Automated Composition of Refactorings

A short demonstration

UNIVERSITY
OF OSLO

Refactoring

Martin Fowler, in his book on refactoring [Fow99], defines a refactoring:

> *Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. [Fow99, p. 53]*

If we leave the motivation behind refactoring out of the definition, it could be rephrased like this:

Definition

A *refactoring* is a transformation done to a program without altering its external behavior.

Composite Refactorings

There are *primitive refactorings*. These refactorings cannot be expressed in terms of other refactorings. And there are *composite refactorings*:

Definition

A *composite refactoring* is a refactoring that can be expressed in terms of two or more other refactorings.

The Extract and Move Method refactoring

This thesis is concentrating on creating a composite refactoring of the *Extract Method* and *Move Method* refactorings. The composition of the two is called the *Extract and Move Method* refactoring.

The Extract Method refactoring

The *Extract Method* refactoring is used to extract a fragment of code from its context and into a new method. A call to the new method is inlined where the fragment was before. It is used to break code into logical units, with names that explain their purpose

```
class C {
  void method() {
    // 1: Some code
    // 2: Fragment
    // 3: More code
  }
}
```

```
class C {
  void method() {
    // 1: Some code
    extractedMethod();
    // 3: More code
  }

  void extractedMethod() {
    // 2: Fragment
  }
}
```

The Move Method refactoring

The *Move Method* refactoring is used to move a method from one class to another. This is useful if the method is using more features of another class than of the class which it is currently defined.

```
class C {
  void method() {
    X x = new X();
    iBelongInX(x);
  }
  void iBelongInX(X x) {
    x.foo(); x.bar();
  }
}

class X {
  void foo(){/*...*/}
  void bar(){/*...*/}
}
```

```
class C {
  void method() {
    X x = new X();
    x.iBelongInX();
  }
}

class X {
  void iBelongInX() {
    foo(); bar();
  }
  void foo(){/*...*/}
  void bar(){/*...*/}
}
```

The Composition

```
// Before
class C {
  void method() {
    X x = new X();
    x.foo(); x.bar();
  }
}

class X {
  void foo(){/*...*/}
  void bar(){/*...*/}
}
```

The Composition

```
// Intermediate step
class C {
  void method() {
    X x = new X();
    extractedMethod(x);
  }
  void extractedMethod(X x) {
    x.foo(); x.bar();
  }
}

class X {
  void foo(){/*...*/}
  void bar(){/*...*/}
}
```

The Composition

```
// Before
class C {
  void method() {
    X x = new X();
    x.foo(); x.bar();
  }
}

class X {
  void foo(){/*...*/}
  void bar(){/*...*/}
}
```

```
// After
class C {
  void method() {
    X x = new X();
    x.extractedMethod();
  }
}

class X {
  void extractedMethod() {
    foo(); bar();
  }
  void foo(){/*...*/}
  void bar(){/*...*/}
}
```

Automation

- Search based
- Heuristics
- Project wide search and perform

Demonstration

- The `LastStatementOfSelectionEndsInReturnOrThrowChecker.visit(IfStatement node)` method
  - Extract and Move on selection
  - Extract and Move, search based, on method
- The `no.uio.ifi.refaktor` project
  - Extract and Move, search based, over whole project

What is left

- Write technical section
- Write up argument for correctness
- Define the final case study
- Run unit tests before and after change
- Make more examples
- Metrics?
- . . .

Bibliography

[Fow99] Martin Fowler. *Refactoring: improving the design of existing code*. Reading, MA: Addison-Wesley, 1999. ISBN: 0201485672.