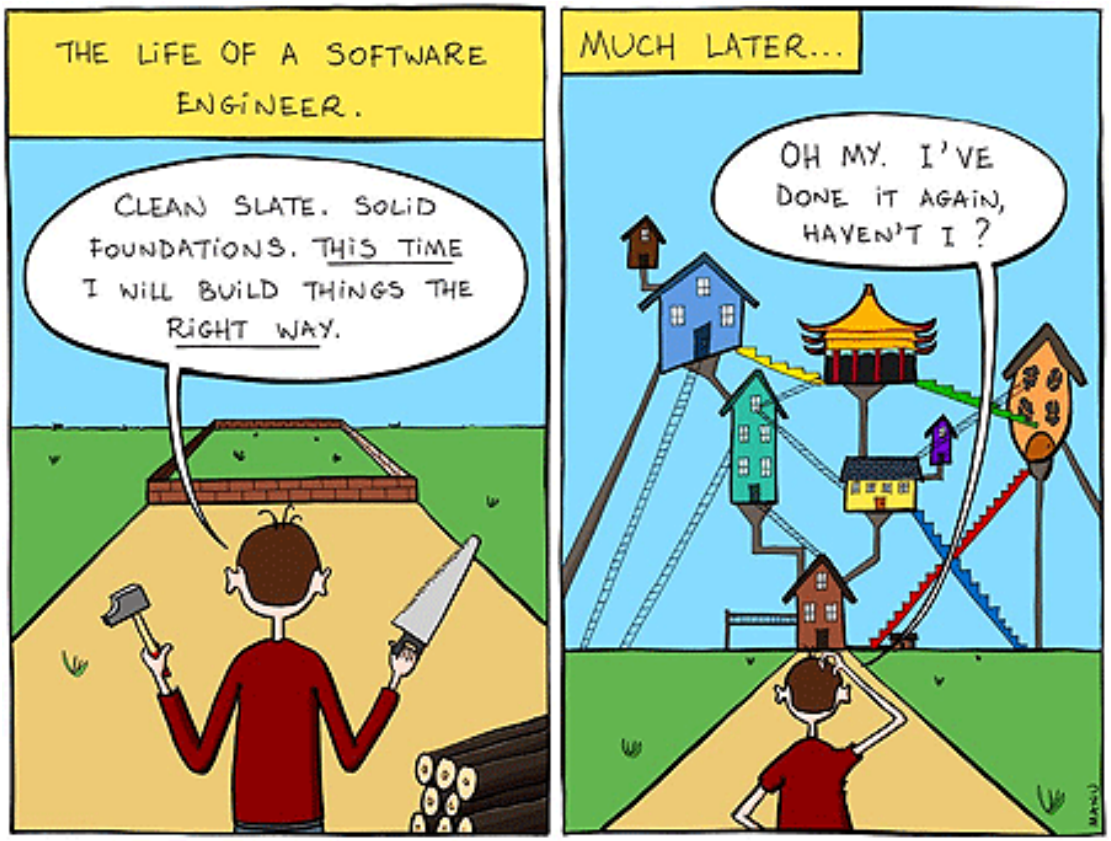
cphbusiness

Refactoring



What is refactoring?

*The process of changing a software system in such a way it does not alter the **external behaviour** of the code, yet improves the **internal structure***

Fowler et. al.: Refactoring

Fowler: Workflows of refactoring

https://www.youtube.com/watch?v=vqEg37e4Mkw

https://www.lynda.com/Developer-Programming-Foundations-tutorials/Welcome/122457/135613-4.html

Code smell

- Examples:

1. Code dublication (put in a method)
2. Very long methods (devide into submethods or even classes)
3. To many comments (use selfdescribing methods)

Why refactor?

- Reality
  - Extremely hard to get the design right the first time
  - Hard to understand the business domain fully
  - Hard to understand the requirements – even if the users do
  - Hard to predict how the system will evolve over the next years
  - The original design is often inadequate
  - The system get fragile over time – and thus harder to maintain
- Refactoring helps by
  - Manipulate she system in a safe environment
  - Recreate a situation where evolution is possible
  - Understand existing code

Write to people not the compiler

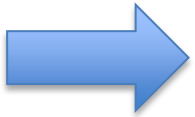
Why refactor?

Economy!!

- Quality
- Professionalism
- The correct

Before

- Unreadable code
- Duplicated code
- Complex code
- Hard to modify



After

- Easier to understand
- Cheaper to modify
- ‘Clean’ code
- Better code

Readability

Which part is the easier to read?

```
if (date.before(summer_start) || (summer_end))
  charge = quantity * winterRate + winterServiceCharge;
else
  charge = quantity * summerRate;
```

```
if (isSummer(date))
  charge = summerCharge(quantity);
else
  charge = winterCharge(quantity);
```

When to refactor?

When:

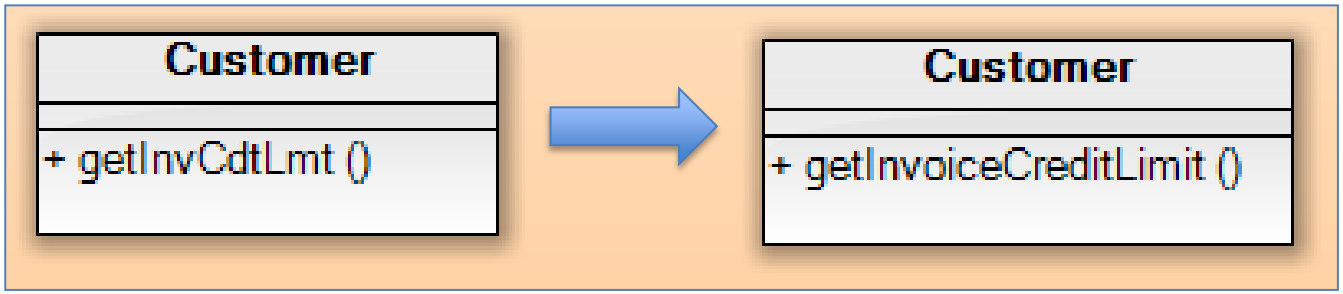
- After new features has been implemented
- Errors are corrected
- During code review
- Only while refactoring
  - Do not add new features while refactoring

When not to refactor:

- While solving a code problem.
- Sometimes the better approach is to ditch it all – and start over

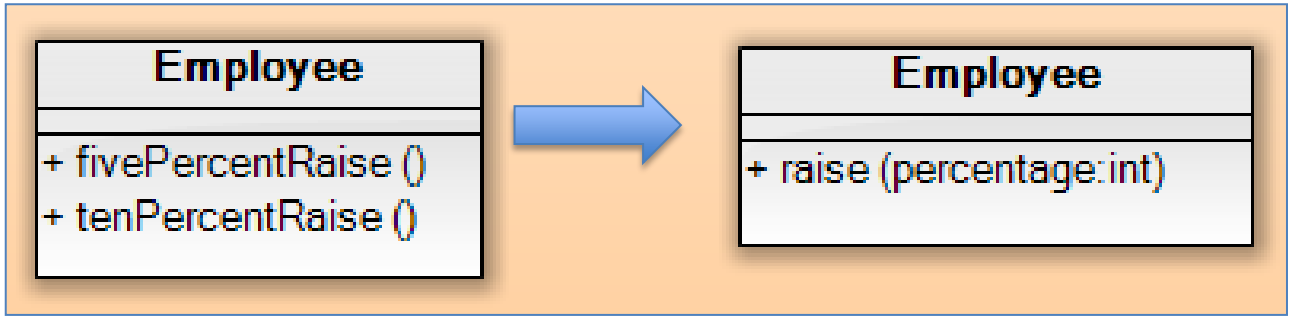
Rename method

- If the name of a method does not reveal its purpose
- Change the name of the method



Parameterize method

- If several methods do similar things but with different values contained in the method body
- Create one method that uses a parameter for the different values.



Preserve whole object

- If you are getting several values from an object and passing these values as parameters in a method call
- Send the whole object instead

```
int low = daysTempRange().getLow();
int high = daysTempRange().getHigh();
withinPlan = plan.withinRange(low, high);
```

```
withinPlan = plan.withinRange(daysTempRange());
```

Inline temp

- If you have a temp that is assigned to once with a simple expression, and the temp is getting in the way of other refactoring
- Replace all references to that temp with the expression

```
double basePrice = someOrder.basePrice();
return (basePrice > 1000)
```

```
return (someOrder.basePrice() > 1000)
```

Inline method

- If 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;
}
```

Replace temp with query

- If you are using a temporary variable to hold the result of an expression
- 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

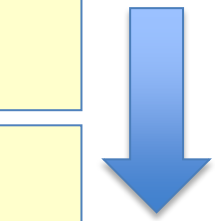
```
double basePrice = quantity * itemPrice;
if (basePrice > 1000)
  return basePrice * 0.95;
else
  return basePrice * 0.98;
```

```
double basePrice() {
  return quantity * itemPrice;
}
...
if (basePrice() > 1000)
  return basePrice() * 0.95;
else
  return basePrice() * 0.98;
```

Extract method (My favorite)

- If you have a code fragment that can be grouped together
- Turn it into a method whose name explains the purpose of the method

```
void printOwing(double amount) {
   printBanner();
! // (comment is a code smell tell):
   //print details
   System.out.println (name: + _name);
   System.out.println (amount + amount);
}
```



```
void printOwing(double amount) {
   printBanner();
   printDetails(amount);
}

void printDetails (double amount) {
   System.out.println (name: + _name);
   System.out.println (amount + amount);
}
```

Decompose conditional

- If you have a complicated conditional (if-then-else) statement
- Extract methods from the condition, **then** part and **else** part

```
if (date.before(summer_start) || (summer_end))
  charge = quantity * winterRate + winterServiceCharge;
else
  charge = quantity * summerRate;
```

```
if (isSummer(date))
  charge = summerCharge(quantity);
else
  charge = winterCharge(quantity);
```

Replace Nested Conditional with Guard Clauses

- If a method has conditional behaviour that does not make clear the normal path of execution.
- Use guarded clauses for all the special cases

```
double getPayAmount() {
  double result;
  if (_isDead) result = deadAmount();
  else {
     if (_isSeparated) result = separatedAmount();
     else {
       if (_isRetired) result = retiredAmount();
       else result = normalPayAmount();
     }; }   return result;
  }
```

```
double getPayAmount() {
  if (_isDead) return deadAmount();
  if (_isSeparated) return separatedAmount();
  if (_isRetired) return retiredAmount();
 return normalPayAmount();   }
```

Introduce assertion

- If a section of code assumes something about the state of the program
- Make the assumption explicit with an assertion

```
double getExpenseLimit() {
//should have either expense limit or a primary project
  return (expenseLimit != NULL_EXPENSE) ?
  expenseLimit:
  primaryProject.getMemberExpenseLimit();
}
```

```
double getExpenseLimit() {
  assert(expenseLimit != NULL_EXPENSE ||
       primaryProject != null);
  return (expenseLimit != NULL_EXPENSE) ?
  expenseLimit:
  primaryProject.getMemberExpenseLimit();
}
```

More about refactoring

- Fowler video: https://www.youtube.com/watch?v=vqEg37e4Mkw
- Catalogue of refactoring:
- http://refactoring.com/catalog/
- Refactoring a first example:
- http://www.cs.unc.edu/~stotts/723/refactor/chap1.html

- Refactoring to patterns:
- http://industriallogic.com/xp/refactoring/catalog.html

Exercise

- Try to refactor the part of your project code that
  1. Is working
  2. Looks messy and is hard to read.