Studying Refactoring supported by *RefactoringMiner* Detection Tool

Flavia Coelho, Tiago Massoni and Everton Alves
Software Practices Laboratory (SPLab)
Federal University of Campina Grande - Brazil

July 1, 2020

Abstract

We run *RefactoringMiner*[1] - a refactoring detection tool which operates on version control commits [1] - for some codes inspired by [2, 3, 4] and ones extracted from experiments using *SaferRefactor* and *JDolly* [5]. In this report, we present our findings.

1 Studying *Inline Method* refactoring

In Listing 1, the inline method refactoring happens on a statement that wrap an expression. Thus, in this case, abstraction was applied.

Question. According Table 2 [1], we suggests that it was obtained

$$| M |=| U_{T_1} |$$

but it was expected

$$| M |>| U_{T_1} |.$$

Would be this suggestion correct?

2 Studying behavior-preserving transformations

Listings 2-8 illustrate a sample of codes extracted from experiments conducted by Soares et al. [5].

These experiments evaluate approaches for identifying behavior-preserving transformations activities on software repositories. The codes are available on `https://github.com/flaviacoelho/refactorings-examples`.

2.1 Studying *Rename Class* refactoring

In Listing 2, after the transformation, the method *test* returns 20 instead of 10. Thus, the behavior was changed.

[1] available on https://github.com/tsantalis/RefactoringMiner

before refactoring

```
int tmp = 0;
public int getZeroValue() {
    return isZero() ? 1 : 0;
}
public boolean isZero() {
    return(tmp == 0);
}
```

after refactoring

```
int tmp = 0;
public int getZeroValue() {
    return (tmp == 0) ? 1 : 0;
}
```

Listing 1: [False Negative] Inline Method
`https://github.com/flaviacoelho/refactorings-toy/commit/14d1ad4a9b4695976a3db45f43aed7a6c78ea211`

2.2 Studying *Rename Method* refactoring

After rename n method to k, the m method returns 0 instead of 1, as shown in Listing 3. Hence, the transformation changes the program behavior.

2.3 Studying *Push Down Method* refactoring

Listing 4 shows that pushing down a method to a class in another package may shadow a class declaration leading to a behavioral change. For this reason before the transformation, method m() yields 1, but after that, it yields 0.

2.4 Studying *Push Down Field* refactoring

Pushing down a field that hides other field may changes program behavior. It follows that after the transformation, the method test returns 10 instead of 20, as we can see in Listing 5. So, the transformation changes the program behavior.

2.5 Studying *Pull Up Method* refactoring

As shown in Listing 6, pulling up a method enables overloading of a private method changing program behavior because after the transformation, the test method returns 2 instead of 1.

2.6 Studying *Pull Up Field* refactoring

Apply the pull up field refactoring to a field f moves all fields f of the subclasses to the super class. If one of the fields is initialized with different value, the

behavior of the program may change.

In Listing 7, the method test returns 10 instead of 20. Therefore, the refactoring changes the program behavior.

2.7 Studying *Move Method* refactoring

In the Listing 8, applying the move method refactoring to move k(int) to B, the test() method returns 2 instead of 1. As a result, moving a method to another class may change program behavior due to an overloading. Hence, this transformation changes the program behavior.

References

[1] Tsantalis, N., Mansouri, M., Eshkevari, L. M., Mazinanian, D., and Dig, D. *Accurate and efficient refactoring detection in commit history*. In Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 2018.

[2] Deitel, P. and Deitel, H. *Java: How to Program*. Tenth Edition. Pearson, 2014.

[3] Fowler, M. *Refactoring: Improving the Design of Existing Programs*. AddisonWesley, 1999.

[4] Eilertsen, A. M. *Making Software Refactorings Safer*. Master Thesis – Department of Informatics – University of Bergen. Bergen, 2016.

[5] G. Soares, R. Gheyi and T. Massoni. *SaferRefactor Experiments*, 2010, `http://www.dsc.ufcg.edu.br/~spg/saferefactor/experiments.html`

before transformation

```java
package a;
public class A {
    public int k() {
        return 20;
    }
}

package a;
import b.*;
public class B extends C {
    public int test() {
        return k();
    }
}

package b;
public class C {
    public int k() {
        return 10;
    }
}
```

after transformation

```java
package a;
public class C { //renamed class
    public int k() {
        return 20;
    }
}

package a;
import b.*;
public class B extends C {
    public int test() {
        return k();
    }
}

package b;
public class C {
    public int k() {
        return 10;
    }
}
```

Listing 2: [False Positive] Rename Class
`https://github.com/flaviacoelho/refactorings-examples/commit/73fa25d00ef11615b0322a294d1bc5ad21d48a31`

before transformation

```
package p2;
import p1.*;
public class B extends A {
  protected long n(int a){
    return 0;
  }
  public long m(){
    return this.k(2);
  }
}
package p1;
public class A {
  public long k(long a){
    return 1;
  }
}
```

after transformation

```
package p2;
import p1.*;
public class B extends A {
  protected long k(int a){ //n renamed to k
    return 0;
  }
  public long m(){
    return this.k(2);
  }
}

package p1;
public class A {
  public long k(  long a){
    return 1;
  }
}
```

Listing 3: [False Positive] Rename Method
`https://github.com/flaviacoelho/refactorings-examples/commit/f706be1cb4907724096bac04e07f3be889d4d33d`

before transformation

```
package p1;
import p2.*;
public class B extends A {
  protected long k(int a){
    return 0;
  }
  public long test(){
    return m();
  }
}
package p2;
public class B extends A {
}
package p2;
import p1.*;
public class A {
  public long k(long a){
    return 1;
  }
  public long m(){
    return new B().k(2);
  }
}
```

after transformation

```
package p1;
import p2.*;
public class B extends A {
  protected long k(int a){
    return 0;
  }
  public long test(){
    return m();
  }
public long m() {
    return new B().k(2);
  }
}
package p2;
public class B extends A {
    public long m() {
        return new B().k(2);
      }
}
package p2;
import p1.*;
public class A {
  public long k(long a){
    return 1;
  }
}
```


Listing 4: [False Positive] Push Down Method
https://github.com/flaviacoelho/refactorings-examples/commit/505c6fa16eba14d29e2c6f65fb422d9725862602

before transformation

```
1  public class A {
2    public int k = 10;
3  }
4  public class B extends A {
5    public int k = 20;
6  }
7  public class C extends B {
8    public int test() {
9      return super.k;
10   }
11 }
```

after transformation

```
1  public class A {
2    public int k = 10;
3  }
4  public class B extends A {
5  }
6  public class C extends B {
7    public int k = 20; //pushed down
         field
8    public int test() {
9      return super.k;
10   }
11 }
```

Listing 5: [False Positive] Push Down Field
https://github.com/flaviacoelho/refactorings-examples/commit/42674a9d4f34af60328e8a857301d40f24f75ff0

before transformation

```
public class A {
  public int k(long l) {
    return 1;
  }
  private int k(int l) {
    return 2;
  }
}
public class B extends A {
  public int m() {
    return k(2);
  }
  public int test() {
    return m();
  }
}
```

after transformation

```
public class A {
  public int k(long l) {
    return 1;
  }
  private int k(int l) {
    return 2;
  }
  public int m() { //pulled up
    return k(2);
  }
}
public class B extends A {
  public int test() {
    return m();
  }
}
```

Listing 6: [False Positive] Pull Up Method
`https://github.com/flaviacoelho/refactorings-examples/commit/7a5c5f63f6c9e8a0764cd7efa474634d4cf639f7`

before transformation

```
public class A {
}
public class B extends A {
  public int k = 10;
}
public class C extends A {
  public int k = 20;
  public long test() {
    return k;
  }
}
```

after transformation

```
public class A {
  public int k = 10; //pulled up
}
public class B extends A {
}
public class C extends A {
  public long test() {
    return k;
  }
}
```

Listing 7: [False Positive] Pull Up Field
`https://github.com/flaviacoelho/refactorings-examples/commit/9a9cd138b9f268c2f0749a2556dd114e57278bfe`

before transformation

```
public class A {
  public B b;
  protected long k(int a) {
    return 2;
  }
}
public class B {
  private long k(long a) {
    return 1;
  }
  public long test() {
    return k(2);
  }
}
```

after transformation

```
public class A {
  public B b;
}
public class B {
  private long k(long a) {
    return 1;
  }
  public long test() {
    return k(2);
  }
  protected long k(int a) { //moved from A
    return 2;
  }
}
```

Listing 8: [False Positive] Move Method
https://github.com/flaviacoelho/refactorings-examples/commit/d4c814d3fbee55a55ffc73fec38cc79145b99bf4