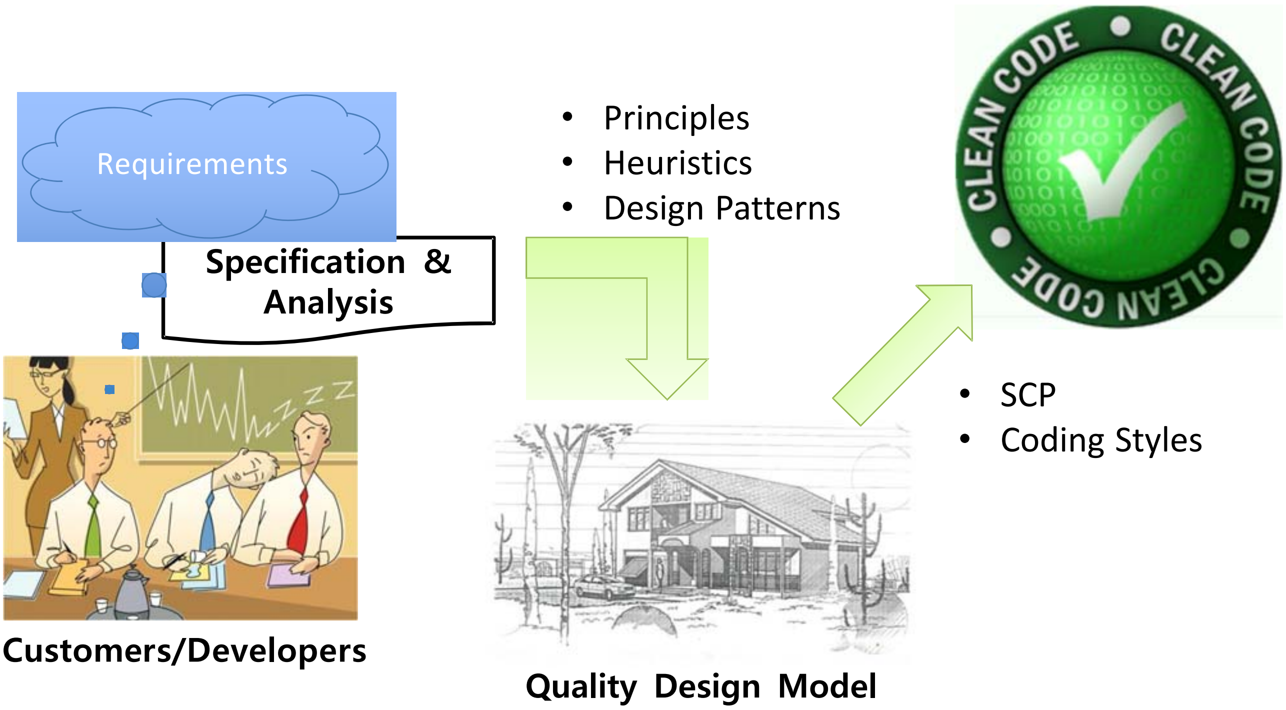
Design Quality and Metrics

SW Design & Coding Concepts



Requirements

Specification & Analysis

Customers/Developers

- Principles
- Heuristics
- Design Patterns

Quality Design Model

- SCP
- Coding Styles

Software Quality

External Quality

- Does SW behave correctly?
- Are the produced results correct?
- Does the software run fast?
- Is the software UI easy to use?

Internal Quality

- Is the code easy to read and understand?
- Is the design & code well structured?
- Is the design & code easy to modify?

Quality Attributes

Quality Attribute	Definition
Understandability	The ease with which the design fragment can be comprehended.
Changeability	The ease with which a design fragment can be modified (without causing ripple effects) when an existing functionality is changed.
Extensibility	The ease with which a design fragment can be enhanced or extended (without ripple effects) for supporting new functionality.
Reusability	The ease with which a design fragment can be used in program context other than the one for which the fragment was originally designed.
Testability	The ease with which a design fragment can support the detection of defects within it via testing.
Reliability	The ease with which a design fragment can the correct realization of the functionality and helps guard against the introduction of runtime problems.

Can You Read ?

```
#include <stdio.h>
main(t,_,a)
char *a;
{
return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):
1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')#\
}'+}##(!!/")
  :t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
    :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);
}
```

The winning code of International Obfuscated C Contest“ in 1988

Code Readability: its Output

```
On the first day of Christmas my true love gave to me
a partridge in a per tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a per tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a per tree.

…

On the twelfth day of Christmas my true love gave to me
twelve drummers drumming, eleven pipers piping, ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a per tree.
```

Readability

- Use a consistent style: new line, space, indent, …

```
double[] doSomething(int x1, int x2, int x3) {double y[] = new double[2]; Q =
x2*x2 – 4*x1*x3 ; if ( Q > 0 ) {
y[0] = ( – x2 + Math.sqrt(Q) ) / (2*x1) ; y[1] = ( – x2 – Math.sqrt(Q) ) / (2*x1) ; }
else
if ( Q == 0 ) y[0] = y[1] = (-x2) / (2*x1) ; return solution ;
}
```

```
double[] doSomething(int x1, int x2, int x3) {
        double y[] = new double[2] ;
        Q = x2*x2 – 4*x1*x3 ;
        if ( Q > 0 ) {
                y[0] = ( – x2 + Math.sqrt(Q) ) / (2*x1) ;
                y[1] = ( – x2 – Math.sqrt(Q) ) / (2*x1) ;
        } else if ( Q == 0 )
                y[0] = y[1] = (-x2) / (2*x1) ;
        return y ;
}
```

Understandability

- The ease with which the design fragment can be comprehended.

```
double[] doSomething(int x1, int x2, int x3) {
        double y[] = new double[2] ;
        Q = x2*x2 – 4*x1*x3 ;
        if ( Q > 0 ) {
                y[0] = ( – x2 + Math.sqrt(Q) ) / (2*x1) ;
                y[1] = ( – x2 – Math.sqrt(Q) ) / (2*x1) ;
        } else if ( Q == 0 )
                y[0] = y[1] = (-x2) / (2*x1) ;
        return y ;
}
```

Understandability

- Meaningful names are important for understandability

```
double[] quadraticEquation(int a, int b, int c) {
  double solution[] = new double[2] ;
  double D = b*b – 4*a*c ;
  if ( D > 0 ) {
    solution[0] = ( - b + Math.sqrt(D) ) / (2*a) ;
    solution[1] = ( - b - Math.sqrt(D) ) / (2*a) ;
  }
  else if ( D == 0 ) {
    solution[0] = solution[1] = (-b) / (2*a) ;
  }
  return solution ;
}
```

Poor Understandability

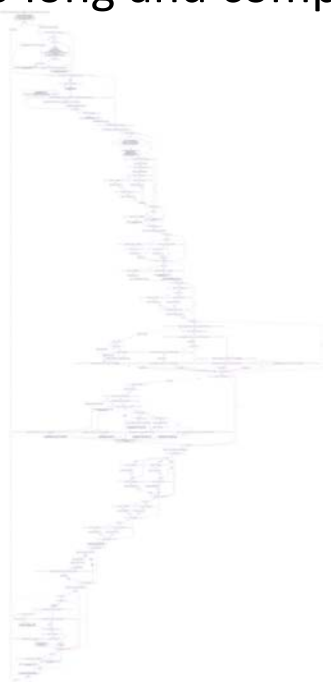
```
   float compute(int k, int x) {
     float result = 0.0F;
     switch (k) {
     case 0: result += 2;
       if ( x> 2)
          result += (x- 2) * 1.5; break;
     case 1:
        result += 1.5;
       if (x> 3)
          result += (x- 3) * 1.5; break;
     case 2:
        result += x * 3; break;
     default: break;
     }
     return result ;
   }
};
```

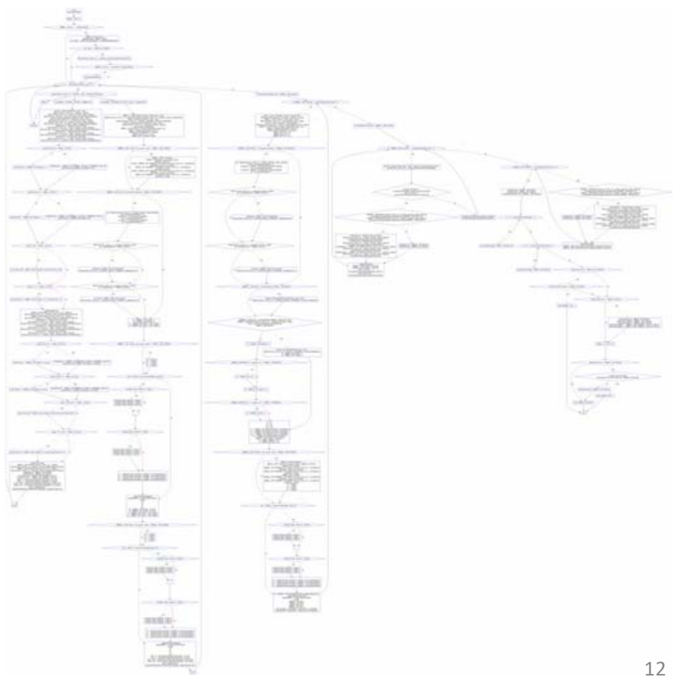
Good Understandability

```
enum MovieKind { REGULAR, CHILDREN, NEW_RELEASE } ;
float getCharge(const MovieKind kind, const int daysRented) {
      float charge = 0.0F;
      switch (kind) {
      case REGULAR:
         charge += 2;
         if (daysRented > 2) {
            charge += (daysRented - 2) * 1.5;
         }
         break;
      case CHILDREN:
         charge += 1.5;
         if (daysRented > 3) {
            charge += (daysRented - 3) * 1.5;
         }
         break;
      case NEW_RELEASE:
         charge += daysRented * 3;
         break;
      default: break;
      }
      return charge;
   }
};
```

Understandability

- Too long and complex code is less understandable





Cyclomatic Complexity (CC)

- A quantitative measure of the number of **linearly independent paths** through a source code.
- Developed by Thomas J. McCabe, Sr. in 1976.
- Computed using the **control flow graph**.
- # test cases == cyclomatic complexity
- **V(G) = decision points + 1**
- Threshhold ≈ 10

CC Example

```
static String flipFlop(int min, int max)
{
      if (max < min) return null;

      String result = "";

      for (int i = min; i < max; i++) {
          if (i % 3 == 0)
              result += "flip";
          if (i % 5 == 0)
              result += "flop";
      }

      return result;
}
```

V(G) =

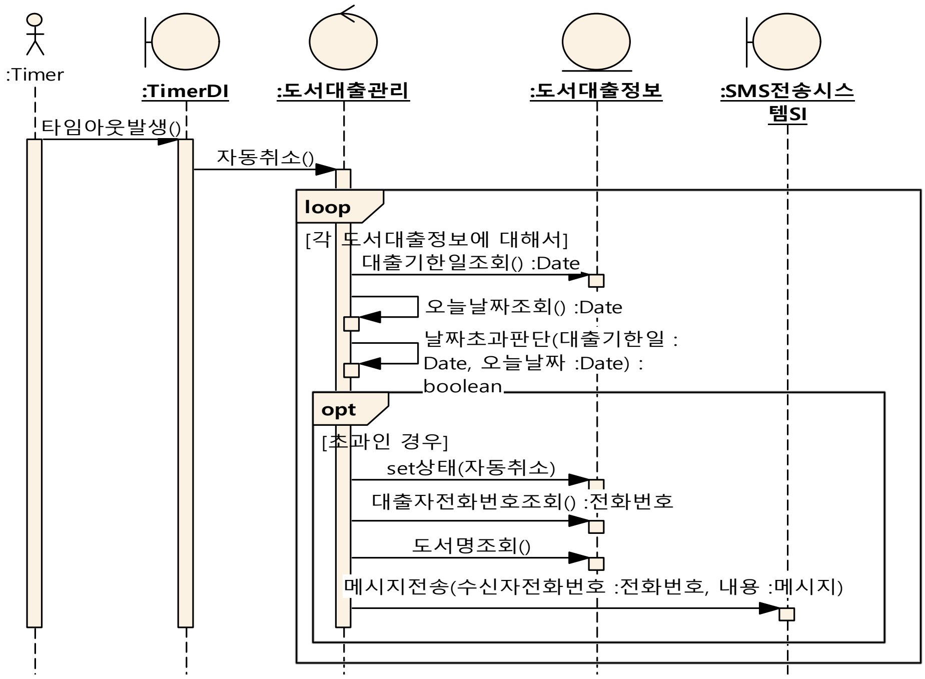
CC and Defect Risk

CC	Description	Risk
1-4	A simple procedure	Low
5-10	A well structured and stable procedure	Low
11-20	A more complex procedure	Moderate
21-50	A complex procedure, **alarming**	High
>50	An error-prone, extremely troublesome procedure	Very High

http://www.aivosto.com/project/help/pm-complexity.html

CC at Design Phase

- CC of 도서대출관리::자동취소()



Nesting Depth

- Number of Structuring Levels

```
public void function1(int i) {
  // ...
  if ( i >= 0 )
    // ...
  else
    // ...
}
```

Nesting Depth = 1

```
int function2(int x) {
  int y = 0 ;
  for ( int  i = 0 ; i < x ; i ++ ) {
    if ( i >= 10 )
      // ...
    else
      // y = ..
  }
  if ( y >= 0 ) return y ;
  else return –y ;
}
```

Nesting Depth = 2

NPath

- The number of **acyclic execution paths** through a method.
- Threshold ≈ 200

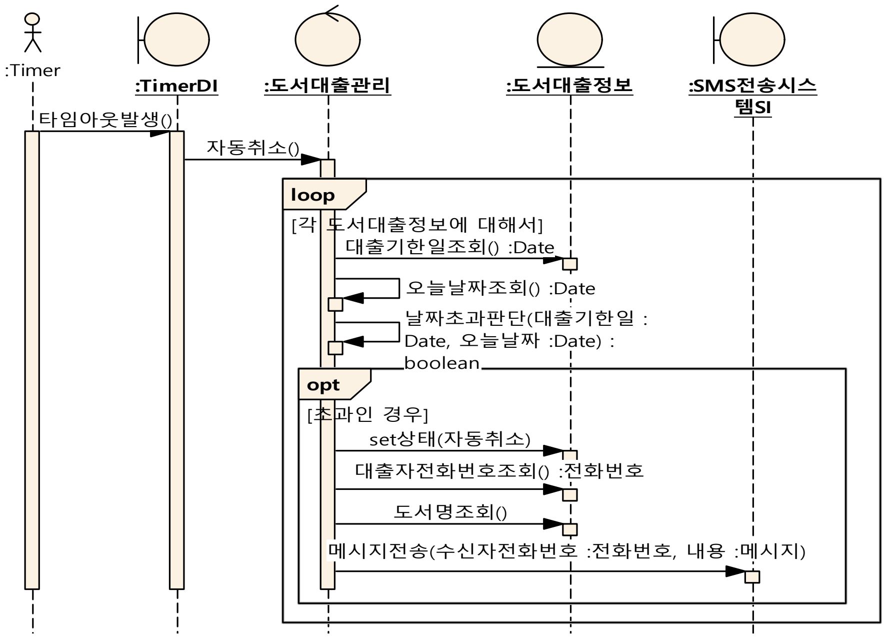
NPath Example

```
static String flipFlop(int min, int max)
{
      if (max < min) return null;

      String result = "";

      for (int i = min; i < max; i++) {
          if (i % 3 == 0)
              result += "flip";
          if (i % 5 == 0)
              result += "flop";
      }

      return result;
}
```

NPath =

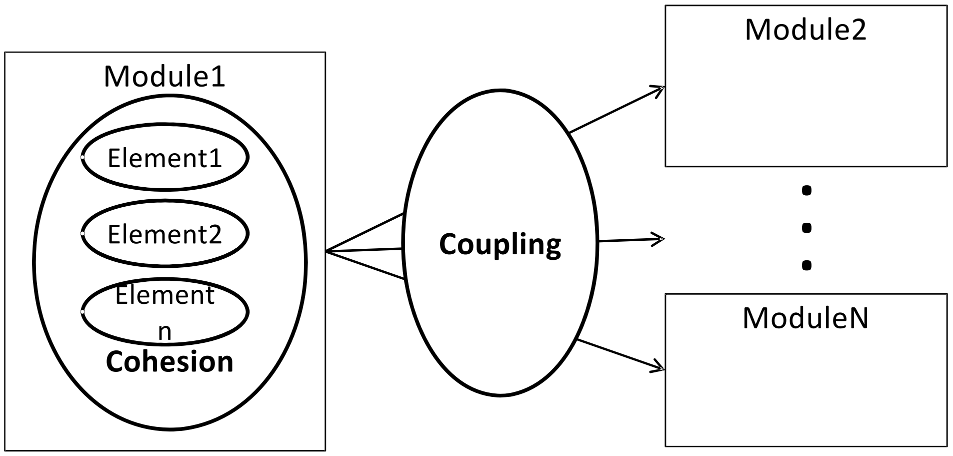
Nesting Depth and NPath at Design Phase

- Nesting Depth of 도서대출관리::자동취소()
- NPath of 도서대출관리::자동취소()



Cohesion and Coupling

- **Cohesion**
  - Strength of functional relatedness of elements within a module
  - The degree to which a class has a single, well-focused purpose
- **Coupling**:
  - Degree of interdependence between modules



Module1

Element1

Element2

Element n

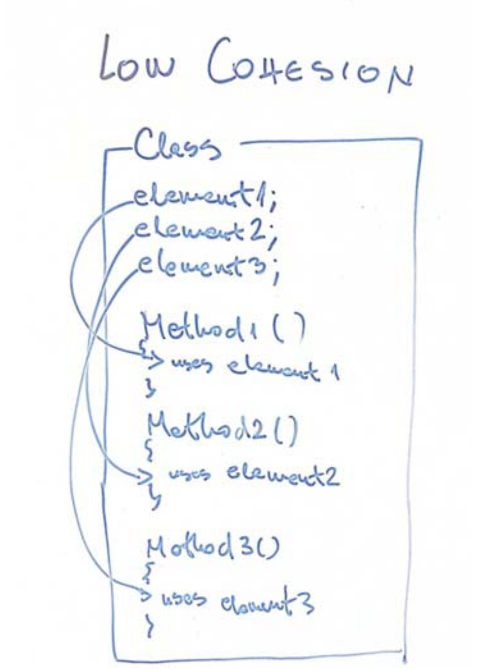
Cohesion

Coupling

Module2

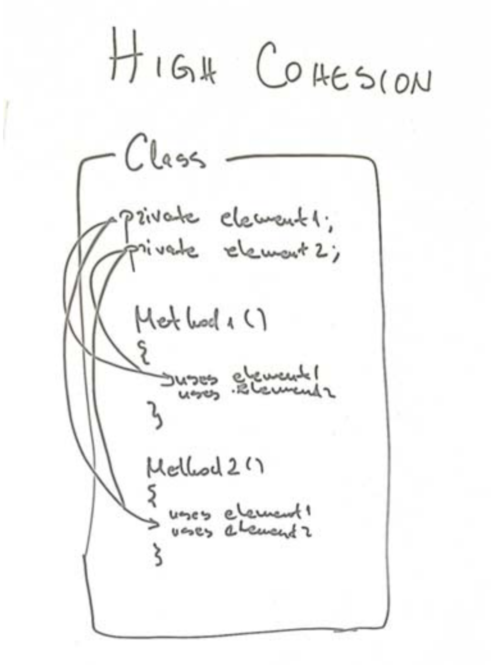
ModuleN

Low Cohesion vs. High Cohesion



```
Class
element1;
element2;
element3;

Method1 ()
{
 uses element 1
}
Method2 ()
{
 uses element2
}

Method3()
{
 uses element3
}
```

COINCIDENTAL COHESION (WORST)



```
Class
private element1;
private element2;

Method1 ()
{
 uses element1
 uses element2
}

Method2 ()
{
 uses element1
 uses element2
}
```

FUNCTIONAL COHESION (BEST)

Types of Cohesion

- Coincidental Cohesion (WORST)
- Logical Cohesion
- Temporal Cohesion
- Procedural Cohesion
- Communicational/Informational Cohesion
- Sequential Cohesion
- Functional Cohesion (BEST)

Which One is More Cohesive?

```
int sumOrProduct(boolean flag, List<Integer> values) {
      if (flag)
            return values.stream()
                                    .reduce(0, (sum, v) -> sum + v);
      else
            return values.stream()
                                    .reduce(1, (product, v) -> product * v);
}
```

```
int sum(List<Integer> values) {
      return values.stream()
                              .reduce(0, (sum, v) -> sum + v);
}

int product(List<Integer> values) {
      return values.stream()
                              .reduce(1, (product, v) -> product * v);
}
```

LCOM (Lack of Cohesion Metrics)

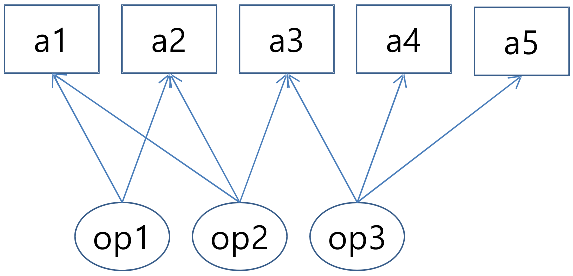
- For each pair of methods in the class:
  - If access disjoint sets of instance variables, increase P by one.
  - If share at least one variable access, increase Q by one.

LCOM1 = P - Q , if P > Q

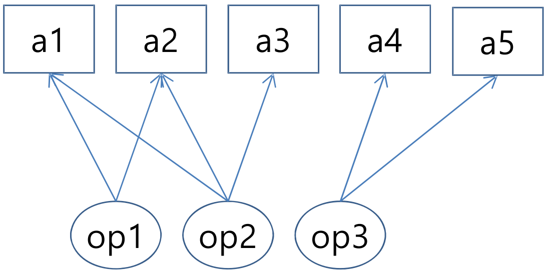
LCOM1 = 0   otherwise

- LCOM1 = 0 indicates a cohesive class.
- LCOM1 > 0 indicates that the class needs or can be split into two or more classes.

LCOM Example



LCOM = 1 – 2 = 0 → 0

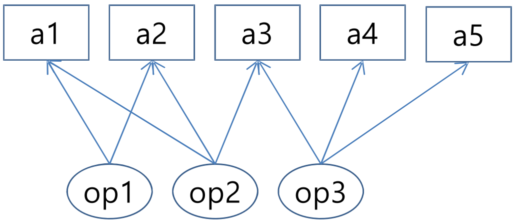


LCOM = 2 – 1 = 1

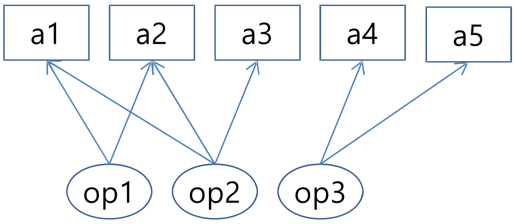
Variations of LCOM

- LCOM2 = 1 – (sum(MF)/M*F) : [0..1]
- LCOM HS (Hendersons-Seller) =
  (M – sum(MF)/F) / (M-1) [0..2]

- M is the number of methods in class
- F is the number of instance fields in the class.
- MF is the number of methods of the class accessing a particular instance field.
- Sum(MF) is the sum of MF over all instance fields of the class



LCOM = 1 – (8/15) = 0.47
LCOM HS = (3-8/5) / 2 = 0.7

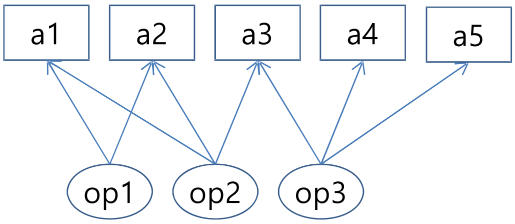


LCOM = 1 – (7/15) = 0.53
LCOM HS = (3-7/5) / 2 = 0.8

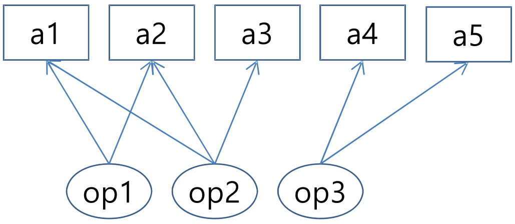
Tight Class Cohesion & Loose Class Cohesion

NP = maximum number of method pairs
 = N * (N-1) / 2 where N is the number of methods
NDC = number of method pairs with direct connections
NIC = number of method pairs with indirect connections

TCC = NDC / NP
LCC = (NDC+NIC) / NP

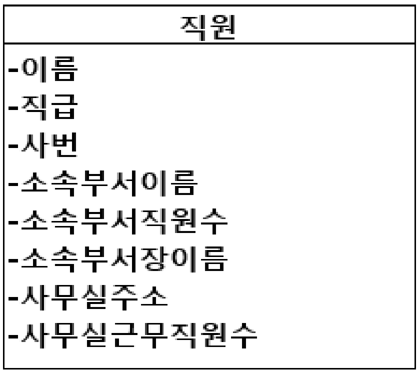


TCC = 2 / 3
LCC = 3 / 3



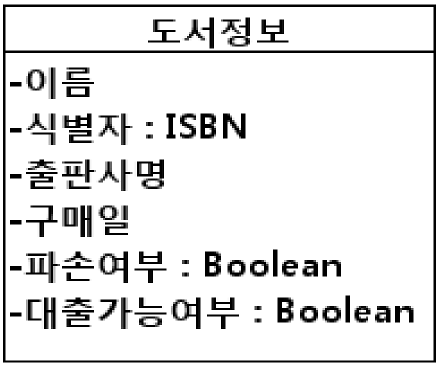
TCC = 1 / 3
LCC = 1 / 3

Another examples of less cohesive classes



직원

-이름
-직급
-사번
-소속부서이름
-소속부서직원수
-소속부서장이름
-사무실주소
-사무실근무직원수

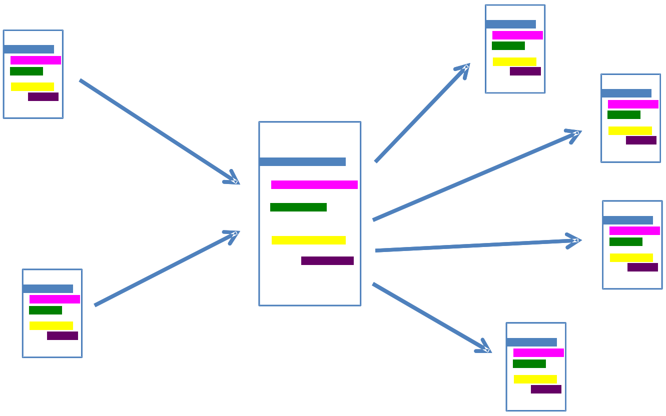


도서정보

-이름
-식별자 : ISBN
-출판사명
-구매일
-파손여부 : Boolean
-대출가능여부 : Boolean

Fan Out and Fan In

- Fan out
  - The number of called modules (via outbound calls)
- Fan in
  - The number of calling modules (via incoming calls)



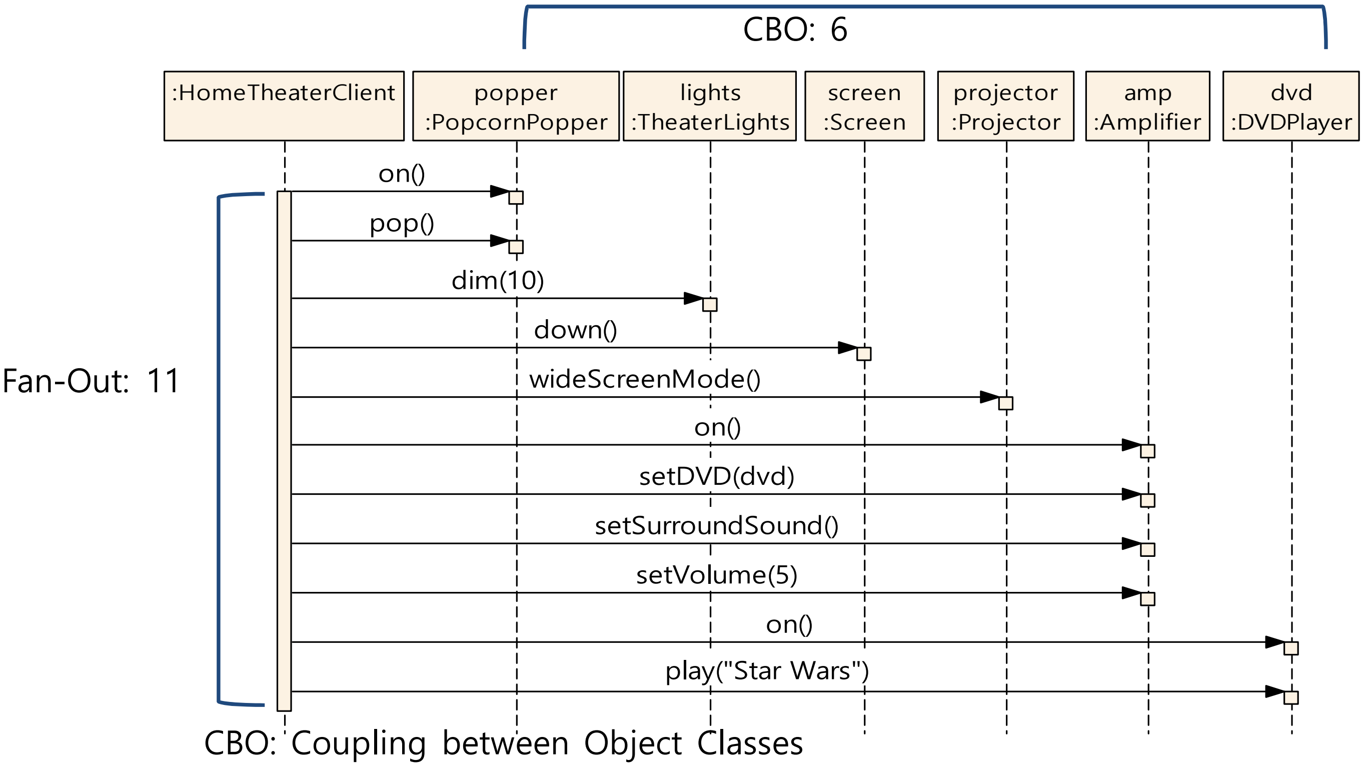
Afferent Coupling (C_a) and Efferent Coupling (C_e)

- Afferent == Incoming
- Efferent == Outgoing
- Instability of a package (I) $= \dfrac{C_e}{C_e + C_a}$

Coupling Metrics (CBO & RFC)

- CBO (Coupling Between Objects) measures coupling in terms of **classes**
  - the number of classes that a class referenced (Fan Out) +
  - the number of classes that referenced the class (Fan In)
- RFC (Response For Class) measures coupling in terms of **method calls**
  - the number of methods in the class (not including inherited methods) +
  - the number of distinct method calls made by the methods in the class

Coupling Metrics for Class



CBO: Coupling between Object Classes

Evolvability

- Code should be easily changed and extended.
- Changeability: The ease with which a design fragment can be modified (without causing ripple effects) when an existing functionality is changed.
- Extensibility: The ease with which a design fragment can be enhanced or extended (without ripple effects) for supporting new functionality

Evolvability

- getSum is both readable and understandable.

```
int getSum(const int values[], const int size) {
    int sum = 0;
    for (unsigned int i = 0; i < size; i++)
        sum += values[i];
    return sum;
}
```

- How about maintainability?
- For example, when you want to sum only positive numbers?

Poor Evolvability

- Define new but similar function by copy&pasting.

```
int getSum2(const int values[], const int size) {
    int sum = 0;
    for (unsigned int i = 0; i < size; i++)
        if ( values[i] >= 0 )
          sum += values[i];
    return sum;
}
```

Poor Evolvability

- When you want to add only odd numbers,

```
int getSum3(const int values[], const int size) {
    int sum = 0;
    for (unsigned int i = 0; i < size; i++)
        if ( (values[i] % 2) == 0 )
          sum += values[i];
    return sum;
}
```

- Codes are modified to support the changes. ➔ less maintainable!

Good Evolvability

- Identify the changed code fragments and make it varied by parameter.

```
int getSum(const int values[], const int size, bool(*include)(const int) ) {
    int sum = 0;
    for (unsigned int i = 0; i < size; i++)
        if ( include(values[i]) )
            sum += values[i];
    return sum;
}
```

Good Evolvability

- For adding positive numbers only

```
bool isPositive(const int v) {
    return v >= 0;
}

int main() {
    int values[SIZE] = { 10, 20, -10, 30, -20 };
    int sum = getSum(values, SIZE, isPositive);
    cout << sum << endl;
}
```

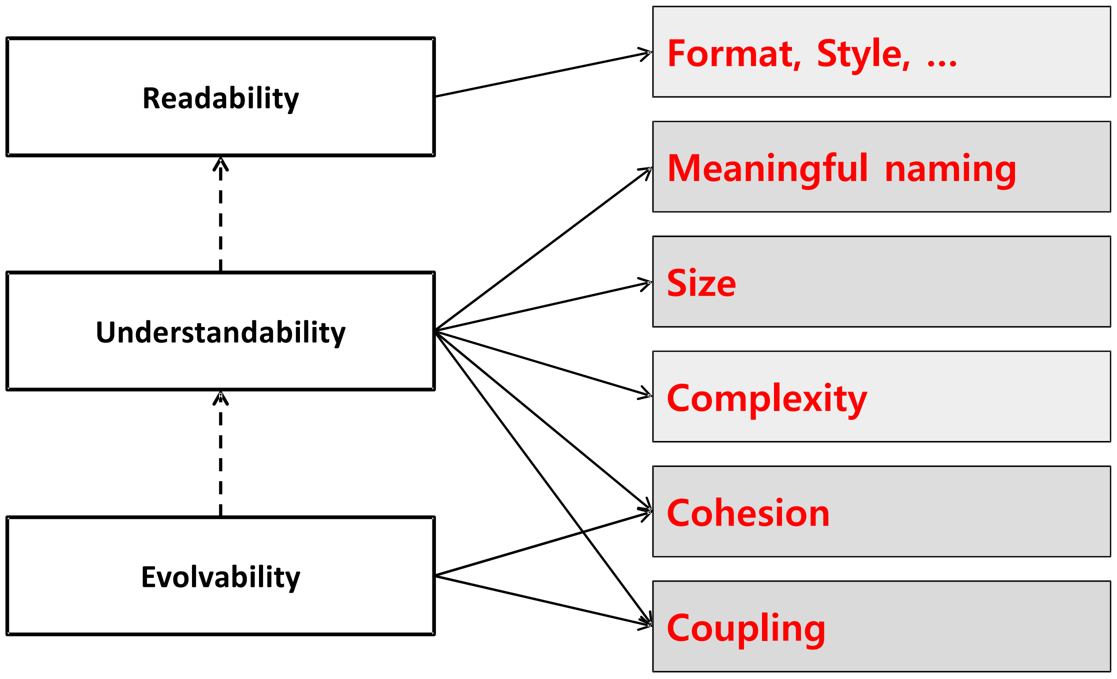
- No changes to existing getSum() !
- The existing code can be easily extended.

Good Evolvability

- For adding even numbers only

```
bool isEvenNumber(const int v) {
    return ( v % 2 ) ==  0 ;
}

int main() {
    int values[SIZE] = { 10, 20, -10, 30, -20 };
    int sum = getSum(values, SIZE, isEvenNumber);
    cout << sum << endl;
}
```

Design Quality and Principle



Metrics in Industry Standards

Type	Metric	Limit				
		Standard				
		MISRA	SCR-G	JPL	JSF	HIS
Size	Method Lines of Code(LOC)	80		60	200	50
	Comment Frequency	50%	30%	-	-	-
Complexity	Cyclomatic Complexity(CC)	15	-	-	20	10
	Number of Execution Paths(NPath)	75	-	-	-	80
	Number of Structuring Levels	6	-	-	-	4
Coupling	Number of Parameters	-	-	6	6	5
	Fan In	-	-	-	-	5
	Fan Out	-	-	-	-	7
	Number of Calling Levels	8	-	-	-	4

* MISRA: MISRA Report 5, Software Metrics
* SCR-G: 무기체계 소프트웨어 코딩규칙
* JPL: JPL(Jet Propulsion Lab.) Coding Standard for the C
* JSF: Joint Strike Fighter Air Vehicle C++ Coding Standards
* HIS: HIS(Audi, BMW 등 5개 자동차 업체 그룹) Source Code Metrics