

# **COSC 310:**

# **Software Engineering**

Dr. Bowen Hui

University of British Columbia Okanagan

[bowen.hui@ubc.ca](mailto:bowen.hui@ubc.ca)

# Good Code? Bad Code?

```
q = ((p<=1) ? (p ? 0 : 1) : (p== -4) ? 2 : (p+1));
```

# Good Code? Bad Code? (2)

```
while (*a++ = *b--) ;
```

# Good Code? Bad Code? (3)

```
char b[2][10000], *s, *t=b, *d, *e=b+1, **p; main(int
c, char**v) {int n=atoi(v[1]); strcpy(b, v[2]);
while(n--){for(s=t, d=e; *s; s++)
{for(p=v+3; *p; p++) if(**p==*s){strcpy(d, *p+2); d+=
strlen(d); goto x;} *d++=*s; x:} s=t; t=e; e=s;
*d++=0; } puts(t); }
```

# Yes, But ...

- "nobody" really writes code like that
- your code is (probably) great
  
- but ...
  - others alter your code
  - you alter other people's code
  - collaboration = better product, right?

# Some Causes of Smelly Code

- code like that emerges after a collaborative effort
  - you won't get it right the first time around
  - defects show up at all points of the lifecycle
  - code is constantly being revisited
- accumulated modifications lead to this type of code
  - no such thing as a "perfect coder"
  - external conditions can affect the quality of work
    - approaching deadlines,
    - stress,
    - skunks, ...

# Some Causes of Smelly Code (cont.)

- agile process with very small upfront design
  - the simplest thing that could possibly work
  - this code is expected in an agile process

in fact, if smelly code doesn't show up, then you are probably not using an agile process

# Solution?

1. improve the code without changing its overall behaviour
2. spend more time designing upfront
  - which is better?



# Motivating Thought Experiment

- Case:
  - You've written all the input analysis and response generation code for your project.
  - The code accidentally got deleted (no backups).
- Questions:
  - How much time would it take you to rewrite the code from scratch?
    - same, more, less?
  - Would the code have a better design the second time around?

# Code Evolves

- rule of thumb:
  - it's harder to maintain (someone else's) code than to write code from scratch
  - extreme: NIH syndrome (not invented here)
- reality:
  - most programmers spend their time evolving and maintaining code
- advice:
  - it pays off to keep code simple and clean

# What is Refactoring?

*"[Refactoring is] the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure"*

-- Martin Fowler

# Refactoring Changes

- changes made to a system that:
  - do not change observable behaviour
  - remove duplication or needless complexity
  - enhance software quality
  - make code easier and simpler to understand
  - make code more flexible
  - make code easier to change

# Why Refactor?

- bad code incurs in huge penalty in the long run
  - cost in reviewing, maintaining, fixing bad code
  - cost in repeatedly having to do those activities
- benefits of refactoring:
  - not usually a short term benefit in cost/time
  - long term investment in the quality of code and its structure

# When to Refactor?

- **NOT:** 2 weeks every 6 months
- opportunistic refactoring = do it as you develop
- boy scout principle: leave it better than you found it
- times when you might recognize a bad smell:
  - when you add a function
  - when you fix a bug
  - when you code review

# The Rule of Three

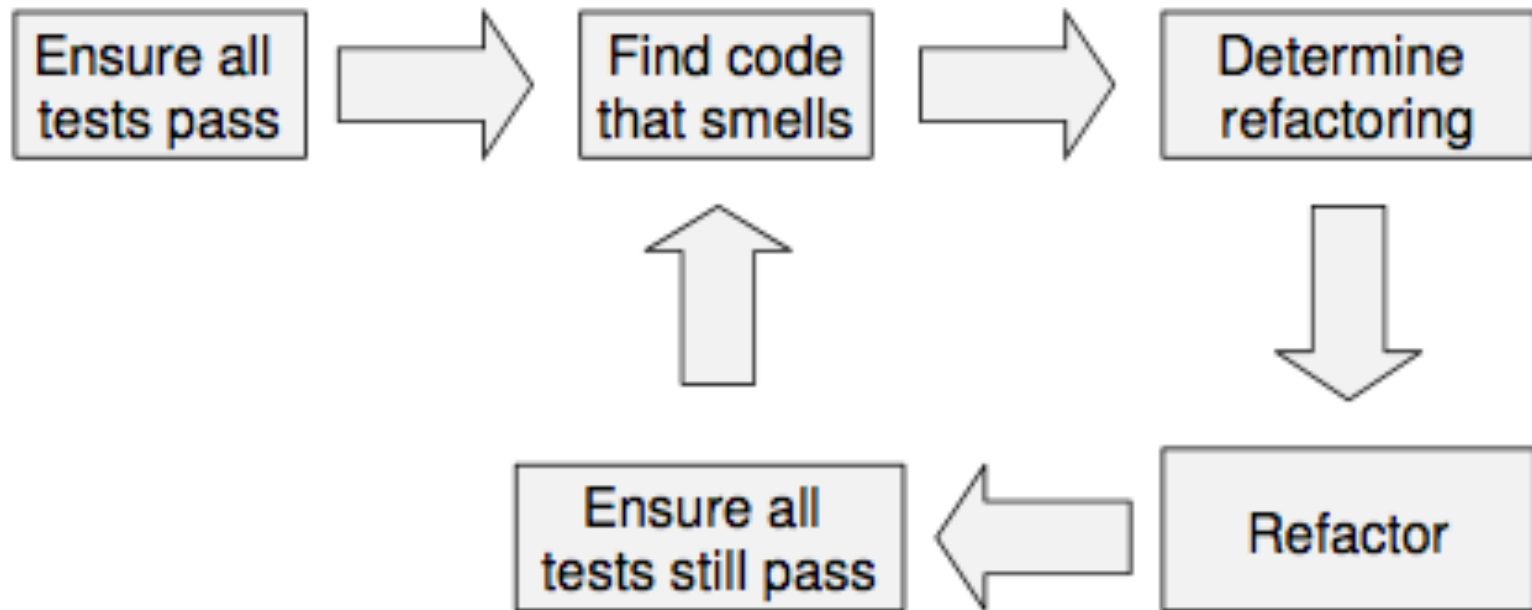
1. the first time, just do it!
2. need it somewhere else? cut and paste it!
3. the third time, refactor!

# When Not to Refactor?

- when tests are failing
  - why?
- when you should just rewrite the code
  - why?
- when you have impending deadlines
  - why?
- when the smell is tolerable
  - why?



# Refactoring Process




repeat until the smell is gone!

# Simple Example

```
int fa = 1;
for( int i=2; i<a; ++i ) fa *= i;

int fb = 1;
for( int i=2; i<b; ++i ) fb *= i;
```



```
int fact( int x ) {
    return (x==1) ? 1 : fact(x-1)*x;
}

fa = fact(a);
fb = fact(b);
```

# Example Refactoring: Pull Up Method

Refactoring: Pull up method - If there are identical methods in more than one subclass, move it to the superclass

eg.



# Example Refactoring: Pull Up Method

```
1: public abstract class Vehicle
2: {
3:     // other methods
4: }
5:
6: public class Car : Vehicle
7: {
8:     public void Turn(Direction direction)
9:     {
10:         // code here
11:     }
12: }
13:
14: public class Motorcycle : Vehicle
15: {
16: }
17:
18: public enum Direction
19: {
20:     Left,
21:     Right
22: }
```

# Example Refactoring: Pull Up Method

```
1: public abstract class Vehicle
2: {
3:     // other methods
4: }
5:
6: public class Car : Vehicle
7: {
8:     public void Turn(Direction direction)
9:     {
10:         // code here
11:     }
12: }
13:
14: public class Motorcycle : Vehicle
15: {
16: }
17:
18: public enum Direction
19: {
20:     Left,
21:     Right
22: }
```



```
1: public abstract class Vehicle
2: {
3:     public void Turn(Direction direction)
4:     {
5:         // code here
6:     }
7: }
8:
9: public class Car : Vehicle
10: {
11: }
12:
13: public class Motorcycle : Vehicle
14: {
15: }
16:
17: public enum Direction
18: {
19:     Left,
20:     Right
21: }
```

# Example Refactoring: Rename

```
1: public class Person
2: {
3:     public string FN { get; set; }
4:
5:     public decimal ClcHrlyPR()
6:     {
7:         // code to calculate hourly payrate
8:         return 0m;
9:     }
10: }
```

# Example Refactoring: Rename

```
1: public class Person
2: {
3:     public string FN { get; set; }
4:
5:     public decimal ClcHrlyPR()
6:     {
7:         // code to calculate hourly payrate
8:         return 0m;
9:     }
10: }
```



```
1: // Changed the class name to Employee
2: public class Employee
3: {
4:     public string FirstName { get; set; }
5:
6:     public decimal CalculateHourlyPay()
7:     {
8:         // code to calculate hourly payrate
9:         return 0m;
10:    }
11: }
```

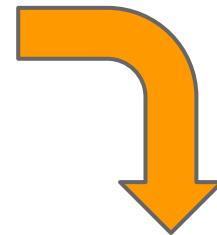
# Example Refactoring: Extract Method

```
1: public class Receipt
2: {
3:     private IList<decimal> Discounts { get; set; }
4:     private IList<decimal> ItemTotals { get; set; }
5:
6:     public decimal CalculateGrandTotal()
7:     {
8:         decimal subTotal = 0m;
9:         foreach (decimal itemTotal in ItemTotals)
10:             subTotal += itemTotal;
11:
12:         if (Discounts.Count > 0)
13:         {
14:             foreach (decimal discount in Discounts)
15:                 subTotal -= discount;
16:         }
17:
18:         decimal tax = subTotal * 0.065m;
19:
20:         subTotal += tax;
21:
22:         return subTotal;
23:     }
24: }
```



# Example Refactoring: Extract Method

```
1: public class Receipt
2: {
3:     private IList<decimal> Discounts { get; set; }
4:     private IList<decimal> ItemTotals { get; set; }
5:
6:     public decimal CalculateGrandTotal()
7:     {
8:         decimal subTotal = 0m;
9:         foreach (decimal itemTotal in ItemTotals)
10:             subTotal += itemTotal;
11:
12:         if (Discounts.Count > 0)
13:         {
14:             foreach (decimal
15:                 subTotal -=
16:             }
17:
18:             decimal tax = subTotal
19:
20:             subTotal += tax;
21:
22:             return subTotal;
23:         }
24: }
```



```
1: public class Receipt
2: {
3:     private IList<decimal> Discounts { get; set; }
4:     private IList<decimal> ItemTotals { get; set; }
5:
6:     public decimal CalculateGrandTotal()
7:     {
8:         decimal subTotal = CalculateSubTotal();
9:
10:        subTotal = CalculateDiscounts(subTotal);
11:
12:        subTotal = CalculateTax(subTotal);
13:
14:        return subTotal;
15:     }
16: }
```

# Example Refactoring: Introduce Parameter Object

```
1: public class Registration
2: {
3:     public void Create(decimal amount, Student student,
4:                         IEnumerable<Course> courses, decimal credits)
5:     {
6:         // do work
7:     }
8: }
```

# Example Refactoring: Introduce Parameter Object

```
1: public class Registration
2: {
3:     public void Create(decimal amount, Student student,
4:                         IEnumerable<Course> courses, decimal credits)
5:     {
6:         // do work
7:     }
8: }
```



```
1: public class RegistrationContext
2: {
3:     public decimal Amount { get; set; }
4:     public Student Student { get; set; }
5:     public IEnumerable<Course> Courses { get; set; }
6:     public decimal Credits { get; set; }
7: }
8:
9: public class Registration
10: {
11:     public void Create(RegistrationContext registrationContext)
12:     {
13:         // do work
14:     }
15: }
```

# Refactoring Practice

- split up into 5 teams
- work together on the following exercises as a team
- each team select two exercises

# Take Home Message

*Any fool can write code that a computer can understand. Good programmers write code that humans can understand.*

Martin Fowler

Refactoring: Improving the design of existing code