SED

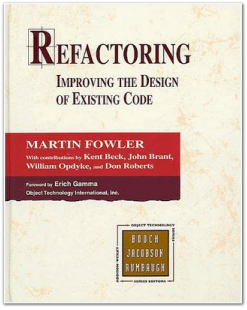
Refactoring

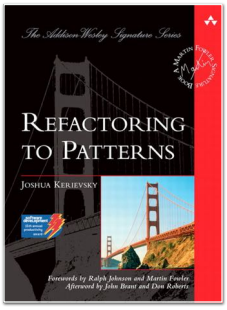
Dr Robert Chatley - rbc@imperial.ac.uk

@rchatley

Refactoring

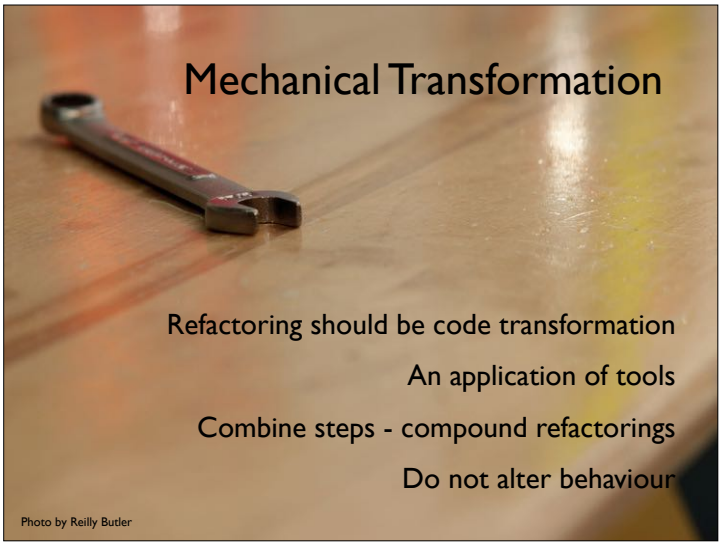
"By continuously improving the design of code, we make it easier and easier to work with. This is in sharp contrast to what typically happens: little refactoring and a great deal of attention paid to expediently adding new features. If you get into the hygienic habit of refactoring continuously, you'll find that it is easier to extend and maintain code."





Technical Debt

Photo by Robert Scoble



Mechanical Transformation

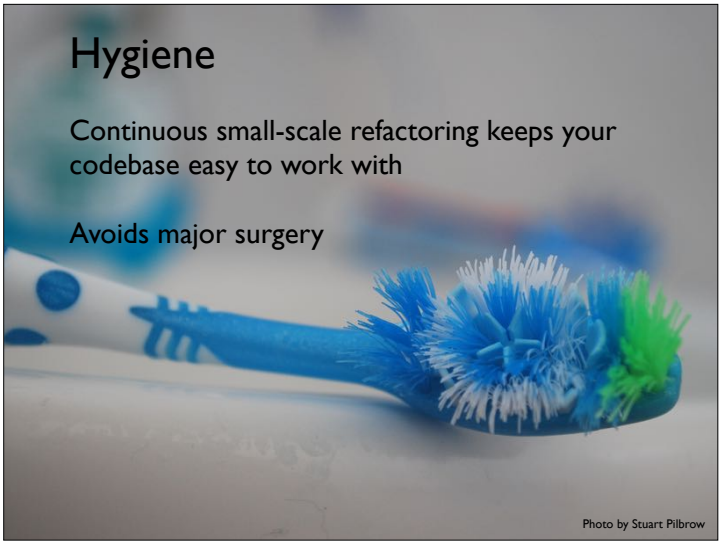
Refactoring should be code transformation

An application of tools

Combine steps - compound refactorings

Do not alter behaviour

Photo by Reilly Butler



Hygiene

Continuous small-scale refactoring keeps your codebase easy to work with

Avoids major surgery

Photo by Stuart Pilbrow

Clone this repo

https://gitlab.doc.ic.ac.uk/sed/yahtzee

Identify some improvements

https://gitlab.doc.ic.ac.uk/sed/yahtzee

Refactor the code **in small steps** to improve the code's design

Run tests before and after each change

A Catalogue of Refactorings

What would you do with this code?

```
class Invoice {

  ...

  public void print(PrintStream stream) {

    stream.println("ACME Ltd");
    stream.println("123 High St");
    stream.println("London");
    stream.println("SW12 1AA");

    int totalCost = 0;

    for(LineItem item : items) {
      StringBuffer line = new StringBuffer();
      line.append(item.name);
      line.append("\t");
      line.append(item.price);
      line.append("\t");
      line.append(item.quantity);
      line.append("\t");
      line.append(item.quantity * item.price);
      totalCost += item.quantity * item.price;
      stream.println(line.toString());
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }
}
```

Compose Method

```
class Invoice {

  ...

  public void print(PrintStream stream) {

    stream.println("ACME Ltd");
    stream.println("123 High St");
    stream.println("London");
    stream.println("SW12 1AA");

    int totalCost = 0;

    for(LineItem item : items) {
      StringBuffer line = new StringBuffer();
      line.append(item.name);
      line.append("\t");
      line.append(item.price);
      line.append("\t");
      line.append(item.quantity);
      line.append("\t");
      line.append(item.quantity * item.price);
      totalCost += item.quantity * item.price;
      stream.println(line.toString());
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }
}
```

```
class Invoice {

  ...

  public void print(PrintStream stream) {

    printAddress(stream);

    int totalCost = 0;

    for (LineItem item : items) {
      StringBuffer line = new StringBuffer();
      line.append(item.name);
      line.append("\t");
      line.append(item.price);
      line.append("\t");
      line.append(item.quantity);
      line.append("\t");
      line.append(item.quantity * item.price);
      totalCost += item.quantity * item.price;
      stream.println(line.toString());
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private void printAddress(PrintStream stream) {
    stream.println("ACME Ltd");
    stream.println("123 High St");
    stream.println("London");
    stream.println("SW12 1AA");
  }
```

extract method

Compose Method

```java
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    int totalCost = 0;

    for (LineItem item : items) {
      StringBuffer line = new StringBuffer();
      line.append(item.name);
      line.append("\t");
      line.append(item.price);
      line.append("\t");
      line.append(item.quantity);
      line.append("\t");
      line.append(item.quantity * item.price);
      totalCost += item.quantity * item.price;
      stream.println(line.toString());
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private void printAddress(PrintStream stream) {
    stream.println("ACME Ltd");
    stream.println("123 High St");
    stream.println("London");
    stream.println("SW12 1AA");
  }
```

```java
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    int totalCost = 0;

    for (LineItem item : items) {
      printLineItem(stream, item);
      totalCost += item.quantity * item.price;
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private void printLineItem(PrintStream stream, LineItem
item) {
    StringBuffer line = new StringBuffer();
    line.append(item.name);
    line.append("\t");
    line.append(item.price);
    line.append("\t");
    line.append(item.quantity);
    line.append("\t");
    line.append(item.quantity * item.price);
    stream.println(line.toString());
  }

  private void printAddress(PrintStream stream) {
    stream.println("ACME Ltd");
    stream.println("123 High St");
    stream.println("London");
    stream.println("SW12 1AA");
  }
```

Separate Responsibility

```java
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    int totalCost = 0;

    for (LineItem item : items) {
      printLineItem(stream, item);
      totalCost += item.quantity * item.price;
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private void printLineItem(PrintStream stream, LineItem item) {
      StringBuffer line = ....
```

```java
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    for (LineItem item : items) {
      printLineItem(stream, item);
    }

    int totalCost = 0;
    for (LineItem item : items) {
      totalCost += item.quantity * item.price;
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private void printLineItem(PrintStream stream, LineItem item) {
    StringBuffer line = ....
```

Compose Method

```java
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    for (LineItem item : items) {
      printLineItem(stream, item);
    }

    int totalCost = 0;
    for (LineItem item : items) {
      totalCost += item.quantity * item.price;
    }

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private void printLineItem(PrintStream stream, LineItem item) {
    StringBuffer line = ....
```

```java
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    for (LineItem item : items) {
      printLineItem(stream, item);
    }

    int totalCost = totalCost();

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private int totalCost() {
    int totalCost = 0;
    for (LineItem item : items) {
      totalCost += item.quantity * item.price;
    }
    return totalCost;
  }

  private void printLineItem(PrintStream stream, LineItem item
    StringBuffer line = ....
```

Inline Variable

```
class Invoice {

    ....

  public void print(PrintStream stream) {

    printAddress(stream);

    for (LineItem item : items) {
      printLineItem(stream, item);
    }

    int totalCost = totalCost();

    stream.println("Total" + "\t\t\t" + totalCost);
  }

  private int totalCost() {
    int totalCost = 0;
    for (LineItem item : items) {
      totalCost += item.quantity * item.price;
    }
    return totalCost;
  }

  private void printLineItem(PrintStream stream, LineItem item) {
    StringBuffer line = ....
```

```
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    for (LineItem item : items) {
      printLineItem(stream, item);
    }

    stream.println("Total" + "\t\t\t" + totalCost());
  }

  private int totalCost() {
    int totalCost = 0;
    for (LineItem item : items) {
      totalCost += item.quantity * item.price;
    }
    return totalCost;
  }

  private void printLineItem(PrintStream stream, LineItem item)
    StringBuffer line = ....
```

inline

Compose Method

```
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    for (LineItem item : items) {
      printLineItem(stream, item);
    }

    stream.println("Total" + "\t\t\t" + totalCost());
  }

  private int totalCost() {
    int totalCost = 0;
    for (LineItem item : items) {
      totalCost += item.quantity * item.price;
    }
    return totalCost;
  }

  private void printLineItem(PrintStream stream, LineItem item) {
    StringBuffer line = ....
```

```
class Invoice {

  ....

  public void print(PrintStream stream) {

    printAddress(stream);

    for (LineItem item : items) {
      printLineItem(stream, item);
    }

    printTotal(stream);
  }

  private void printTotal(PrintStream stream) {
    stream.println("Total" + "\t\t\t" + totalCost());
  }

  private int totalCost() {
    int totalCost = 0;
    for (LineItem item : items) {
      totalCost += item.quantity * item.price;
    }
    return totalCost;
}

  private void printLineItem(PrintStream stream, LineItem ite
    StringBuffer line = ....
```

Duplication between classes

```
class Invoice {

  public void print(PrintStream p) {

    p.println("ACME Ltd");
    p.println("123 High St");
    p.println("London");
    p.println("SW12 1AA");

    int totalCost = 0;

    for(LineItem item : items) {
      StringBuffer line = new StringBuffer();
      buffer.append(item.name);
      buffer.append("\t");
      buffer.append(item.price);
      buffer.append("\t");
      buffer.append(item.quantity);
      buffer.append("\t");
      buffer.append(item.quantity * item.price);
      totalCost += item.quantity * item.price;
      p.println(line.toString());
    }

    p.println("Total" + "\t\t\t" + totalCost);
  }
}
```

```
class HeadedLetter {

  public void print(PrintStream stream) {

    stream.println("ACME Ltd");
    stream.println("123 High St");
    stream.println("London");
    stream.println("SW12 1AA");

      // ...
  }
}
```

extract to common class

Extract Class

```
class HeadedLetter {

    public void print(PrintStream stream) {

        stream.println("ACME Ltd");
        stream.println("123 High St");
        stream.println("London");
        stream.println("SW12 1AA");

        // ...
    }
}
```

1) extract method

```
class HeadedLetter {

    public void print(PrintStream stream) {

        printAddress(stream);

        // ...
    }

    private void printAddress(PrintStream stream) {
        stream.println("ACME Ltd");
        stream.println("123 High St");
        stream.println("London");
        stream.println("SW12 1AA");
    }

}

class Address {

}
```

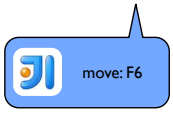
2) create (empty) class

Extract Class

```
class HeadedLetter {

    public void print(PrintStream stream) {

        printAddress(stream, new Address());

        // ...
    }

    private void printAddress(PrintStream stream, Address address) {
        stream.println("ACME Ltd");
        stream.println("123 High St");
        stream.println("London");
        stream.println("SW12 1AA");
    }
}

class Address {

}
```

3) add (unused) parameter

4) move method onto parameter




```
class HeadedLetter {

    public void print(PrintStream stream) {

        new Address().printAddress(stream);

        // ...
    }

}

class Address {

    void printAddress(PrintStream stream) {
        stream.println("ACME Ltd");
        stream.println("123 High St");
        stream.println("London");
        stream.println("SW12 1AA");
    }
}
```

Duplication between classes

use extracted class

```
class Invoice {

  public void print(PrintStream p) {

    new Address().printAddress(p);

    int totalCost = 0;

    for(LineItem item : items) {
      StringBuffer line = new StringBuffer();
      buffer.append(item.name);
      buffer.append("\t");
      buffer.append(item.price);
      buffer.append("\t");
      buffer.append(item.quantity);
      buffer.append("\t");
      buffer.append(item.quantity * item.price);
      totalCost += item.quantity * item.price;
      p.println(line.toString());
    }

    p.println("Total" + "\t\t\t" + totalCost);
  }
}
```

```
class HeadedLetter {

    public void print(PrintStream stream) {

        new Address().printAddress(stream);

        // ...
    }
}
```

Rename to Tidy Up

```java
class Invoice {

  public void print(PrintStream p) {

    new Address().printTo(p);

    int totalCost = 0;

    for(LineItem item : items) {
      StringBuffer line = new StringBuffer();
      buffer.append(item.name);
      buffer.append("\t");
      buffer.append(item.price);
      buffer.append("\t");
      buffer.append(item.quantity);
      buffer.append("\t");
      buffer.append(item.quantity * item.price);
      totalCost += item.quantity * item.price;
      p.println(line.toString());
    }

    p.println("Total" + "\t\t\t" + totalCost);
  }
}
```

```java
class HeadedLetter {

  public void print(PrintStream stream) {

    new Address().printTo(stream);

    // ...
  }
}
```

rename

Shift-F6

Replace Conditional with Polymorphism

```java
HeadedLetter letter = new HeadedLetter();

if (letter.isImportant()) {
    letter.sendByCourierTo(recipient, address);
} else {
    letter.sendByStandardMailTo(recipient, address);
}
```

1) extract & move method

```java
HeadedLetter letter = new HeadedLetter();

letter.sendTo(recipient, address);

class HeadedLetter implements Correspondence {
  public void sendTo(Person recipient, Address address) {
    if (isImportant()) {
      sendByCourierTo(recipient, address);
    } else {
      sendByStandardMailTo(recipient, address);
    }
  }
}
```

2) extract interface

```java
public interface Correspondence {
  void sendTo(Person recipient, Address address);
}
```

Replace Conditional with Polymorphism

```java
class ConfidentialLetter implements Correspondence {

    ...

    public void sendTo(Person recipient, Address address) {
            sendByCourierTo(recipient, address);
    }

    private void sendByCourierTo(Person recipient, Address address) {
        ...
    }
}
```

3) create two variants that implement the same interface

```java
class GeneralCircularLetter implements Correspondence {

    ...

    public void sendTo(Person recipient, Address address) {
        sendByStandardMailTo(recipient, address);
    }

    private void sendByStandardMailTo(Person recipient, Address address) {
        ...
    }
}
```

4) use polymorphically

```java
for (Correspondence letter : postBag) {
    letter.sendTo(recipient, address);
}
```