METHOD DIFFERENTIATOR USING SLICE-BASED COHESION METRICS

Akira Goto (Osaka University)
Norihiro Yoshida (Nara Institute of Science and Technology)
Masakazu Ioka (Osaka University)
Eunjong Choi (Osaka University)
Katsuro Inoue (Osaka University)
Source code differentiation

- Developers need to understand differences between a pair of methods in Java source code
  - When developers refactor a method pair into a module
  - When developers modify a method pair simultaneously
    - For enhancement, bug fix

However, manual differentiation is tedious and error prone.
Example of similar methods

It is hard to understand the differences between the method pair

```java
public void similarMethodA(){
    if(finalBuffer.remaining() < 8){
        while(finalBuffer.remaining() > 0){
            finalBuffer.put((byte)0);
        }
    }
    finalBuffer.position(0);
    transform(finalBuffer);
    finalBuffer.position(0);
}
finalBuffer.putLong(length << 3);
finalBuffer.position(0);
transform(finalBuffer);

public void similarMethodB(){
    if(finalBuffer.remaining() < 8){
        while(finalBuffer.remaining() > 0){
            finalBuffer.put((byte)0);
        }
    }
    finalBuffer.position(0);
    transform(finalBuffer.array(),0);
    finalBuffer.position(0);
}
finalBuffer.putLong(length << 3);
finalBuffer.position(0);
transform(finalBuffer.array(),0);
```
Automatic code differentiation

- **Textual Diff**
  - e.g., GNU Diff

- **Syntactic Diff**
  - e.g., Eclipse

- **PDG-based Diff** [Xue2011]
  - Comprehension support for differences in terms of dependency type
  - For a pair of duplicated code fragments

Example of similar methods

Syntactic differences are insufficient to support refactoring, enhancement and bug fix.

```java
public void similarMethodA(){
   if(finalBuffer.remaining() < 8){
      while(finalBuffer.remaining() > 0){
         finalBuffer.put((byte)0);
      }
   finalBuffer.position(0);
   transform(finalBuffer);
   finalBuffer.position(0);
}
   finalBuffer.putLong(length << 3);
   finalBuffer.position(0);
   transform(finalBuffer);

   :
}

public void similarMethodB(){
   if(finalBuffer.remaining() < 8){
      while(finalBuffer.remaining() > 0){
         finalBuffer.put((byte)0);
      }
   finalBuffer.position(0);
   transform(finalBuffer.array(),0);
   finalBuffer.position(0);
}
   finalBuffer.putLong(length << 3);
   finalBuffer.position(0);
   transform(finalBuffer.array(),0);

   :}
```
Example of similar methods

Syntactic differences are insufficient to support refactoring, enhancement and bug fix.

Which part should be merged into one method?
Example of similar methods

Syntactic differences are insufficient to support refactoring, enhancement and bug fix.

```
public void similarMethodA(){
    if(finalBuffer.remaining() < 8){
        while(finalBuffer.remaining() > 0){
            finalBuffer.put((byte)0);
        }
    }
    finalBuffer.position(0);
    transform(finalBuffer);
    finalBuffer.position(0);
}
finalBuffer.putLong(length << 3);
finalBuffer.position(0);
transform(finalBuffer);
```

```
public void similarMethodB(){
    if(finalBuffer.remaining() < 8){
        while(finalBuffer.remaining() > 0){
            finalBuffer.put((byte)0);
        }
    }
    finalBuffer.position(0);
    transform(finalBuffer.array(),0);
    finalBuffer.position(0);
}
finalBuffer.putLong(length << 3);
finalBuffer.position(0);
transform(finalBuffer.array(),0);
```

Which part should be modified simultaneously for enhancement and bug fix?
Motivation

• Developers often try to understand which part of a source file corresponds to each functionality.

  • During refactoring
    • A code fragment that implements a single functionality is a good candidate for code extraction.
  • During enhancement and bug fix
    • Simultaneous editing should be considered for each code fragment that implements a single functionality.

The existing differentiators do not take into account functionalities implemented in a source file.
Research overview

• We have developed a method differentiator MEDICO.

• The differentiation is based on slice-based cohesion metrics.
  • In order to identify a set of code fragments, each of which implements a single functionality

• Cohesion metric is generally defined as
  • a measure which expresses, in order for each of the constituent parts to realize a specific feature, the extent to which they work together. [Stevens1974]

Cohesion metrics for code fragment

• Most of the existing cohesion metrics are aimed at calculating the degree of the cohesion of a class/method.

• However, cohesion metrics for calculating the cohesion of a code fragment are needed.
  • In order to identify a code fragment that includes a set of cohesive statements and implements a single functionality

• We modified slice-based cohesion [Weiser1981] to calculate the cohesion of a code fragment.
  • Calculate slice-based metrics after the code fragment is extracted
  • Tightness, Coverage, Overlap → FTightness, FCoverage, FOverlap

Steps of the differentiation by MEDICO

Input: A pair of Java methods

AST-based Differencing

Enumerating all possible differences

Ranking of differences using cohesion metrics

Rank
1
2
3
4