Utilization of Method Graphs to Measure Cohesion in Object Oriented Software

Atakan Aral, Tolga Ovatman

Istanbul Technical University
Faculty of Computer and Informatics
Outline

• Introduction
• Motivation
• Contribution
• Method Graphs
• Clustering & Similarity Measurement
• Results & Discussion
• Conclusion
• Future Work
Introduction

- Cohesion is the *harmony* of software components in performing responsibilities.
- Hard to measure quantitatively.
- Cohesion of a class is usually measured by examining *common data usage of its methods*.

![Diagram showing different cohesion levels.](attachment:image.png)
Motivation

• Conventional techniques analyze *object code* and require *compilation*
• Hard to detect data usage from *source code* – Indirect ways to access a variable
• Prevents evaluating incomplete code

• Our aim is to derive a *pure source code based cohesion measure*
A cohesion measure that
– depends purely on static analysis
– considers only method-to-method relations
– is graph based and software-wide
– is highly correlated to LCOM metric
• Cooperating Methods (CM) relation
  – involves method couples that are *called together* from another *host method*.

```java
Class A{
    MA1(){
        MA2 ();
        B.MB ();
    }
    MA2 (){
        B.MB ();
        MA3 ();
        MA4 ();
    }
    MA3 (){}
    MA4 (){}
}
```
Method Graphs

• Method Call (MC) relation
  – involves method couples that call one another.

```java
class A {
    MA1 () {
        MA2 () ;
        B.MB () ;
    }
    MA2 () {} 
    MA3 () {
        MA4 () ;
    }
    MA4 () {
        MA1 () ;
    }
}
```
Method Graphs

- Method Layout (ML) relation
  - involves method couples that are in the same class.

```plaintext
Class A{
    MA1()
    MA2()
    MA3()
    MA4()
}

Class B{
    MB1()
    MB2()
}
```
Method Graphs

• Internal Call (IC) relation
  – involves method couples that call one another and are in the same class.

```java
Class A{
    MA1(){
        MA2();
        B.MB();
    }
    MA2(){}
    MA3(){
        MA4();
    }
    MA4(){
        MA1();
    }
}
```
Clustering & Similarity

• 4 graphs are clustered using Weak Component Clustering

• 6 clustering couples are compared using adjusted Rand index
  – CM—MC: Are the methods calling each other, also called together from another body?
  – IC—ML: What is the level of fragmentation within classes?
• 14 most popular open-source Java projects from *GitHub* and *SourceForge*

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>RAND INDICES</th>
<th>LCOM HS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$G_{CM} - G_{MC}$</td>
<td>$G_{CM} - G_{IC}$</td>
</tr>
<tr>
<td>borg</td>
<td>0.183</td>
<td>0.012</td>
</tr>
<tr>
<td>clojure</td>
<td>0.210</td>
<td>0.077</td>
</tr>
<tr>
<td>freemind</td>
<td>0.181</td>
<td>0.010</td>
</tr>
<tr>
<td>freeplane</td>
<td>0.234</td>
<td>0.006</td>
</tr>
<tr>
<td>jabref</td>
<td>0.191</td>
<td>0.005</td>
</tr>
<tr>
<td>javaParser</td>
<td>0.325</td>
<td>0.507</td>
</tr>
<tr>
<td>jedit</td>
<td>0.334</td>
<td>0.124</td>
</tr>
<tr>
<td>junit</td>
<td>0.057</td>
<td>0.018</td>
</tr>
<tr>
<td>logisim</td>
<td>0.171</td>
<td>0.004</td>
</tr>
<tr>
<td>pdfsam</td>
<td>0.374</td>
<td>0.007</td>
</tr>
<tr>
<td>StickyRoyale</td>
<td>0.414</td>
<td>0.364</td>
</tr>
<tr>
<td>tuxguitar</td>
<td>0.245</td>
<td>0.007</td>
</tr>
<tr>
<td>wurfl</td>
<td>0.109</td>
<td>0.049</td>
</tr>
<tr>
<td>ytd2</td>
<td>0.308</td>
<td>0.181</td>
</tr>
</tbody>
</table>
Results & Discussion

<table>
<thead>
<tr>
<th></th>
<th>$G_{CM}$ _ $G_{MC}$</th>
<th>$G_{CM}$ _ $G_{IC}$</th>
<th>$G_{CM}$ _ $G_{ML}$</th>
<th>$G_{MC}$ _ $G_{IC}$</th>
<th>$G_{MC}$ _ $G_{ML}$</th>
<th>$G_{IC}$ _ $G_{ML}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_{CM}$ _ $G_{IC}$</td>
<td>0.585</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_{CM}$ _ $G_{ML}$</td>
<td>0.320</td>
<td>0.808</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_{MC}$ _ $G_{IC}$</td>
<td></td>
<td>0.644</td>
<td>0.876</td>
<td>0.445</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_{MC}$ _ $G_{ML}$</td>
<td>0.430</td>
<td>0.882</td>
<td>0.938</td>
<td>0.602</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_{IC}$ _ $G_{ML}$</td>
<td>0.225</td>
<td>0.593</td>
<td>0.778</td>
<td>0.245</td>
<td>0.677</td>
<td></td>
</tr>
<tr>
<td>MED LCOM</td>
<td>0.767</td>
<td>0.474</td>
<td>0.211</td>
<td>0.572</td>
<td>0.393</td>
<td>0.168</td>
</tr>
<tr>
<td>AVG LCOM</td>
<td>0.770</td>
<td>0.476</td>
<td>0.168</td>
<td>0.580</td>
<td>0.393</td>
<td>0.233</td>
</tr>
</tbody>
</table>
• Calling two methods, that call each other, from the body of another method, decreases the overall cohesion of software
Conclusion

• We suggest a novel technique that allows to measure **software-wide** cohesion.

• It uses **static source code analysis**.

• Results indicate that its correlation to LCOM HS is 77%.
Future Work

• to adapt our measure to class level
• to use direction—weight data to improve accuracy
• to build a mathematical basis for the relation between our measure and the LCOM metric
• to study on the other correlated groups
Thank you!