On the automation of challenging refactorings (through advanced method extraction techniques)

Ran Ettinger
(RanE@il.ibm.com, http://researcher.ibm.com/person/il-RanE)
IBM Research – Haifa
In IDC Herzliya’s CS for Real seminar
19 March 2013
Background

• What is Refactoring?
  – The process of gradually improving the design of an existing software system by performing source code transformations that improve its quality in such a way that it becomes easier to maintain the system and reuse parts of it, while preserving the behavior of the original system.

• For example: Extract Method

```java
void printOwing(double amount) {
    printBanner();

    // print details
    print("name:" + _name);
    print("amount:" + amount);
}
```

Source: Martin Fowler's online refactoring catalog
Refactoring to Design Patterns
Enterprise Architecture Patterns
Extract Computation: A Slice-Extraction Refactoring

- **Make a slice of code reusable**
  - Not merely copying and pasting it
  - Update the original code too, like in the “Extract Method” refactoring
- **Turn a non-contiguous slice into a contiguous fragment of code, before applying “Extract Method”**
  - Rearrange the rest of the code
    - Prepare parameters
    - Use slice results
    - Prevent unwanted side-effects
    - Compute further results
  - A kind of **defragmentation**
Extract Computation Example

```java
out.println("<table border=0>");
int start = page * 20;
int end = start + 20;
end = Math.min(end,
    album.getPictures().size());
for (int i = start; i < end; i++) {
    Picture picture = album.getPicture(i);
    printPicture(out, picture);
}
out.println("</table>");
```

```java
int start = page * 20;
int end = start + 20;
end = Math.min(end,
    album.getPictures().size());
Queue<Picture> pictures =
    new LinkedList<Picture>();
for (int i = start; i < end; i++) {
    Picture picture =
    album.getPicture(i);
    pictures.add(picture);
}
display(out, start, end, pictures);
```

```java
void display(PrintStream out, int start,
int end, Queue<Picture> pictures) {
    out.println("<table border=0>");
    for (int i = start; i < end; i++) {
        printPicture(out, pictures.remove());
    }
    out.println("</table>");
}
```
Extract Computation as a Building Block

- **Examples of basic refactorings from Martin Fowler’s catalog**
  - Split Loop
  - Replace Temp with Query
  - Separate Query from Modifier
  - Extract Method (non-contiguous flavor, e.g. near-clone elimination)

- **Examples of refactorings to patterns from Joshua Kerievsky’s catalog**
  - Compose Method
  - Form Template Method

- **Examples of big refactorings from Martin Fowler and Kent Beck**
  - Separate Domain from Presentation
  - Convert Procedural Design to Objects
Split Loop [Fowler]

• **Situation:** “You have a loop that is doing two things.”

• **Recommendation:** “Duplicate the loop.”

• **Link:** [http://www.refactoring.com/catalog/splitLoop.html](http://www.refactoring.com/catalog/splitLoop.html)
Before Split Loop on `averageAge`

```java
void printValues() {
    double averageAge = 0;
    double totalSalary = 0;
    for (int i = 0; i < people.length; i++) {
        averageAge += people[i].age;
        totalSalary += people[i].salary;
    }
    averageAge = averageAge / people.length;
    System.out.println(averageAge);
    System.out.println(totalSalary);
}
```
After Split Loop on `averageAge`

```java
void printValues() {
    double totalSalary = 0;
    for (int i = 0; i < people.length; i++) {
        totalSalary += people[i].salary;
    }
    double averageAge = 0;
    for (int i = 0; i < people.length; i++) {
        averageAge += people[i].age;
    }
    averageAge = averageAge / people.length;
    System.out.println(averageAge);
    System.out.println(totalSalary);
}
```
Advanced Split Loop Example

```java
for (int i = start; i < end; i++) {
    Picture picture = album.getPicture(i);
    printPicture(out, picture);
}
```

Queue<Picture> pictures = new LinkedList<Picture>();
for (int i = start; i < end; i++) {
    Picture picture = album.getPicture(i);
    pictures.add(picture);
}
for (int i = start; i < end; i++) {
    printPicture(out, pictures.remove());
}
```
Replace Temp with Query (RTwQ) [Fowler]

- **Situation**: “You are using a temporary variable to hold the result of an expression.”
- **Recommendation**: “Extract the expression into a method. Replace all references to the temp with the expression. The new method can then be used in other methods.”
- **Link**: http://www.refactoring.com/catalog/replaceTempWithQuery.html
public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Enumeration rentals = _rentals.elements();
    String result = "Rental Record for " + getName() + "\n";
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        frequentRenterPoints += each.getFrequentRenterPoints();
        //show figures for this rental
        result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf(each.getCharge()) + "\n";
        totalAmount += each.getCharge();
    }
    // add footer lines
    result += "Amount owed is " +
              String.valueOf(totalAmount) + "\n";
    result += "You earned " +
              String.valueOf(frequentRenterPoints) +
              "frequent renter points\n";
    return result;
}

class Customer ...
public String statement() {
    int frequentRenterPoints = 0;
    Enumeration rentals = _rentals.elements();
    String result = “Rental Record for “ + getName() + “\n”;
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        frequentRenterPoints += each.getFrequentRenterPoints();
        //show figures for this rental
        result += “\t” + each.getMovie().getTitle() + “\t” +
                String.valueOf(each.getCharge()) + “\n”;
    }
    // add footer lines
    result += “Amount owed is “ +
              String.valueOf(getTotalCharge()) + “\n”;
    result += “You earned “ +
              String.valueOf(frequentRenterPoints) + “frequent renter points\n”;
    return result;
}
class Customer ...
private double getTotalCharge() {
    double result = 0;
    Enumeration rentals = _rentals.elements();
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        result += each.getCharge();
    }
    return result;
}
class Customer ...
public String statement() {
    int frequentRenterPoints = 0;
    Enumeration rentals = _rentals.elements();
    String result = “Rental Record for “ + getName() + “
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        frequentRenterPoints += each.getFrequentRenterPoints();
        //show figures for this rental
        result += “\t” + each.getMovie().getTitle() + “\t” +
                String.valueOf(each.getCharge()) + “\n”;
    }
    // add footer lines
    result += “Amount owed is “ +
              String.valueOf(getTotalCharge()) + “\n”;
    result += “You earned “ +
              String.valueOf(frequentRenterPoints) +
              “frequent renter points\n”;
    return result;
}
class Customer ...
public String statement() {
    Enumeration rentals = _rentals.elements();
    String result = “Rental Record for “ + getName() + “\n”;
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        //show figures for this rental
        result += “\t” + each.getMovie().getTitle() + “\t” +
                  String.valueOf(each.getCharge()) + “\n”;
    }
    // add footer lines
    result += “Amount owed is “ +
              String.valueOf(getTotalCharge()) + “\n”;
    result += “You earned “ +
              String.valueOf(getFrequentRenterPoints()) + “ frequent renter points\n”;
    return result;
}
class Customer ...
private double getFrequentRenterPoints() {
    int result = 0;
    Enumeration rentals = _rentals.elements();
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        result += each.getFrequentRenterPoints();
    }
    return result;
}
Separate Query from Modifier (SQfM) [Fowler]

- **Situation**: “You have a method that returns a value but also changes the state of an object.”
- **Recommendation**: “Create two methods, one for the query and one for the modification.”
- **Link**: http://www.refactoring.com/catalog/separateQueryFromModifier.html
Before SQfM on foundMiscreant

```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 ";
}

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

```java
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;
        }
    }
}
```
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 checkSecurity(String[] people) {
    sendAlert(people);
    String found = foundPerson(people);
    someLaterCode(found);
}
Extract Method [Fowler]

• **Situation**: “You have a code fragment that can be grouped together.”

• **Recommendation**: “Turn the fragment into a method whose name explains the purpose of the method.”

• **Link**: [http://www.refactoring.com/catalog/extractMethod.html](http://www.refactoring.com/catalog/extractMethod.html)
Extract Method: Challenges

• Extract multiple fragments (into a single method)
  – Aka. non-contiguous code
  – Where to place the call?
  – Which statements to delete?
  – Parameters?
  – Backup variables and/or renaming?

• Clone elimination
• Extract incomplete fragments (i.e. with non-extracted holes)
• Loop untangling with second loop reusing intermediate values
• Exiting jumps
Compose Method [Kerievsky]

- **Situation**: “You can't rapidly understand a method's logic.”

- **Recommendation**: “Transform the logic into a small number of intention-revealing steps at the same level of detail.”

- **Link**: http://www.industriallogic.com/xp/refactoring/composeMethod.html
class List...

public void add(Object element) {
    if (!readOnly) {
        int newSize = size + 1;
        if (newSize > elements.length) {
            Object[] newElements = new Object[elements.length+10];
            for (int i = 0; i < size; i++)
                newElements[i] = elements[i];
            elements = newElements;
        }
        elements[size++] = element;
    }
}
After Compose Method

class List...
public void add(Object element) {
    if (readOnly) {
        return;
    } else if (atCapacity())
        grow();
    addElement(element);
}

private void addElement(Object element) {
    elements[size++] = element;
}

private void grow() {
    Object[] newElements =
        new Object[elements.length + GROWTH_INCREMENT];
    for (int i = 0; i < size; i++)
        newElements[i] = elements[i];
    elements = newElements;
}

private boolean atCapacity() {
    return (size + 1) > elements.length;
}
Form Template Method (FTM)

• Fowler
  – **Situation**: “You have two methods in subclasses that perform similar steps in the same order, yet the steps are different.”
  – **Recommendation**: “Get the steps into methods with the same signature, so that the original methods become the same. Then you can pull them up.”

• Kerievsky
  – **Situation**: “Two methods in subclasses perform similar steps in the same order, yet the steps are different.”
  – **Recommendation**: “Generalize the methods by extracting their steps into methods with identical signatures, then pull up the generalized methods to form a Template Method.”
Convert Procedural Design to Objects [Fowler & Beck]

• **Situation**: “You have code written in a procedural style.”

• **Recommendation**: “Turn the data records into objects, break up the behavior, and move the behavior to the objects.”

• **Link**: [http://sourcemaking.com/refactoring/convert-procedural-design-to-objects](http://sourcemaking.com/refactoring/convert-procedural-design-to-objects)
Separate Domain from Presentation [Fowler & Beck]

• **Situation:** “You have GUI classes that contain domain logic.”

• **Recommendation:** “Separate the domain logic into separate domain classes.”

• **Link:** [http://sourcemaking.com/refactoring/separate-domain-from-presentation](http://sourcemaking.com/refactoring/separate-domain-from-presentation)
Conclusion

• Program slicing can help in building automated tools for refactoring
• Fine slicing is a generalization of program slicing
  – It can be used to compute meaningful sub-programs
  – Can be used to compute the complementary code required for correct refactoring
• This work is part of a long-term research agenda focusing on advanced enterprise refactoring tools, aiming to assist in daily software development on the one hand and in legacy modernization on the other
• The new Extract Computation refactoring for isolating fine slices is a crucial building block in this endeavor
  – It will be used to enhance the automation for complex code-motion refactorings in order to support enterprise transformations such as the move to MVC [WRT08-09]
• Present and future work
  – Refactoring tools for Cobol in RDz
  – Refactoring tools for Java in Eclipse [ECOOP12]
  – Fine slicing tool to assist in program comprehension
  – Big refactorings too: composition; meaningful error reporting
References

- Abadi, Ettinger, and Feldman, WRT08: Re-approaching the refactoring rubicon. *Workshop on Refactoring Tools @ OOPSLA08*.
- Abadi, Ettinger, and Feldman, WRT09: Fine slicing for advanced method extraction. *Workshop on Refactoring Tools @ OOPSLA09*.
- Abadi, Ettinger, and Feldman, FASE @ ETAPS12: Fine slicing: Theory and applications for computation extraction.
- My ECOOP12 paper: Program sliding.
Thanks!