Technische Universität Braunschweig

ISF

Structural Feature Interaction Patterns - Case Studies and Guidelines

VaMoS 2014, January 22–24, Nice, France

Sven Schuster, Sandro Schulze, Ina Schaefer, January 23, 2014

Feature Interactions

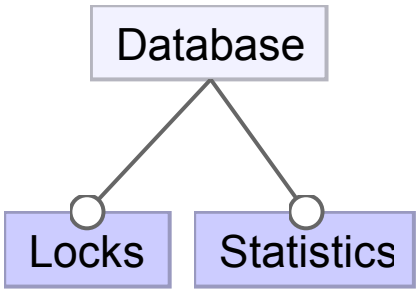
Behavioral Feature Interactions

- Fire & Flood Control

Feature Interactions

Structural Feature Interactions

- Database Locks & Statistics



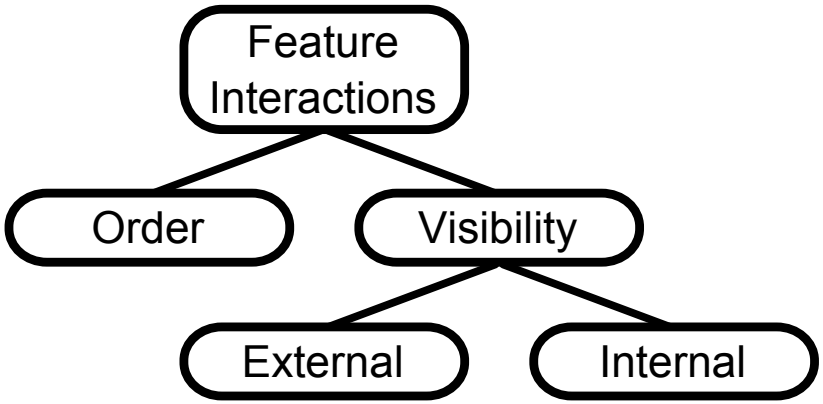
- Locks (blue), Statistics (red)
- Overlapping (violet)
  - Locking the calculation of Statistics
  - Statistics of Locking
- Optional Feature Problem

```
1  class Database {
2    List locks;
3    void lock() { /*...*/ }
4    void unlock() { /*...*/ }
5    void put(Object key, Object data) {
6      lock();
7      /*...*/
8      unlock();
9    }
10   Object get(Object key) {
11     lock();
12     /*...*/
13     unlock();
14   }
15   int getOpenLocks() {
16     return locks.size();
17   }
18   int getDbSize() {
19     return calculateDbSize();
20   }
21   static int calculateDbSize() {
22     lock();
23     /*...*/
24     unlock();
25   }
26 }
```

Problem Statement

Categorization by Apel et al.

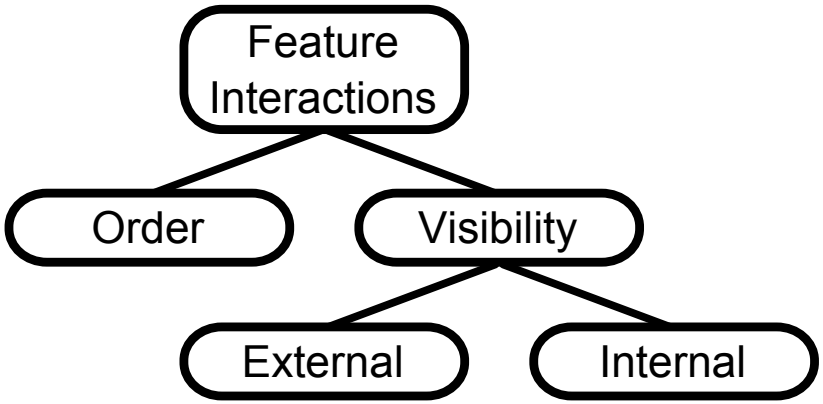
- Which FIs exist?
- How do they manifest?



Problem Statement

Categorization by Apel et al.

- Which FIs exist?
- How do they manifest?



What about...

- Why do they exist?
- Are they intentional?

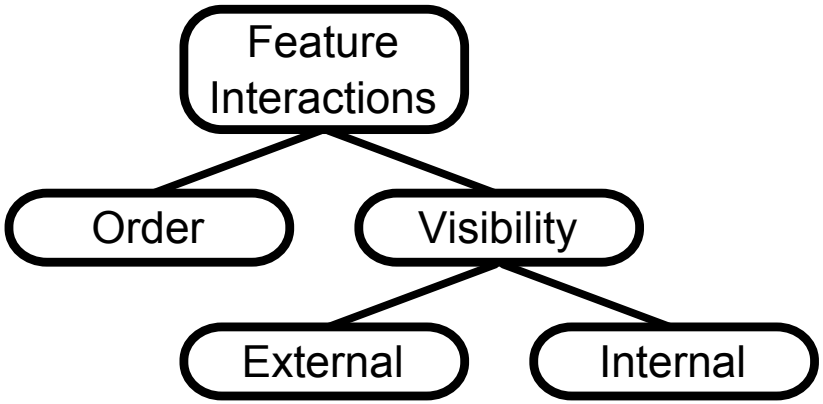
Focussing on...

- Internal Interactions
  - Structural Interactions

Problem Statement

Categorization by Apel et al.

- Which FIs exist?
- How do they manifest?



What about...

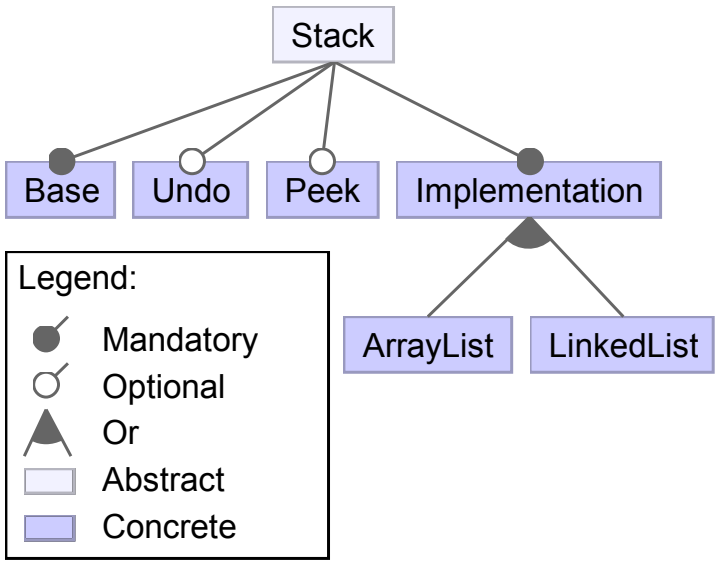
- Why do they exist?
- Are they intentional?

Focussing on...

- Internal Interactions
  - Structural Interactions

$\Rightarrow$ Structural Feature Interaction Patterns

SPLs & FOP



Feature Base

```
class Stack
  void push(int v)
    { /* ... */ }
  int pop() { /* ... */ }
}
```

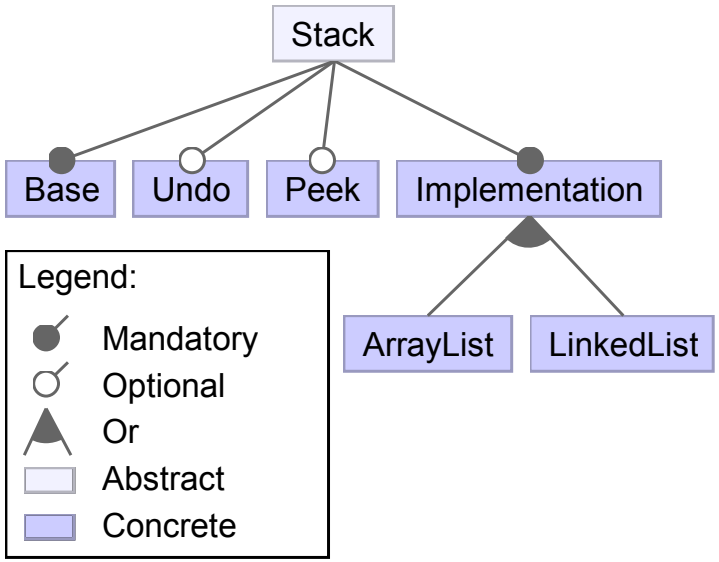
Feature Undo

```
class Stack { /* ... */
  int backupPush;
  void undo() { /* ... */ }
  void push(int v) {
    backupPush=v;
    original(v);
  }
}
```

Superimposition

```
class Stack { /* ... */
  int backupPush;
  int pop() { /*...*/ }
  void undo() { /*...*/ }
  void push(int v) {
    backupPush=v;
    /* original push */
  }
}
```

SPLs & FOP



Feature Base

```
class Stack
  void push(int v)
    { /* ... */ }
  int pop() { /* ... */ }
}
```

Feature Undo

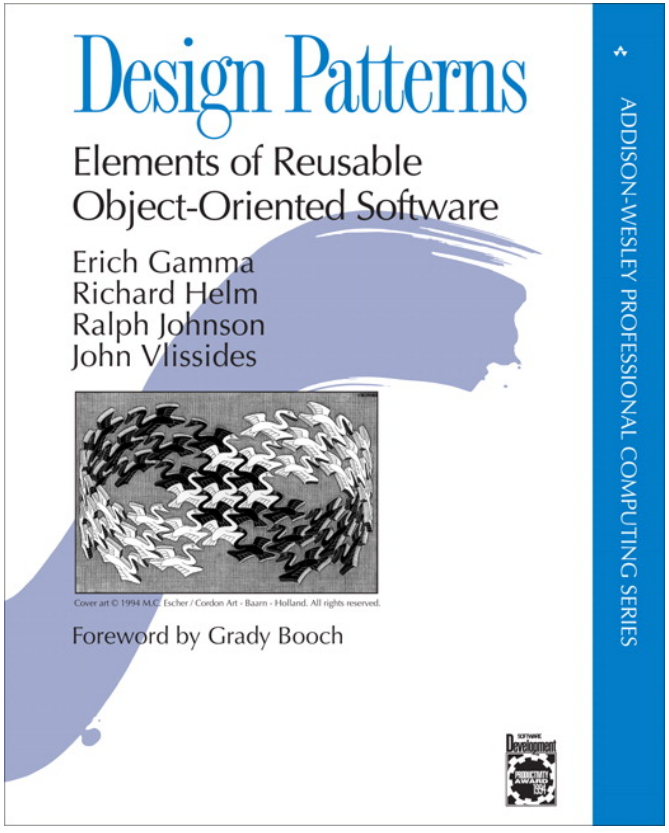
```
class Stack { /* ... */
  int backupPush;
  void undo() { /* ... */ }
  void push(int v) {
    backupPush=v;
    original(v);
  }
}
```

Superimposition

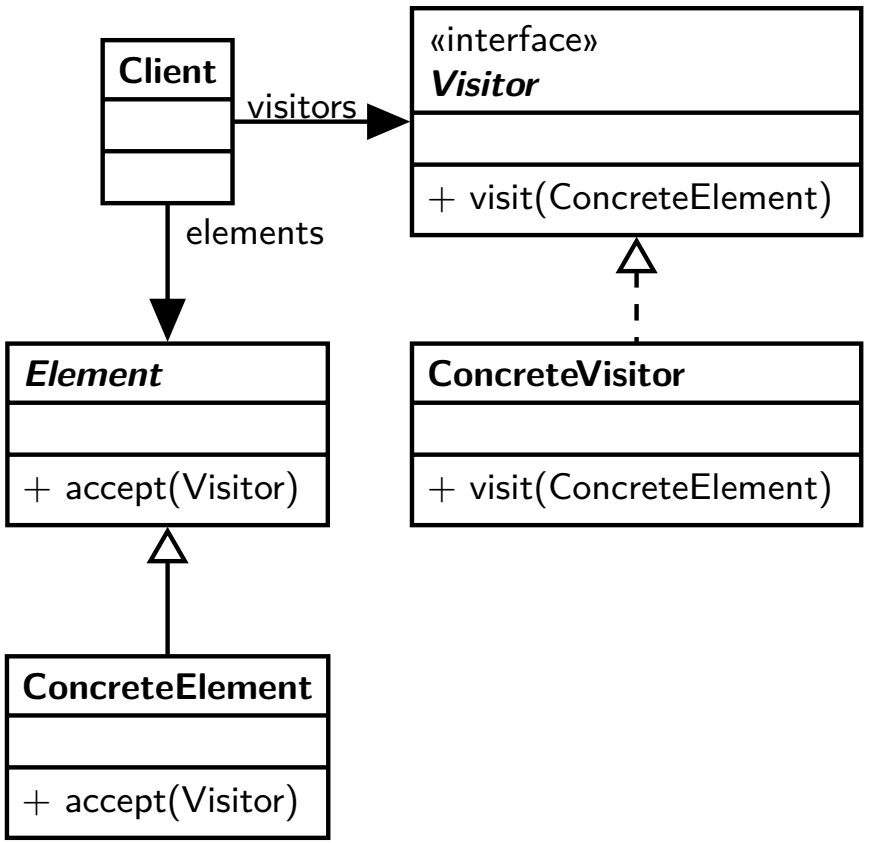
```
class Stack { /* ... */
  int backupPush;
  int pop() { /*...*/ }
  void undo() { /*...*/ }
  void push(int v) {
    backupPush=v;
    /* original push */
  }
}
```

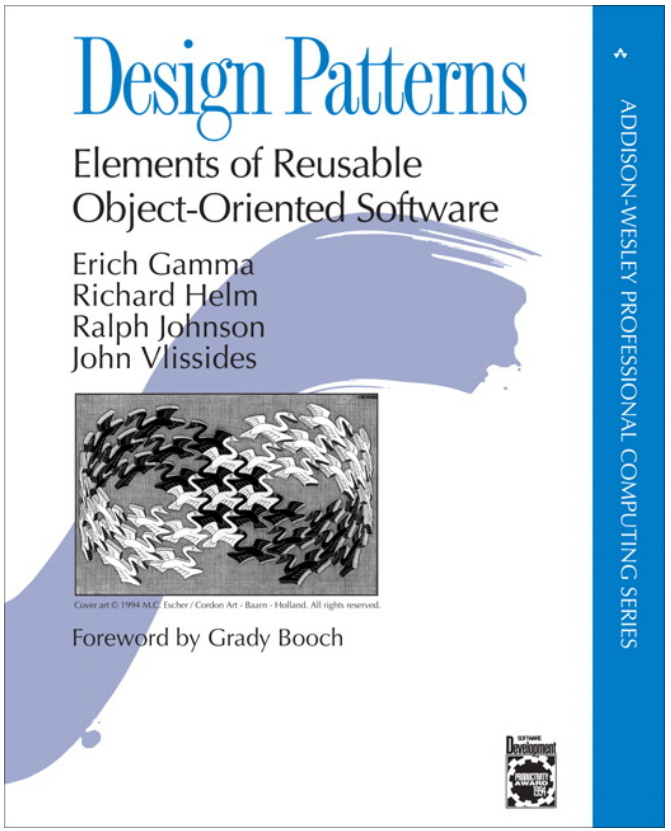
⇒ FeatureHouse / Fuji

Design Patterns

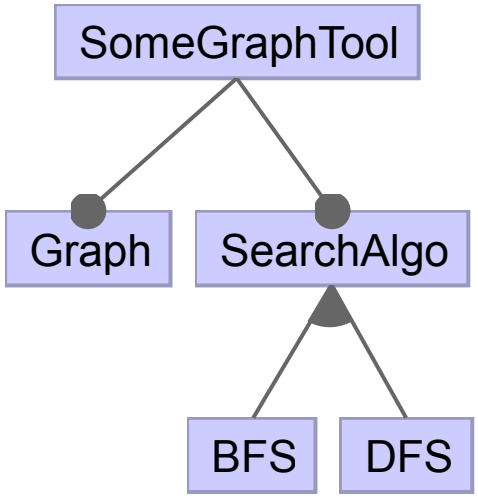


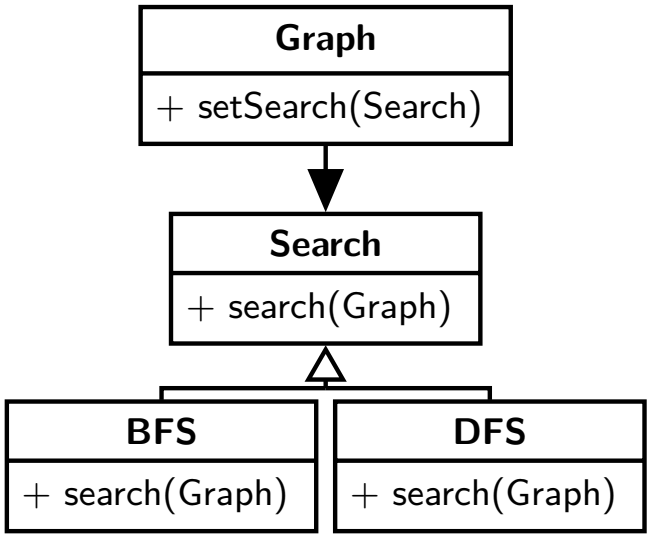
Design Patterns





Feature-Oriented Design Patterns





⇒ One-to-one mapping between Features & Strategies[1]

[1] Sven Schuster, Sandro Schulze, *Object-Oriented Design in Feature-Oriented Programming*, FOSD'12

Feature Template Method

Feature *DirectedOnlyVertices*

```
1 public class Graph {
2   public void run(Vertex s) {
3     /*empty */
4   }
5 }
```

Feature *TestProg*

```
1 class Main {
2   public static void main( String[] args ) {
3     Graph g = new Graph();
4     /* ... */
5     g.run(g.getVertices().next());
6   }
7 }
```

Feature *Number*

```
1  public class Graph {
2    public void run(Vertex s) {
3      System.out.println("Number");
4      NumberVertices();
5      original(s);
6    }
7    public void NumberVertices() {
8      /* ... */
9    }
10 }
```

⇒ Implicit constraint to fill/refine template method in later features[2]

[2] Sven Schuster, Sandro Schulze, *Object-Oriented Design in Feature-Oriented Programming*, FOSD'12

Feature Template Method

Feature *DirectedOnlyVertices*

```
1  public class Graph {
2    public void run(Vertex s) {
3      /*empty */
4    }
5  }
```

Feature *TestProg*

```
1  class Main {
2    public static void main( String[] args ) {
3      Graph g = new Graph();
4      /* ... */
5      g.run(g.getVertices().next());
6    }
7  }
```

Feature *Number*

```
1  public class Graph {
2    public void run(Vertex s) {
3      System.out.println("Number");
4      NumberVertices();
5      original(s);
6    }
7    public void NumberVertices() {
8      /* ... */
9    }
10 }
```

⇒ Implicit constraint to fill/refine template method in later features[2]

[2] Sven Schuster, Sandro Schulze, *Object-Oriented Design in Feature-Oriented Programming*, FOSD'12

Feature Template Method

Feature *DirectedOnlyVertices*

```
1  public class Graph {
2    public void run(Vertex s) {
3      /*empty */
4    }
5  }
```

Feature *TestProg*

```
1  class Main {
2    public static void main( String[] args ) {
3      Graph g = new Graph();
4      /* ... */
5      g.run(g.getVertices().next());
6    }
7  }
```

Feature *Number*

```
1  public class Graph {
2    public void run(Vertex s) {
3      System.out.println("Number");
4      NumberVertices();
5      original(s);
6    }
7    public void NumberVertices() {
8      /* ... */
9    }
10 }
```

⇒ Implicit constraint to fill/refine template method in later features[2]

[2] Sven Schuster, Sandro Schulze, *Object-Oriented Design in Feature-Oriented Programming*, FOSD'12

Research Questions

⇒ **RQ1:** *To what extent do design patterns exist in feature-oriented SPLs?*

⇒ **RQ2:** *Do feature interactions occur in the context of design patterns and if so, how do the features interact?*

Setup & Methodology

Study

- Automated pattern detection technique

Setup & Methodology

Study

- Automated pattern detection technique
- SPLs of different sizes and domains

SPLs

- Ahead
- BerkeleyDB
- GameOfLife
- GPL
- GUIDSL
- TankWar
- Violet

Setup & Methodology

Study

- Automated pattern detection technique
- SPLs of different sizes and domains
- Common design patterns

SPLs

- Ahead
- BerkeleyDB
- GameOfLife
- GPL
- GUIDSL
- TankWar
- Violet

Patterns

- Visitor
- Observer
- Strategy
- FTM

Design Pattern Detection for FOP

Automated detection approach

- Adapted from Heuzeroth et al.
- Family-based $\Rightarrow$ static analysis

Design Pattern Detection for FOP

Automated detection approach

- Adapted from Heuzeroth et al.
- Family-based ⇒ static analysis
- Searching for tuples of elements using Fuji AST

Visitor Pattern

```
VisitorCandidate = {
  Visitor.visit(ConcreteElement),
  ConcreteVisitor.visit(ConcreteElement),
  Element.accept(Visitor),
  ConcreteElement.accept(Visitor)
}
```

Design Pattern Detection for FOP

Automated detection approach

- Adapted from Heuzeroth et al.
- Family-based ⇒ static analysis
- Searching for tuples of elements using Fuji AST
- Manual elimination of false positives

Visitor Pattern

```
VisitorCandidate = {
  Visitor.visit(ConcreteElement),
  ConcreteVisitor.visit(ConcreteElement),
  Element.accept(Visitor),
  ConcreteElement.accept(Visitor)
}
```

Design Pattern Detection - Visitor in GPL

Feature *DFS*

```
1  public class WorkSpace {
2    public void preVisitAction(Vertex v) {}
3  }
4
5  public class Vertex {
6    public void nodeSearch(WorkSpace w) {
7      /*...*/
8      w.preVisitAction(this);
9      /*...*/
10   }
11 }
```

Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
```

Design Pattern Detection - Visitor in GPL

`Visitor.visit(ConcreteElement)`

Feature *DFS*

```
1  public class WorkSpace {
2    public void preVisitAction(Vertex v) {}
3  }
4
5  public class Vertex {
6    public void nodeSearch(WorkSpace w) {
7      /*...*/
8      w.preVisitAction(this);
9      /*...*/
10   }
11 }
```

Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
```

Design Pattern Detection - Visitor in GPL

`Element.accept(Visitor)`

Feature *DFS*

```
1  public class WorkSpace {
2    public void preVisitAction(Vertex v) {}
3  }
4
5  public class Vertex {
6    public void nodeSearch(WorkSpace w) {
7      /*...*/
8      w.preVisitAction(this);
9      /*...*/
10   }
11 }
```

Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
```

Design Pattern Detection - Visitor in GPL

`ConcreteVisitor.visit(ConcreteElement)`

Feature *DFS*

```
1  public class WorkSpace {
2    public void preVisitAction(Vertex v) {}
3  }
4
5  public class Vertex {
6    public void nodeSearch(WorkSpace w) {
7      /*...*/
8      w.preVisitAction(this);
9      /*...*/
10   }
11 }
```

Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
```

Design Pattern Detection - Visitor in GPL

`ConcreteElement.accept(Visitor)`

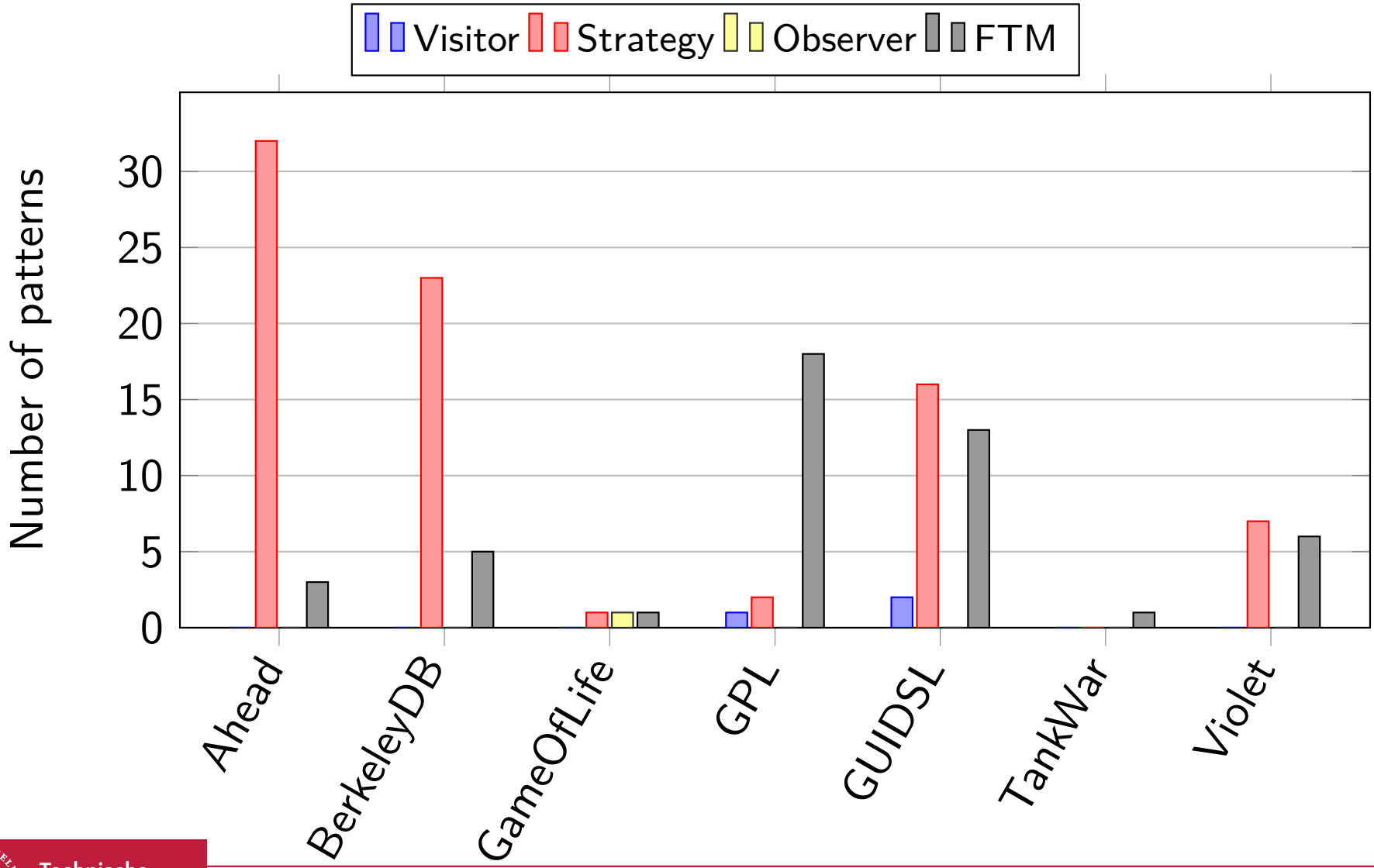
Feature *DFS*

```
1  public class WorkSpace {
2    public void preVisitAction(Vertex v) {}
3  }
4
5  public class Vertex {
6    public void nodeSearch(WorkSpace w) {
7      /*...*/
8      w.preVisitAction(this);
9      /*...*/
10   }
11 }
```

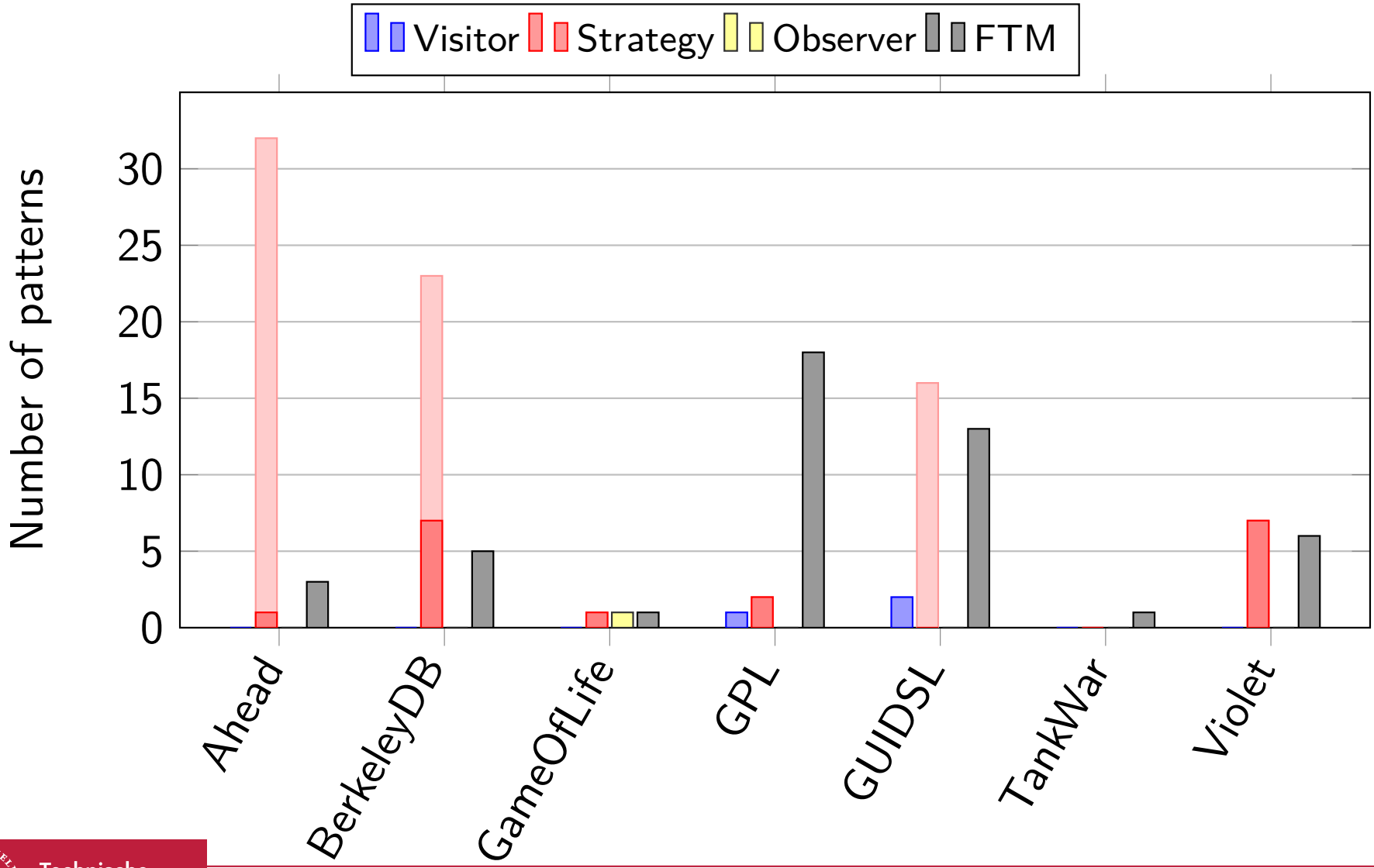
Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
```

Results - Complete Number of Patterns



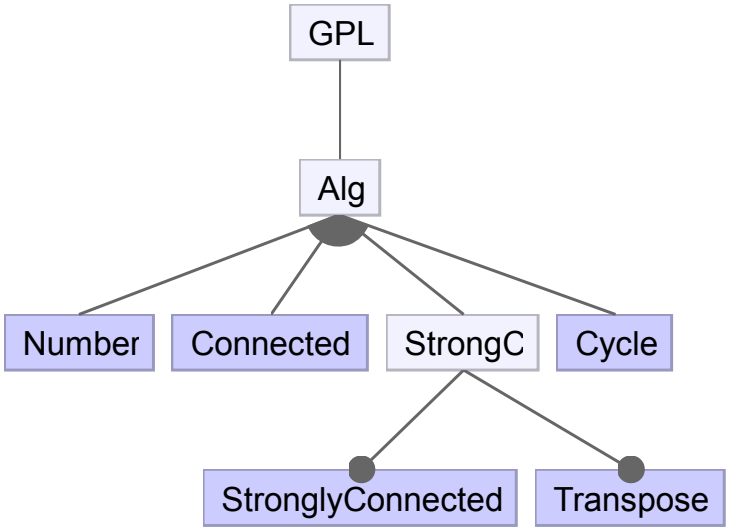
Results - Number of decomposed Patterns



Results - Decomposition of Visitor

Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
```



Feature *Cycle*

```
1  public class CycleWorkSpace extends WorkSpace {
2    public void preVisitAction(Vertex v) {
3      /* ... */
4    }
5  }
```

Feature *Connected*

```
1  public class ConnectedWorkSpace extends WorkSpace {
2    public void preVisitAction(Vertex v) {
3      /* ... */
4    }
5  }
```

Results - Feature Template Method

Feature *DFS*

```
1  public class WorkSpace {
2    public void preVisitAction}(Vertex v) {}
3  }
4  public class Vertex {
5    public void nodeSearch(WorkSpace w) {
6      /*...*/
7    }
8  }
9  public class Graph {
10   public void GraphSearch(WorkSpace w) {
11     /*...*/
12     v.nodeSearch(w);
13     /*...*/
14   }
15 }
```

Feature *DirectedOnlyVertices*

```
1  public class Graph {
2    public void run(Vertex s) {
3      /*empty */
4    }
5  }
```

Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
12
13 public class Graph {
14   public void run(Vertex s) {
15     System.out.println("Number");
16     NumberVertices();
17     original(s);
18   }
19   public void NumberVertices() {
20     GraphSearch(new NumberWorkSpace());
21   }
22 }
```

Results - Feature Template Method

Feature *DFS*

```
1  public class WorkSpace {
2    public void preVisitAction}(Vertex v) {}
3  }
4  public class Vertex {
5    public void nodeSearch(WorkSpace w) {
6      /*...*/
7    }
8  }
9  public class Graph {
10   public void GraphSearch(WorkSpace w) {
11     /*...*/
12     v.nodeSearch(w);
13     /*...*/
14   }
15 }
```

Feature *DirectedOnlyVertices*

```
1  public class Graph {
2    public void run(Vertex s) {
3      /*empty */
4    }
5  }
```

Feature *Number*

```
1  public class NumberWorkSpace extends WorkSpace {
2    int vertexCounter;
3    public NumberWorkSpace() {
4      vertexCounter = 0;
5    }
6    public void preVisitAction(Vertex v) {
7      if (v.visited != true) {
8        v.VertexNumber = vertexCounter++;
9      }
10   }
11 }
12
13 public class Graph {
14   public void run(Vertex s) {
15     System.out.println("Number");
16     NumberVertices();
17     original(s);
18   }
19   public void NumberVertices() {
20     GraphSearch(new NumberWorkSpace());
21   }
22 }
```

Results - Summary

Design Patterns in SPLs

- Design Patterns do occur in feature-oriented SPLs...
  - ... frequently.
  - ... across several features.

Feature Interactions within Design Patterns

- Addition of concrete pattern classes in sibling features
- No refinements of the actual pattern implementation
- Feature Template Method to register/call new objects

Towards Guidelines

Application Scenarios

- Similar to design patterns in OOP
  - Visitor to traverse through grammar
  - Observer to realize MVC
  - Strategy to interchange similar algorithms

Feature Structure

- Concrete classes of decomposed patterns introduced by sibling features
- Design patterns are encapsulated by feature group

$\Rightarrow$ Reflection of design pattern in feature model

$\Rightarrow$ Synergy of modularity of variability model and Implementation

Conclusion & Future Work

We have...

... introduced the idea of structural feature interaction patterns

... proposed a detection technique for design patterns in FOP

... conducted a case study detecting structural feature interaction within design patterns

... introduced ideas for guidelines

Conclusion & Future Work

We have...

... introduced the idea of structural feature interaction patterns

... proposed a detection technique for design patterns in FOP

... conducted a case study detecting structural feature interaction within design patterns

... introduced ideas for guidelines

We should...

... improve and extend the detection technique

... conduct more studies do derive more detailed guidelines

... reason about how to apply such guidelines (automatically)?