2023年春季学期

软件设计实践

软件重构与维护

谢 涛　　马 郓

软件重构与维护

软件维护

童话故事

(Demeyer et al., 2002)

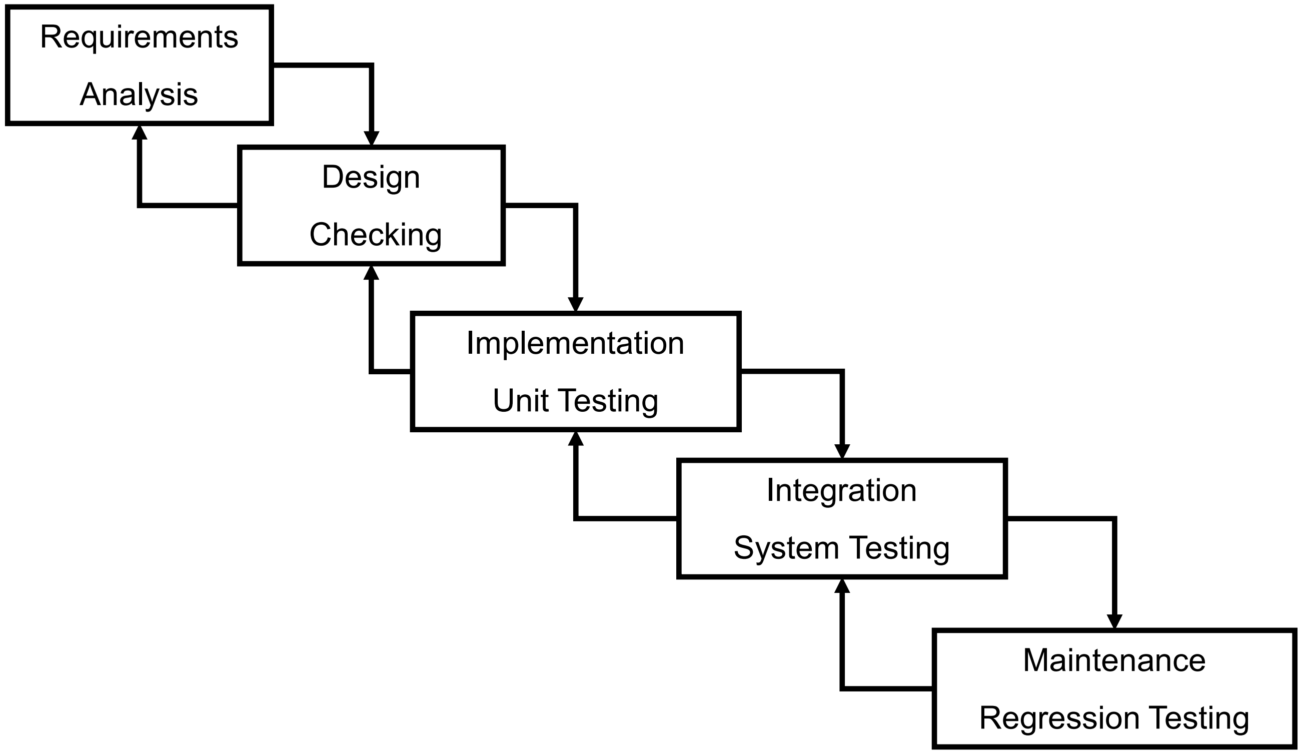
"Once upon a time there was a Good Software Engineer whose Customers knew exactly what they wanted.

The Good Software Engineer worked very hard to design the Perfect System that would solve all the Customers' problems now and for decades.

When the Perfect System was designed, implemented and finally deployed, the Customers were very happy indeed.

The Maintainer of the system had very little to do to keep the Perfect System up and running, and the Customers and the Maintainer lived happily every after."

现实: 软件变更


Requirements
Analysis
Design
Checking
Implementation
Unit Testing
Integration
System Testing
Maintenance
Regression Testing

现实: 软件越来越复杂

(Potvin & Levenberg, 2016)

Google 代码库 (2015年1月情况)

- > 2.5万 Google 开发者
- ~ 10亿 文件
  - ~ 900万 源文件
  - ~ 20亿 行源代码
- ~ 3.5 千万commits （18年间）
  - ~1.5 千万行代码变更/周 （2014年）
- ~ 4万 commits/每工作日

软件维护类型

(IEEE, 2006)

- 软件维护: modifications of software performed after delivery

Correction

- Corrective 修正式维护
  - Correct discovered faults
- Preventive 预防式维护
  - Prevent occurrences of faults/failures (e.g., refactoring, document improvement)

Enhancement

- Adaptive 适配式维护
  - Adapt to a changed or changing environment
- Perfective 完善式维护
  - Implement new or changed user requirements

维护类型比较

Maintenance Type	Request often from customer?	Effect of change visible to customers?	Need testing? (if yes, for what?)
Corrective	X	X	X (reproduce failure, confirm fix, not break)
Preventative			X (not break)
Adaptive		X?	X (confirm adaptation, not break)
Perfective	X	X	X (test new features, not break)

文献

- Demeyer, S., Ducasse, S., & Nierstrasz, O. (2002). *Object-oriented reengineering patterns (1st Edition)*. Morgan Kaufmann. ISBN: 9781558606395
- Potvin, R., & Levenberg, J. (2016). Why Google stores billions of lines of code in a single repository. *Communications of the ACM, 59(7)*, 78-87. doi: https://doi.org/10.1145/2854146
- IEEE. (2006). IEEE 14764-2006. *ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance*. Retrieved from https://standards.ieee.org/standard/14764-2006.html

软件重构与维护

软件规模和复杂度度量

规模和复杂度度量

- 代码规模（size）
  - Number of lines of code (LOC)
  - Number of methods
  - Number of classes
  - Number of files

- 代码复杂度（complexity）
  - Cyclomatic complexity
  - Object-oriented-specific metrics (in short as OO metrics)

代码行 (LOC)

- aka Source Lines of Code (SLOC)
- Easy to measure
- Correlating to time to build/maintain
- Between two programs of “equal functionality,” it may be better to have fewer lines of code

度量代码行的挑战

- Blanks? Comments?

Program 1:

```
copy(char *p,*q) {while(*p) *q++ = *p++;}
```

Program 2:

```
copy(char *p,*q) {
   while(p) {
        *q++ = *p++;
   }
}
```

代码行: 语言是什么很关键

- Assembly code may be 2-3X longer than C code
- C code may be 2-3X longer than C++/Java code
- C++/Java code may be 2-3X longer than …

- Lines of code is valid metric when
  - Same language
  - Standard formatting
  - Code having been reviewed

复杂度度量

- Complex software is
  - Hard to understand
  - Hard to change
  - Hard to reuse

- Some metrics for complexity
  - Cyclomatic complexity
  - Object-oriented-specific metrics (in short as OO metrics)

Cyclomatic 复杂度

(McCabe, 1976)

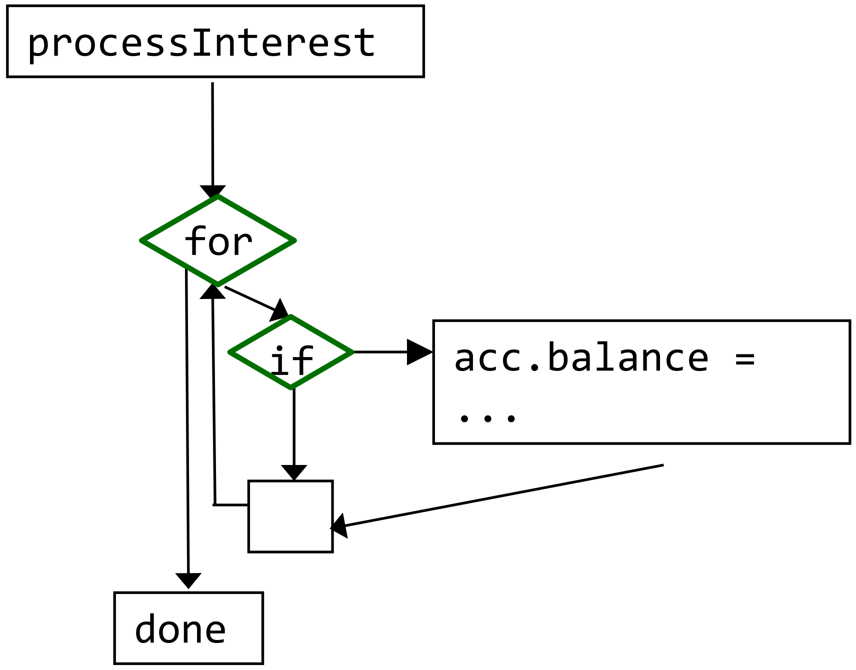
- Measuring logical complexity at the method level
- Informing #tests needed to execute a method comprehensively
  - upper bound #tests necessary to achieve complete branch coverage
- Simple calculation: #decisionPoints + 1
  - Such calculation valid only when there is 1 exit point; otherwise use
    - #decisionPoints - #exitPoints + 2
  - Decision points counted at the lowest, machine-level instructions
    - E.g., if (x > 0) && (y > 0)) counted as 2 decision points
- A method with a cyclomatic complexity > 10 considered as hard to test and fault prone

if, while, for等

度量Cyclomatic复杂度样例

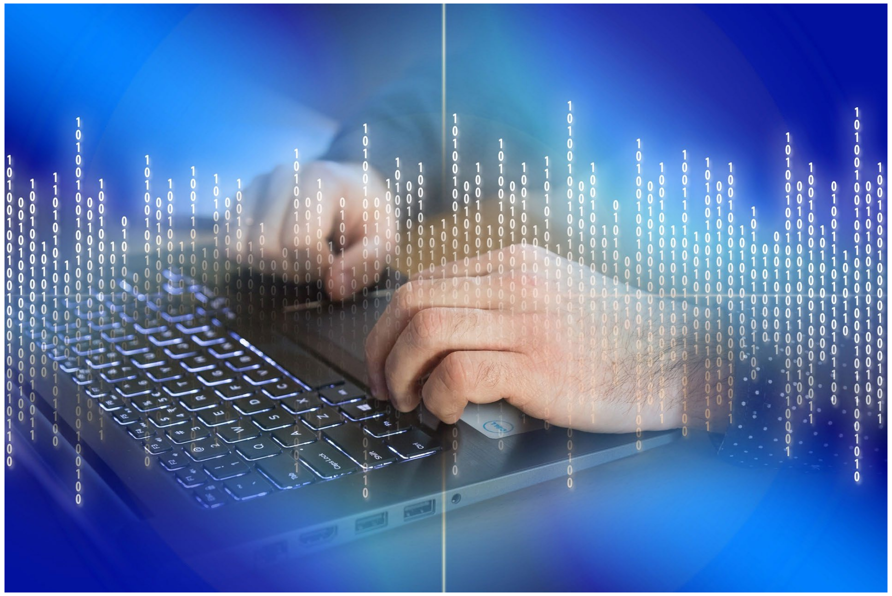
(McCabe, 1976)

```
void processInterest() {
   for (acc : accounts) {
         if (hasInterest(acc)) {
               acc.balance = acc.balance +
              acc.balance * acc.interest;
         }
    }
```



- #decisionPoints + 1
- #edges - #nodes + 2
- #regions of the flow graph

利用规模/复杂度度量的方式


Refactoring Candidate

Code Review/Testing Focus

好的度量值

Metric	**Desirable Value**
Lines of Code	Lower
Cyclomatic Complexity	Lower

文献

- McCabe, T.J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering* (4), 308–320. doi: https://doi.org/10.1109/TSE.1976.233837

软件重构与维护

面向对象度量

面向对象度量

(Chidamber & Kemerer, 1994)

- ❑Weighted Methods Per Class (WMC)
- ❑Depth of Inheritance Tree (DIT)
- ❑Number of Children (NOC)
- ❑Coupling between Object Classes (CBO)
- ❑Response for a Class (RFC)
- ❑Lack of Cohesion in Methods (LCOM)

Weighted Methods Per Class (WMC)

- WMC for a class: sum of complexities of the methods in the class
- Possible ways of measuring complexities of a method
  - 1 (i.e., WMC: number of methods)
  - #Lines of code
  - #method calls
  - Cyclomatic complexity
- Used to predict time/effort required to develop and maintain a class

- The larger #methods in a class, the greater the potential impact on children
- Classes with large #methods are more likely to be application specific and less reusable

Depth of Inheritance Tree (DIT)

- DIT for a class: length from the class to the root of the class hierarchy tree, measured by #ancestor classes.
  - multiple inheritance: max length from the class to the root of the tree

- The deeper a class is in the hierarchy, the more methods it reuses
- The deeper a class is in the hierarchy, the more methods it inherits and so it is harder to predict its behavior, and deeper trees are more complex

Number of Children (NOC)

- NOC for a class: #immediate subclasses
- A class with more children is more reused on the methods defined on the class
- A class might have a lot of children because of misuse of subclassing
- A class with a large number of children is probably very important (due to effects on changes on all the children) and needs a lot of testing

Coupling Between Object Classes (CBO)

- CBO for a class: #other classes to which the class is coupled
  - Class A is coupled to class B: a method of A invokes a method of B
- A class is desirable to be coupled with only abstract classes high in the inheritance hierarchy
- Higher coupling makes the class harder to test
- Higher coupling makes designs harder to change
- Higher coupling makes classes harder to reuse

Response For a Class (RFC)

- RFC for a class:
  |methods in the class U methods called by the class|
  - A set of methods that can potentially be executed in response to a message received by an object of that class

- If a large #methods can be invoked in response to a message, testing becomes more complicated
- The more methods that can be invoked from a class, the greater the complexity of the class

Lack of Cohesion in Methods (LCOM)

- LCOM for a class: #pairs of methods that don’t share instance variables - #pairs of methods that share instance variables

- Cohesiveness of methods is a sign of encapsulation
- Lack of cohesion implies classes should be split

好的度量值

Metric	**Desirable Value**
Weighted Methods Per Class (WMC)	Low (tradeoff)
Depth of Inheritance Tree (DIT)	Low (tradeoff)
Number of Children (NOC)	Low (tradeoff)
Coupling between Object Classes (CBO)	Lower
Response for a Class (RFC)	Lower
Lack of Cohesion of Methods (LCOM)	Lower

文献

- Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6), 476-493. doi: https://doi.org/10.1109/32.295895

软件重构与维护

重构

重构

(Fowler et al., 1999)

- Changing the internal structure of a program without changing its external behavior
  - Done to make code easier to understand and cheaper to modify

- Key part of extreme programming (XP)
- Key part of software evolution
- Key part of making reusable software

- Examples refactorings
  - Extract Method
  - Extract Variable

样例: Extract Method

- A code portion in need of being grouped together ➔ a method with informative name to explain its purpose
- Increase code reusability and readability

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

   //print details

   System.out.println (“name” + _name);
   Sysetm.out.println (“amount” + amount);
}
```

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

void printDetails(double amount) {

   System.out.println (“name” + _name);
   Sysetm.out.println (“amount” + amount);
}
```

Extract Method: Steps to Perform

- Create a new method and name it to reflect its purpose
- Copy extracted code into new target method
- Replace extracted code with method call
- Scan extracted code for references to variable local to source method ➔ turn the references to parameters
- Check whether local scope variables are modified by extracted code
  - One: return it as result
  - More: cannot extract as it stands

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

   //print details

   System.out.println (“name” + _name);
   Sysetm.out.println (“amount” + amount);
}
```

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

void printDetails(double amount) {

   System.out.println (“name” + _name);
   Sysetm.out.println (“amount” + amount);
}
```

样例: Extract Variable

- Complicated expression ➔ a temp variable with informative name to explain its purpose
- Help break a complex expression (e.g., conditional logic) down into something less error prone

```
if ((platform.toUpperCase().indexOf("MAC") > -1) &&
     (browser.toUpperCase().indexOf("IE") > -1)   &&
     (wasInitialized() && resize > 0)   {

          // do something

}
```

```
final boolean isMacOs =      platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;
final boolean wasResized = resize > 0;

if (isMacOS && isIEBrowser && wasInitialized() && wasResized) {
    // do something
}
```

Steps:

- Declare a final temp variable and set to the result of (or parts of) a complex expression
- Replace the expression (or its parts) with the value of the temp variable

更多重构的例子

(Fowler, 2018)

- Extract method
  - Inline method
- Extract variable
  - Inline variable
- Rename, rename, rename
- Move method
- … more see online catalog of refactorings (Fowler, 2018)

Change Design XOR Functionality

- Separate changing behavior from refactoring
  - Changing behavior requires new tests
  - Refactoring must pass all tests

- Only refactor when you need to
  - Before you change behavior
  - After you change behavior
  - To understand

重构工具支持

- Deciding where to refactor
  - Tools for measuring cohesion, size, etc.
  - Tools for measuring code duplication/cloning
- Performing the change
  - Many modern IDEs

具体样例：Introduce Parameter Object

(Fowler, 1999)

- There is a group of parameters that naturally go together
  - Many methods have same parameters
  - Parameters are passed unchanged from one method to another
  - Method has too many parameters

Major steps:

- Make a new class for the group of parameters
- Add a new parameter whose type is the new class
- Incrementally remove each original parameter and replace its references (in the method body) to the value stored in the new parameter

Code under refactoring of Introduce Parameter Object

```
class Account {
       …
  double getFlowBetween(Date start, Date end) {
    double result = 0;
    Iterator e = _entries.elements();
    while (e.hasNext()) {
       Entry each = (Entry) e.next();
       Date date = each.getDate();
       if (date.equals(start) || date.equals(end) ||
          (date.after(start) && date.before(end))) {
          result += each.getValue();
       }
    }
    return result;
  }
```

Step 1: Make a new class for the group of parameters

```
class DateRange {
  private final Date _start;
  private final Date _end;

  DateRange (Date start, Date end) {
    _start = start;
    _end = end;
  }
  Date getStart() {
    return _start;
  }
  Date getEnd() {
    return _end;
  }
}
```

Step 2: Add a new parameter whose type is the new class

```
class Account {
  ...
  double getFlowBetween(Date start, Date end, DateRange range)
  {
    double result = 0;
    Iterator e = _entries.elements();
    while (e.hasNext()) {
       Entry each = (Entry) e.next();
       Date date = each.getDate();
       if (date.equals(start) || date.equals(end) ||
          (date.after(start) && date.before(end))) {
          result += each.getValue();
       }
    }
    return result;
  }
```

Step 3: Use a new object for the parameter in all the callers

```
double flow = anAccount.getFlowBetween(startDate, endDate);
```

➔

```
double flow = anAccount.getFlowBetween(startDate, endDate,
   new DateRange(null, null))
```

Step 4a: For each of the original parameters, modify caller to store parameter in the new object and omit parameter from call

```
double flow = anAccount.getFlowBetween(startDate, endDate,
   new DateRange(null, null))
```

➔

```
double flow = anAccount.getFlowBetween(endDate,
   new DateRange(startDate, null))
```

Step 4b: For each of the original parameters, modify method body to replace the references to the original parameter with the value stored in the new parameter

```
class Account {
   ...
  double getFlowBetween(Date end, DateRange range) {
    double result = 0;
    Iterator e = _entries.elements();
    while (e.hasNext()) {
       Entry each = (Entry) e.next();
       Date date = each.getDate();
       if (date.equals(range.getStart()) || date.equals(end) ||
          (date.after(range.getStart()) && date.before(end))) {
          result += each.getValue();
       }
    }
    return result;
  }
```

Step 5a: For each of the original parameters, modify caller to store parameter in the new object and omit parameter from call

```
double flow = anAccount.getFlowBetween(endDate,
   new DateRange(startDate, null))
```

➔

```
double flow = anAccount.getFlowBetween(
   new DateRange(startDate, endDate))
```

Step 5b: For each of the original parameters, modify method body to replace the references to the original parameter with the value stored in the new parameter

```
class Account {
   ...
   double getFlowBetween(DateRange range) {
     double result = 0;
     Iterator e = _entries.elements();
     while (e.hasNext()) {
        Entry each = (Entry) e.next();
        Date date = each.getDate();
        if (date.equals(range.getStart()) ||
           date.equals(range.getEnd()) ||
           (date.after(range.getStart()) &&
                  date.before(range.getEnd()))) {
           result += each.getValue();
        }
     }
     return result;
   }
```

Introduce Parameter Object → Extract Method

After introducing a parameter object, check see whether code should be moved to its methods

```
double getFlowBetween(DateRange range) {
  ...
   while (e.hasNext()) {
        Entry each = (Entry) e.next();
        Date date = each.getDate();
        if (date.equals(range.getStart()) ||
             date.equals(range.getEnd()) ||
             (date.after(range.getStart()) &&
             date.before(range.getEnd()))) {
             result += each.getValue();
        }
   }
   ...
}
```

```
class DateRange {
     ...
        boolean includes(Date arg) {
        return (arg.equals(_start) ||
  arg.equals(_end) ||
              (arg.after(_start) && arg.before(_end)));
        }
  }

if (range.includes(date) {
     result += each.getValue();
}
```

重构指南

- Refactorings should be small
- Check after each step to make sure you didn’t make a mistake
- One refactoring leads to another
- Major change requires multiple refactorings

文献

- Fowler M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code (1st Edition).* Addison-Wesley Professional. ISBN: 9780201485677
- Fowler M. (2018). online catalog of refactorings. Retrieved from https://www.refactoring.com/catalog/

软件重构与维护

代码味道（Code Smell）

Code Smells

(Fowler et al., 1999)

If it stinks, change it.

-- Grandma Beck, discussing child-rearing philosophy

- "... certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring."
- They are clear signs that your design is starting to decay
- Long term decay leads to "software rot"

Example Code Smells

(Fowler et al., 1999)

- ❑ Duplicated code
- ❑ Long method
- ❑ Long parameter list
- ❑ …

Duplicated Code

- **Difficult to maintain**
  - **E.g., You fix faults in some instances of duplicated code but not all instances**
- **Duplicate methods in subclasses**
  - **Refactor with move to superclass, possibly create superclass**
- **Duplicate expressions in the same class**
  - **Refactor with Extract Method**
- **Duplicate expressions in different classes**
  - **Refactor with Extract Method, move to common component**

Example of Duplicated Code

```
sqrt(pow(loc1.getX() - loc2.getX(), 2) +
     pow(loc1.getY() - loc2.getY(), 2))
```

➔

```
sqrt(square(loc1.getX() - loc2.getX()) +
     square(loc1.getY() - loc2.getY()))

double square(double d) {
    return pow(d, 2);
}
```

Long Method

- Won’t fit on a page
- Can’t think of whole thing at once; difficult to understand

- Refactor with Extract Method
  - Loop body
  - Places where there is (or should be) a comment (don’t write comments but name methods)

Long Parameter List

- Hard to understand
- Unstable, easily expanded when more data are needed to pass along

- Refactor with Introduce parameter object

- Worthwhile especially when there are several methods with the same parameter list, and they call each other

Object-Oriented Code Smells

(Fowler et al., 1999)

- Code smells in object-oriented code

- Large Class
- Message Chain
- Feature Envy
- Data Class
- …

Large Class

- Difficult to understand or maintain
- > 24 methods, or 6 instance variables

- Refactor with
  - Extract Class
    - Group instance variables into component classes, e.g., `depositAmount` and `depositCurrency`
  - Extract Subclass

Message Chain

- Long list of method calls, e.g.,

```
customer.getAddress().getState();
window.getBoundingbox().getOrigin().getX();
```

➔

- Shorter calls, e.g.,

```
customer.getState()
window.leftBoundary()
```

- A client asks an object for another object and then asks that object for another object etc.
  - Bad because client depends on the structure of the navigation

Feature Envy

- ❑ Code "wishes" it were in another class
- ❑ Refactor with Move Method

Example:

```
    teacher.getClasses().add(thisClass);
    teacher.setClassLoad(teacher.getClassLoad() + 1);
```

➔

```
    teacher.addClass(thisClass);
```

Data Class

- Class has no methods except for getter and setters
- Refactor with
  - Looking for missing methods (feature envy?) and move them to the class
  - Merging with another class
- Example: DateRange

Example Data Class

```
class DateRange {
    private final Date _start;
   private final Date _end;
   DateRange (Date start, Date end) {
      _start = start;
      _end = end;
   }
   Date getStart() {
      return _start;
   }
   Date getEnd() {
      return _end;
   }
}
```

```
double getFlowBetween(DateRange range) {
  ...
   while (e.hasNext()) {
        Entry each = (Entry) e.next();
        Date date = each.getDate();
        if (date.equals(range.getStart()) ||
            date.equals(range.getEnd()) ||
            (date.after(range.getStart()) &&
            date.before(range.getEnd()))) {
            result += each.getValue();
        }
   }
   ...
}
```

```
if (range.includes(date) {
     result += each.getValue();
}
```

```
boolean includes(Date arg) {
return (arg.equals(_start) || arg.equals(_end) ||
   (arg.after(_start) && arg.before(_end)));
}
```

文献

- Fowler M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code (1st Edition).* Addison-Wesley Professional. ISBN: 9780201485677