On the separation of queries from modifiers

Ran Ettinger, IBM Research – Haifa

CREST Open Workshop, University College London

24 January 2011

Separate Query from Modifier (SQfM)

- A refactoring technique by Martin Fowler*
  - "*You have a method that returns a value but also changes the state of an object.*"
  - "Create two methods, one for the query and one for the modification."
- Inspired by Bertrand Meyer's Command Query Separation (CQS)
- This talk:
  - Outline of a first algorithm to support the automation of this refactoring
  - Based on program slicing, with reference to other refactoring techniques
  - A prototype tool integrated into Eclipse
  - Open source implementation in WALA (http://wala.sourceforge.net)
    - Developed by Eli Kfir and Daniel Lemel (Technion, Israel Institute of Technology)
    - Contributions by Alex Libov (Technion), Dima Rabkin and Vlad Shumlin (Haifa University)
    - Based on a slicer for Java by Stephen J. Fink (IBM Research) and the WALA contributors

* See http://www.refactoring.com/catalog/separateQueryFromModifier.html and http://sourcemaking.com/refactoring/separate-query-from-modifier

Fowler’s Example (Before SQfM)

```java
String foundMiscreant(String[] people) {
   for (int i=0; i<people.length; i++) {
      if (people[i].equals("Don")) {
         sendAlert();
         return "Don";
      }
      if (people[i].equals("John")) {
         sendAlert();
         return "John";
      }
   }
   return "";
}
```

```java
void checkSecurity(String[] people) {
   String found = foundMiscreant(people);
   someLaterCode(found);
}
```

Fowler’s Example (After SQfM)

```
String foundPerson(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            return "Don";
        }
        if (people[i].equals("John")) {
            return "John";
        }
    }
    return "";
}
```

```
void sendAlert(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            sendAlert();
            return;
        }
        if (people[i].equals("John")) {
            sendAlert();
            return;
        }
    }
}
```

```
void checkSecurity(String[] people) {
    sendAlert(people);
    String found = foundPerson(people);
    someLaterCode(found);
}
```

Fowler's Example (Beyond SQfM)

```
String foundPerson(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            return "Don";
        }
        if (people[i].equals("John")) {
            return "John";
        }
    }
    return "";
}
```

```
void sendAlert(String[] people) {
    if (! foundPerson(people).equals(""))
        sendAlert();
}
```

```
void checkSecurity(String[] people) {
    sendAlert(people);
    String found = foundPerson(people);
    someLaterCode(found);
}
```

Outline of a Separation Algorithm

1. Optional: Add a temporary variable for the returned value
2. Extract the slice of the returned value into a new method (Q), adjusting the original method accordingly
3. Optional: Inline Temp (on the result of Q)
4. Extract Method (for M), after updating its return statements
5. Inline Method (on the original method)

Step 1: Add a Temporary Variable

```
String foundMiscreant(String[] people) {
    String result;
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            sendAlert();
            result = "Don";
            return result;
        }
        if (people[i].equals("John")) {
            sendAlert();
            result = " John";
            return result;
        }
    }
    result = "";
    return result;
}
```

```
void checkSecurity(String[] people) {
    String found = foundMiscreant(people);
    someLaterCode(found);
}
```

Step 2: Extract Q (Slice of `result`)

```java
String foundPerson(String[] people) {
    String result;
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            result = "Don";
            return result;
        }
        if (people[i].equals("John")) {
            result = "John";
            return result;
        }
    }
    result = "";
    return result;
}
```

```java
String foundMiscreant(String[] people) {
    String result = foundPerson(people);
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            sendAlert();
            return result;
        }
        if (people[i].equals("John")) {
            sendAlert();
            return result;
        }
    }
    return result;
}
```

```java
void checkSecurity(String[] people) {
    String found = foundMiscreant(people);
    someLaterCode(found);
}
```

Step 3: Inline Temp (`result`)

```
String foundPerson(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            return "Don";
        }
        if (people[i].equals("John")) {
            return "John";
        }
    }
    return "";
}
```

```
String foundMiscreant(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            sendAlert();
            return foundPerson(people);
        }
        if (people[i].equals("John")) {
            sendAlert();
            return foundPerson(people);
        }
    }
    return foundPerson(people);
}
```

```
void checkSecurity(String[] people) {
    String found = foundMiscreant(people);
    someLaterCode(found);
}
```

Step 4: Extract Method (M)

```
String foundPerson(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            return "Don";
        }
        if (people[i].equals("John")) {
            return "John";
        }
    }
    return "";
}
```

```
String sendAlert(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            sendAlert();
            return;
        }
        if (people[i].equals("John")) {
            sendAlert();
            return;
        }
    }
}
```

```
String foundMiscreant(String[] people) {
    sendAlert();
    return foundPerson(people);
}
```

```
void checkSecurity(String[] people) {
    String found = foundMiscreant(people);
    someLaterCode(found);
}
```

Step 5: Inline Method

```
String foundPerson(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            return "Don";
        }
        if (people[i].equals("John")) {
            return "John";
        }
    }
    return "";
}
```

```
void sendAlert(String[] people) {
    for (int i=0; i<people.length; i++) {
        if (people[i].equals("Don")) {
            sendAlert();
            return;
        }
        if (people[i].equals("John")) {
            sendAlert();
            return;
        }
    }
}
```

```
void checkSecurity(String[] people) {
    sendAlert(people);
    String found = foundPerson(people);
    someLaterCode(found);
}
```

Conditions for Behavior Preservation

- The two new method names must be legal and cause no conflict
- The code of Q must be free of side effects
  - Otherwise, can some measures be taken to prevent the effects?
  - Further SQfM of called methods might be needed, requiring further user interaction
- Legal selection of a method
  - It should be non-void and with side effects (or M would be empty)
  - If it participates in overriding special treatment is needed
    - Example: A Java **`Iterator`**'s **`next()`** method

Some Challenges

- How not to fail when the Query has side effects
  - Idea: assuming Q will follow M, try to reuse some of M's results in Q instead of re-computing them; so it is the slice of the side effects that will be extracted, instead of that of the returned value
- How to minimize code duplication, correctly
  - which extraction technique (of Q or of M) should be preferred?
- How not to fail in the final (Inline Method) step
  - When the call is inside a loop's condition the Modifier's invocation location is non-trivial
  - The Eclipse "Inline" treatment is not always correct