ON THE CONGRUENCE OF MODULARITY AND CODE COUPLING

University of Trier

Fabian Beck and Stephan Diehl
Modular Software Systems

„For human beings, the only way to manage a complex system or solve a problem is to break it up.“

(Baldwin and Clark, 2000)

Software systems

- Complex systems
- Modularized on different levels
Modularity Principles

- **Low Coupling and High Cohesion** (Stevens, Myers, Constantine, 1974)
  few dependencies between modules (low coupling), many dependencies in the single modules (high cohesion)

- **Information Hiding** (Parnas, 1971/72)
  hide design decisions into modules

- **Conway’s Law** (Conway, 1968)
  the modularization reflects the organization of the development team

and others  RC Martin’s principles of package design, Separation of Concerns, Domain Knowledge, etc.
Research Question

Modularity principles

Theory

Reality
Research Question

Modularity principles

Which principles are really used in practice?
Approach

- **Empirical study** on software projects
  - 16 open source Java systems
  - modularity = classes contained in packages
- Analyze different concepts of code coupling
  - Coupling concepts are related to design principles
- Measure the congruence of coupling and modularity
  - Infer information on the usage of design principles
Code Coupling Concepts
Example

```java
/**
 * @author fabian
 */
class A {
    String name;
    B b = new B();

    void doSomething() {
        b.doSomethingElse();
    }

    String toString() {
        return name;
    }
}

/**
 * @author fabian
 */
class B {
    String name;

    void doSomethingElse() {
        ...
    }

    String toString() {
        return name;
    }
}
```
Example

//**
 //  * @author fabian
 //  */
 class A {
     String name;
     B b = new B();

     void doSomething() {
         b.doSomethingElse();
     }

     String toString() {
         return name;
     }
 }

//**
 //  * @author fabian
 //  */
 class B {
     String name;

     void doSomethingElse() {
         ...
     }

     String toString() {
         return name;
     }
 }
Example

```java
/**
 * @author fabian
 */
class A {
    String name;

    B b = new B();

    void doSomething() {
        b.doSomethingElse();
    }

    String toString() {
        return name;
    }
}
```
Example

```java
/**
 * @author fabian
 */
class A {
    String name;
    B b = new B();

    void doSomething() {
        b.doSomethingElse();
    }

    String toString() {
        return name;
    }
}
```

```java
/**
 * @author fabian
 */
class B {
    String name;

    void doSomethingElse() {
        ...
    }

    String toString() {
        return name;
    }
}
```

Code clone
Example

```java
/**
 * @author fabian
 */
class A {
    String name;
    B b = new B();

    void doSomething() {
        b.doSomethingElse();
    }

    String toString() {
        return name;
    }
}

/**
 * @author fabian
 */
class B {
    String name;

    void doSomethingElse() {
        ...
    }

    String toString() {
        return name;
    }
}
```
Example

```java
/**
 * @author fabian
 */
class A {
    String name;
    B b = new B();
    void doSomething() {
        b.doSomethingElse();
    }
    String toString() {
        return name;
    }
}

/**
 * @author fabian
 */
class B {
    String name;
    void doSomethingElse() {
        ...
    }
    String toString() {
        return name;
    }
}

Code coupling is **multi-dimensional**.

I. Structural Dependencies
II. Evolution & Ownership
III. Code Clones & Semantic
I. Structural Dependencies

Structural static code dependencies

Inheritance

Aggregation

Usage

Direct

Indirect (fan-out)

\[ 9 \text{ combinations of structural dependencies} \]
II. Evolution & Ownership

- Evolutionary Coupling (co-change)
  Two classes are coupled if they were frequently changed together.

- Code Ownership
  Two classes are coupled if they have similar owners.
III. Code Clones & Semantic

- **Code Clones**
  
  *Two classes are coupled if they share a code clone.*

- **Semantic Similarity**
  
  *Two classes are coupled if they use similar vocabulary.*
## Relationships to Modularity Principles

<table>
<thead>
<tr>
<th>Coupling Concept</th>
<th>Related Design Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct structural dependencies</td>
<td>Low Coupling and High Cohesion</td>
</tr>
<tr>
<td>Indirect structural dependencies (external)</td>
<td>Separation of Concerns</td>
</tr>
<tr>
<td>Indirect structural dependencies (internal)</td>
<td>Information Hiding</td>
</tr>
<tr>
<td>Evolutionary coupling</td>
<td>Information Hiding</td>
</tr>
<tr>
<td>Code ownership</td>
<td>Conway’s Law</td>
</tr>
<tr>
<td>Code clones</td>
<td>?</td>
</tr>
<tr>
<td>Semantic similarity</td>
<td>Domain Knowledge</td>
</tr>
</tbody>
</table>

based on theoretical considerations
Coupling-Modularity Congruence

Coupling Concept I

Coupling Concept II
Coupling-Modularity Congruence

Coupling Concept I

Coupling Concept II

congruent

non-congruent
Coupling-Modularity Congruence Congruence Metric (simplified)
The percentage of couplings not crossing module borders

\[
\text{congruence} = \frac{|\rightarrow|}{|\rightarrow| + |\rightarrow|}
\]

Non-simplified: considering the hierarchical structure of the modularization
Results

Coupling-modularity congruence (median)

- Direct structural
- Indirect structural (external)
- Indirect structural (internal)
- Evolutionary coupling
- Code ownership
- Code clones
- Semantic similarity

Baseline
High congruence for:
- structural coupling
- evolutionary coupling
- code clones
Results

**Inheritance** reaches the highest congruence

1. **internal indirect structural coupling**
2. **direct structural coupling**
Different Types of Modules

Similar congruence pattern for all package types
(even for utility packages)
Modularity Principles

- None of the principles is dominating
- Principles are applied to all types of modules alike

**Stronger principles**

- **Low Coupling and High Cohesion**
  *Explanation:* High congruence of direct structural dependencies

- **Information Hiding**
  *Explanation:* High congruence of internal fan-out and evolutionary coupling

**Weaker principles**

- **Conway’s Law, Separation of Concerns, Domain Knowledge**
Summary

- Empirical study on 16 software projects
- High congruence between coupling and modularity
  - structural dependencies (in particular, inheritance)
  - evolutionary coupling
  - code clones
- Different module types
  - no type-specific congruence patterns

Which modularity principles are really used in practice?
  - Low Coupling and High Cohesion
  - Information Hiding