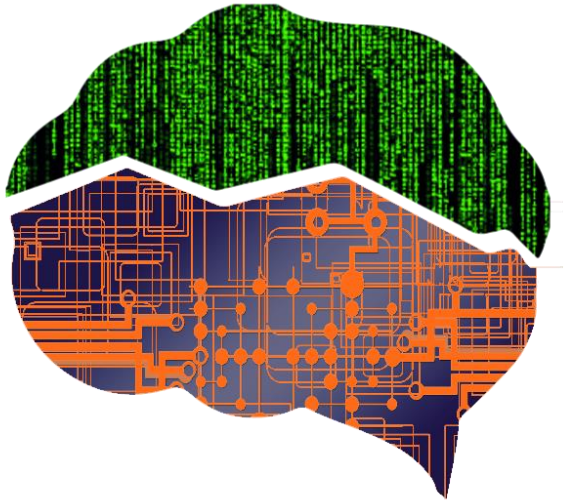
Software Product Line Engineering

Feature-Oriented Development

Christian Kästner (Carnegie Mellon University)
Sven Apel (Universität Passau)
Norbert Siegmund (Bauhaus-Universität Weimar)
Gunter Saake (Universität Magdeburg)



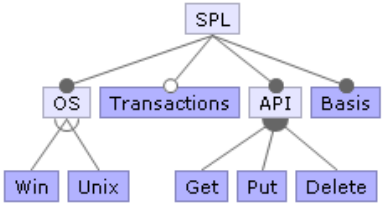
Bauhaus-Universität Weimar

How to implement variability?

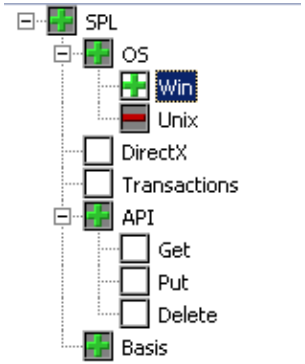
Domain Eng.

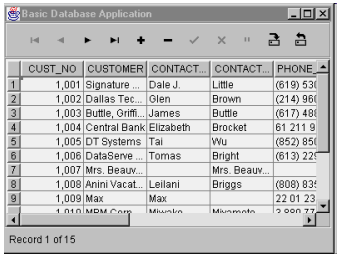
Application Eng.

Feature Model



Reusable Implementation artifacts





Feature Selection

Generator

Resulting Program

Goals

- Novel implementation techniques
- Solving the problems:
  - **Feature traceability**
  - Crosscutting concerns
  - Preplanning problem
  - Inflexible extension mechanisms (especially inheritance)
- Modular implementation of features

Agenda

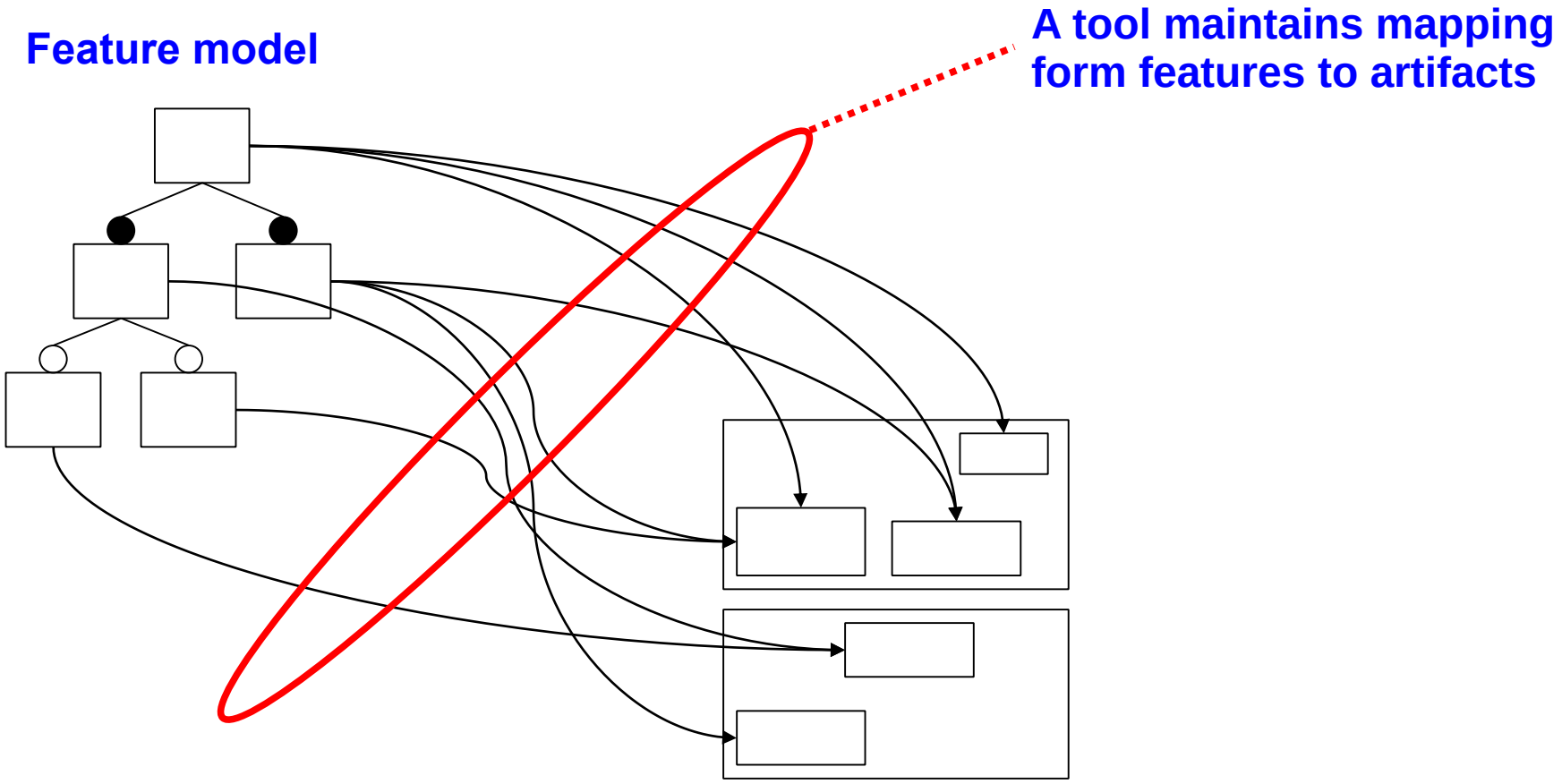
- Basic idea
- Implementation via AHEAD
- Principle of uniformity

Basic Idea

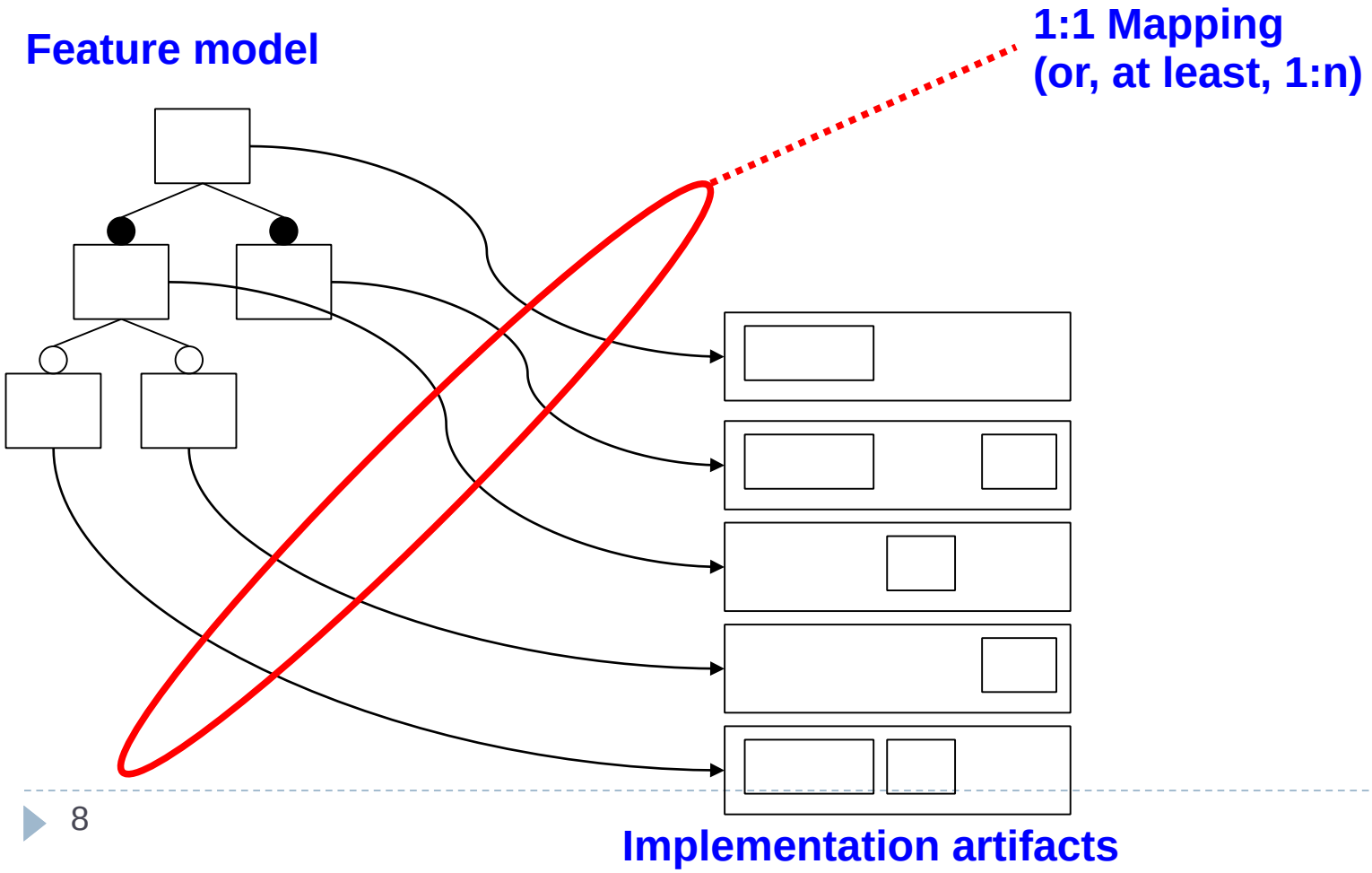
Goal: Feature Cohesion (Feature Traceability Problem)

- Property of a program: Localize **all** implementation artifacts of a single feature an **single place** in the code
  - Features are explicit in the program code
- A question of programming language or environment!
  - Virtual vs. physical separation

Feature Traceability with Tool Support



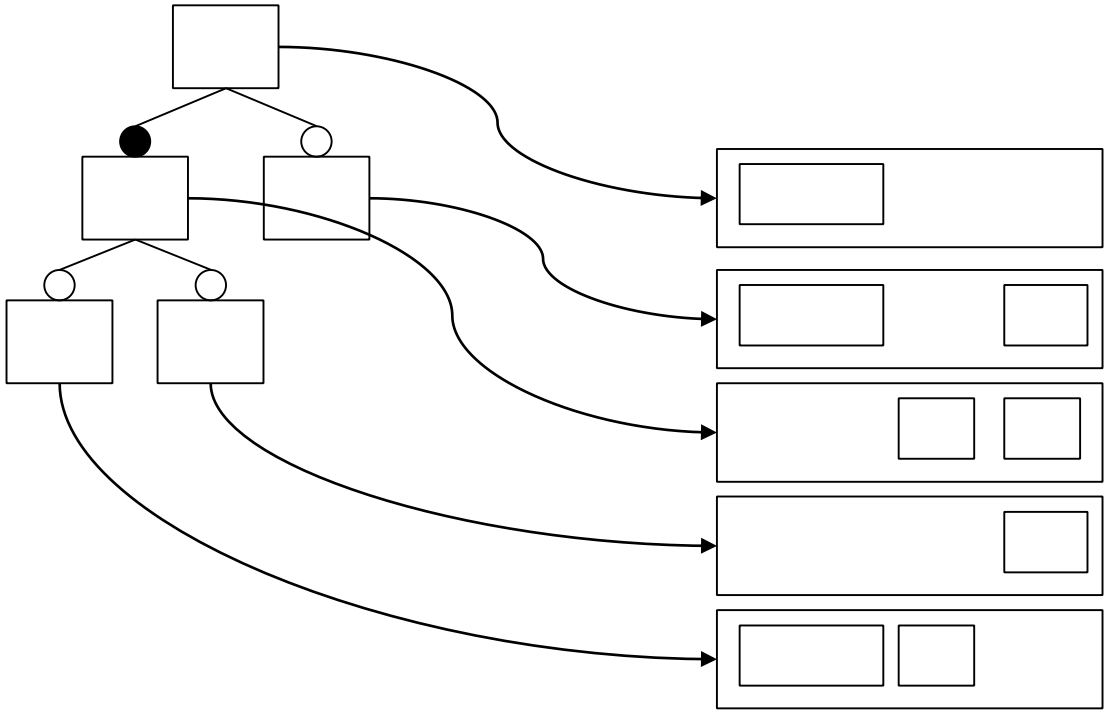
Feature Traceability with Language Support



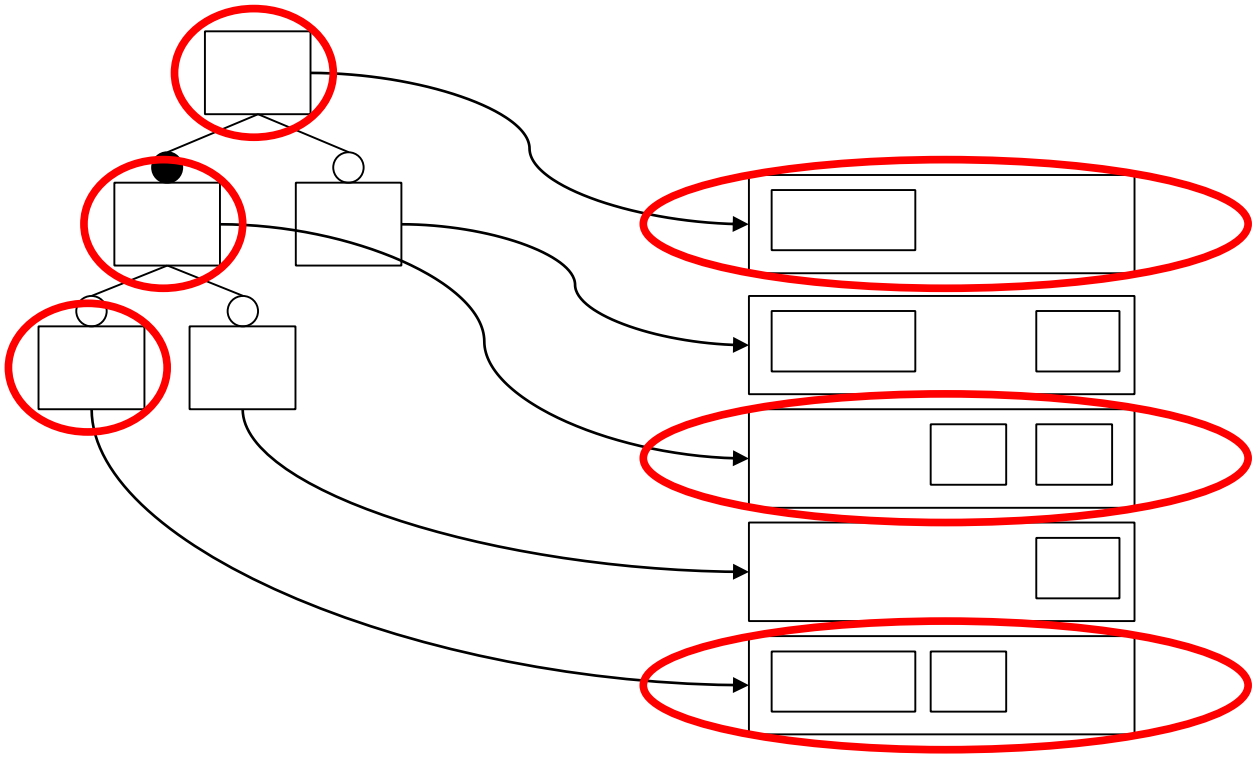
Feature-Oriented Programming

- Prehofer, ECOOP'97 und Batory, ICSE’03
- Language-based approach to overcome the feature traceability problem
- Each feature is implemented by a feature module
  - Good feature traceability
  - Separation and modularization of features
  - Simple feature composition
- Feature-based program generation

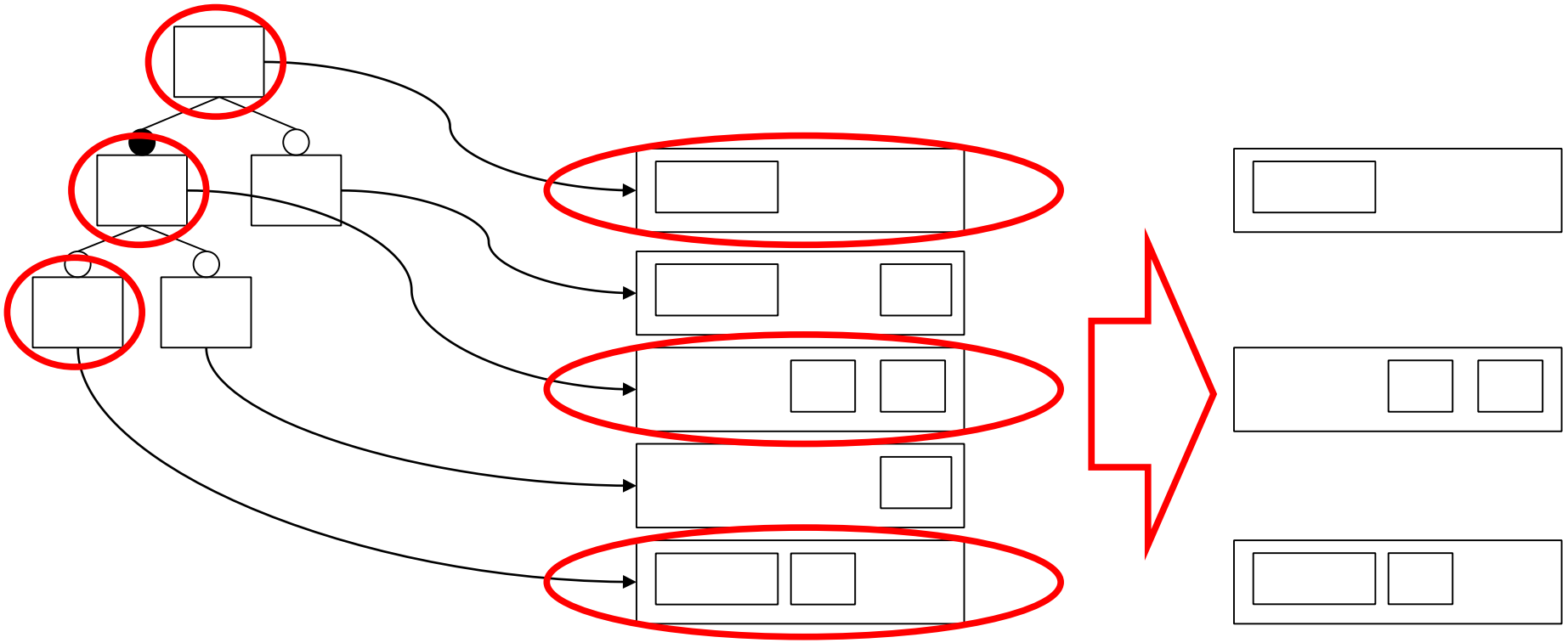
Feature Composition



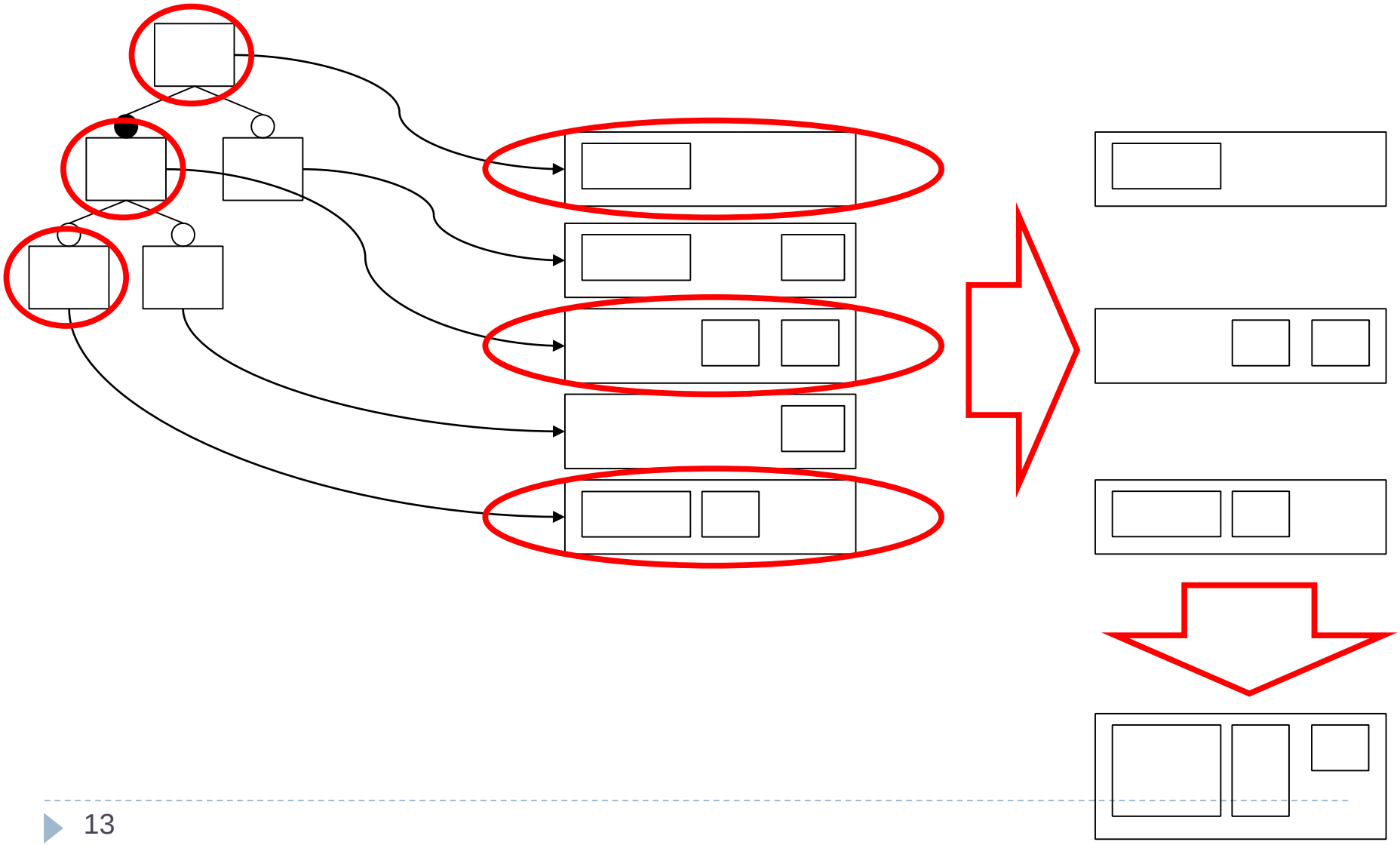
Feature Composition



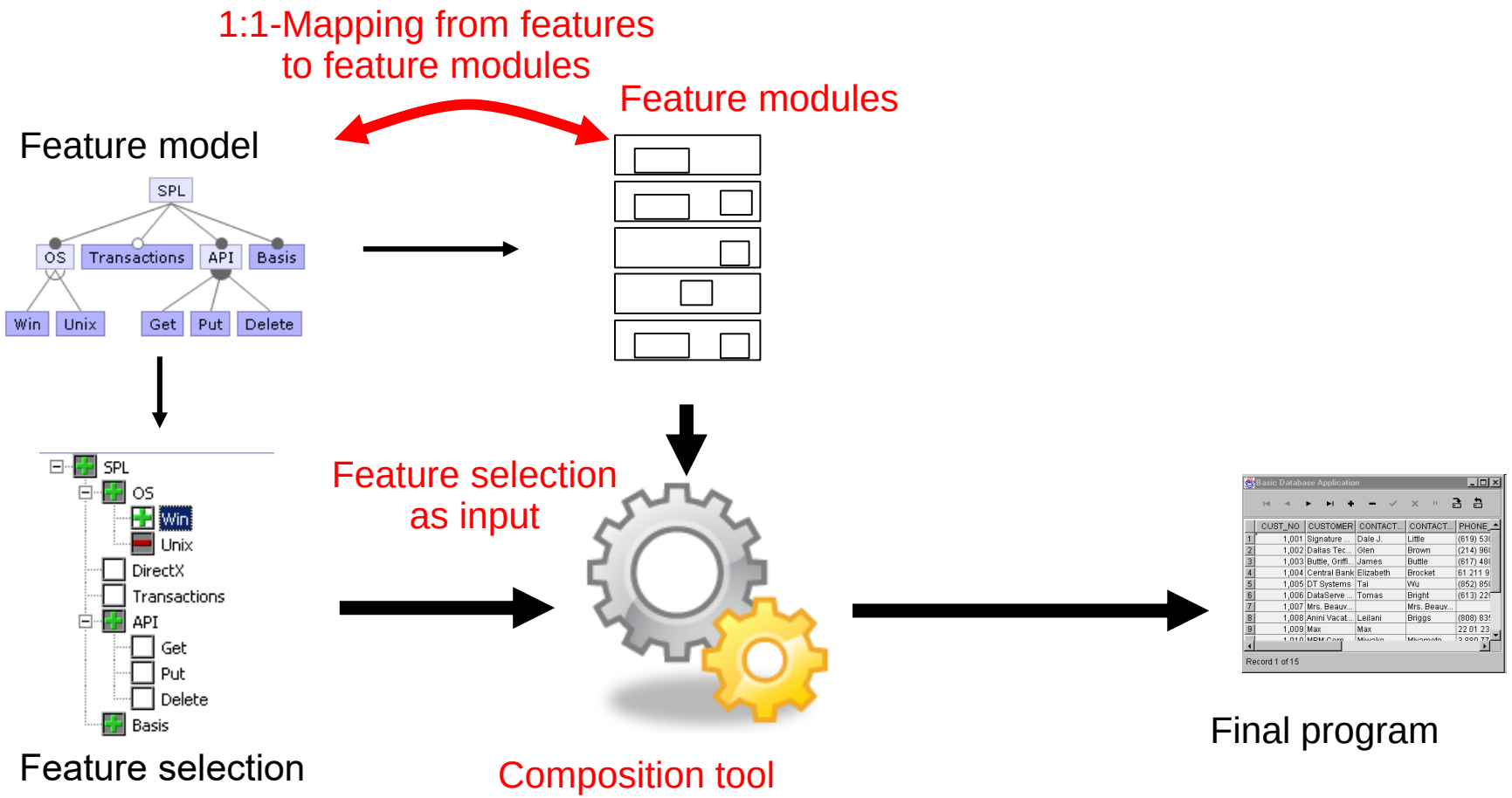
Feature Composition



Feature Composition



Product Lines with Feature Modules



Implementation via AHEAD

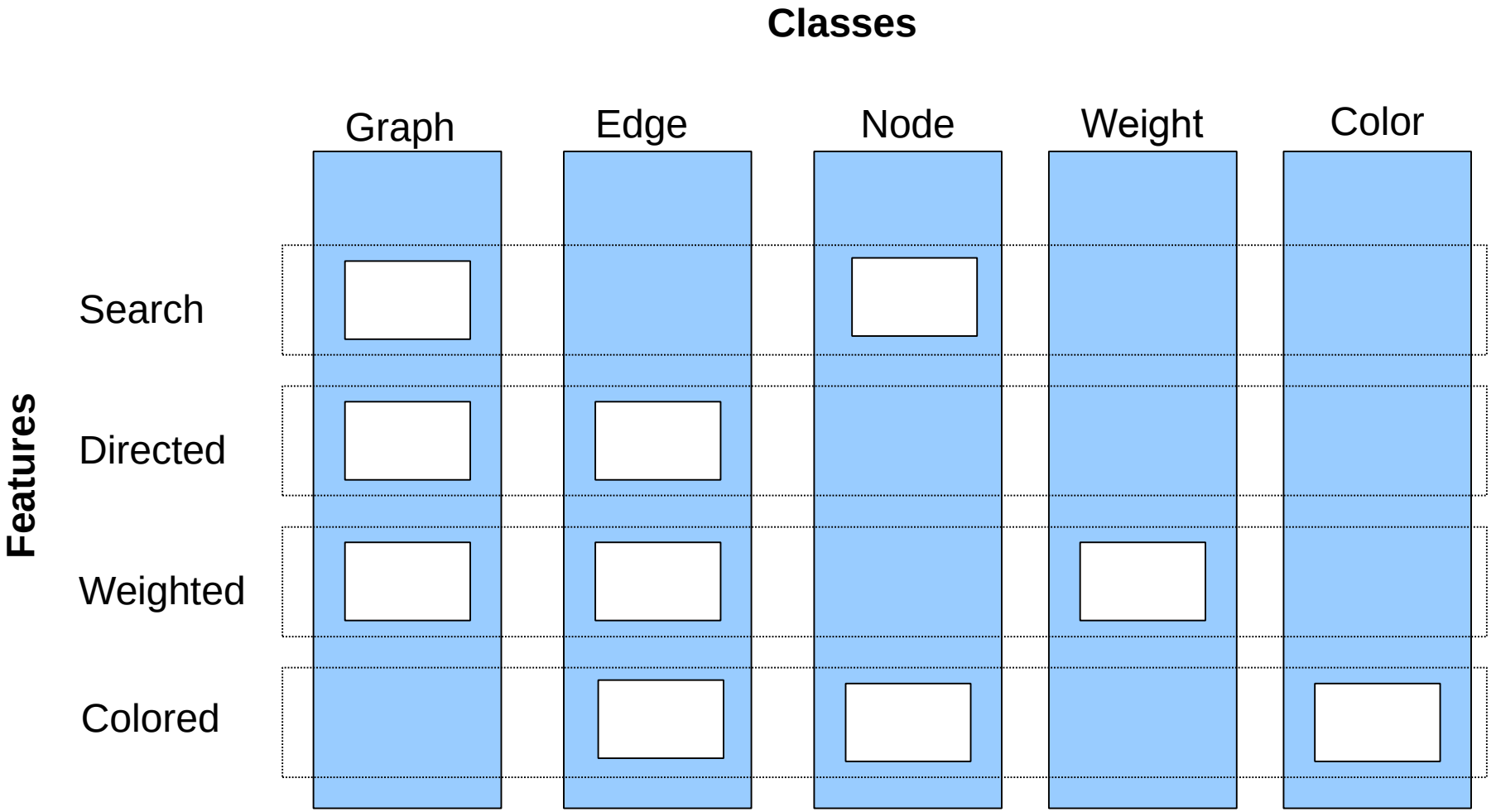
Implementing Feature Modules

- Separation in multiple classes is established and suitable as base structure
- Features are often implemented in multiple classes
- Classes often implement more than a single feature

- Idea: Keep class structure, but further decompose classes based on features

- AHEAD (Algebraic Hierarchical Equations for Application Design) or FeatureHouse as possible tools

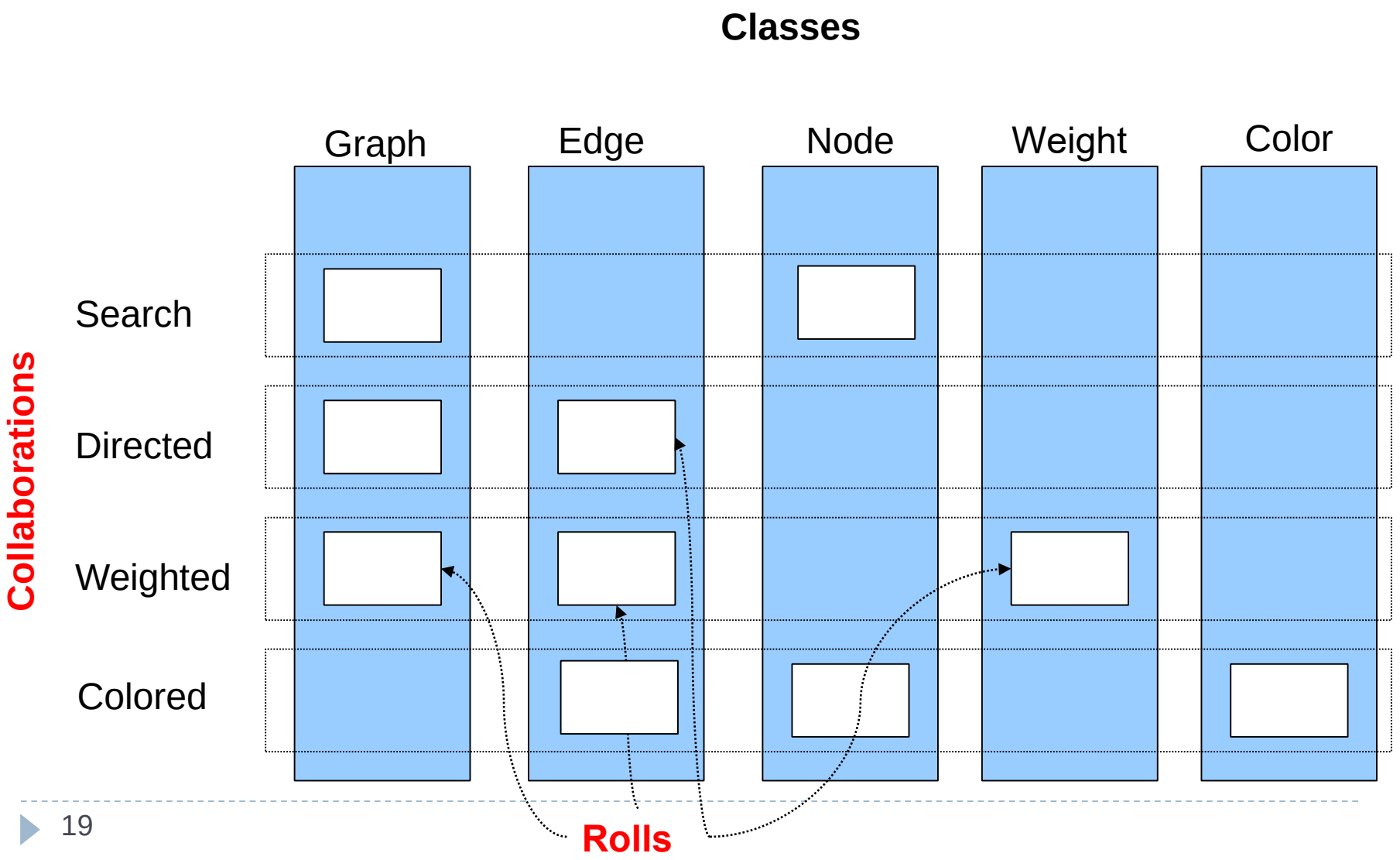
Decomposition of Classes



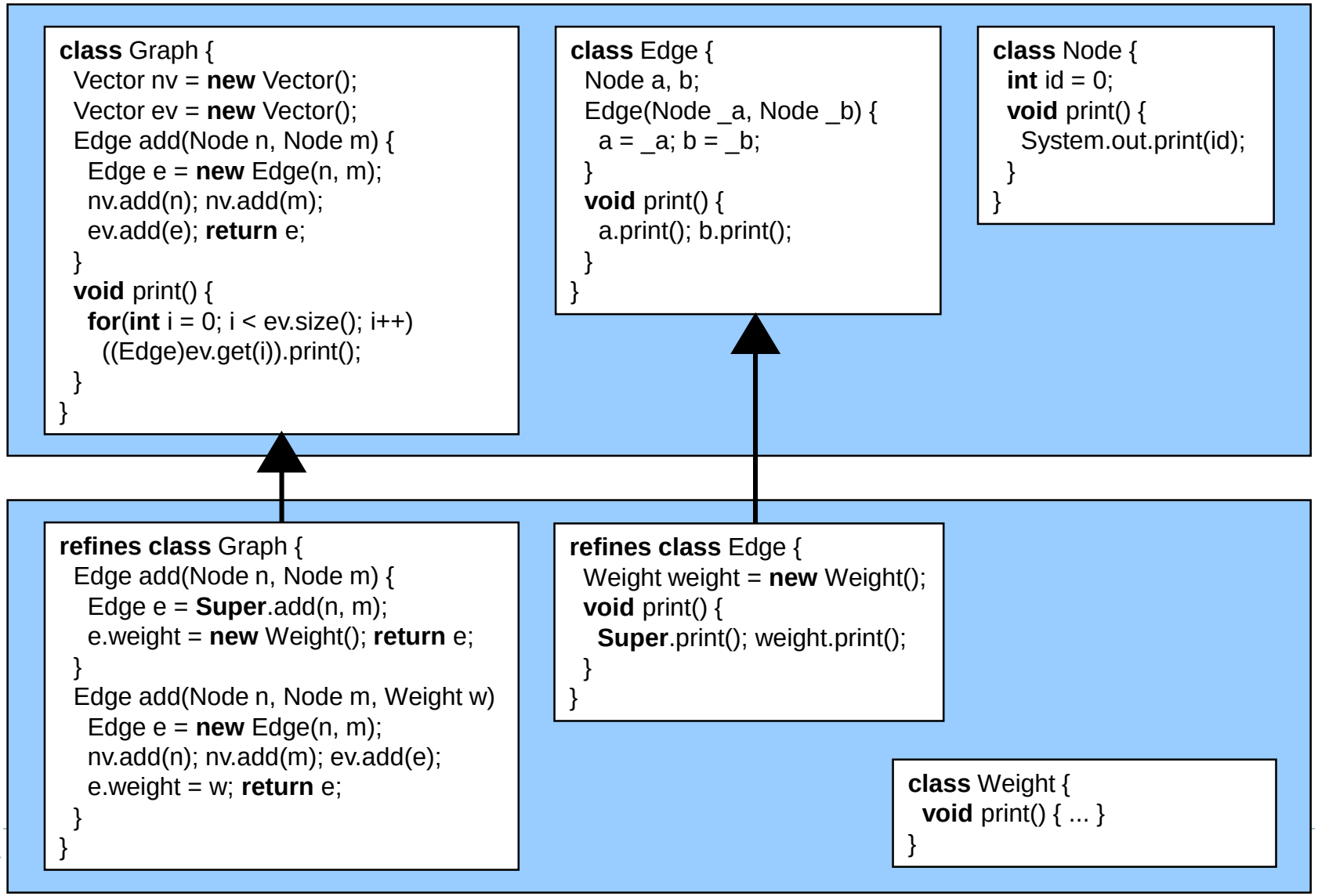
Collaborations & Rolls

- Collaboration: a set of classes that interaction with each other to implement a feature
- Different classes play different rolls within a collaboration
- A class plays different rolls in different collaborations
- A role encapsulates the behavior / the functionality of a class that is relevant for a collaboration

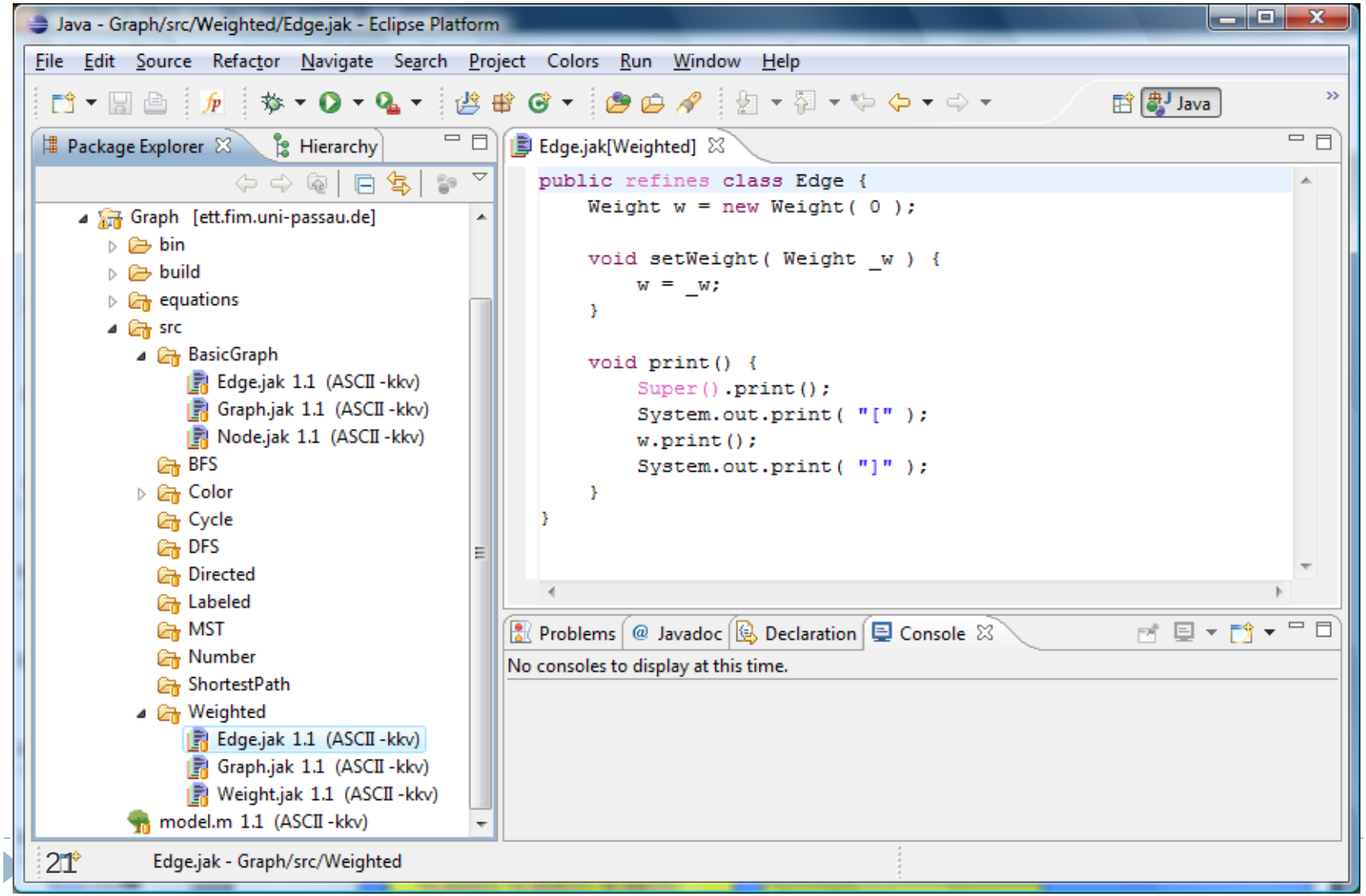
Collaborations & Rolls



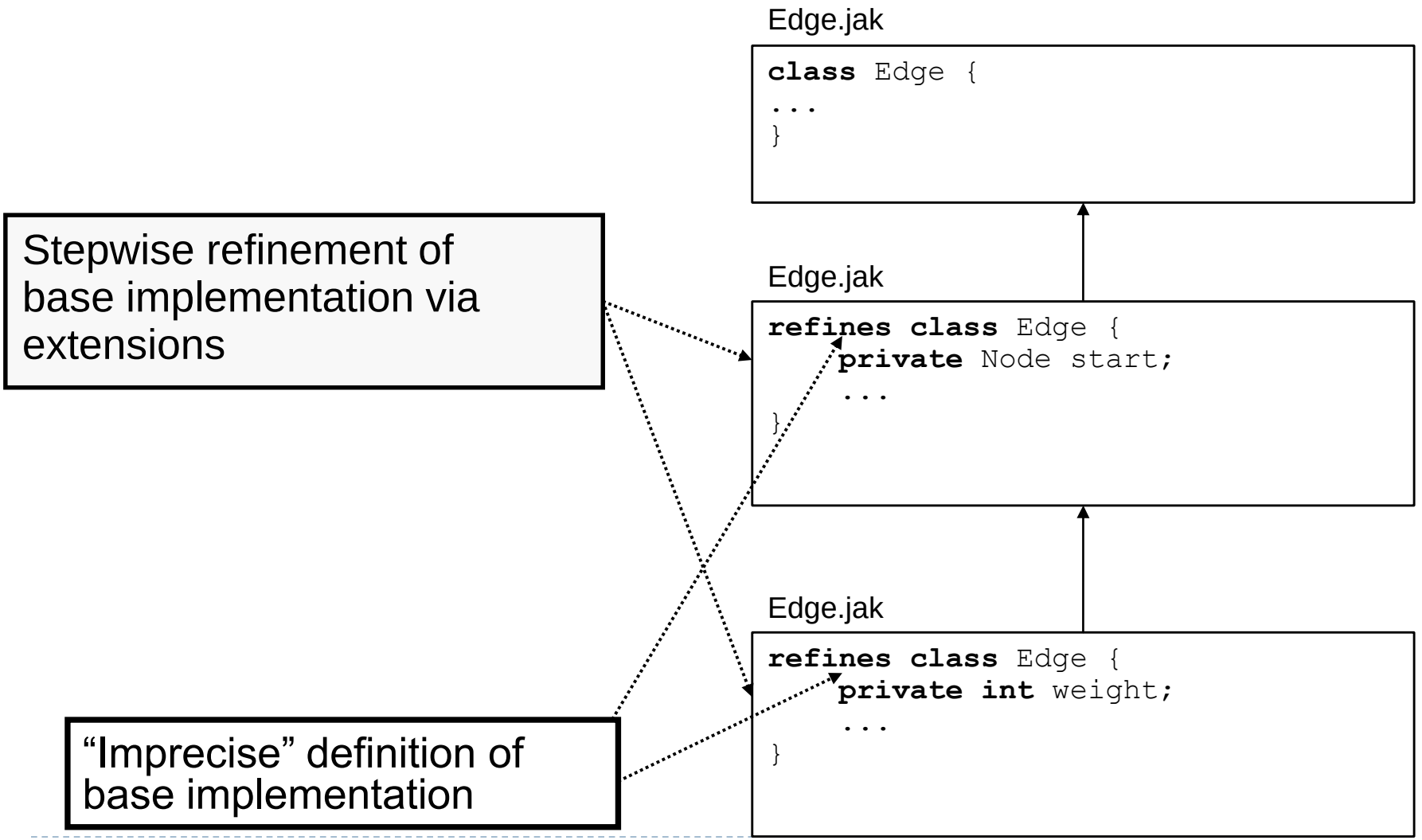
Collaboration Design



Folder Hierarchy



Example: Class Refinements



Method Refinements (AHEAD)

- Methods can be added or extended in every refinement
- Overriding of methods
- Calling the method of the previous refinement via **`Super`***
- Similar to inheritance

```
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);
    }
}
```

* Due to technical reasons, we have to give the expected types of the method behind the `Super` keyword, for example, `Super(String,int).print('abc', 3)`

Method Refinement (FeatureHouse)

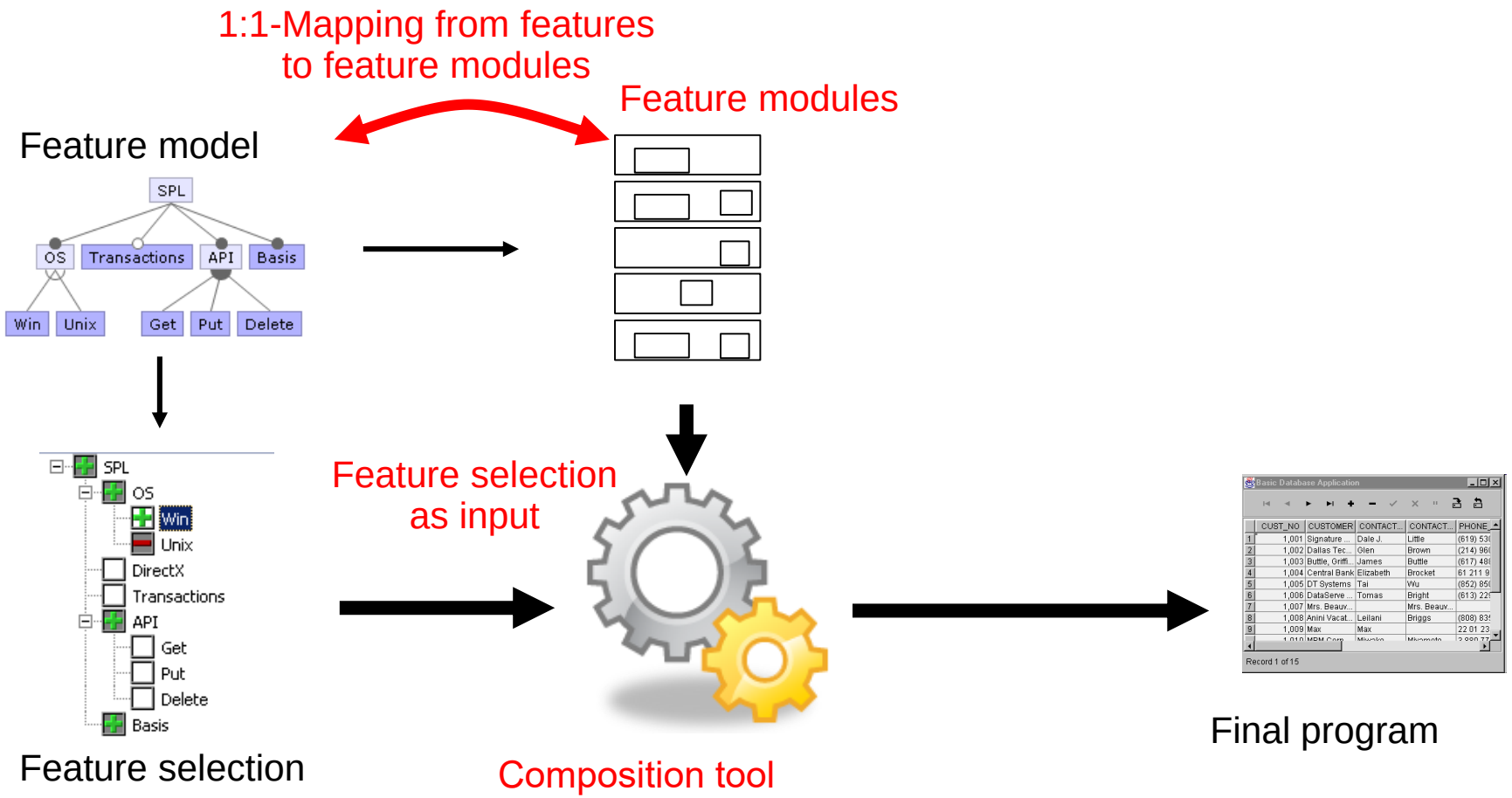
- No **`refines`** necessary
- Methods can be added and extended in every refinement
- Overriding of methods
- Calling the method of a previous refinement via **`original`**
- Similar to inheritance

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

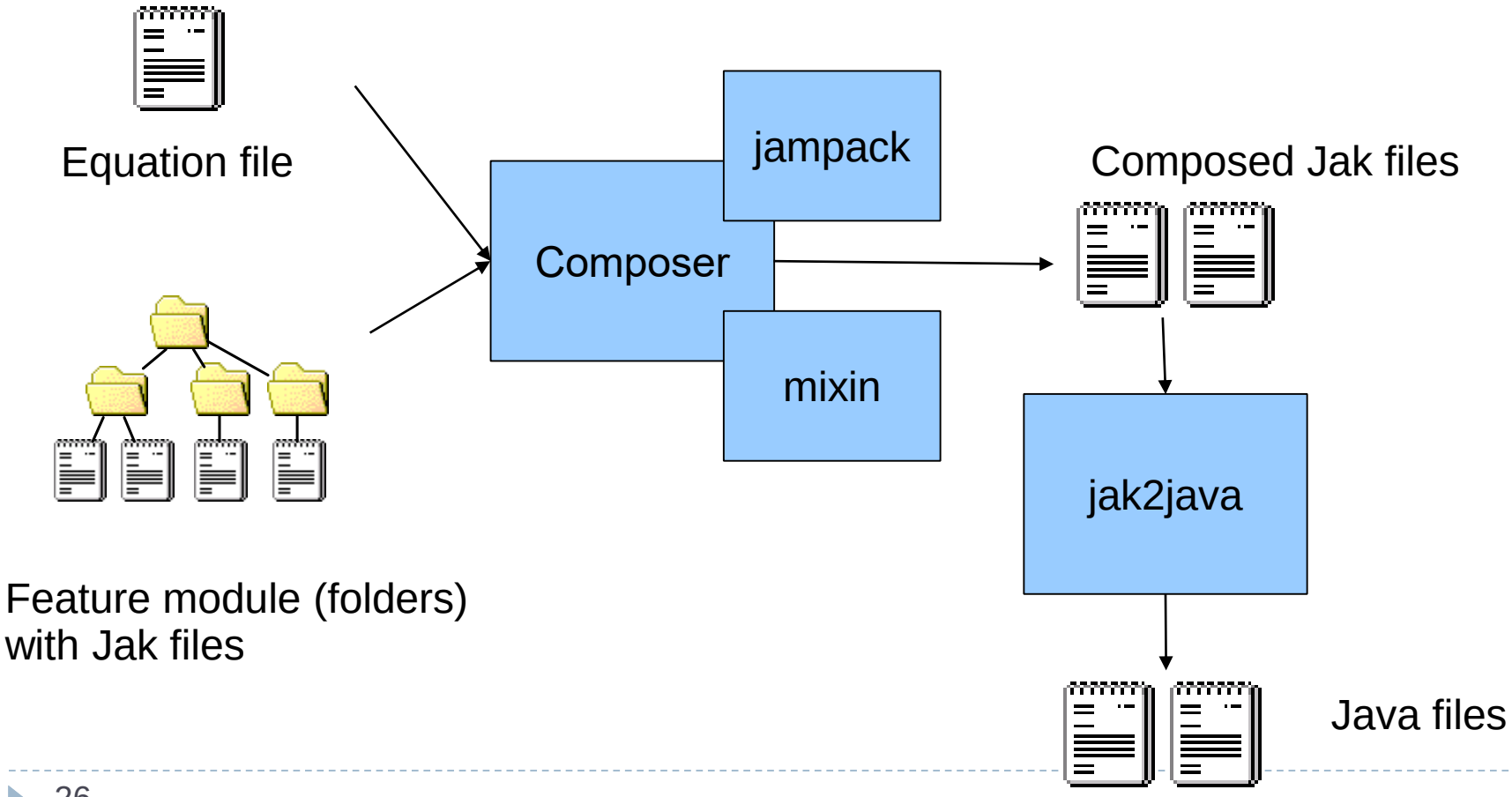
```
class Edge {
    private Node start;
    void print() {
        original();
        System.out.print(
            " directed from " + start);
    }
}
```

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

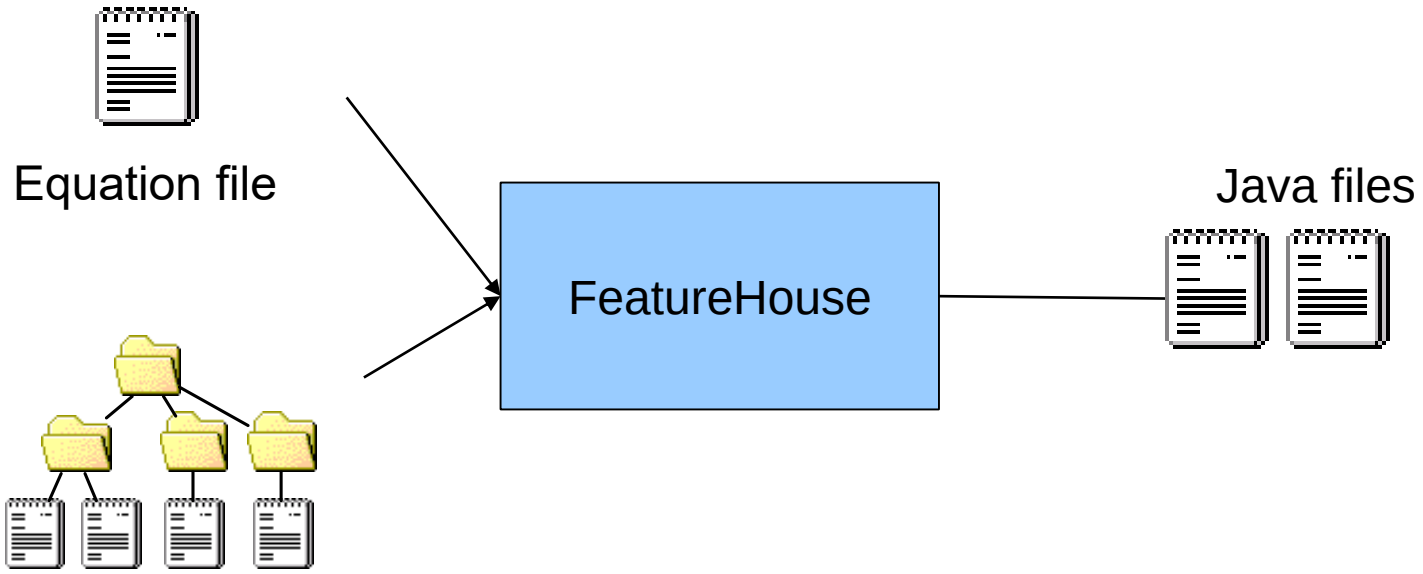
Product Lines with Feature Modules



Composition in AHEAD



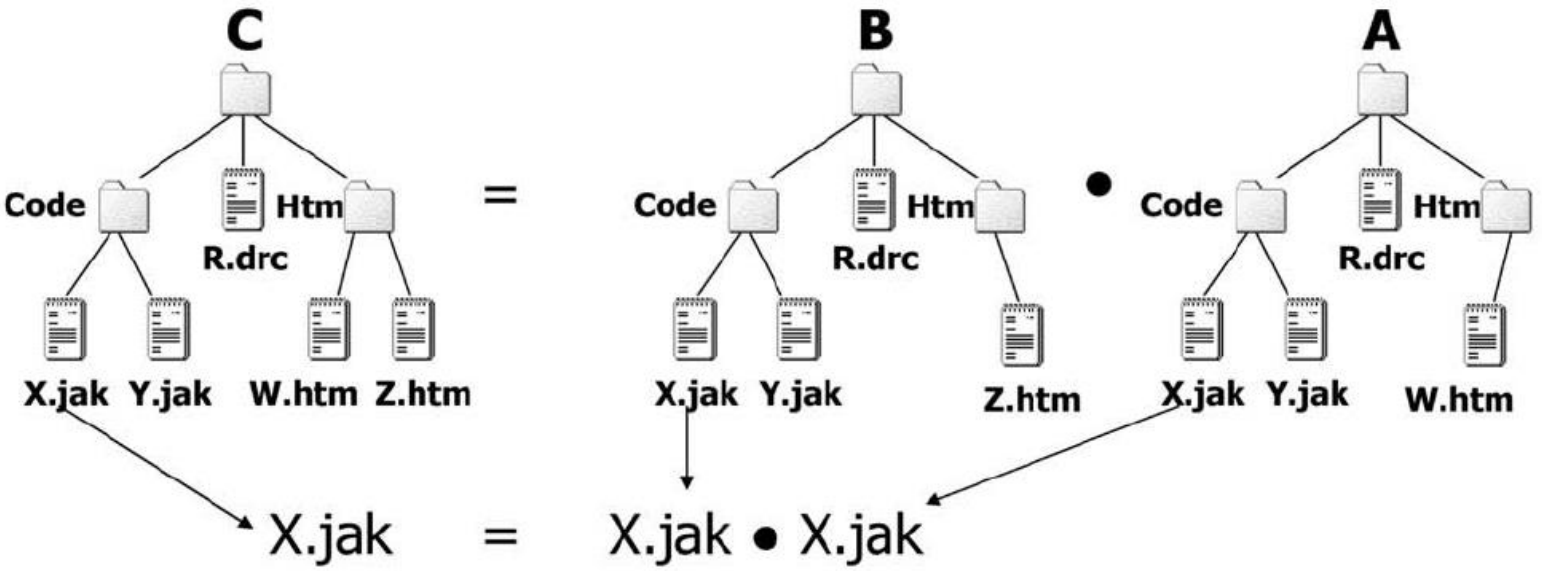
Composition in FeatureHouse



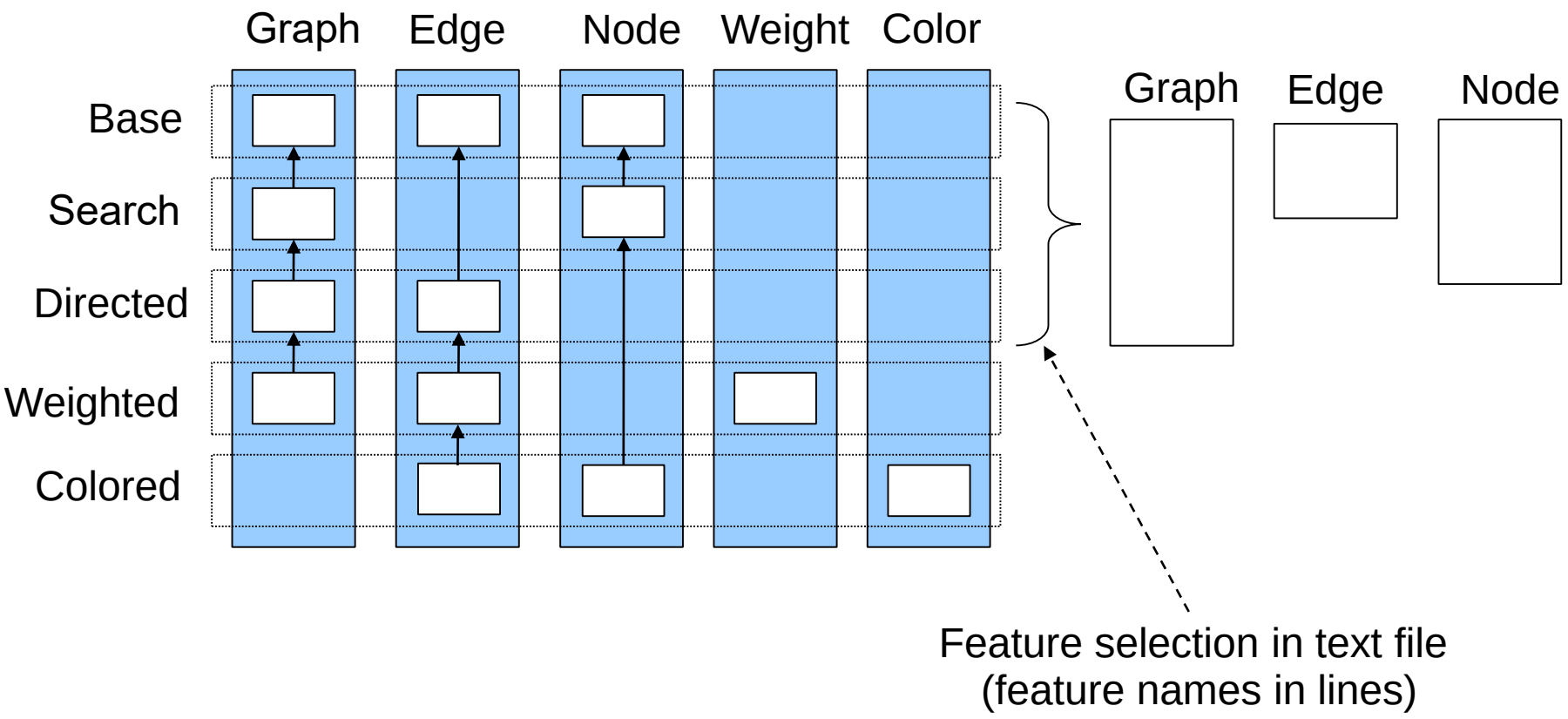
Feature modules (folders)
with Java files

Composition of Folders

- All rolls of a collaboration will be stored into one package/module (usually in a folder)
- Composition of collaborations via composition of classes with all equally named class refinements



Example

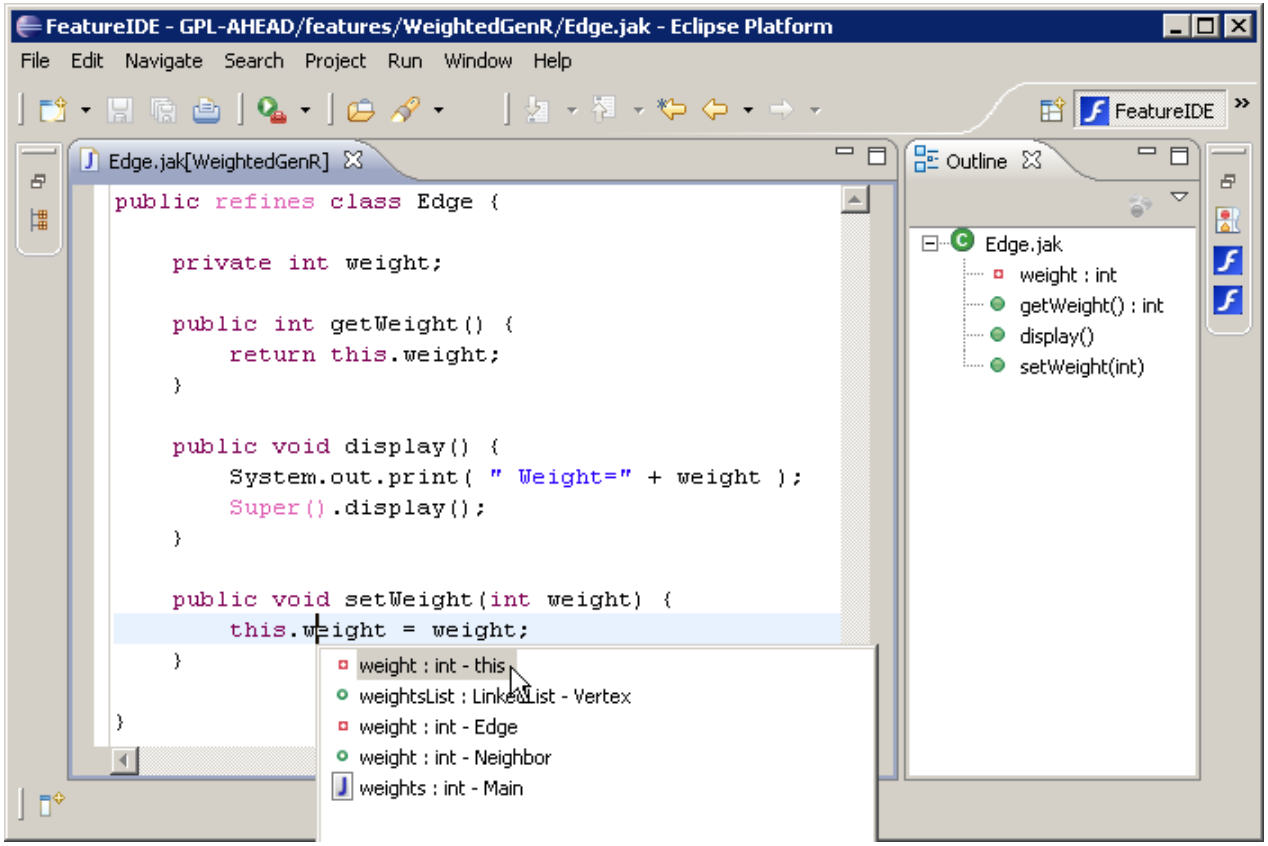

Graph
Edge
Node
Weight
Color
Base
Search
Directed
Weighted
Colored
Graph
Edge
Node
Feature selection in text file
(feature names in lines)

Tools

- AHEAD Tool Suite + Documentation
  - Command-line tool for Jak (Java 1.4 extension)
  - http://www.cs.utexas.edu/users/schwartz/ATS.html
- FeatureHouse
  - Command-line to for Java, C#, C, Haskell, UML, …
  - http://www.fosd.de/fh
- FeatureC++
  - Alternative to AHEAD for C++
  - http://www.fosd.de/fcpp
- FeatureIDE
  - Eclipse-Plugin for AHEAD, FeatureHouse, and FeatureC++
  - Automatically builds variants, provides syntax highlight, etc…
  - http://www.fosd.de/featureide

FeatureIDE – Demo

- Video tutorials at youtube



http://www.cs.utexas.edu/users/dsb/cs392f/Videos/FeatureIDE/

Summary for AHEAD

- A base class + arbitrary refinements (rolls)
- Class refinements can …
  - Add fields
  - Add methods
  - Refine methods
- Feature module (collaboration): Folder with base classes and /or refinements
- During compilation, base classes and refinements of selected features are composed

Principle of Uniformity

Principle of Uniformity

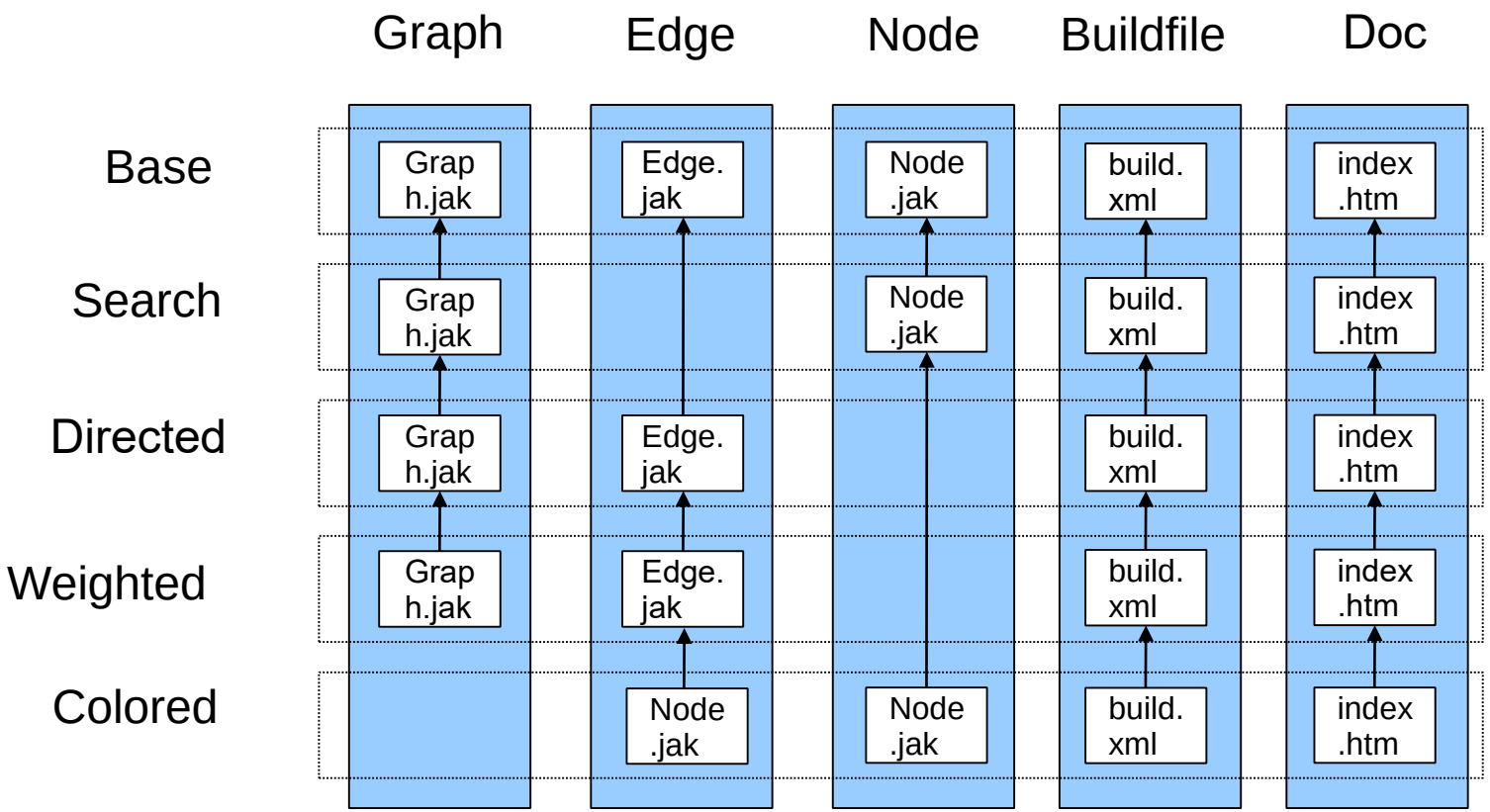
- Software consists of more than only Java source code
  - Other programming languages (z. B. C++, Javascript)
  - Build scripts (Make, XML)
  - Documentation (XML, HTML, PDF, Text, Word)
  - Grammars (BNF, ANTLR, JavaCC, Bali)
  - Models (UML, XMI, ...)
  - …
- All software artifacts must be able to be refined
- Integration of different artifacts in collaborations

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

Example



Further files: Grammars, unit tests, models, specifications, database schemas, etc.

Tool Support

- AHEAD – language-independent concept with different tools for:
  - Jak (Java 1.4)
  - Xak (XML)
  - Bali-Grammars
- FeatureHouse – language-independent tool, easy to extend, supports:
  - Java 1.5
  - C#
  - C
  - Haskell
  - JavaCC- and Bali-Grammars
  - UML

Summary

- Feature-oriented programming solves the feature traceability problem via collaborations and rolls (mapping)
- Implementation via class refinements
- Principle of uniformity

Outlook

- Implementation of crosscutting concerns can become expensive in specific cases
- Features are not always independent. How to implement dependent collaborations?
- Discussion and limitations

Literature

- D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering, 30(6), 2004.
  [Introduction to AHEAD]
- S. Apel, C. Kästner, and C. Lengauer. Language-Independent and Automated Software Composition: The FeatureHouse Experience. IEEE Transactions on Software Engineering, 39(1), 2013.
  [Introduction to FeatureHouse]

Quiz

- How many rolls can a program with 3 classes and 4 features have at (a) maximum and (b) minimum?
- Can we replace class refinements simply by inheritance?
- How does AHEAD solve the preplanning problem?
- How do components and feature orientation match?