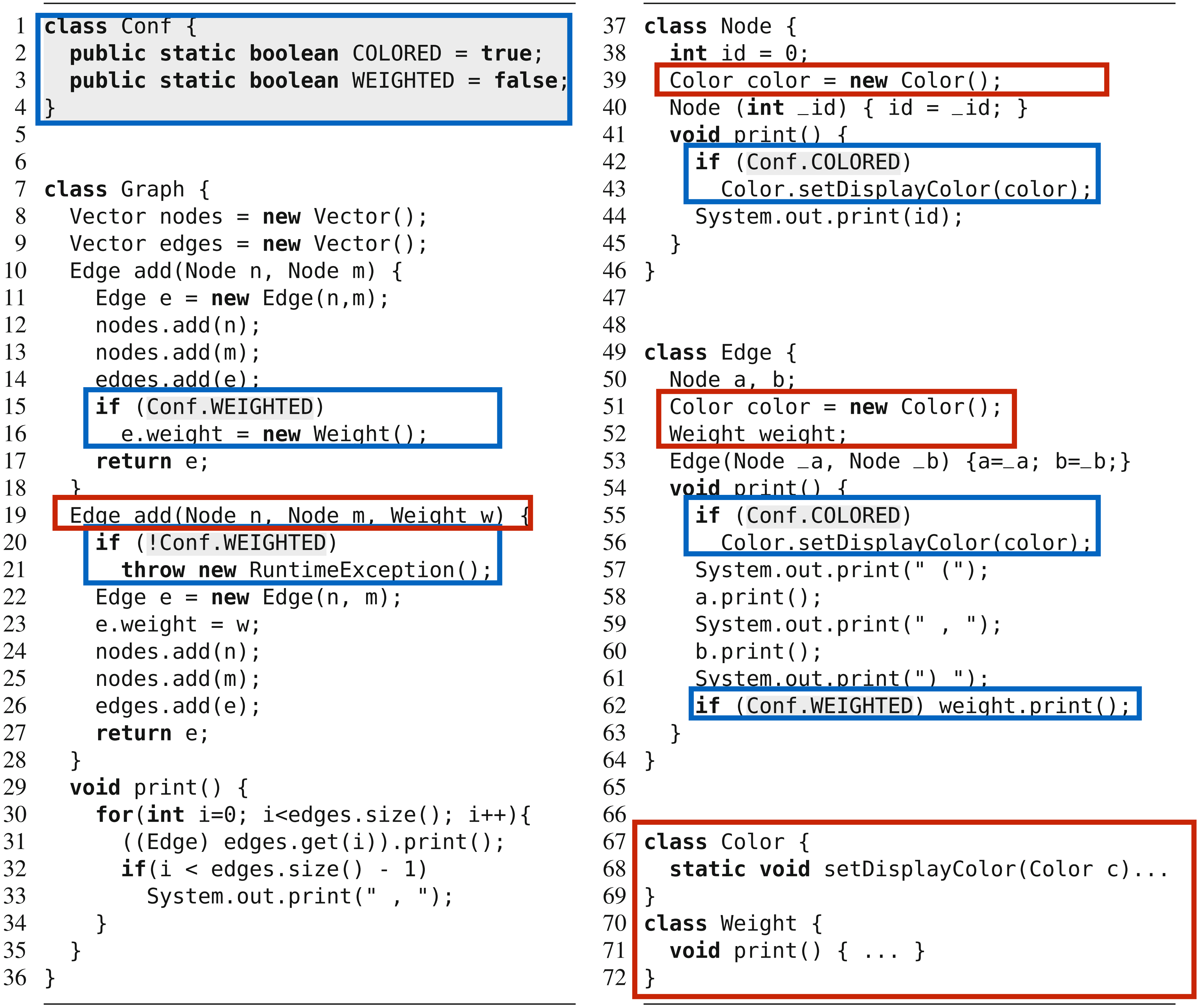
```
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}


class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n,m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    if (Conf.WEIGHTED)
      e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    if (!Conf.WEIGHTED)
      throw new RuntimeException();
    Edge e = new Edge(n, m);
    e.weight = w;
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i=0; i<edges.size(); i++){
      ((Edge) edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
class Node {
  int id = 0;
  Color color = new Color();
  Node (int _id) { id = _id; }
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(id);
  }
}


class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) {a=_a; b=_b;}
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(" (");
    a.print();
    System.out.print(" , ");
    b.print();
    System.out.print(") ");
    if (Conf.WEIGHTED) weight.print();
  }
}


class Color {
  static void setDisplayColor(Color c)...
}
class Weight {
  void print() { ... }
}
```

Variability using Parameters

```
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}


class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n,m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    if (Conf.WEIGHTED)
      e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    if (!Conf.WEIGHTED)
      throw new RuntimeException();
    Edge e = new Edge(n, m);
    e.weight = w;
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i=0; i<edges.size(); i++){
      ((Edge) edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  Node (int _id) { id = _id; }
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(id);
  }
}


class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) {a=_a; b=_b;}
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(" (");
    a.print();
    System.out.print(" , ");
    b.print();
    System.out.print(") ");
    if (Conf.WEIGHTED) weight.print();
  }
}


class Color {
  static void setDisplayColor(Color c)...
}
class Weight {
  void print() { ... }
}
```

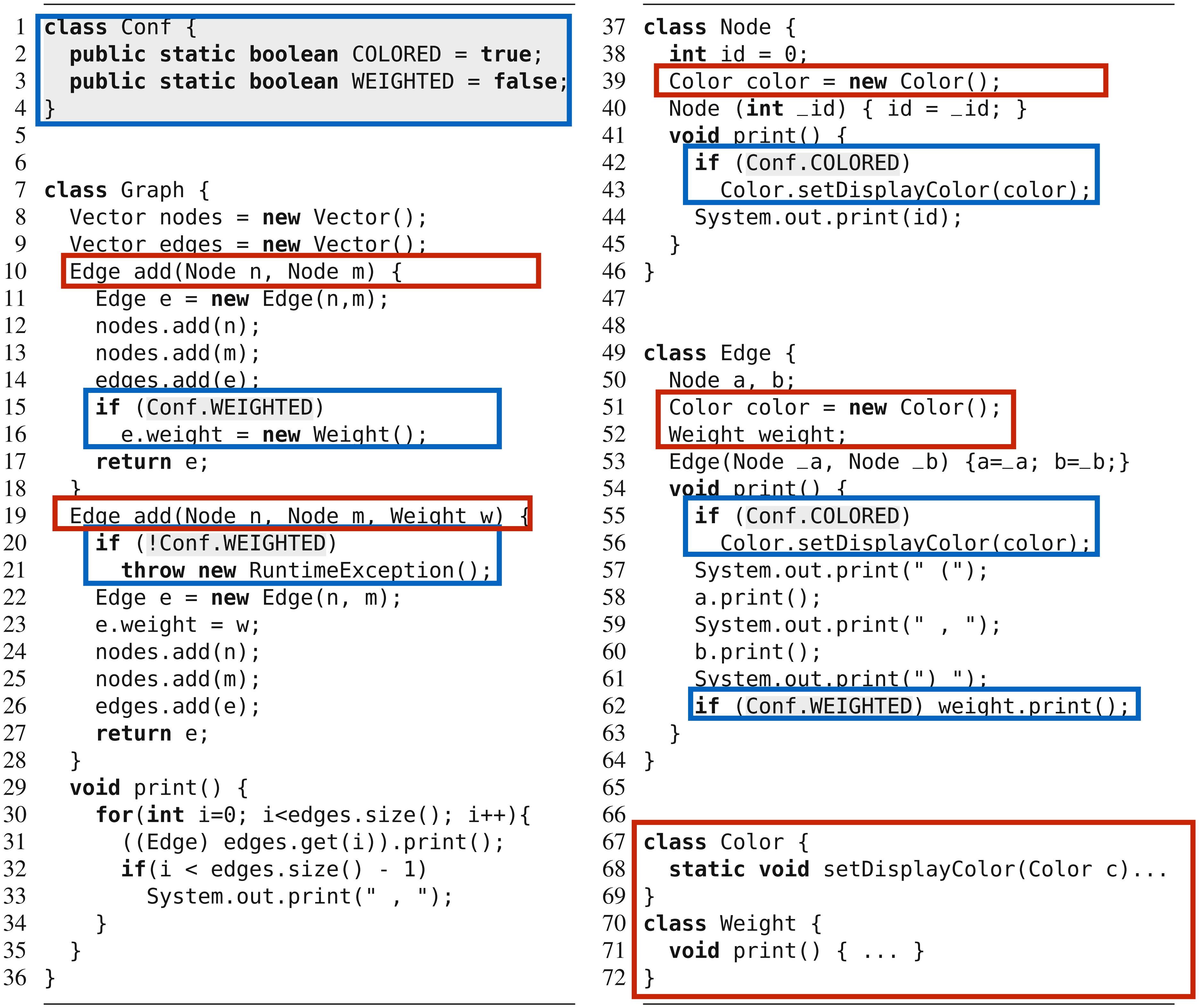
Variability using Parameters



```java
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}


class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n,m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    if (Conf.WEIGHTED)
      e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    if (!Conf.WEIGHTED)
      throw new RuntimeException();
    Edge e = new Edge(n, m);
    e.weight = w;
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i=0; i<edges.size(); i++){
      ((Edge) edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```

```java
class Node {
  int id = 0;
  Color color = new Color();
  Node (int _id) { id = _id; }
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(id);
  }
}


class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) {a=_a; b=_b;}
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(" (");
    a.print();
    System.out.print(" , ");
    b.print();
    System.out.print(") ");
    if (Conf.WEIGHTED) weight.print();
  }
}


class Color {
  static void setDisplayColor(Color c)...
}
class Weight {
  void print() { ... }
}
```

Variability using Parameters



```java
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}


class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n,m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    if (Conf.WEIGHTED)
      e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    if (!Conf.WEIGHTED)
      throw new RuntimeException();
    Edge e = new Edge(n, m);
    e.weight = w;
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i=0; i<edges.size(); i++){
      ((Edge) edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```

```java
class Node {
  int id = 0;
  Color color = new Color();
  Node (int _id) { id = _id; }
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(id);
  }
}


class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) {a=_a; b=_b;}
  void print() {
    if (Conf.COLORED)
      Color.setDisplayColor(color);
    System.out.print(" (");
    a.print();
    System.out.print(" , ");
    b.print();
    System.out.print(") ");
    if (Conf.WEIGHTED) weight.print();
  }
}


class Color {
  static void setDisplayColor(Color c)...
}
class Weight {
  void print() { ... }
}
```

Variability using Parameters

+ simple
+ flexible
+ language support
- code bloat
- computing overhead
- non-modular solution

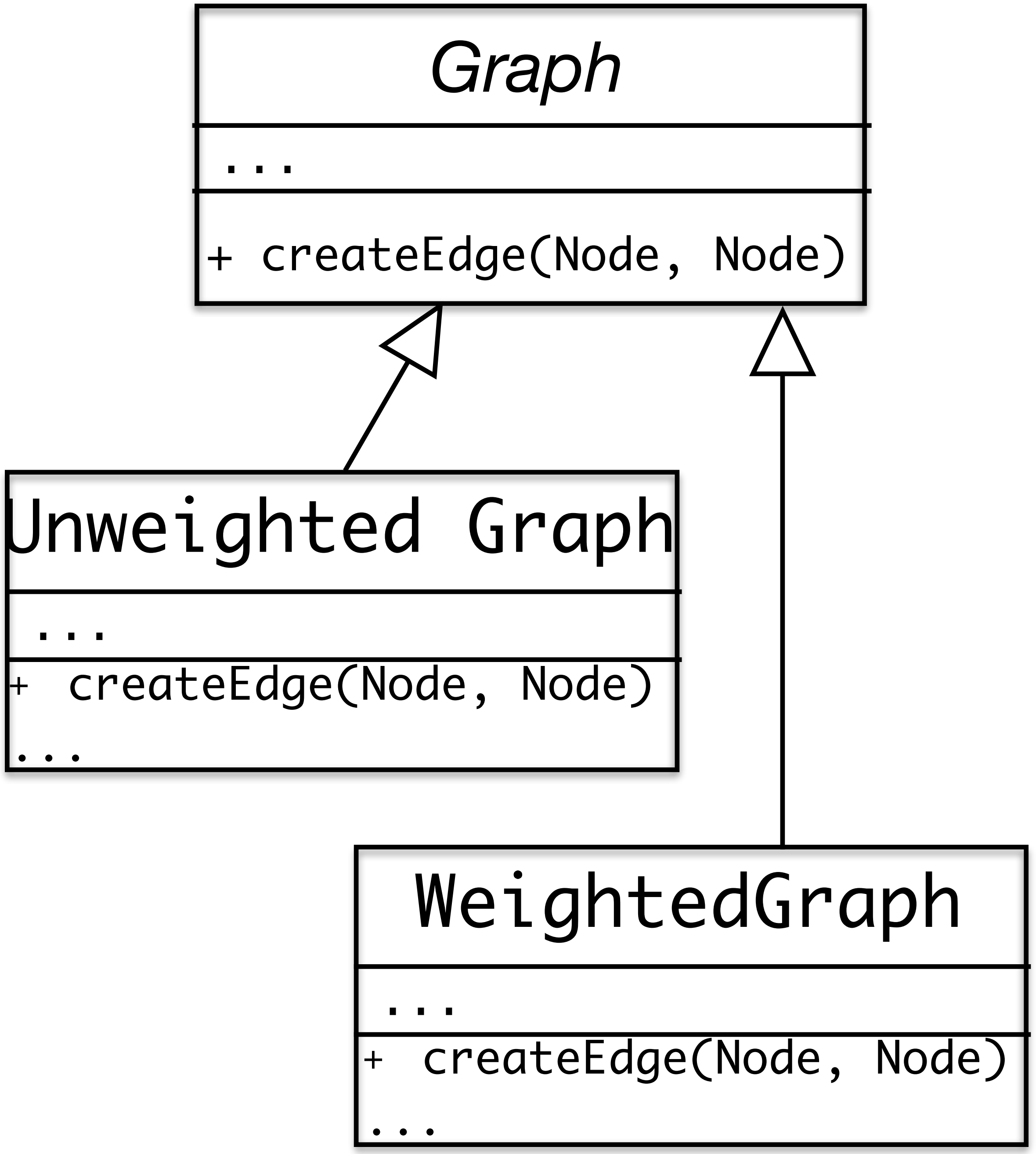
Variability using Parameters

Annotation
Run-time

+ simple
+ flexible
+ language support
- code bloat
- computing overhead
- non-modular solution

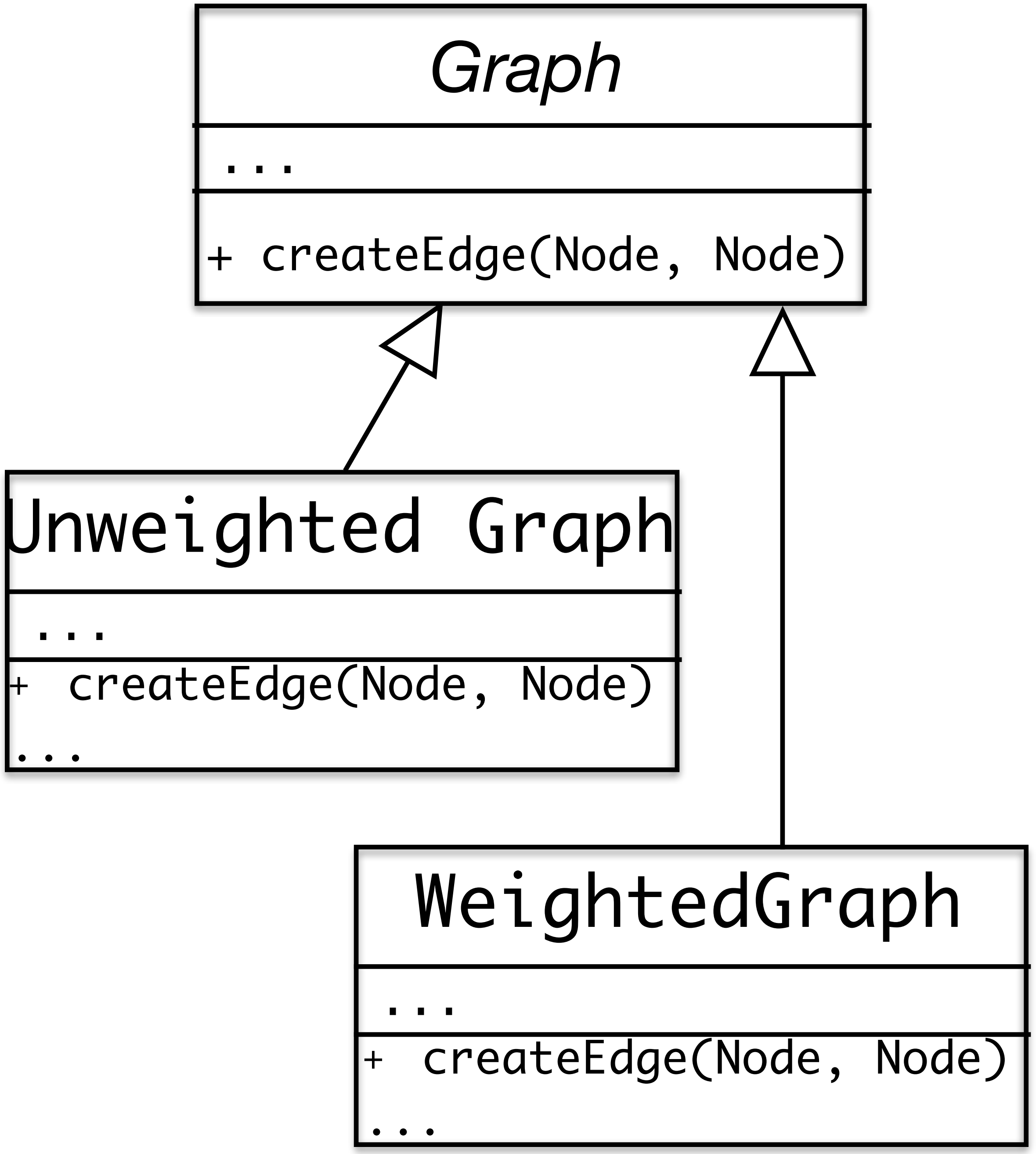
Design Patterns
(Templates)

Templates



```
abstract class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = createEdge(n, m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  protected abstract Edge createEdge(Node n, Node m);
  ...
}

class UnweightedGraph extends Graph {
  protected Edge createEdge(Node n, Node m) {
    return new Edge(n, m);
  }
}

class WeightedGraph extends Graph {
  protected Edge createEdge(Node n, Node m) {
    WeightedEdge e = new WeightedEdge(n, m);
    e.weight=new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    WeightedEdge e = (WeightedEdge) createEdge(n, m);
    e.weight = w;
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
}
```

Templates



Graph

...

+ createEdge(Node, Node)

Unweighted Graph

...

+ createEdge(Node, Node)

...

WeightedGraph

...

+ createEdge(Node, Node)

...

```
abstract class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = createEdge(n, m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  protected abstract Edge createEdge(Node n, Node m);
  ...
}

class UnweightedGraph extends Graph {
  protected Edge createEdge(Node n, Node m) {
    return new Edge(n, m);
  }
}

class WeightedGraph extends Graph {
  protected Edge createEdge(Node n, Node m) {
    WeightedEdge e = new WeightedEdge(n, m);
    e.weight=new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    WeightedEdge e = (WeightedEdge) createEdge(n, m);
    e.weight = w;
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
}
```

Variability using Design Patterns

+ Well-established

+ Easy to communicate design decisions

- Architecture overhead

- Need to preplan extensions

Variability using Design Patterns

Composition
Run-time

+ Well-established

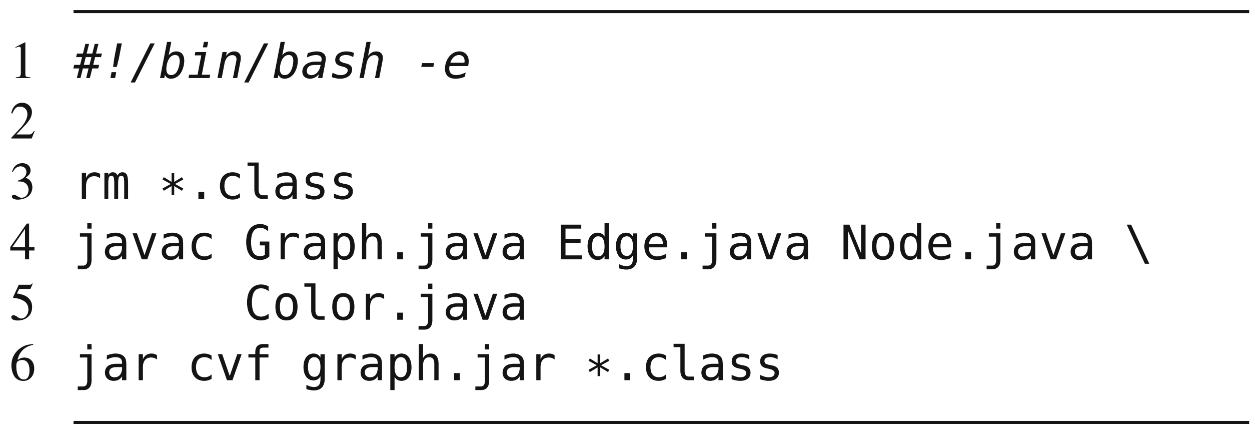
+ Easy to communicate design decisions

- Architecture overhead

- Need to preplan extensions

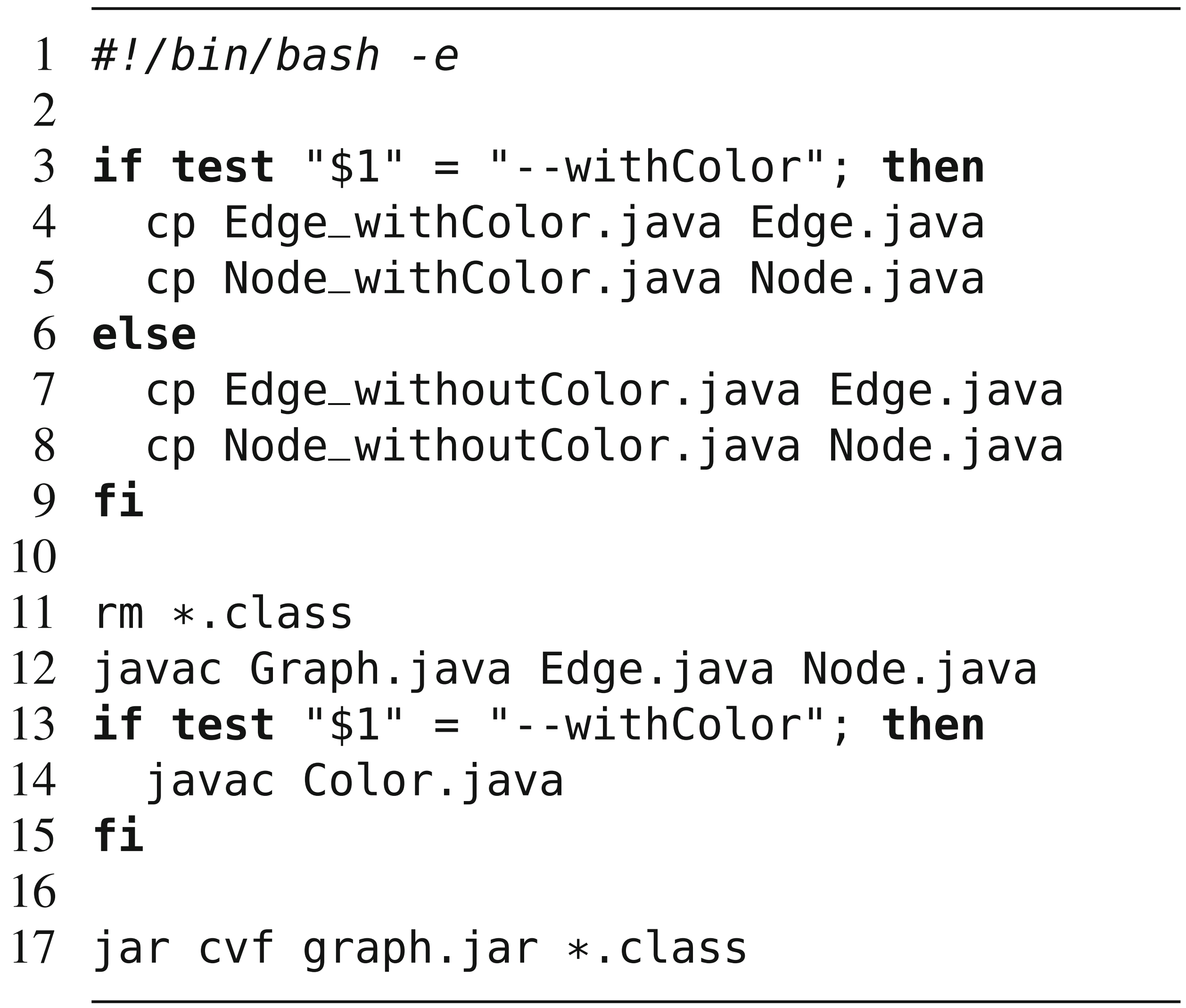
Build Systems

Variability Using Build Scripts



```
#!/bin/bash -e

rm *.class
javac Graph.java Edge.java Node.java \
      Color.java
jar cvf graph.jar *.class
```

No variability



```
#!/bin/bash -e

if test "$1" = "--withColor"; then
  cp Edge_withColor.java Edge.java
  cp Node_withColor.java Node.java
else
  cp Edge_withoutColor.java Edge.java
  cp Node_withoutColor.java Node.java
fi

rm *.class
javac Graph.java Edge.java Node.java
if test "$1" = "--withColor"; then
  javac Color.java
fi

jar cvf graph.jar *.class
```

With variability

Variability Using Build Scripts

```
#!/bin/bash -e

rm *.class
javac Graph.java Edge.java Node.java \
      Color.java
jar cvf graph.jar *.class
```

No variability

```
#!/bin/bash -e

if test "$1" = "--withColor"; then
  cp Edge_withColor.java Edge.java
  cp Node_withColor.java Node.java
else
  cp Edge_withoutColor.java Edge.java
  cp Node_withoutColor.java Node.java
fi

rm *.class
javac Graph.java Edge.java Node.java
if test "$1" = "--withColor"; then
  javac Color.java
fi

jar cvf graph.jar *.class
```

With variability

Variability Using Build Scripts

Annotation
Compile-time

+ simple if features can be mapped into files

+ can control other types of parameters

- code duplication if finer level of granularity needed

- hard to analyze

Preprocessors

Variability Using Preprocessors

```
class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n,m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    /*IF[FEAT_WEIGHTED]*/
    e.weight = new Weight();
    /*END[FEAT_WEIGHTED]*/
    return e;
  }
  /*IF[FEAT_WEIGHTED]*/
  Edge add(Node n, Node m, Weight w) {
    Edge e = new Edge(n, m);
    e.weight = w;
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  /*END[FEAT_WEIGHTED]*/
  void print() {
    for(int i=0; i<edges.size(); i++){
      ((Edge) edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}


/*IF[FEAT_COLORED]*/
class Color {
  static void setDisplayColor(Color c)...
}
/*END[FEAT_COLORED]*/
class Node {
  int id = 0;
  /*IF[FEAT_COLORED]*/
  Color color = new Color();
  /*END[FEAT_COLORED]*/
  Node (int _id) { id = _id; }
  void print() {
    /*IF[FEAT_COLORED]*/
    Color.setDisplayColor(color);
    /*END[FEAT_COLORED]*/
    System.out.print(id);
  }
}

class Edge {
  Node a, b;
  /*IF[FEAT_COLORED]*/
  Color color = new Color();
  /*END[FEAT_COLORED]*/
  /*IF[FEAT_WEIGHTED]*/
  Weight weight;
  /*END[FEAT_WEIGHTED]*/
  Edge(Node _a, Node _b) {a=_a; b=_b;}
  void print() {
    /*IF[FEAT_COLORED]*/
    Color.setDisplayColor(color);
    /*END[FEAT_COLORED]*/
    System.out.print(" (");
    a.print();
    System.out.print(" , ");
    b.print();
    System.out.print(") ");
    /*IF[FEAT_WEIGHTED]*/
    weight.print();
    /*END[FEAT_WEIGHTED]*/
  }
}

/*IF[FEAT_WEIGHTED]*/
class Weight {
  void print() { ... }
}
/*END[FEAT_WEIGHTED]*/
```

Variability using the C Preprocessor

Can you spot the error?

```c
int a = 1;
int b = 0;
#ifdef A
int c = a;
#else
char c = a;
#endif
if (c) {
#ifdef B
  c += a;
  c /= b;
}
#endif
```

Variability using the C Preprocessor

Can you spot the error?

```
int a = 1;
int b = 0;
#ifdef A
int c = a;
#else
char c = a;
#endif
if (c) {
#ifdef B
  c += a;
  c /= b;
}
#endif
```

Compile time:
no matching closing braces when B is not selected

Variability using the C Preprocessor

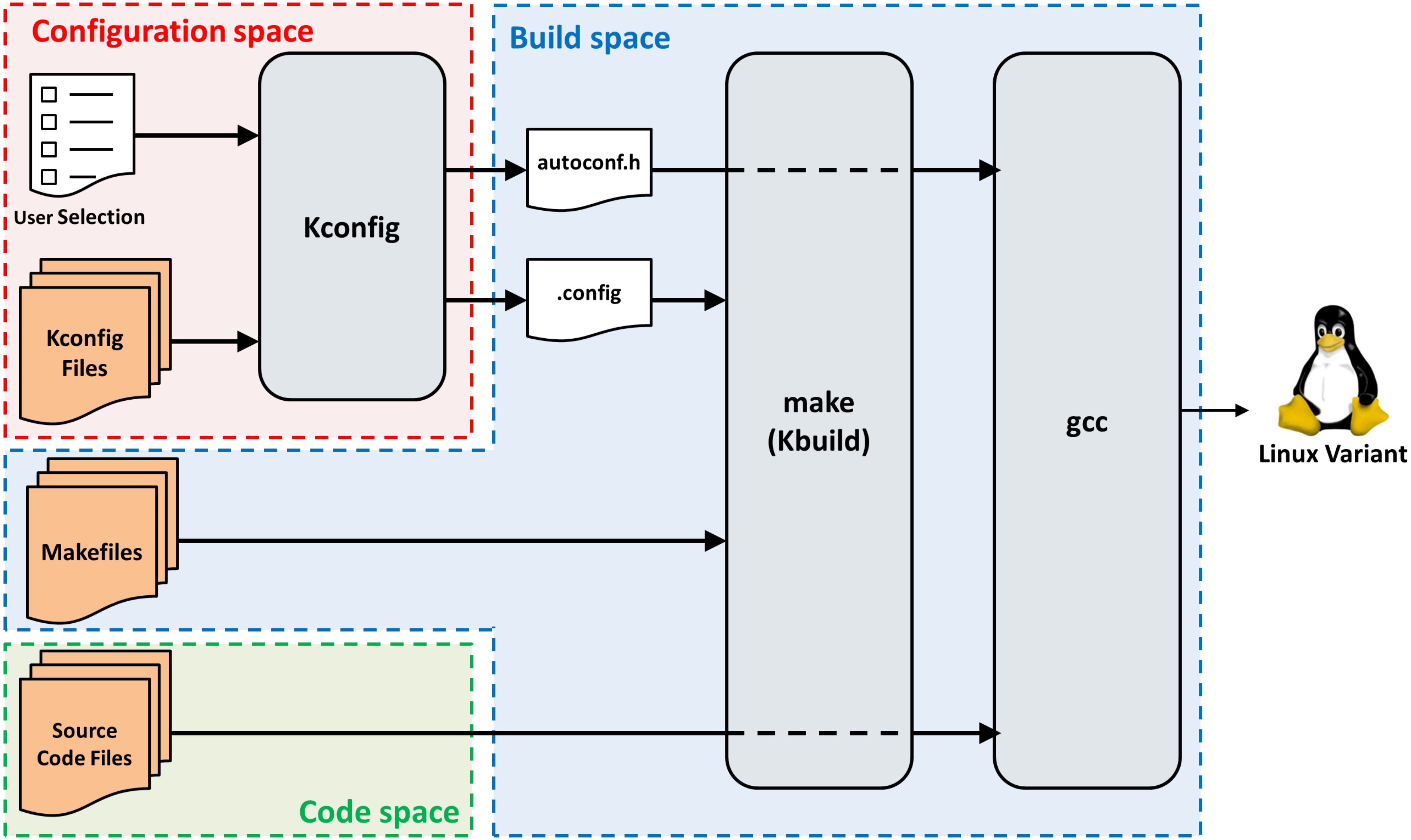
Can you spot the error?

```
int a = 1;
int b = 0;
#ifdef A
int c = a;
#else
char c = a;
#endif
if (c) {
#ifdef B
  c += a;
  c /= b;
}
#endif
```

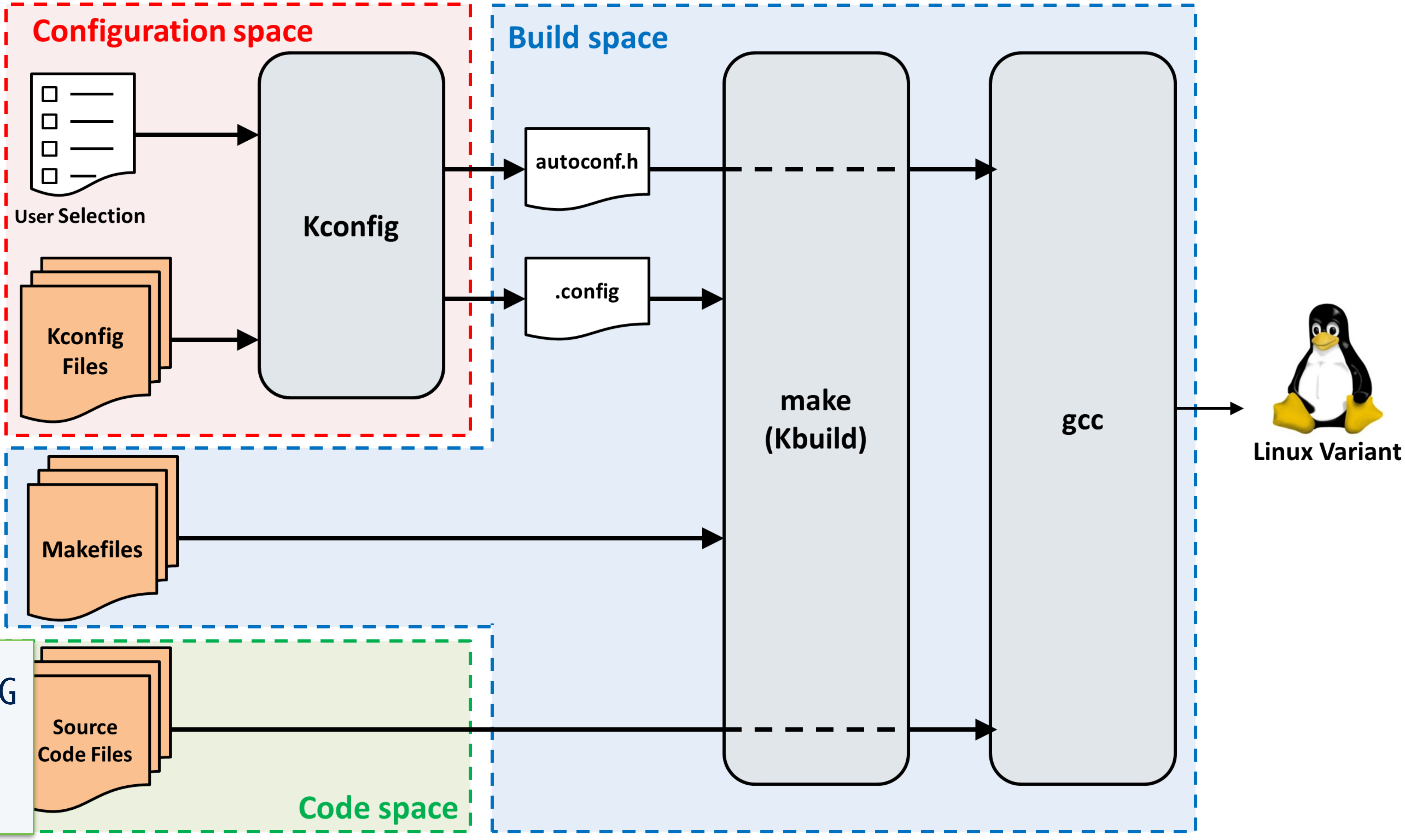
Compile time:
no matching closing braces when B is not selected

Runtime:
division by zero when B is selected

Linux Kernel:
Variability using Build Systems & CPP

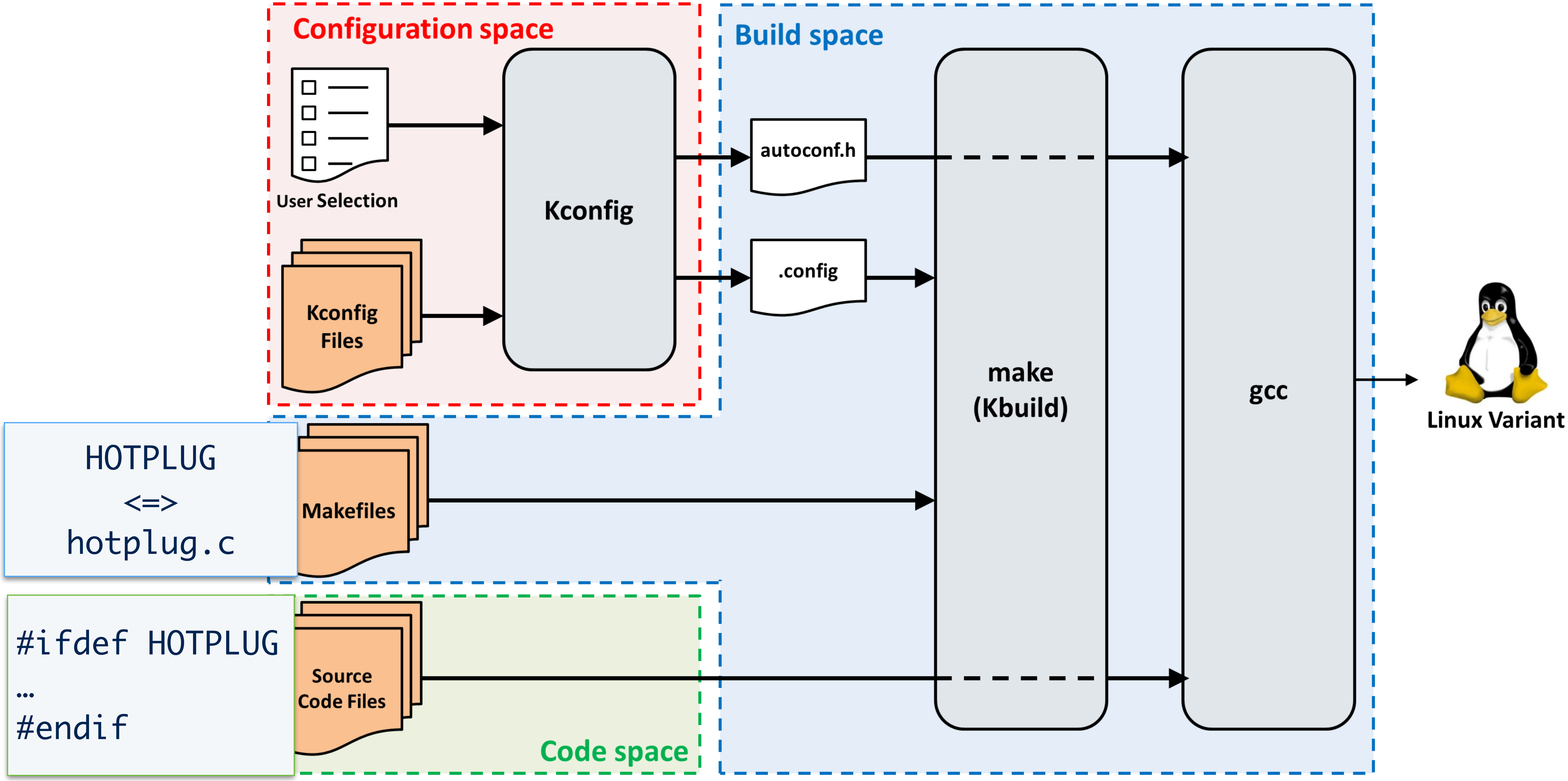


Linux Kernel:
Variability using Build Systems & CPP



```
#ifdef HOTPLUG
…
#endif
```

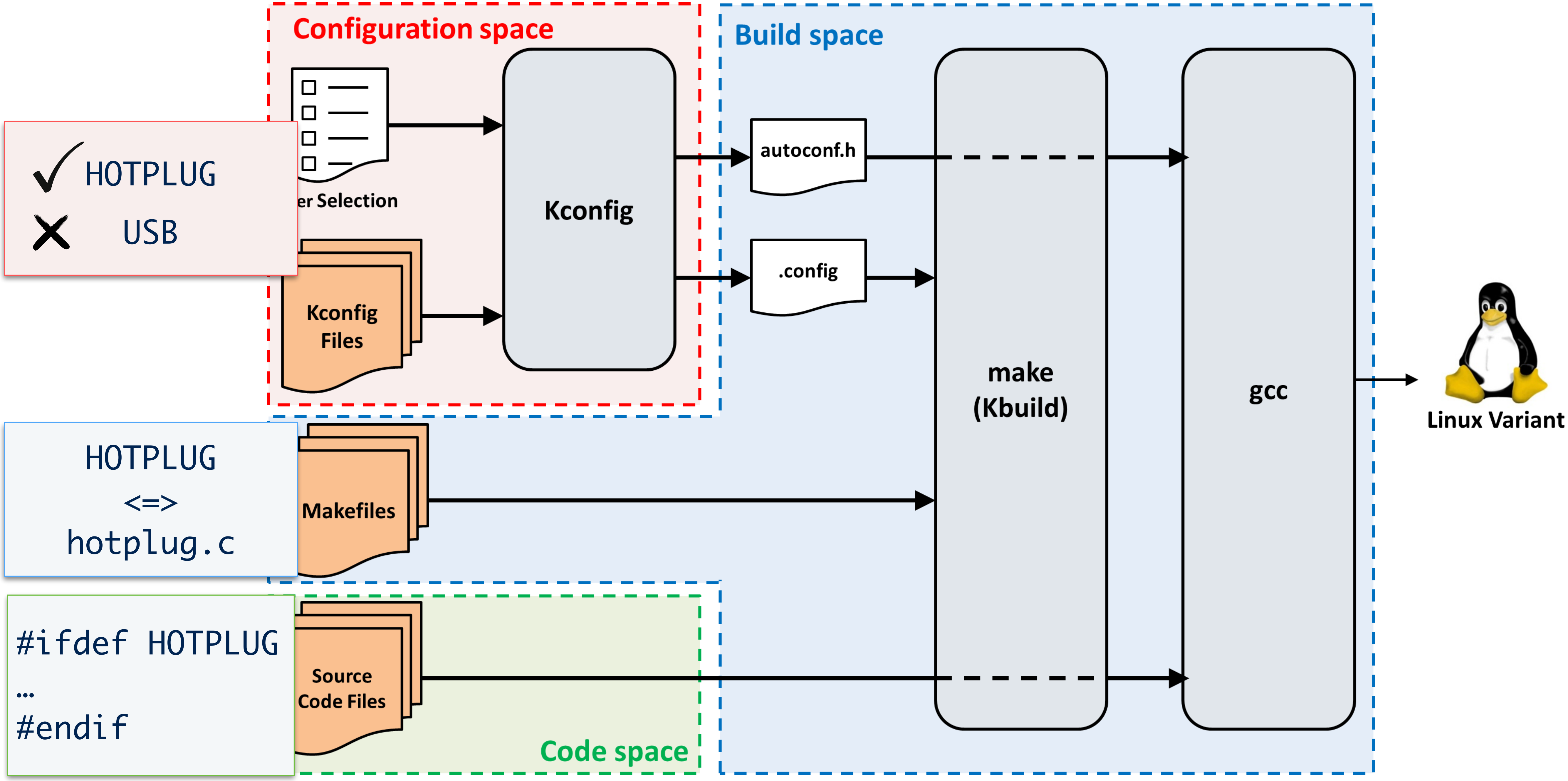
Linux Kernel:
Variability using Build Systems & CPP



```
HOTPLUG
<=>
hotplug.c
```

```
#ifdef HOTPLUG
…
#endif
```

Linux Kernel:
Variability using Build Systems & CPP



✓ HOTPLUG

✗ USB

HOTPLUG
<=>
hotplug.c

```
#ifdef HOTPLUG
…
#endif
```

Variability Using Preprocessors

+ Easy to use, well-known

+ Compile-time customization removes unnecessary code

+ Supports arbitrary levels of granularity

- No separation of concerns (lots of scattering & tangling)

- Can be used in an undisciplined fashion

- Prone to simple (syntactic) errors

Variability Using Preprocessors

+ Easy to use, well-known

+ Compile-time customization removes unnecessary code

+ Supports arbitrary levels of granularity

- No separation of concerns (lots of scattering & tangling)

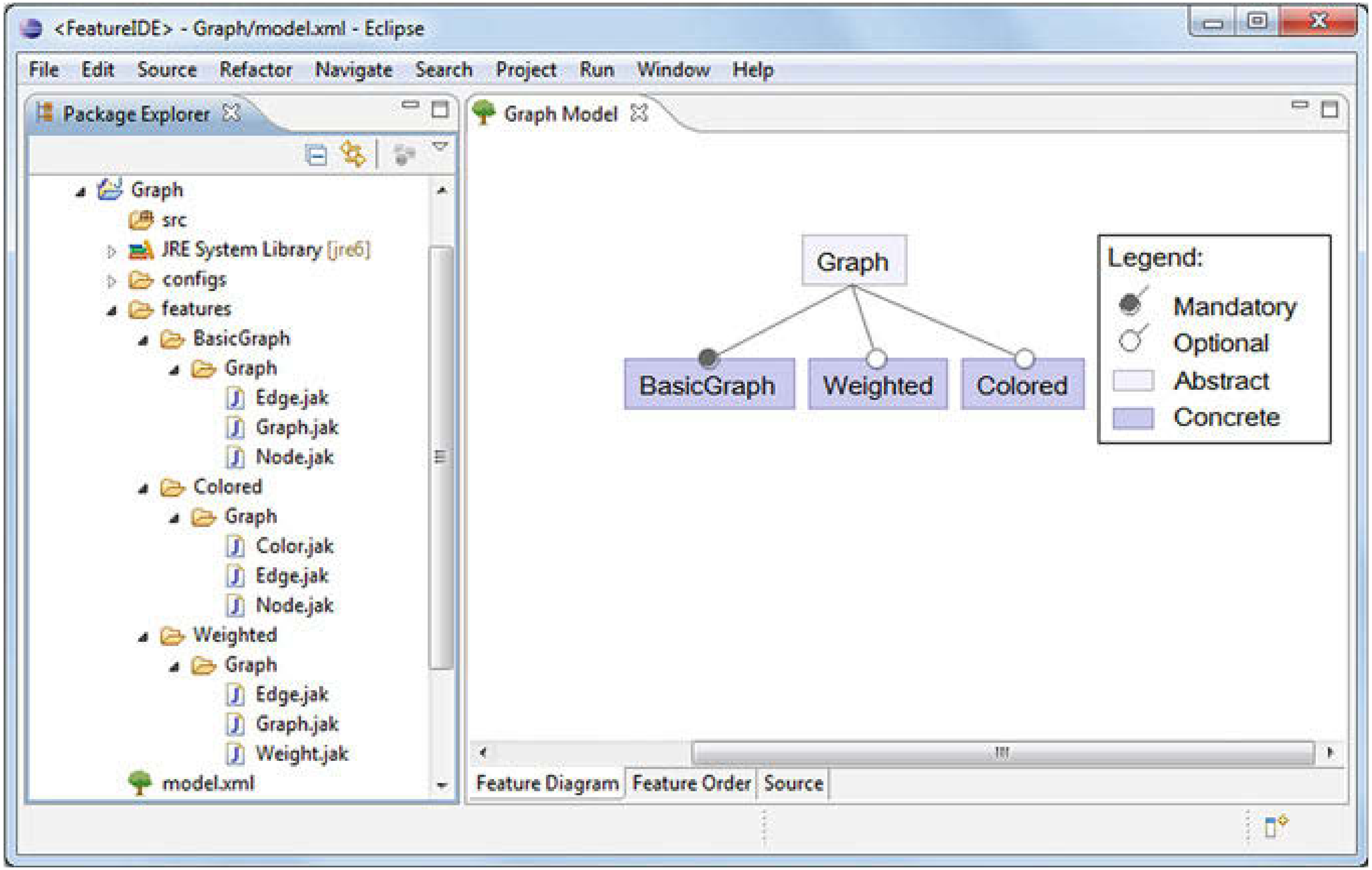
- Can be used in an undisciplined fashion

- Prone to simple (syntactic) errors

Annotation
Compile-time

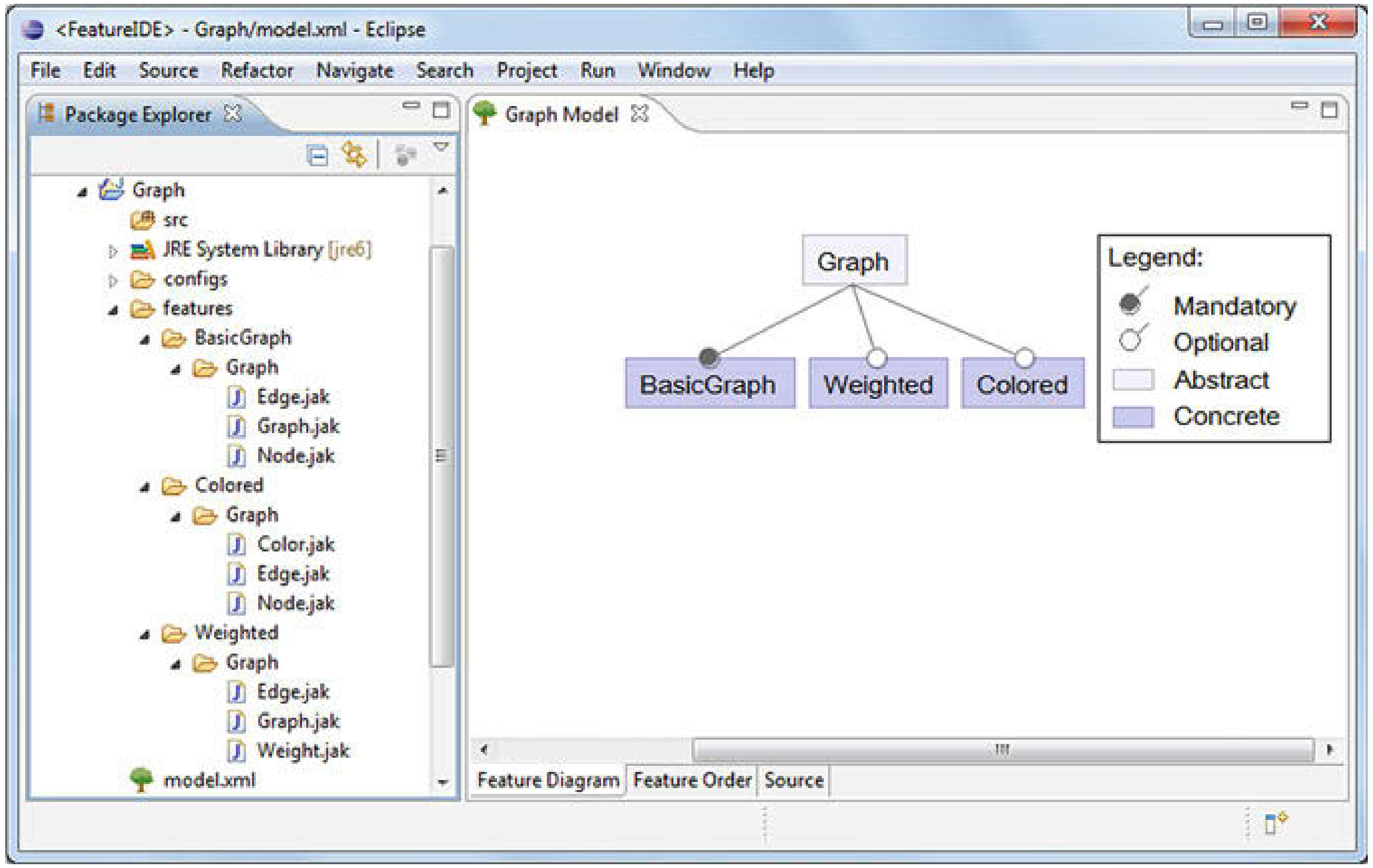
Feature-oriented Programming

Variability Using Feature-oriented Programming



See http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/

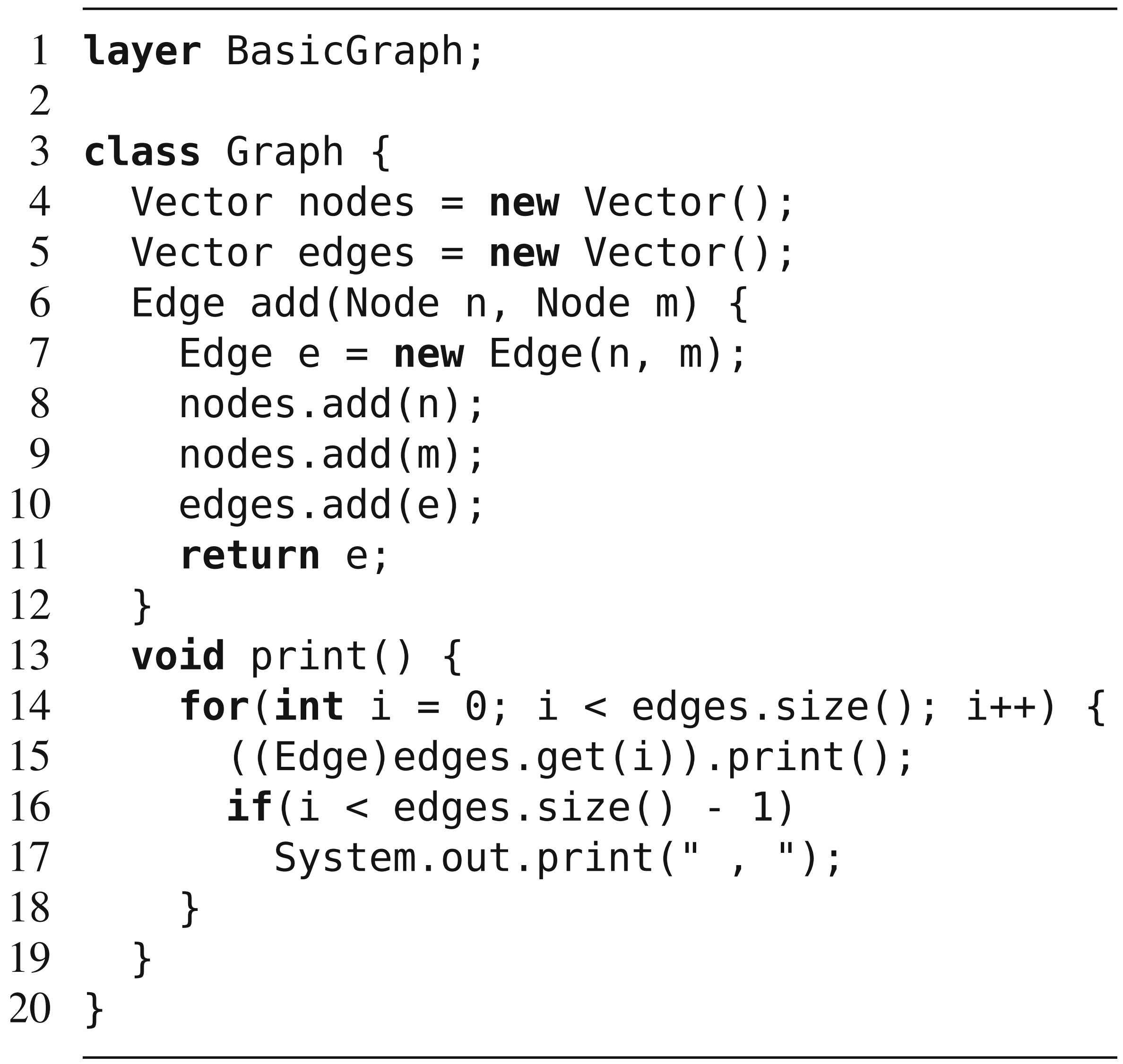
Variability Using Feature-oriented Programming



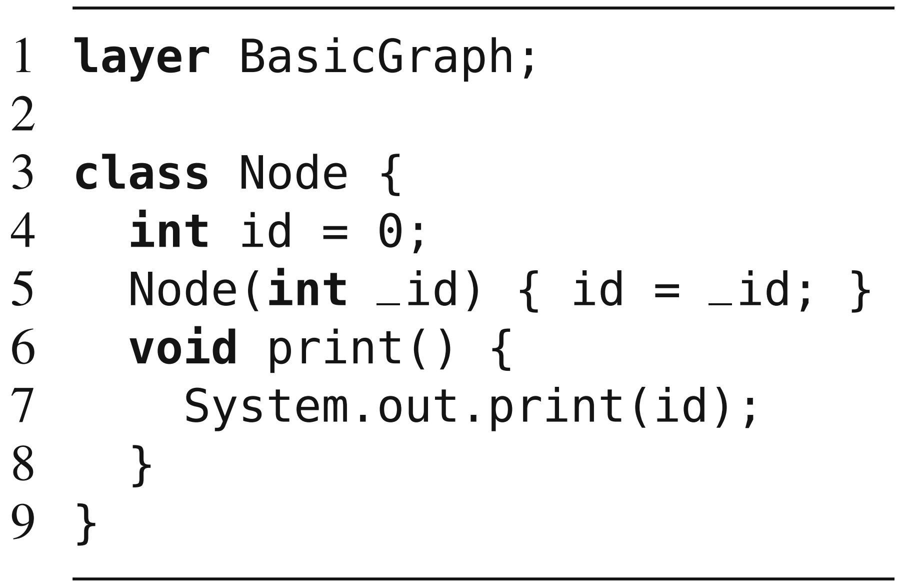
WeightedGraph = Weighted • BasicGraph

ColoredWeightedGraph = Colored • Weighted • BasicGraph

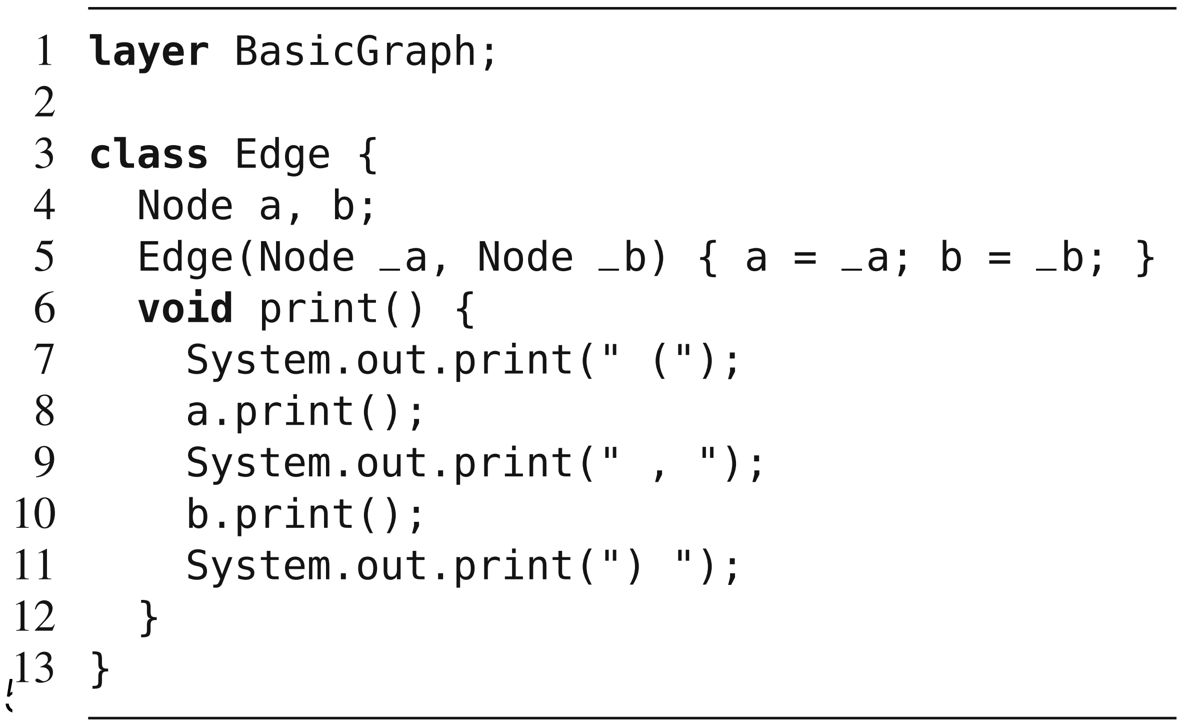
Feature-oriented Graph Implementation



```
layer BasicGraph;

class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```

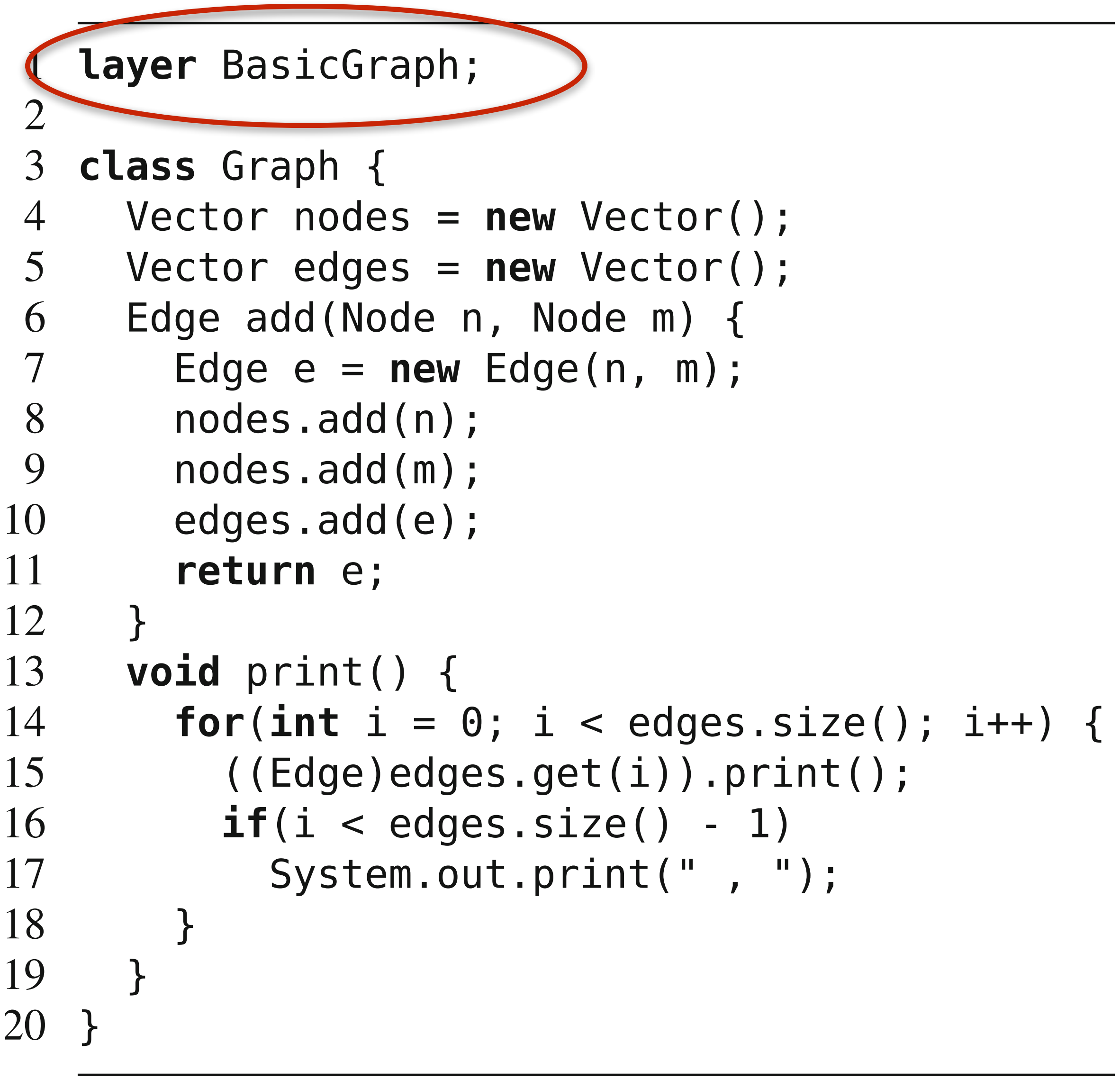


```
layer BasicGraph;

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

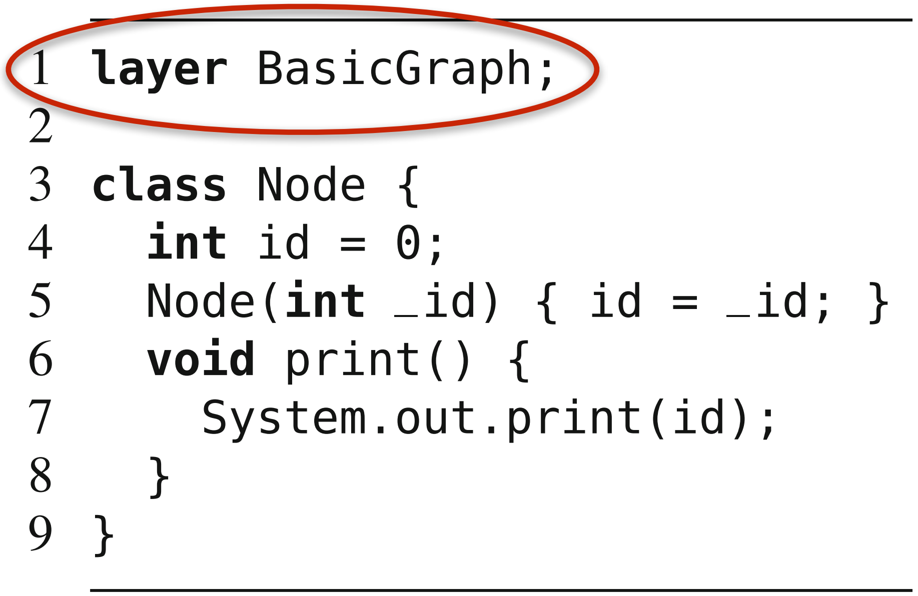


```
layer BasicGraph;

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

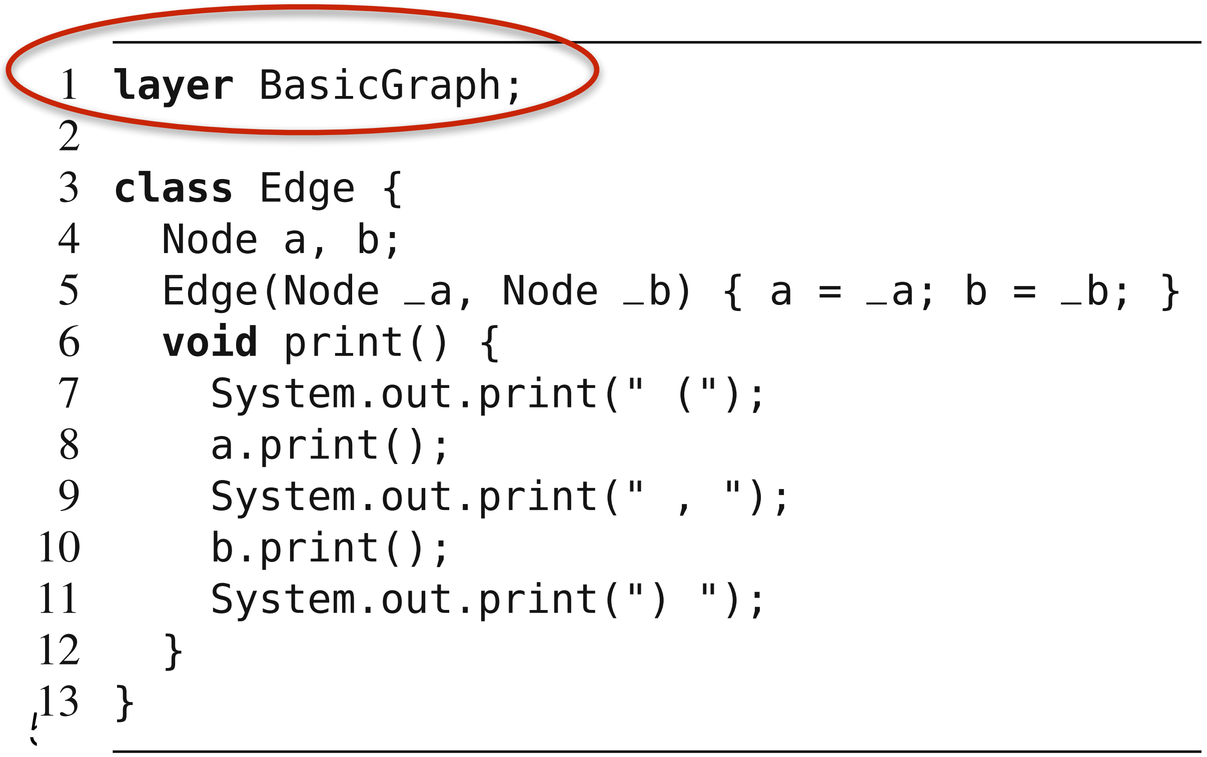
Feature-oriented Graph Implementation



```
layer BasicGraph;

class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```



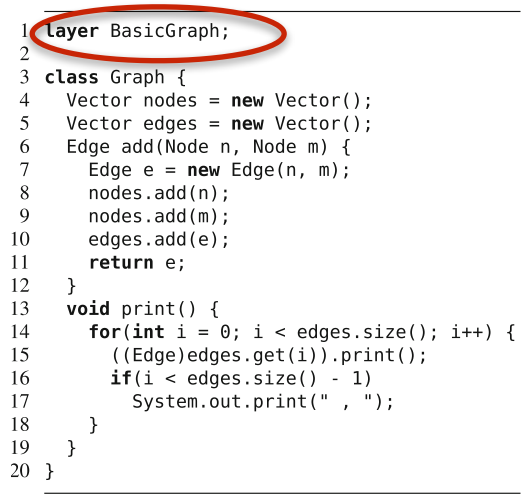
```
layer BasicGraph;

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



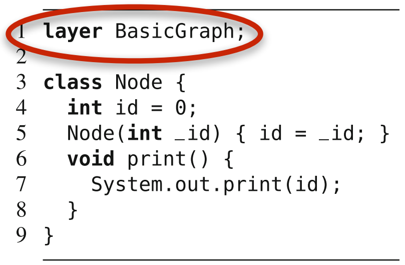
```
layer BasicGraph;

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

same feature

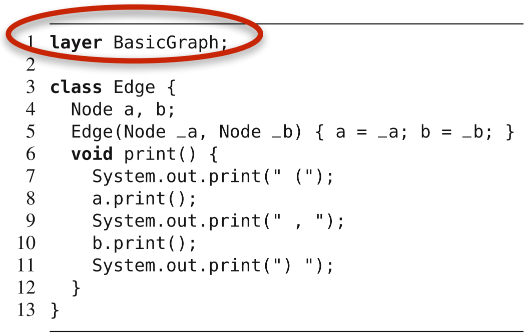
Feature-oriented Graph Implementation



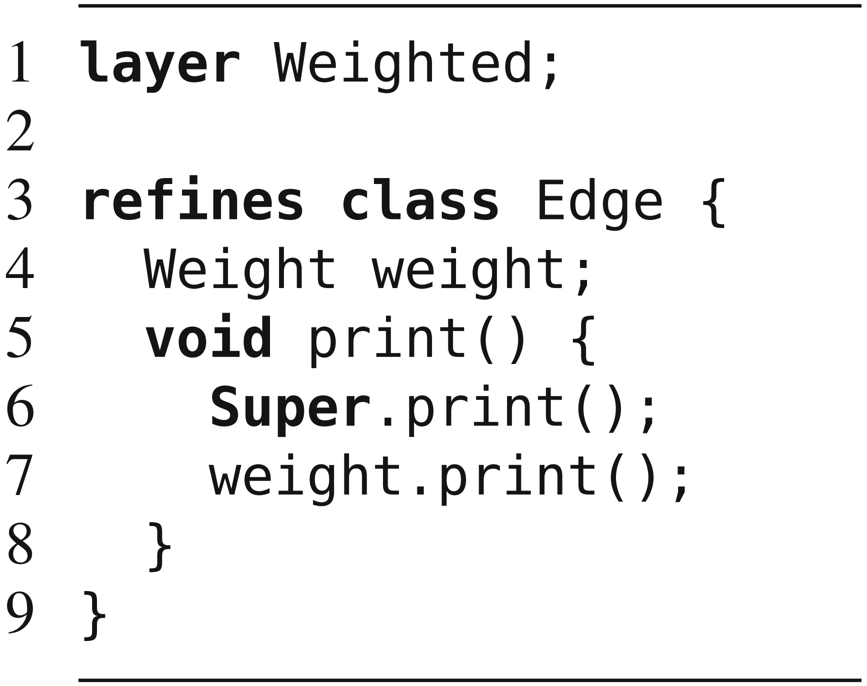
```
layer BasicGraph;

class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```



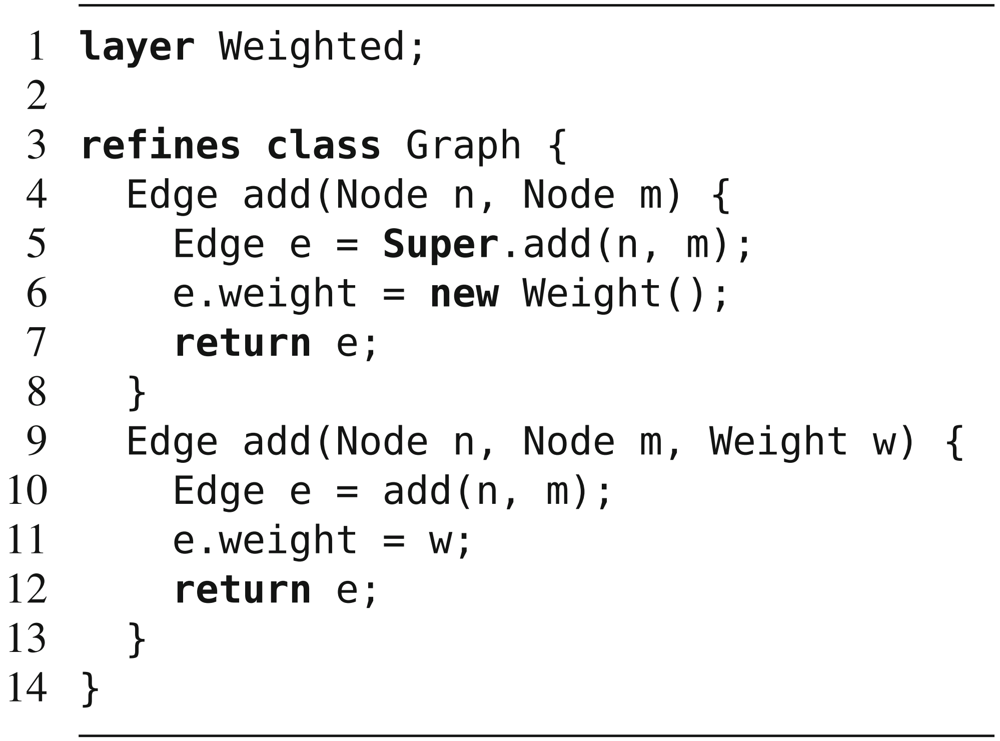
```
layer BasicGraph;

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



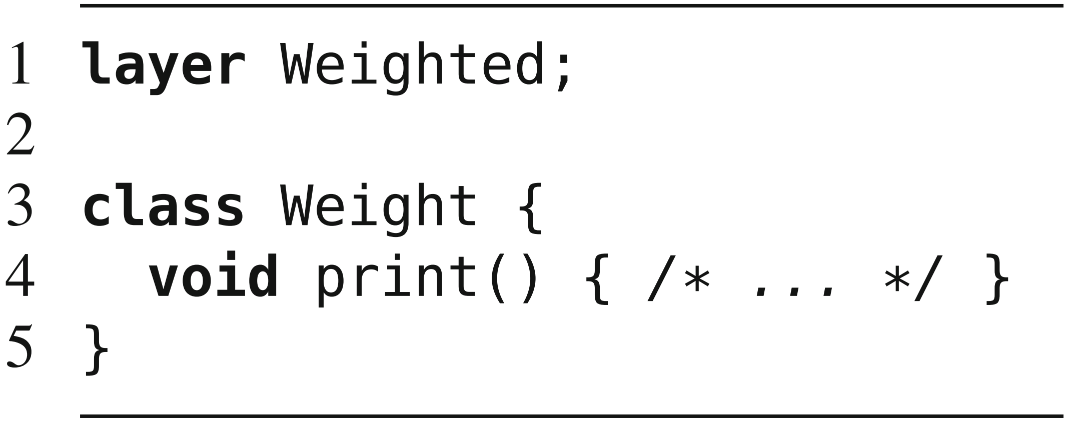
```
layer BasicGraph;

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



```
layer Weighted;

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

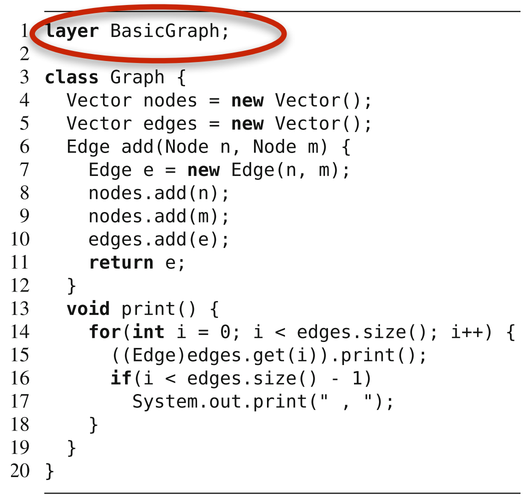


```
layer Weighted;

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

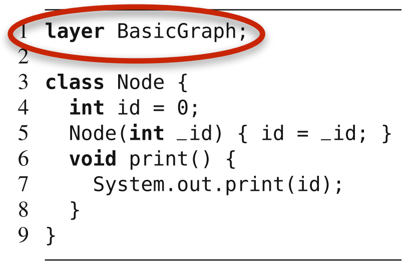


```
layer Weighted;

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

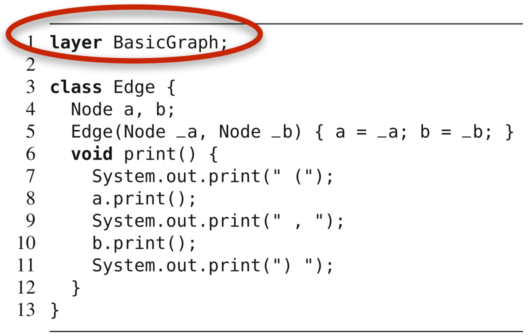
Feature-oriented Graph Implementation



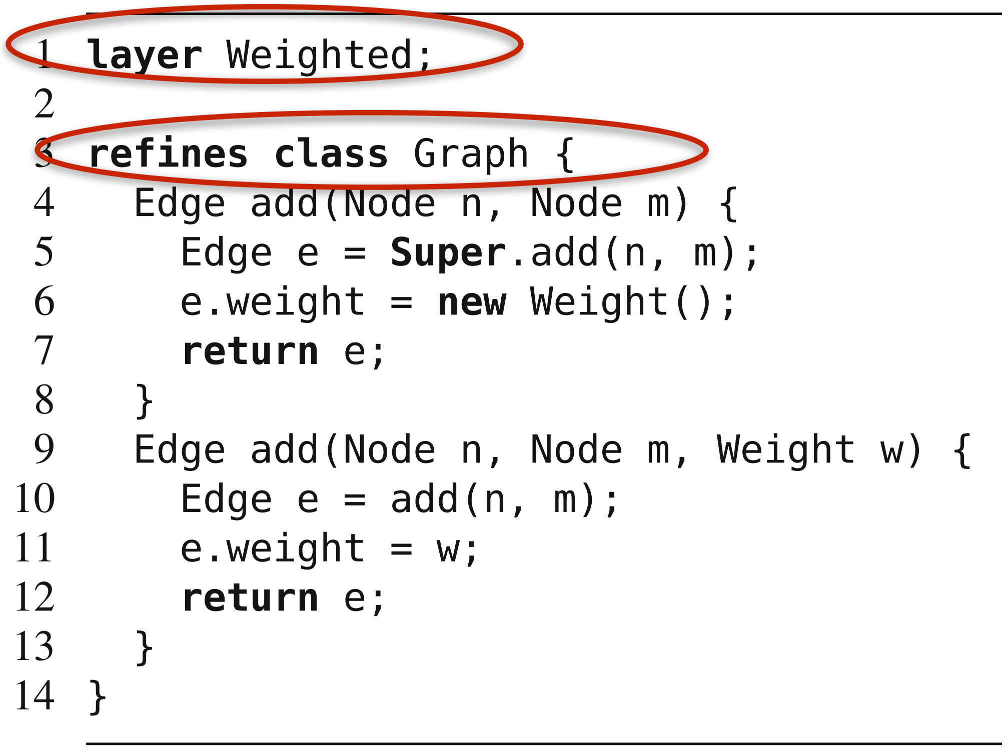
```
layer BasicGraph;

class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```



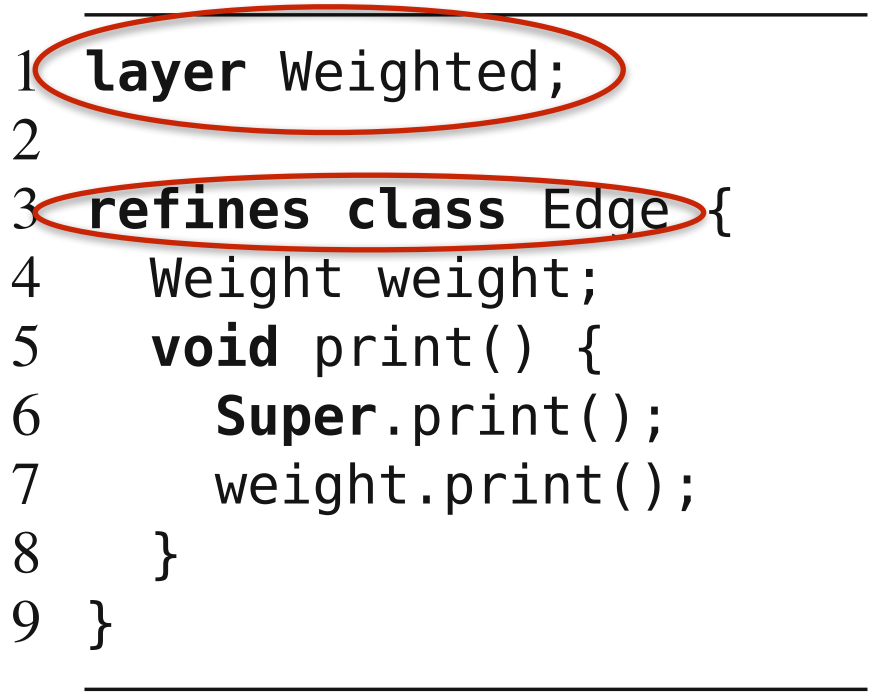
```
layer BasicGraph;

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



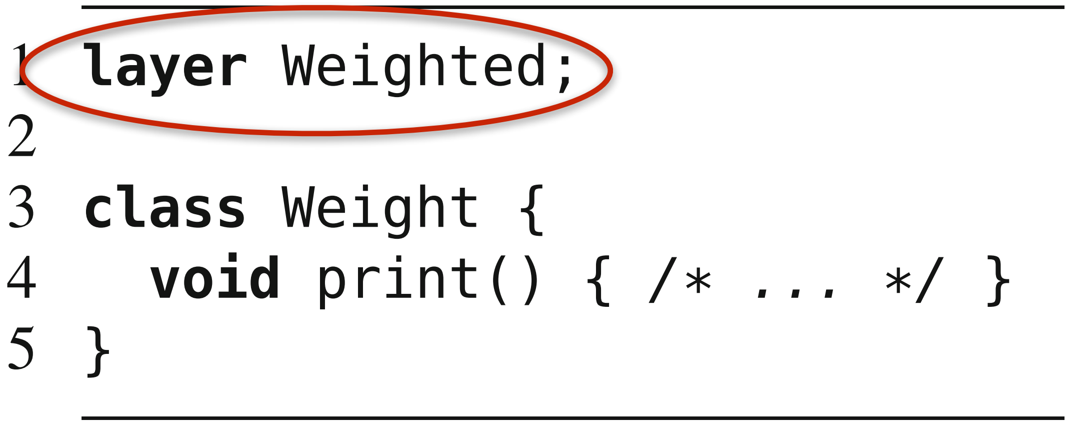
```
layer BasicGraph;

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



```
layer Weighted;

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

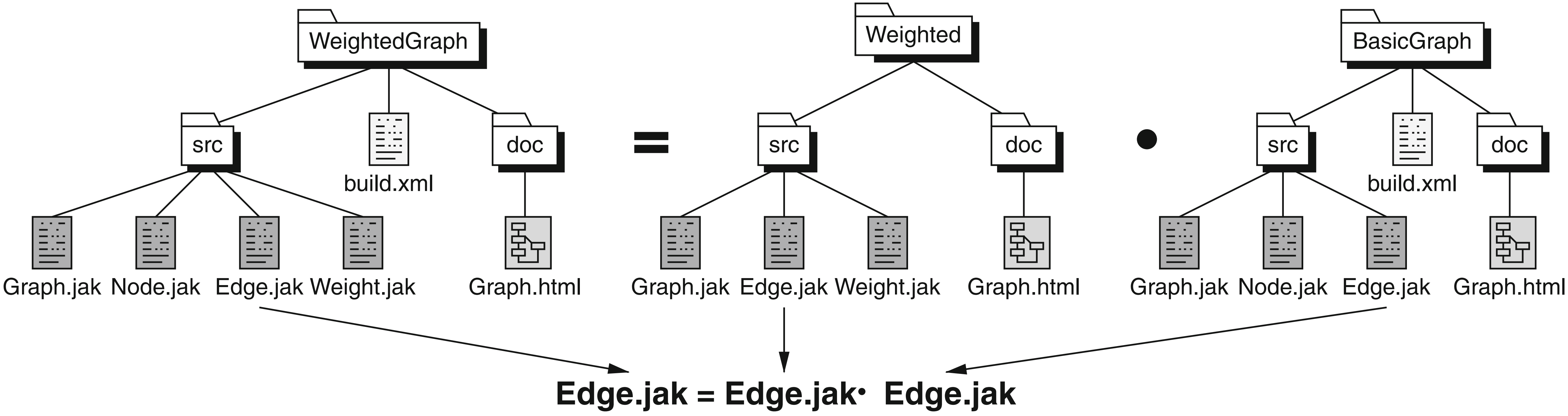


```
layer Weighted;

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



```
layer Weighted;

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

Feature-oriented Graph Implementation



Composed WeightedGraph

```java
class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n);
    nodes.add(m);
    edges.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    Edge e = add(n, m);
    e.weight = w;
    return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
      if(i < edges.size() - 1)
        System.out.print(" , ");
    }
  }
}
```

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

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

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

Variability Using Feature-Oriented Programming

+ Easy-to-use language mechanism, requiring minimal language extensions

+ Compile-time customization of source code

+ Direct feature traceability from a feature to its implementation

- Requires composition tools

- Granularity at level of methods

- Only academic tools so far, little experience in practice

Variability Using Feature-Oriented Programming

+ Easy-to-use language mechanism, requiring minimal language extensions

+ Compile-time customization of source code

+ Direct feature traceability from a feature to its implementation

- Requires composition tools

- Granularity at level of methods

- Only academic tools so far, little experience in practice

Composition
Compile-time

Advanced/Research Topics

Detecting Inconsistencies

```
PCCARD => HOTPLUG
PCMCIA => PCCARD
```

Feature Model

```
#ifdef HOTPLUG
//B1
#else
//B2
#endif
```

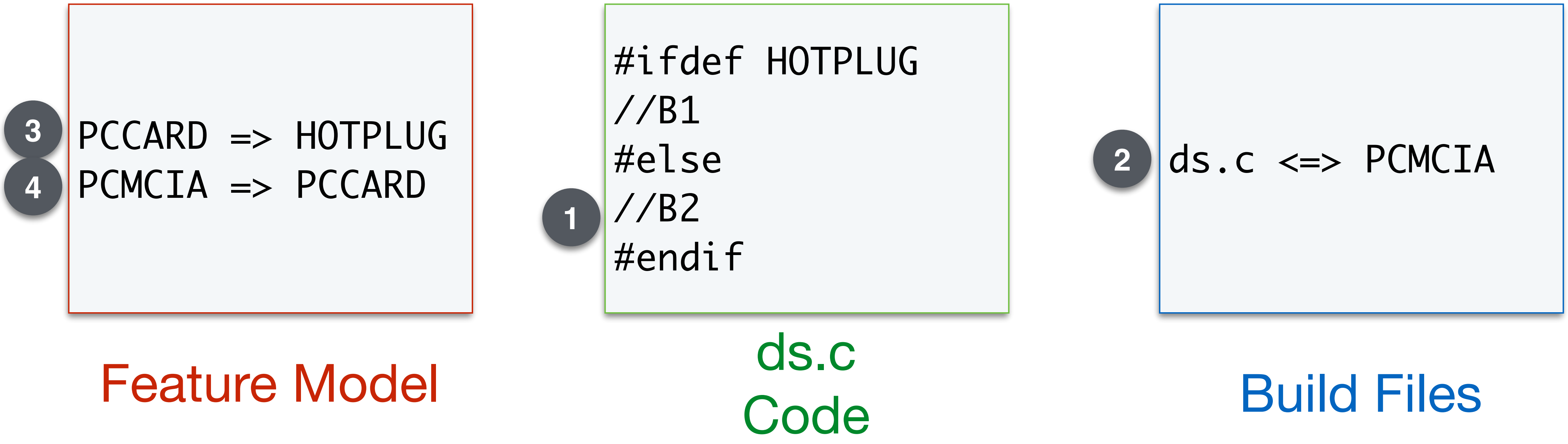
ds.c
Code

```
ds.c <=> PCMCIA
```

Build Files

[Nadi & Holt: CSMR ’12]

Detecting Inconsistencies



```
3 PCCARD => HOTPLUG
4 PCMCIA => PCCARD
```

Feature Model

```
  #ifdef HOTPLUG
  //B1
  #else
1 //B2
  #endif
```

ds.c
Code

```
2 ds.c <=> PCMCIA
```

Build Files

```
  B2 ∧
1  B2 <=> !HOTPLUG ∧
2  PCMCIA ∧
3  PCMCIA => PCCARD ∧
4  PCCARD => HOTPLUG
```

[Nadi & Holt: CSMR ’12]

Detecting Inconsistencies

```
3 PCCARD => HOTPLUG
4 PCMCIA => PCCARD
```

Feature Model

```
  #ifdef HOTPLUG
  //B1
  #else
1 //B2
  #endif
```

ds.c
Code

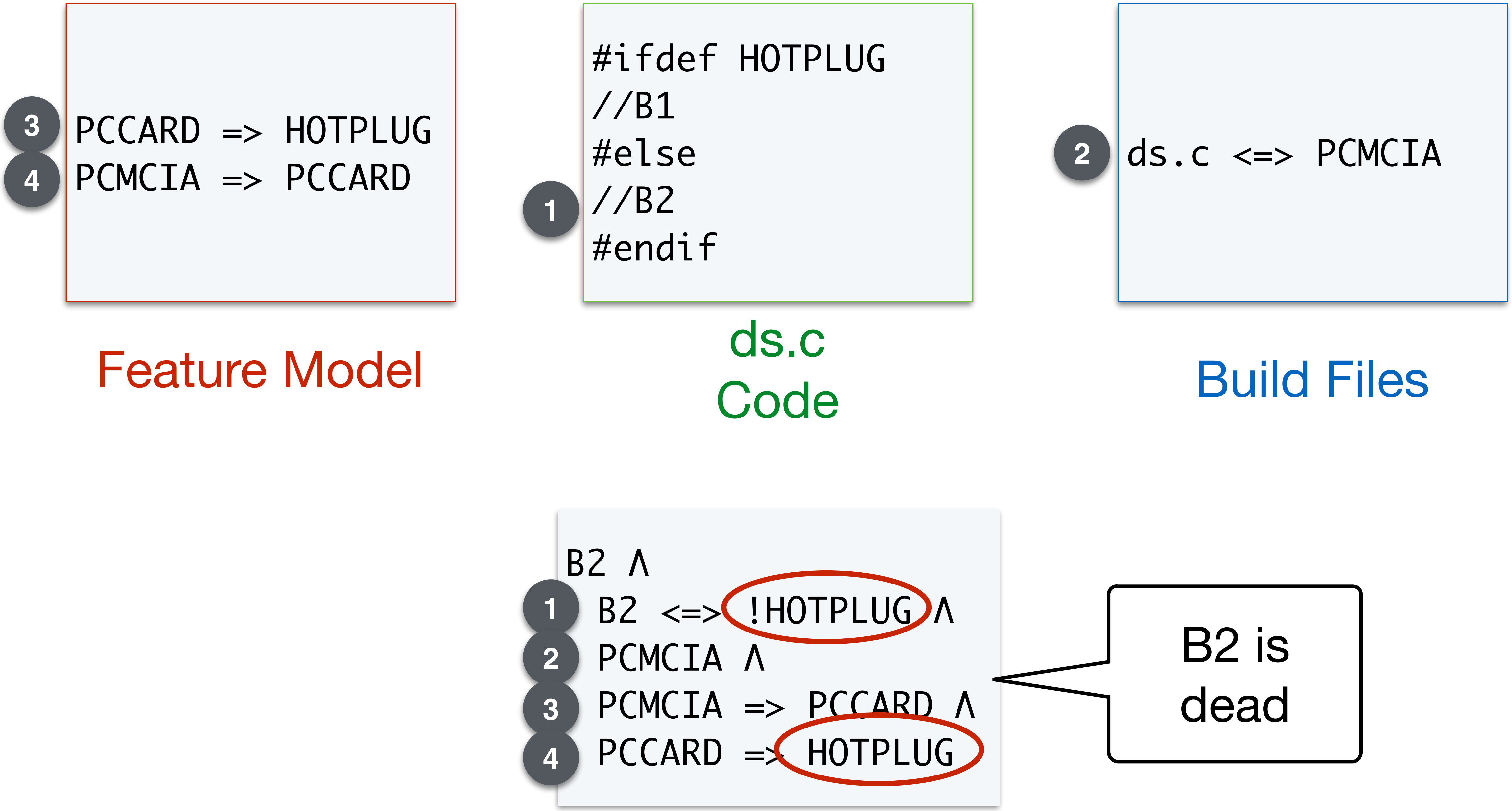
```
2 ds.c <=> PCMCIA
```

Build Files

```
B2 ∧
1 B2 <=> !HOTPLUG ∧
2 PCMCIA ∧
3 PCMCIA => PCCARD ∧
4 PCCARD => HOTPLUG
```

[Nadi & Holt: CSMR ’12]

Detecting Inconsistencies



```
3 PCCARD => HOTPLUG
4 PCMCIA => PCCARD
```

Feature Model

```
  #ifdef HOTPLUG
  //B1
  #else
1 //B2
  #endif
```

ds.c
Code

```
2 ds.c <=> PCMCIA
```

Build Files

```
  B2 ∧
1   B2 <=> !HOTPLUG ∧
2   PCMCIA ∧
3   PCMCIA => PCCARD ∧
4   PCCARD => HOTPLUG
```

B2 is dead

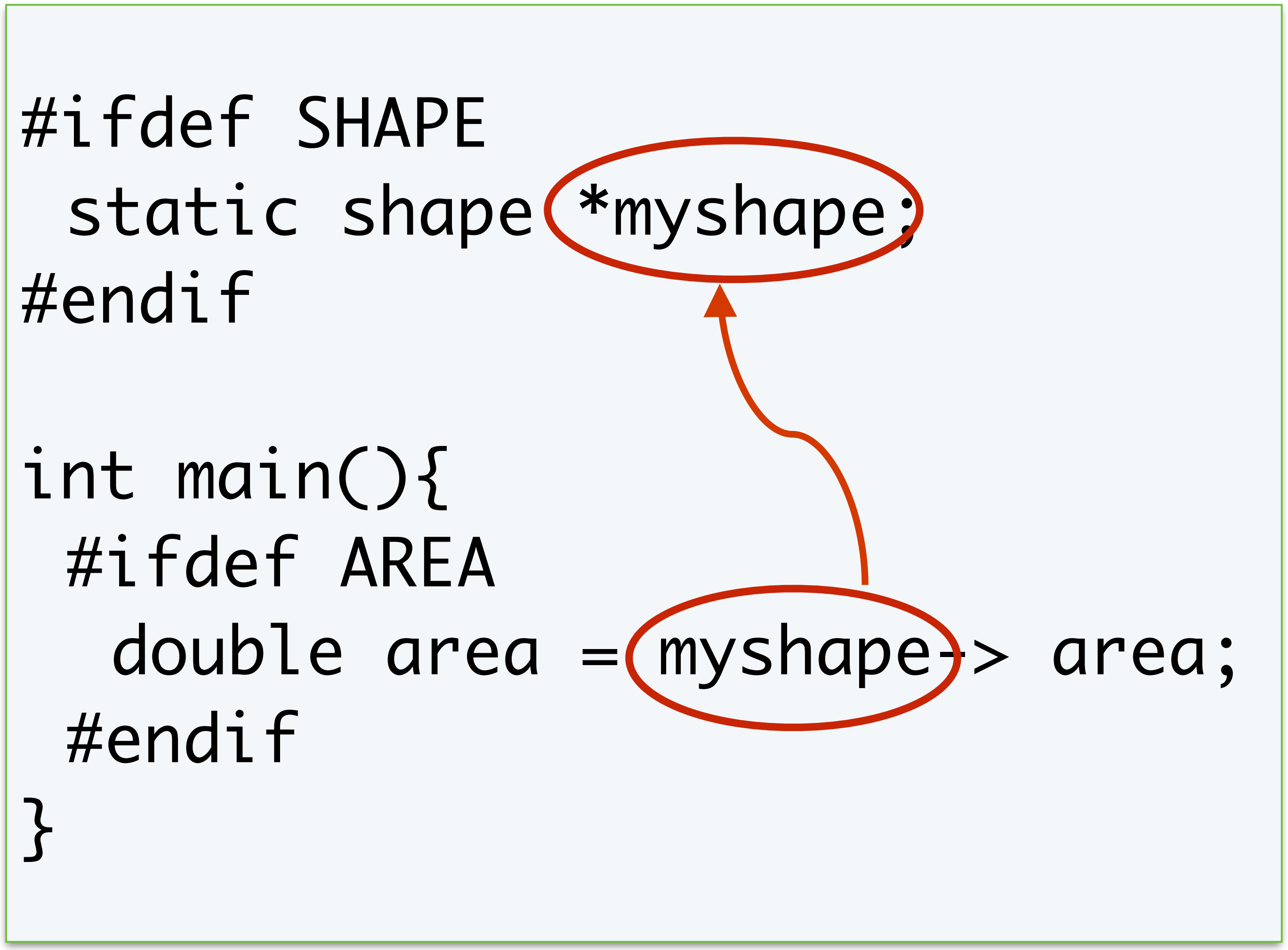
[Nadi & Holt: CSMR ’12]

Detecting Configuration Constraints

```
#ifdef SHAPE
 static shape *myshape;
#endif

int main(){
 #ifdef AREA
  double area = myshape-> area;
 #endif
}
```

[Nadi et al.: ICSE ‘14]

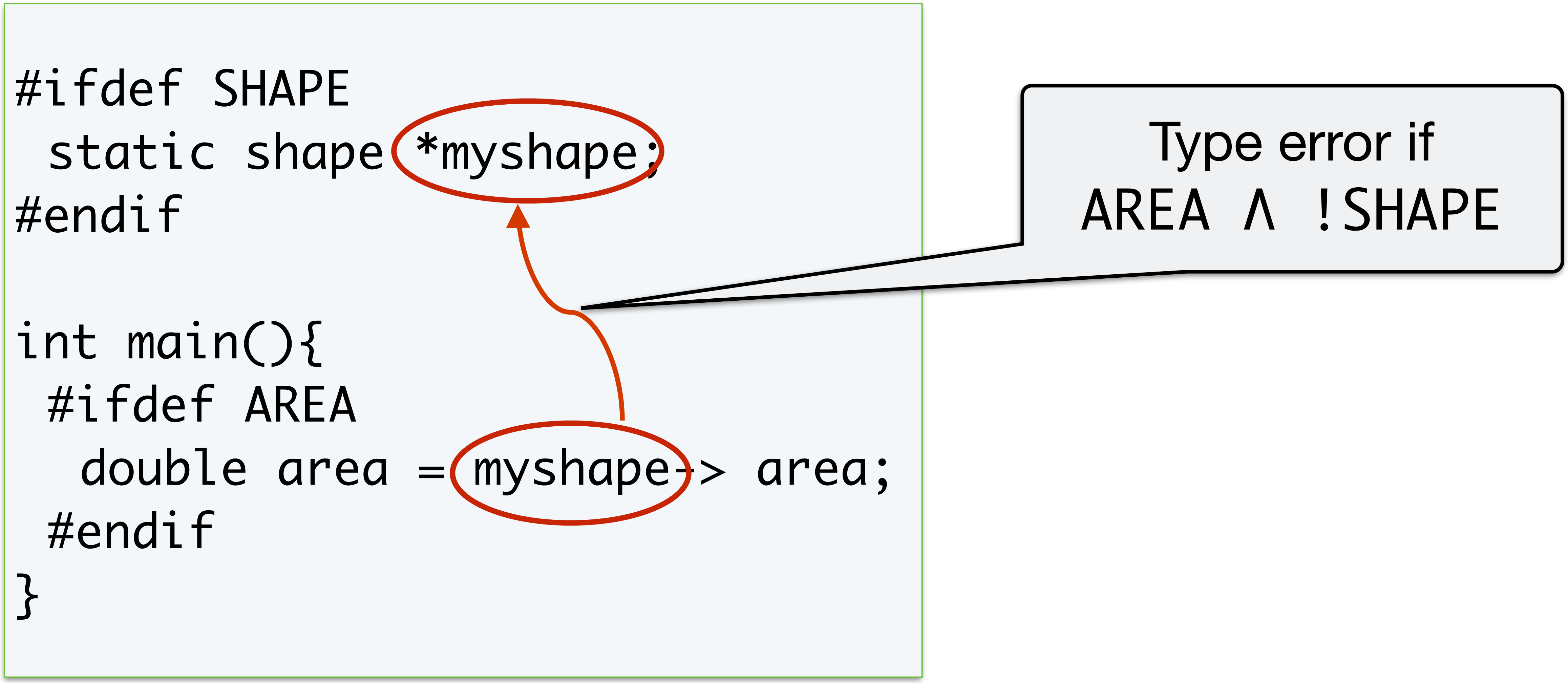
Detecting Configuration Constraints



```
#ifdef SHAPE
 static shape *myshape;
#endif

int main(){
 #ifdef AREA
  double area = myshape-> area;
 #endif
}
```

[Nadi et al.: ICSE ‘14]

Detecting Configuration Constraints

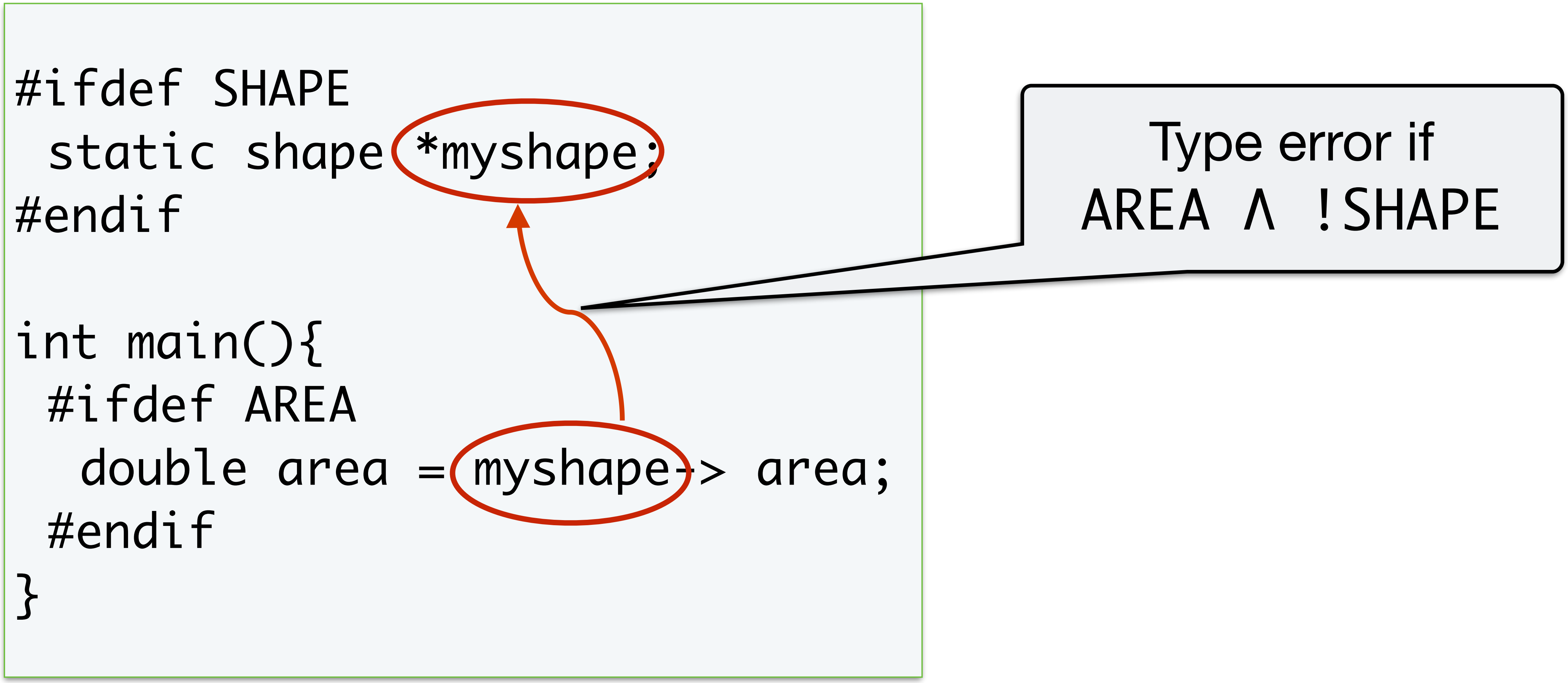


```
#ifdef SHAPE
 static shape *myshape;
#endif

int main(){
 #ifdef AREA
  double area = myshape-> area;
 #endif
}
```

Type error if
AREA ∧ !SHAPE

[Nadi et al.: ICSE ‘14]

Detecting Configuration Constraints



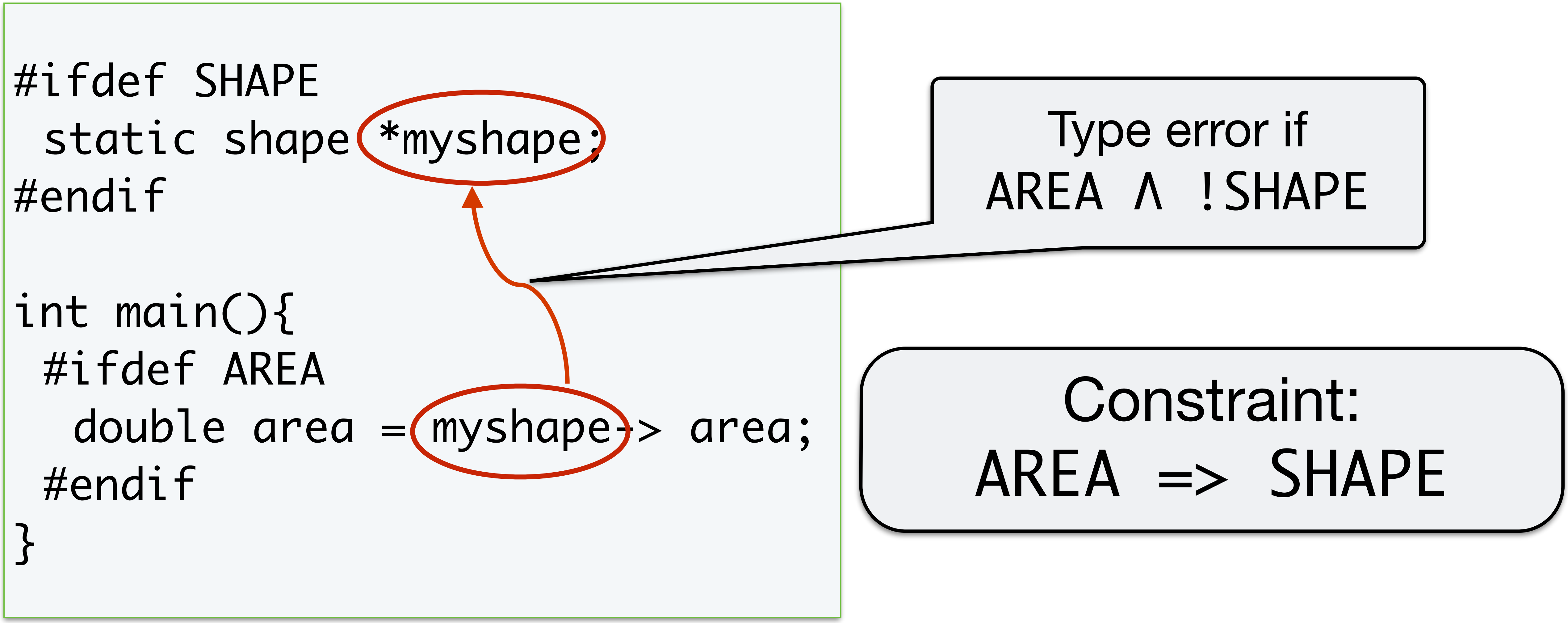
```
#ifdef SHAPE
 static shape *myshape;
#endif

int main(){
 #ifdef AREA
  double area = myshape-> area;
 #endif
}
```

Type error if
AREA ∧ !SHAPE

Feature model should enforce !(AREA ∧ !SHAPE)

[Nadi et al.: ICSE '14]

Detecting Configuration Constraints



```
#ifdef SHAPE
 static shape *myshape;
#endif

int main(){
 #ifdef AREA
  double area = myshape-> area;
 #endif
}
```

Type error if
AREA ∧ !SHAPE

Constraint:
AREA => SHAPE

Feature model should enforce !(AREA ∧ !SHAPE)

[Nadi et al.: ICSE '14]

Detecting Configuration Constraints (Underlying Analysis)

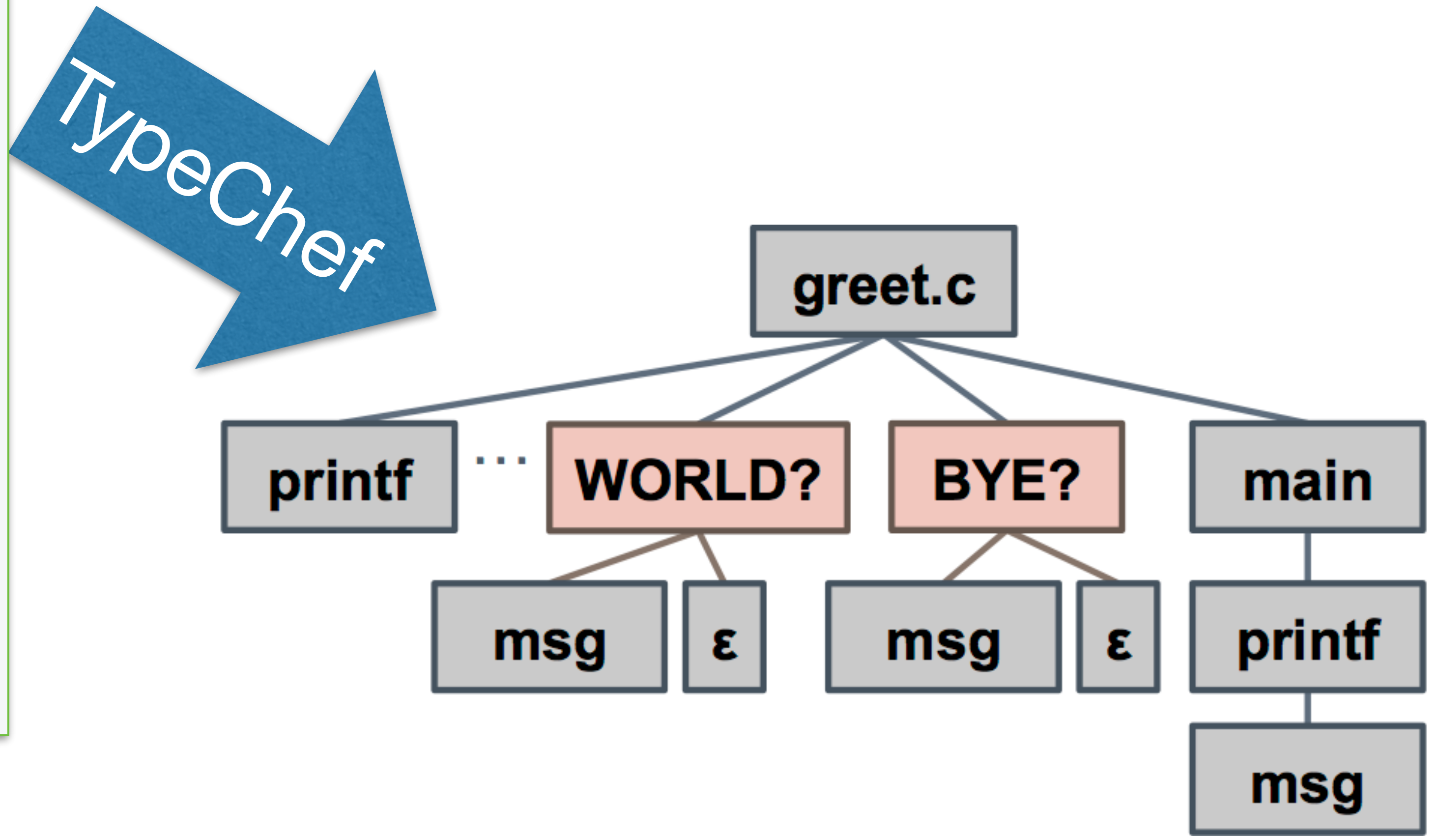
```
#include <stdio.h>

#ifdef WORLD
char * msg = “Hello World”;
#endif

#ifdef BYE
char * msg = “Bye bye!\n”;
#endif

main() {
 print(msg);
}
```

[Kästner et al.: OOPSLA ‘11]

Detecting Configuration Constraints (Underlying Analysis)

```
#include <stdio.h>

#ifdef WORLD
char * msg = “Hello World”;
#endif

#ifdef BYE
char * msg = “Bye bye!\n”;
#endif

main() {
 print(msg);
}
```

https://github.com/ckaestne/TypeChef



AST with variability information

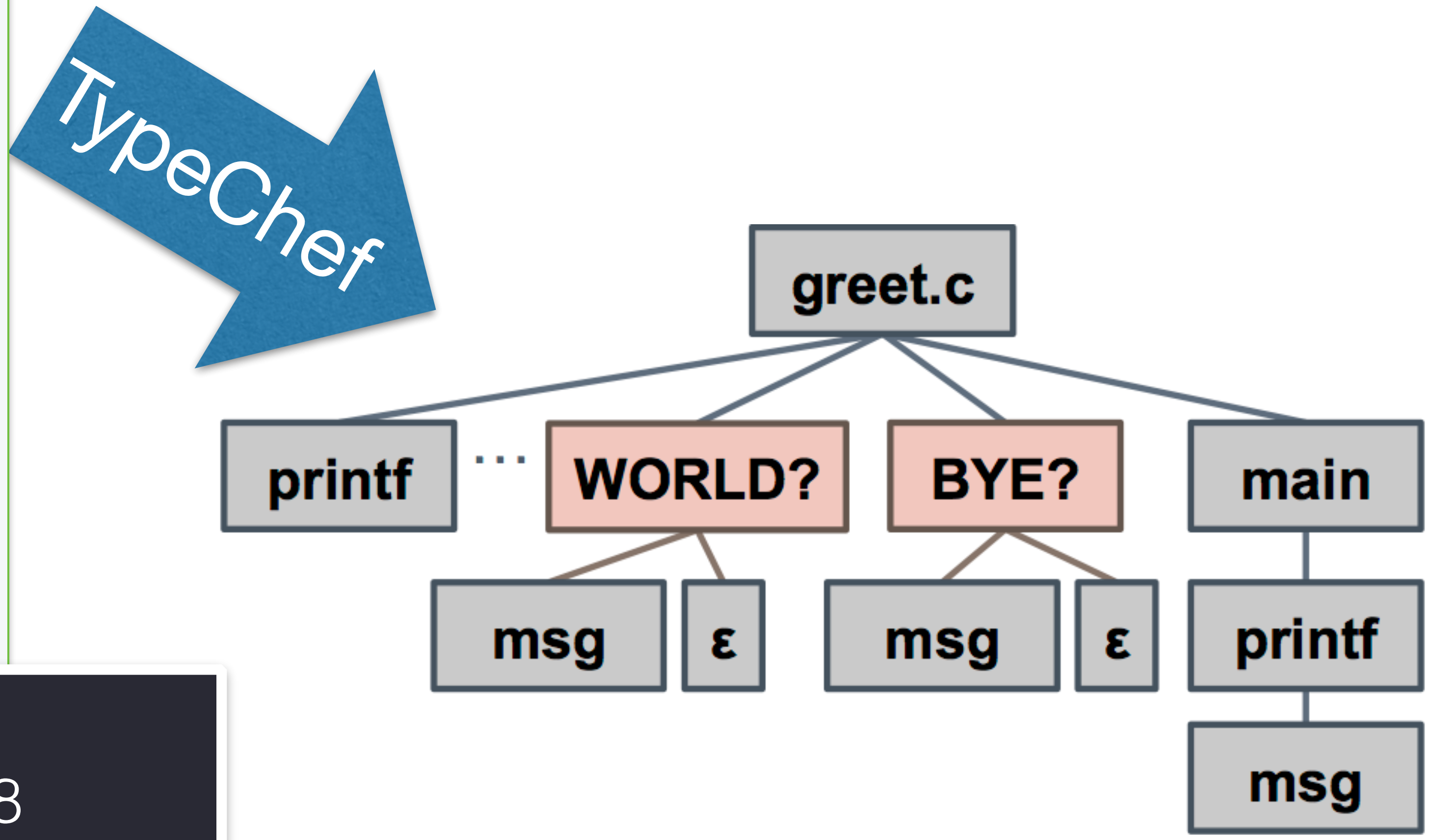
[Kästner et al.: OOPSLA ‘11]

Detecting Configuration Constraints (Underlying Analysis)

```
#include <stdio.h>

#ifdef WORLD
char * msg = “Hello World”;
#endif

#ifdef BYE
char * msg = “Bye bye!\n”;
#endif

main() {
 print(msg);
}
```

```
Found 2 type errors:
 - [WORLD & BYE] file greet.c:7:8
        redefinition of msg
 - [!WORLD & !BYE] file greet.c:11:8
        msg undeclared
```

https://github.com/ckaestne/TypeChef



AST with variability information

[Kästner et al.: OOPSLA ‘11]

Feature Interactions

Weather

Smiley

[Nguyen et al., ICSE ’14]

Feature Interactions

Weather

Smiley

Weather Updates:

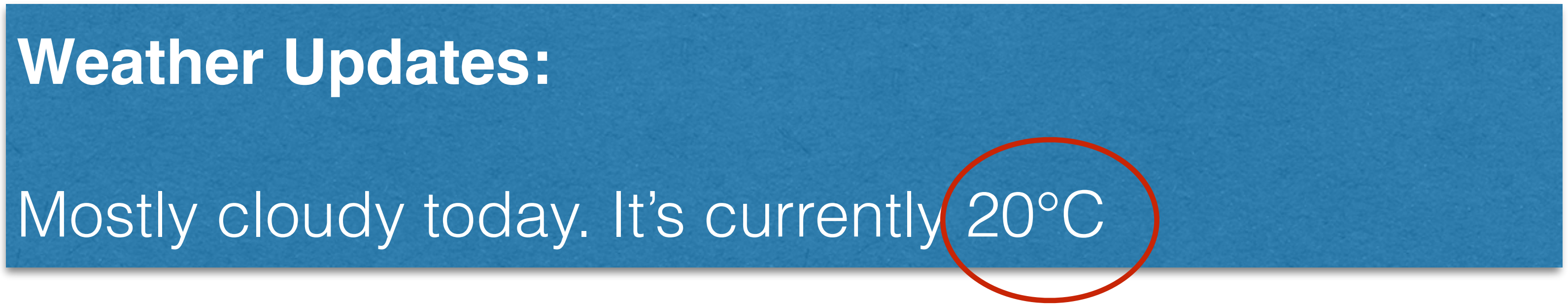
Mostly cloudy today. It’s currently 20°C

[Nguyen et al., ICSE ’14]

Feature Interactions

Weather

Smiley



Weather Updates:

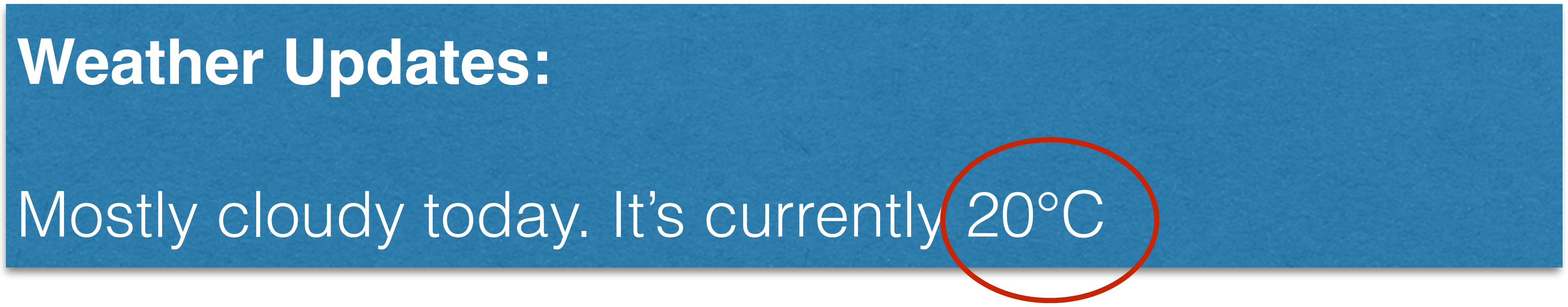
Mostly cloudy today. It’s currently 20°C

[Nguyen et al., ICSE ’14]

Feature Interactions

Weather

Smiley



Weather Updates:

Mostly cloudy today. It’s currently 20°C

Weather

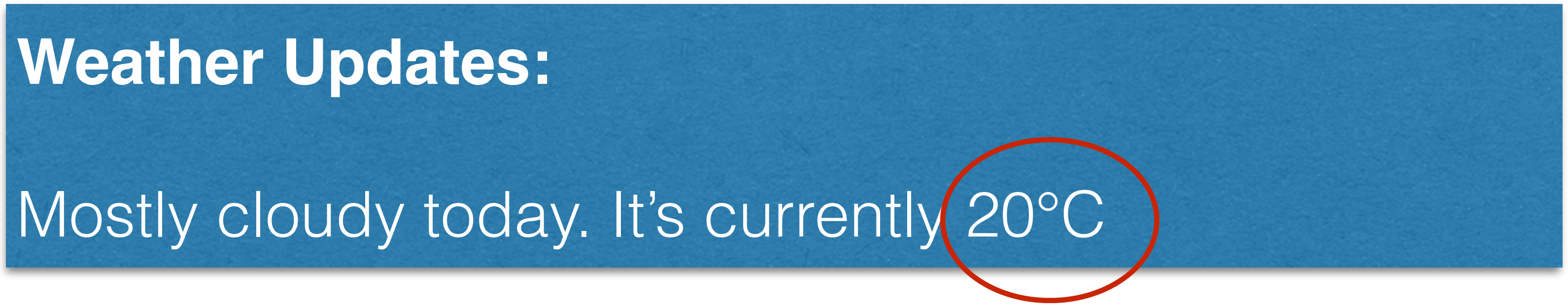
Smiley

[Nguyen et al., ICSE ’14]

Feature Interactions

Weather

Smiley

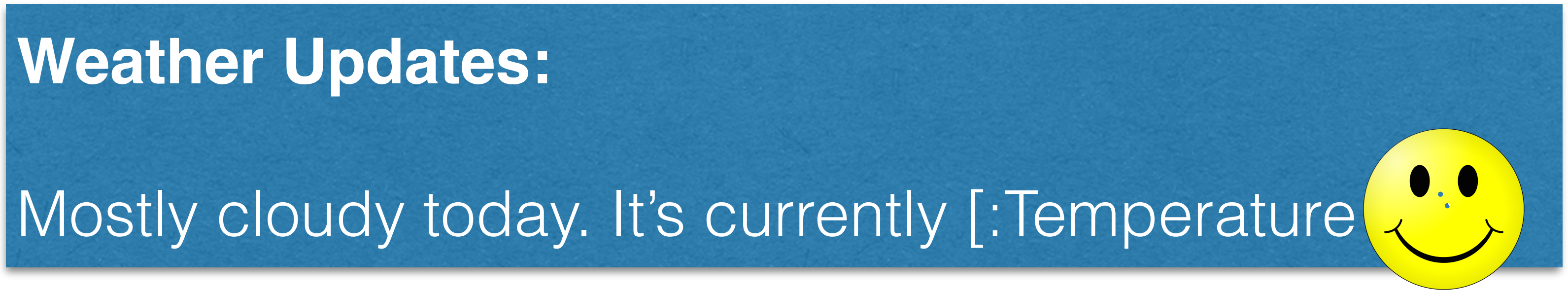


Weather Updates:

Mostly cloudy today. It's currently 20°C

Weather

Smiley

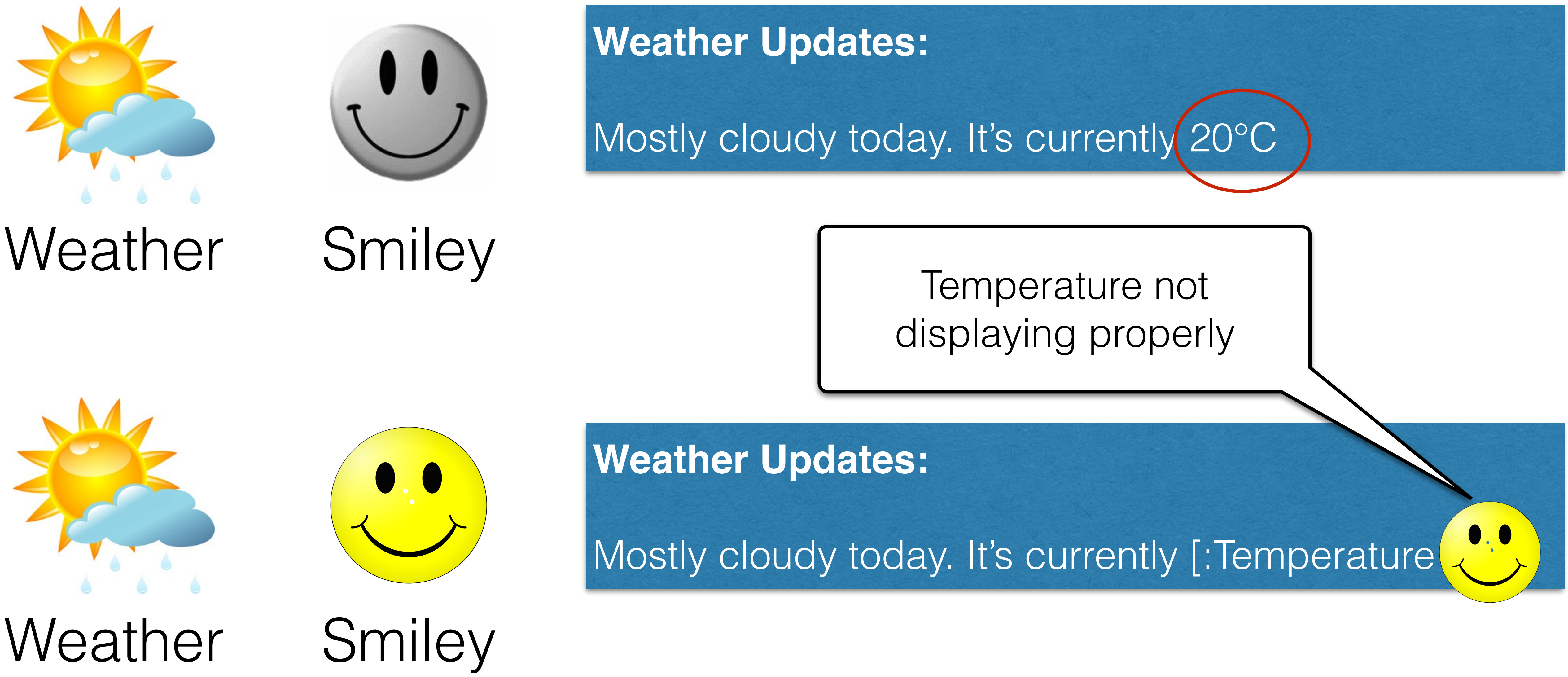


Weather Updates:

Mostly cloudy today. It's currently [:Temperature

[Nguyen et al., ICSE '14]

Feature Interactions



Weather Updates:

Mostly cloudy today. It's currently 20°C

Temperature not displaying properly

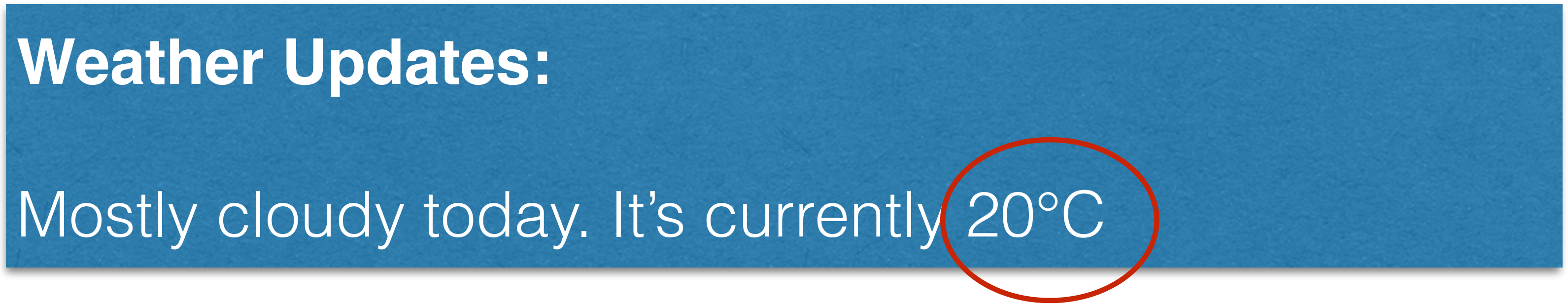
Weather Updates:

Mostly cloudy today. It's currently [:Temperature

[Nguyen et al., ICSE '14]

Feature Interactions

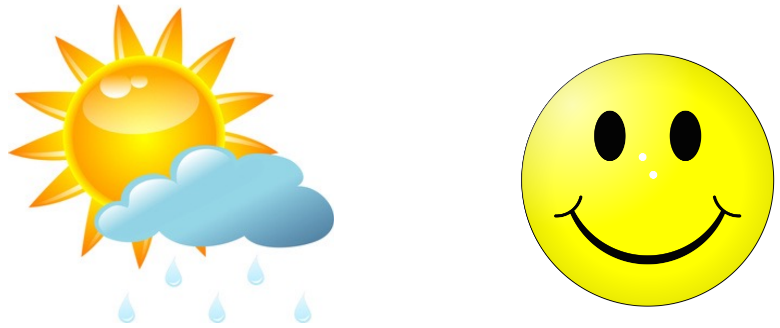
Weather Smiley



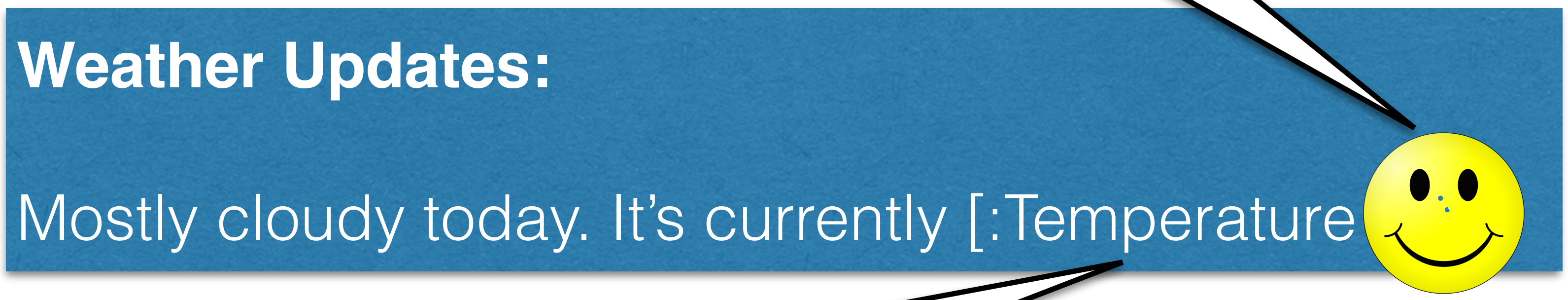
Weather Updates:

Mostly cloudy today. It's currently 20°C

Temperature not displaying properly



Weather Smiley



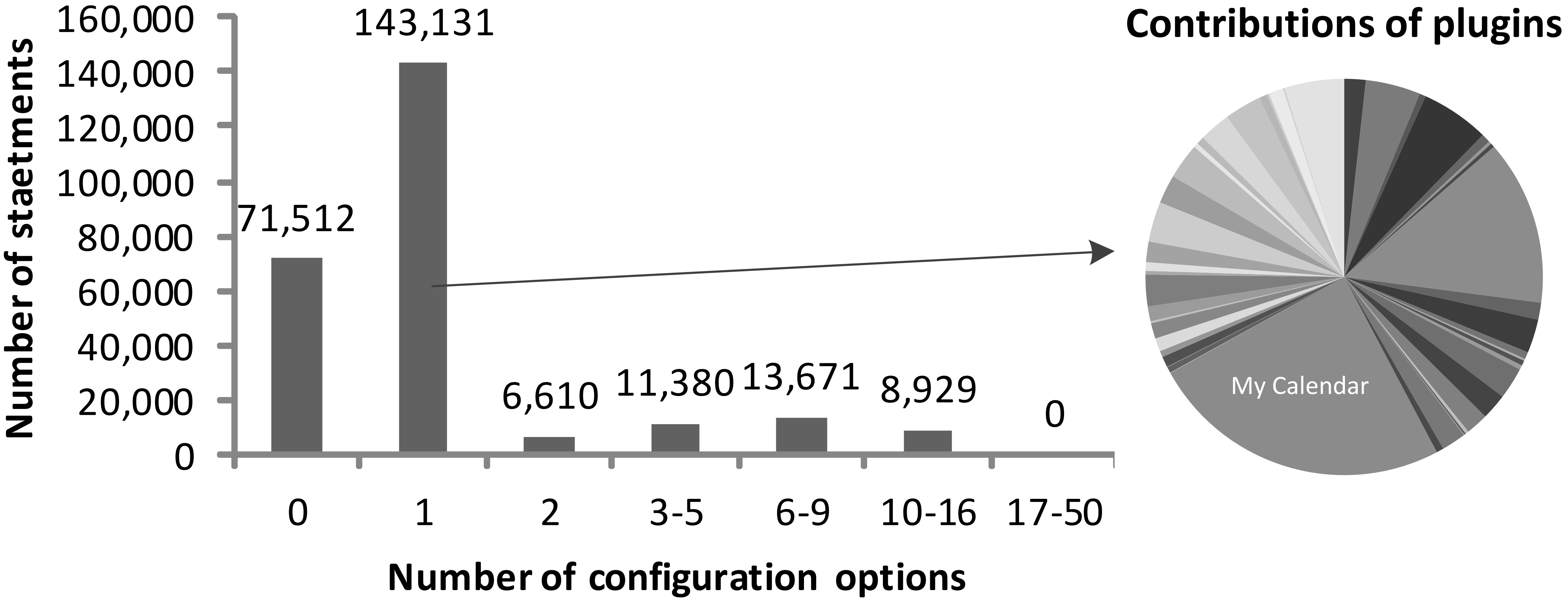
Weather Updates:

Mostly cloudy today. It's currently [:Temperature

Weather replaces [:Temperature:] with value while Smiley replaces :] with a smiley face

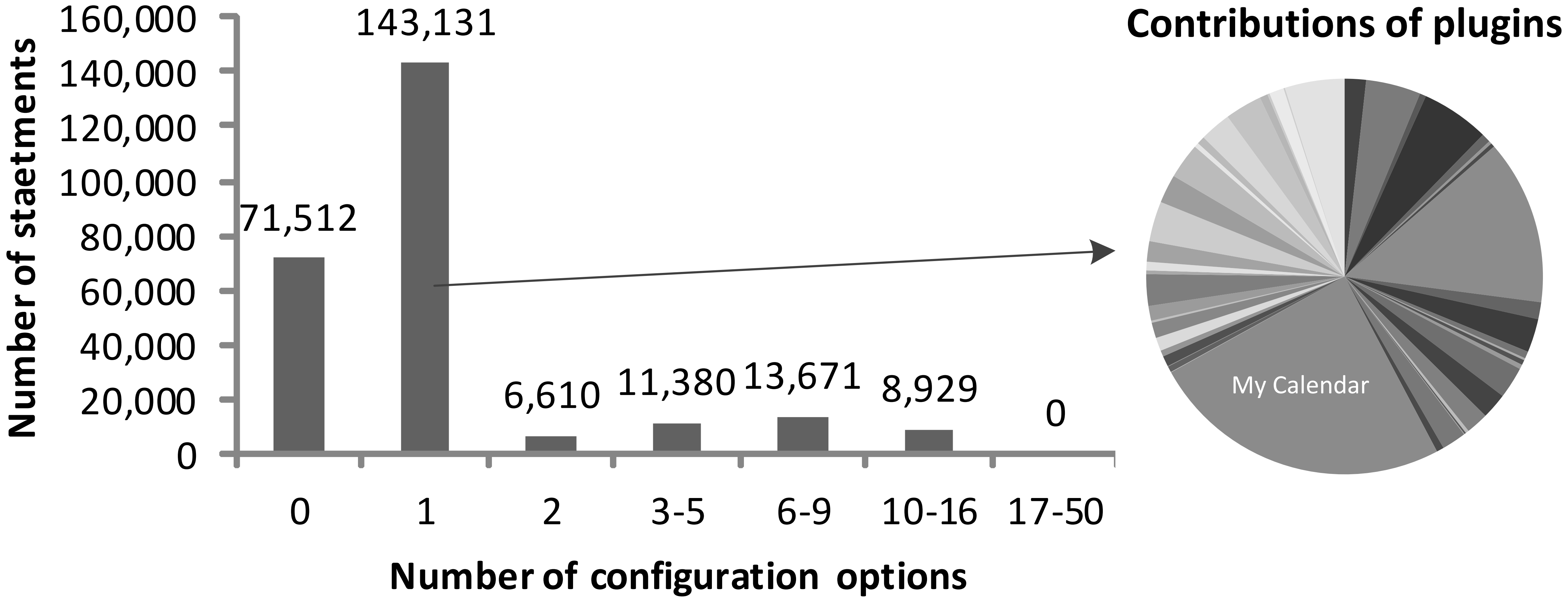
[Nguyen et al., ICSE '14]

Detecting Feature Interactions



[Nguyen et al., ICSE '14]

Detecting Feature Interactions



Intended
vs
Unintended?

[Nguyen et al., ICSE ’14]

Example Thesis Topics

- Identify heuristics to detect *unintended* feature interactions
- Features vs options: nature of configurability in the Linux kernel
- Feature modeling of plugins from build dependencies
- Using feature-oriented programming to guide cryptography API use

sarahnadi.org

Optional Exercise

(1) Familiarize Yourself With Clafer

- Look at clafer.org and familiarize yourself with the syntax and available tools

- You do not need to understand the more advanced features (e.g., quality attributes, multi-objective optimization etc.)

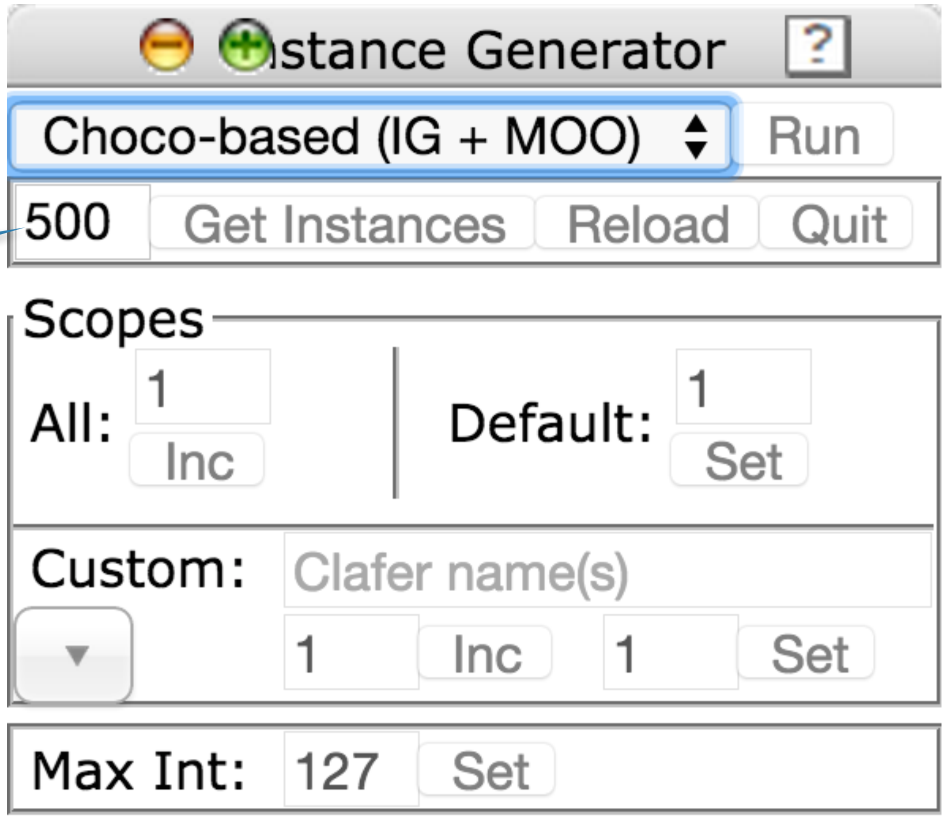
(2) Create a Feature Model!

- Select your favorite car model
- Check out the configurator on the manufacturer's website, and select at least 10 features that describe the car
- Create a feature model in clafer using those features
- Your model should make use of the following
  - optional and mandatory features
  - or and xor groups
- You can write your model directly in the online Clafer configurator (http://t3-necsis.cs.uwaterloo.ca:8093/) and then click the compile button to make sure the syntax is correct

(3) Generate Instances

- Using the same clafer online configurator, generate all possible instances of your model

- **Report how many valid products (i.e., instances) does your car have**



Choco-based (IG + MOO)

500

All: 1

Default: 1

Custom: 1 1

Max Int: 127

Make sure you increase this number to make sure you have covered all valid instances

(4) Create Cross-tree Constraints

- Add at least one cross-tree constraint to your model

- It can be based on real constraints from the car manufacturer or hypothetical constraints you come up with

- **Report how many valid products (i.e., instances) does your car have now**

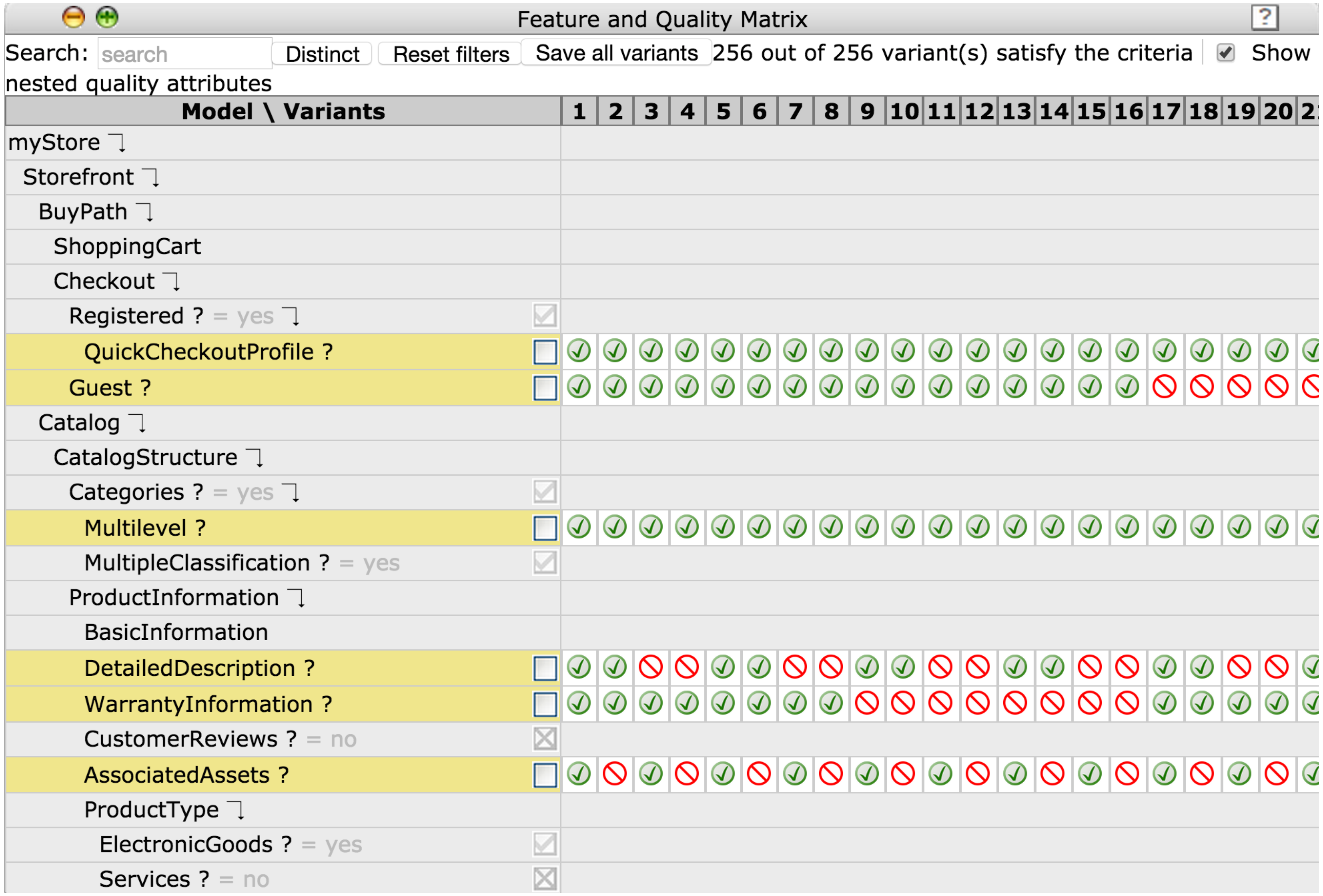
Submit Your Model

- If you want to submit your model, email weiel@st.informatik.tu-darmstadt.de your car_<yourname>.cfr along with the number of instances before and after your added constraints

- Make sure to mark the extra cross-tree constraints you added (using code comments)

Extras: Using the Online Configurator

- You can use the online configurator (feature and quality matrix) to explore the valid products in your product line



Feature and Quality Matrix

Search: search | Distinct | Reset filters | Save all variants | 256 out of 256 variant(s) satisfy the criteria ☑ Show nested quality attributes

Model \ Variants		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	2
myStore ⏋																						
Storefront ⏋																						
BuyPath ⏋																						
ShoppingCart																						
Checkout ⏋																						
Registered ? = yes ⏋	☑																					
QuickCheckoutProfile ?	□	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Guest ?	□	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	🚫	🚫	🚫	🚫	🚫
Catalog ⏋																						
CatalogStructure ⏋																						
Categories ? = yes ⏋	☑																					
Multilevel ?	□	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MultipleClassification ? = yes	☑																					
ProductInformation ⏋																						
BasicInformation																						
DetailedDescription ?	□	✓	✓	🚫	🚫	✓	✓	🚫	🚫	✓	✓	🚫	🚫	✓	✓	🚫	🚫	✓	✓	🚫	🚫	✓
WarrantyInformation ?	□	✓	✓	✓	✓	✓	✓	✓	✓	🚫	🚫	🚫	🚫	🚫	🚫	🚫	🚫	✓	✓	✓	✓	✓
CustomerReviews ? = no	☒																					
AssociatedAssets ?	□	✓	🚫	✓	🚫	✓	🚫	✓	🚫	✓	🚫	✓	🚫	✓	🚫	✓	🚫	✓	🚫	✓	🚫	✓
ProductType ⏋																						
ElectronicGoods ? = yes	☑																					
Services ? = no	☒																					

Software Product Lines

Sarah Nadi
Software Technology Group

🌍 sarahnadi.org

TECHNISCHE
UNIVERSITÄT
DARMSTADT