(11224) INTRODUCTION TO SOFTWARE ENGINEERING

Lecture 19: Code Refactoring (part 1)

Shereen Fouad

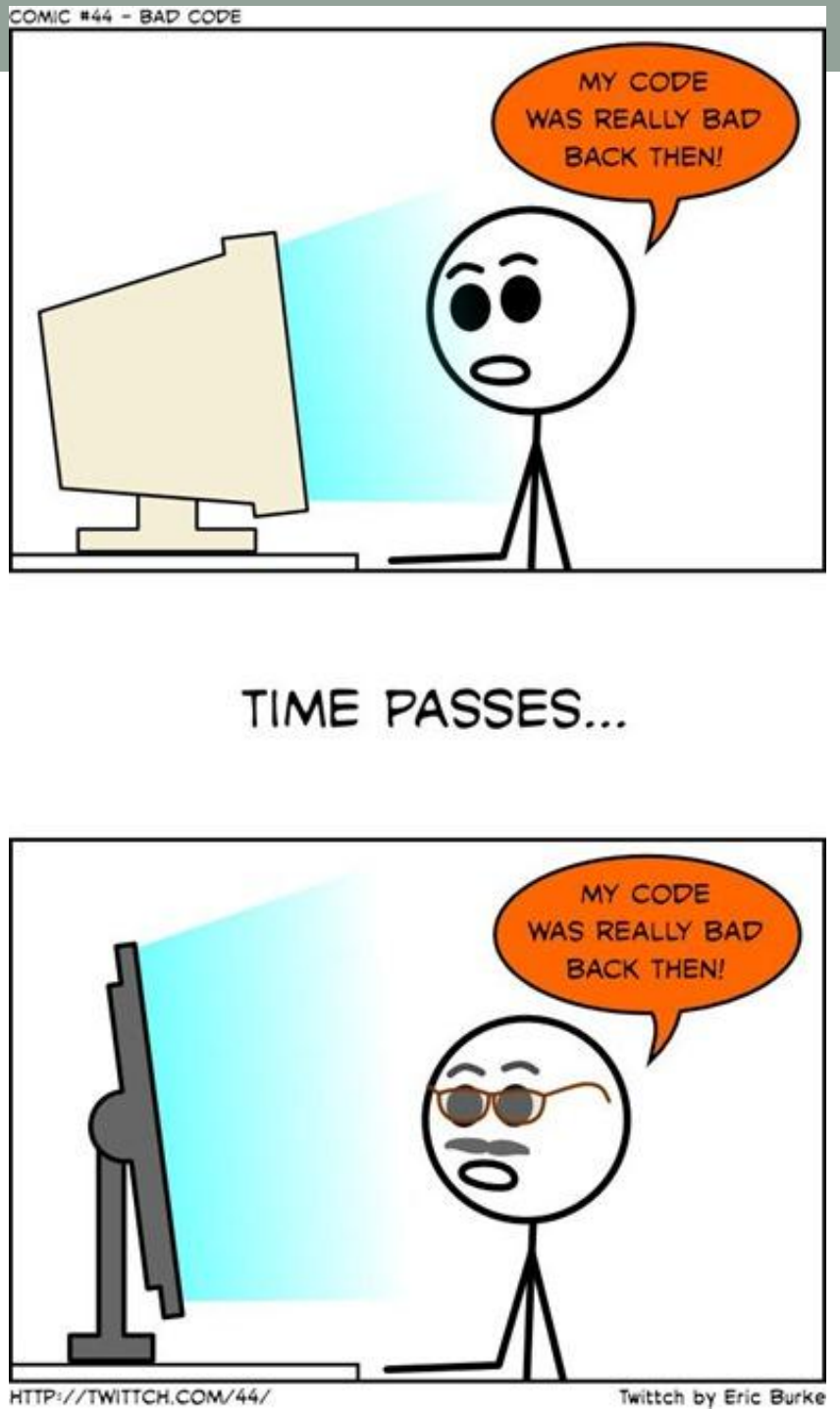
Announcements

- Online test on Friday the 20th of March at 10 am and will close (same day) at 10 pm
  - Counts for 5% of the entire Module Mark
  - Test will be Multiple Choice Questions (25 questions)
  - Once you begin the online test you have only 60 minutes to complete it.
  - You only have one attempt to complete the online test.
  - It will cover all concepts discussed in this module.
- Past years exams are available on canvas.
- Model answer of Exercise 1 (UML class diagram) is available on canvas.

Motivation

- Over the semester we have talked about Software Engineering.
- The overall goal of software engineering is to create high quality software efficiently.
- What if you don’t though? There are always pressures and reasons that software isn’t great

Why do good developers write bad software?

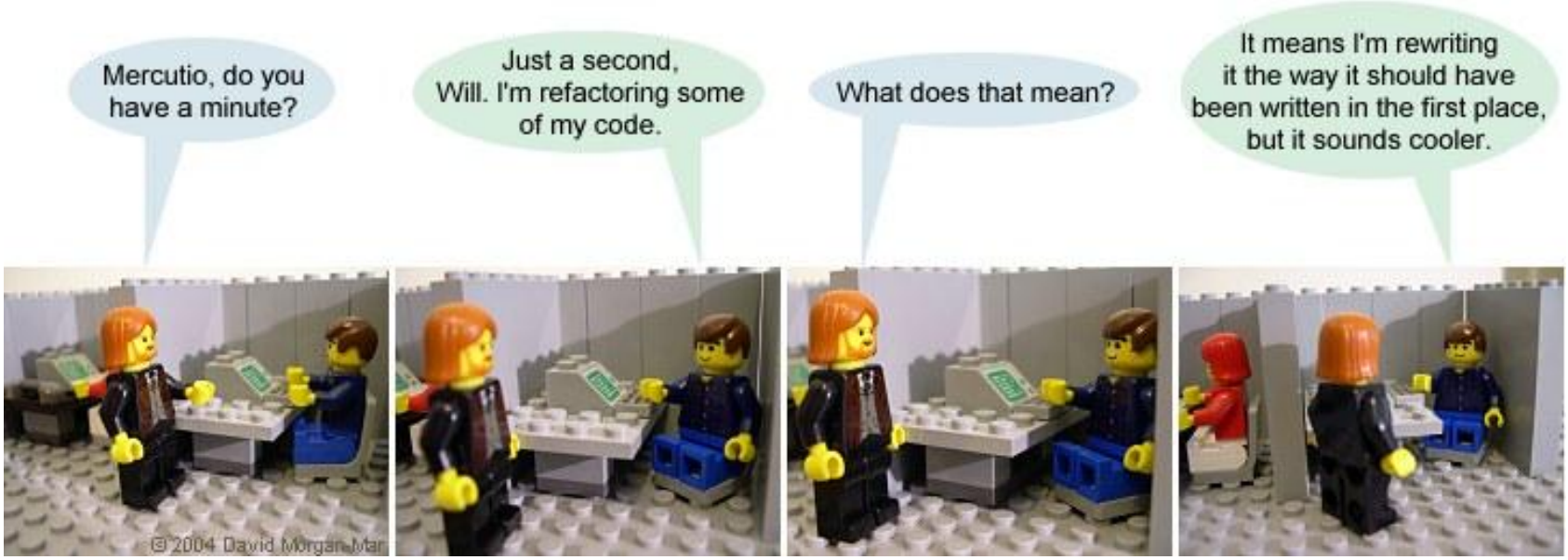


Why do good developers write bad software?

- Requirements change over time, making it hard to update your code (leading to less optimal designs)
- Time and money cause you to take shortcuts
- You learn a better way to do something

Refactoring

- Refactoring modifies software quality to improve its readability without changing what it actually does.
- The goal of refactoring is NOT to add new functionality
- Internal structure of the code is improved



Why refactor??

- Improve its maintainability.
- Improve its extensibility.
- Remove duplicates.
- Easier and quicker to understand the code.
- Easier and quicker to write a code.
- Easier and quicker to find bugs.

Danger!

- Refactoring CAN introduce problems,

Danger!

- Refactoring CAN introduce problems, because anytime you modify software you may introduce bugs!

- **How to solve this problem??**

Danger!

- Refactoring CAN introduce problems, because anytime you modify software you may introduce bugs!
- **How to solve this problem??**
- Testing is an effective and suitable way for ensuring that refactoring has been correctly carried out.

In refactoring we aim to answer Two questions:

1. How do we know our software is “bad”… when it works fine!
2. How do we fix our software?

What makes code hard to work with?

Bad smells in a code

- Duplicated Code
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Data Clumps
- Feature Envy
- Lazy Class
- Data Classes
- Speculative Generality
- Temporary Field
- Comments

Bad smells in a code

- Duplicated Code
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Data Clumps
- Feature Envy
- Lazy Class
- Data Classes
- Speculative Generality
- Temporary Field
- **Comments ???**

Many more code smells

- Given a smell, what refactorings are likely?
  - Check
  - http://wiki.java.net/bin/view/People/SmellsToRefactorings
  - and
  - http://refactoring.com/catalog/

Refactoring Pattern

For each bad smell there are possible refactoring patterns that is applied to the code to mitigate these structural problems.

What is wrong here?

```
1.    void printOwing(double previousAmount)
2.    {
3.        Enumeration e = _orders.elements();
4.        double outstanding = previousAmount * 1.2;
5.
6.        printBanner();
7.
8.        // calculate outstanding
9.        while (e.hasMoreElements())
10.       {
11.           Order each = (Order) e.nextElement();
12.           outstanding += each.getAmount();
13.       }
14.
15.       printDetails(outstanding);
16.   }
```

Refactoring Pattern: Extract Method

- The most important and common refactoring pattern
- Supported by most of the popular Java IDEs.
- This pattern deals with the situation when there is a number of lines of code that belongs together and, in some way, makes sense apart from the code that directly precedes or follows it.

Refactoring Pattern: Extract Method

```
1.  void printOwing(double previousAmount)
2.  {
3.      Enumeration e = _orders.elements();
4.      double outstanding = previousAmount * 1.2;
5.
6.      printBanner();
7.
8.      // calculate outstanding
9.      while (e.hasMoreElements())
10.     {
11.         Order each = (Order) e.nextElement();
12.         outstanding += each.getAmount();
13.     }
14.
15.     printDetails(outstanding);
16. }
```

(Comment or Long Method code smell)

lines 9 to 13 belong together and makes sense apart from the code that directly precedes or follows it.

Refactoring Pattern: Extract Method

```
1.  void printOwing(double previousAmount)
2.  {
3.          double outstanding = previousAmount * 1.2;
4.          printBanner();
5.          outstanding = getOutstanding(outstanding);
6.          printDetails(outstanding);
7.  }
8.
9.  double getOutstanding(double initialValue)
10. {
11.         double outstanding = initialValue;
12.         Enumeration e = _orders.elements();
13.         while (e.hasMoreElements())
14.         {
15.             Order each = (Order) e.nextElement();
16.             outstanding += each.getAmount();
17.         }
18.         return outstanding;
19. }
```

Make code fragment as readable as comments

Refactoring Pattern: Extract Method

```
1.  void printOwing(double previousAmount)
2.  {
3.          double outstanding = previousAmount * 1.2;
4.          printBanner();
5.          outstanding = getOutstanding(outstanding);
6.          printDetails(outstanding);
7.  }
8.
9.  double getOutstanding(double initialValue)
10. {
11.         double outstanding = initialValue;
12.         Enumeration e = _orders.elements();
13.         while (e.hasMoreElements())
14.         {
15.             Order each = (Order) e.nextElement();
16.             outstanding += each.getAmount();
17.         }
18.         return outstanding;
19. }
```

If any variables in the extracted lines are used only within those lines, declare them as local variables in the target method.

Refactoring Pattern: Inline Method

- If you are getting lost in a mess of many methods that do nothing more than redirect by calling another method, then inlining these methods can simplify the situation.
- This is the inverse to Extract Method.
- This refactoring pattern is only suitable for simple methods.

A method's body is just as clear as its name.

Put the method's body into the body of its callers and remove the method.

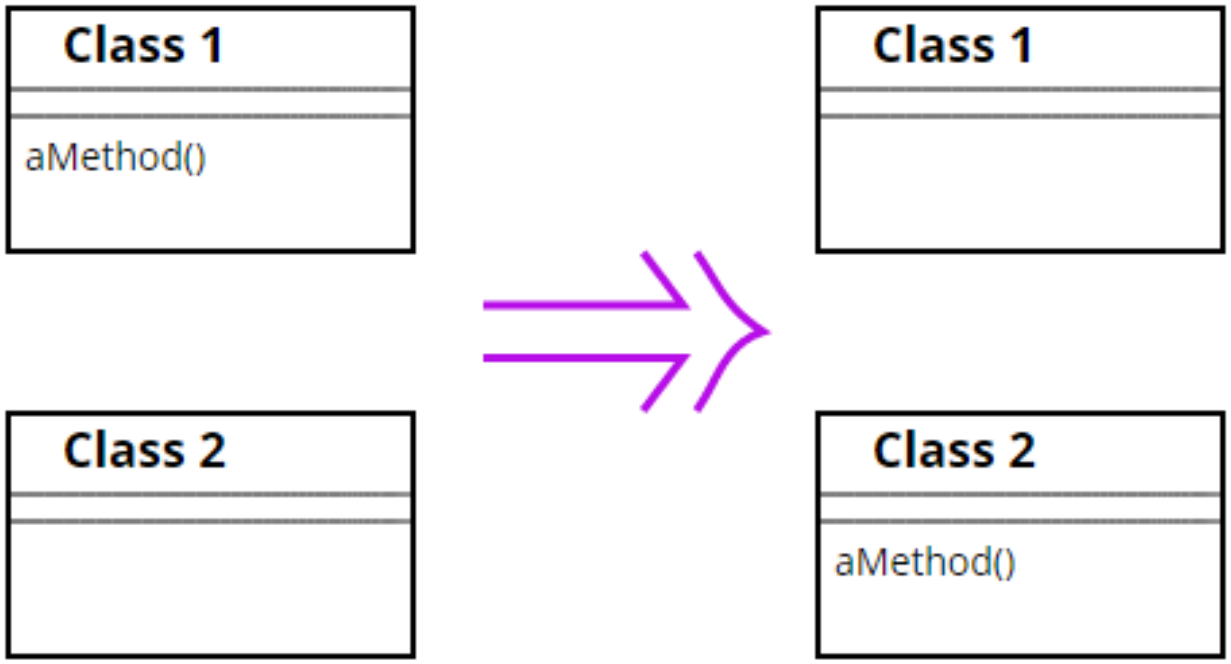
```
int getRating() {
  return (moreThanFiveLateDeliveries()) ? 2 : 1;
}
boolean moreThanFiveLateDeliveries() {
  return _numberOfLateDeliveries > 5;
}
```

```
int getRating() {
  return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

inverse of *Extract Method*

Refactoring Pattern: Move Method

- A method is, or will be, using or used by more features of another class than the class on which it is defined.
- Move a method from one class to o*ther class it uses most.*



References

- A number of slides in this talk is based on:
  - Alan P. Sexton hand-outs (Introduction to Software Engineering. The University of Birmingham. Spring Semester 2014)
  - Martin Fowler, Refactoring, Addison Wesley, 1999.
  - http://refactoring.com/catalog

Thank YOU ☺