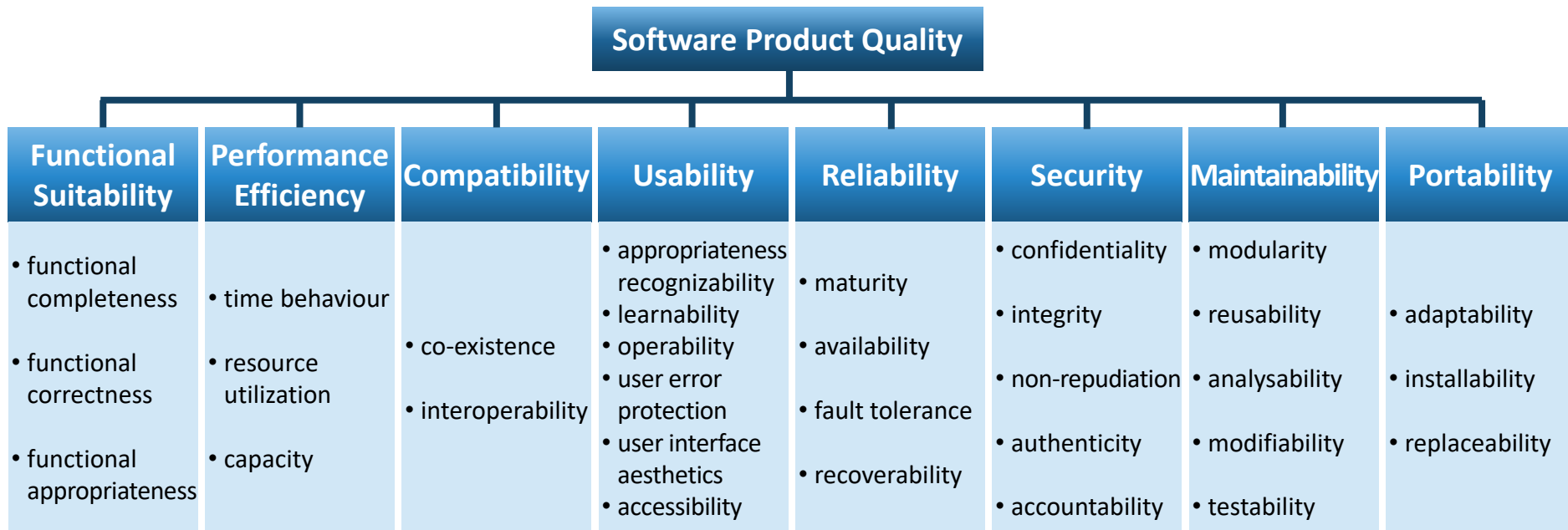


Software Quality

DESIGN MAINTAINABILITY

Software Quality

- defined by IEEE :
 - degree to which a system, component, or process
 - meets specified requirements
 - meets customer or user needs or expectations
- ISO defines software product quality model (ISO/IEC 25010)



Software Quality

- software **design goal** is to achieve **quality software**
- quality of software design
 - fit for purpose (meet all requirements)
 - easily **understandable** and highly **maintainable**
- to improve software design quality :
 - define appropriate goal clearly and concisely
 - use appropriate design principles
 - apply design pattern
 - evaluate or measure quality attributes and redesign
 - remove bad smell
 - refactoring

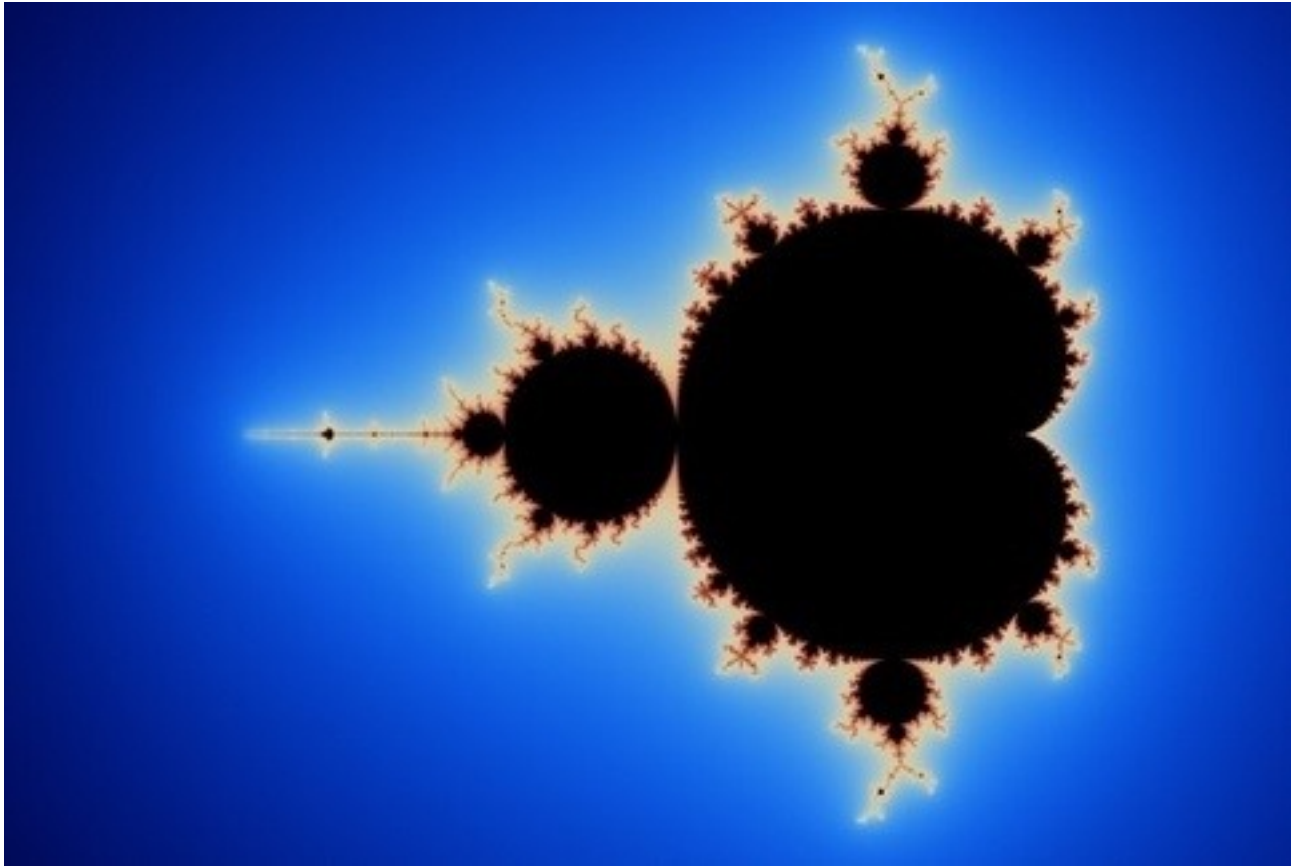
Design Maintainability

- Understandability
 - how easy is it to **understand**
- Adaptability
 - how easy is it to **change** a component
- Extensibility
 - how easy is it to **extend** a new functionality
- Cohesion
 - how closely are the parts of a component **related**
- Coupling
 - how **independent** is a component

Understandability

```
1  -                                     = (
2                                     255,
3                                     lambda
4                                     V      ,B,c
5                                     :c  and Y(V*V+B,B, c
6                                     -1)if(abs(V)<6)else
7                                     (      2+c-4*abs(V)**-0.4)/i
8                                     ) ;v,      x=1500,1000;C=range(v*x
9                                     );import struct;P=struct.pack;M,\
10      j  ='<QIIHHHH',open('M.bmp','wb').write
11  for X in j('BM'+P(M,v*x*3+26,26,12,v,x,1,24))or C:
12      i  ,Y=_;j(P('BBB',*(lambda T:(T*80+T**9
13          *i-950*T  **99,T*70-880*T**18+701*
14          T  **9      ,T*i**(1-T**45*2)))(sum(
15          [      Y(0,(A%3/3.+X%v+(X/v+
16              A/3/3.-x/2)/1j)*2.5
17              /x  -2.7,i)**2 for \
18              A      in C
19              [:9]])
20              /9)
21              )  )
```

Understandability



Understandability

```
public List<int[]> getThem() {  
    List<int[]> l = new ArrayList<int[]>();  
    // this is for loop and if statement  
    for (int[] x : l1){if(x[0] == 4)l.add(x);}return l;}
```

```
// get all the flagged cells from game board  
public List<int[]> getThem() {  
    List<int[]> l = new ArrayList<int[]>();  
    for (int[] x : l1) {  
        if(x[0] == 4) {  
            l.add(x);  
        }  
    }  
    return l;  
}
```

Understandability

```
// get all the flagged cells from game board
public List<int[]> getThem() {
    List<int[]> l = new ArrayList<int[]>();
    for (int[] x : l1) {
        if(x[0] == 4) { l.add(x); }
    }
    return l;
}
```

```
// get all the flagged cells from game board
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard) {
        if (cell.isFlagged()) {
            flaggedCells.add(cell);
        }
    }
    return flaggedCells;
}
```


HYU

```

1  let adb_command(tst argc, char *argv)
2  {
3      char *buffer;
4      int ndevnum = 0;
5      int is_server = 0;
6      int portnum = 0;
7      int i;
8      int quiet;
9      int transport_type = kTransportType;
10     char serial = NULL;
11     char *server_port_str = NULL;
12
13     /* If defined, this should be an
14      * = the directory containing all
15      * the files in this directory.
16      */
17     /* For a particular product, if
18      * = command containing this before
19      * = specify the path using "-p".
20     */
21     gReadoutPath = getenv("ANDROID_PRODUCT_PATH");
22     if (gReadoutPath == NULL) {
23         gReadoutPath = "/dev";
24     }
25     // TODO: also try TARGET_PRODUCT/TARGET
26     serial = getenv("ANDROID_SERIAL");
27
28     /* Validate and assign the server port
29     server_port_str = DEFAULT_ADB_PORT;
30     if (server_port_str == NULL) {
31         server_port = (tst) atoi(server_port_str);
32         if (server_port == 0) {
33             "adb: fix var ANDROID_SERVER_PORT\n";
34             return usage();
35         }
36     }
37     /* Modify the flags */
38     while (argc > 0) {
39         if (strcmp(argv[i], "-server") == 0) {
40             is_server = 1;
41         } else if (strcmp(argv[i], "-ndevnum") == 0) {
42             ndevnum = atoi(argv[i]);
43             /* this is a special flag used
44             * to specify the number of devices
45             */
46         } else if (strcmp(argv[i], "-port") == 0) {
47             portnum = atoi(argv[i]);
48             /* If portnum is 0, then
49             * const char *product = NULL;
50             * If (portnum == 0) {
51             * if (argc > 2) return usage;
52             * product = argv[i];
53             * } else {
54             * product = argv[i] + 1;
55             * }
56         }
57         gReadoutPath = find_product;
58         if (gReadoutPath == NULL) {
59             "adb: no product\n";
60             return usage();
61         }
62         if (argc > 10) {
63             "adb: too many args\n";
64             return usage();
65         }
66         if (argc > 1) {
67             "adb: too many args\n";
68             return usage();
69         }
70         if (argc > 1) {
71             "adb: too many args\n";
72             return usage();
73         }
74         if (argc > 1) {
75             "adb: too many args\n";
76             return usage();
77         }
78         if (argc > 1) {
79             "adb: too many args\n";
80             return usage();
81         }
82         if (argc > 1) {
83             "adb: too many args\n";
84             return usage();
85         }
86         if (argc > 1) {
87             "adb: too many args\n";
88             return usage();
89         }
90         if (argc > 1) {
91             "adb: too many args\n";
92             return usage();
93         }
94         if (argc > 1) {
95             "adb: too many args\n";
96             return usage();
97         }
98         if (argc > 1) {
99             "adb: too many args\n";
100            return usage();
101        }
102        if (argc > 1) {
103            "adb: too many args\n";
104            return usage();
105        }
106        if (argc > 1) {
107            "adb: too many args\n";
108            return usage();
109        }
110        if (argc > 1) {
111            "adb: too many args\n";
112            return usage();
113        }
114        if (argc > 1) {
115            "adb: too many args\n";
116            return usage();
117        }
118        if (argc > 1) {
119            "adb: too many args\n";
120            return usage();
121        }
122        if (argc > 1) {
123            "adb: too many args\n";
124            return usage();
125        }
126        if (argc > 1) {
127            "adb: too many args\n";
128            return usage();
129        }
130        if (argc > 1) {
131            "adb: too many args\n";
132            return usage();
133        }
134        if (argc > 1) {
135            "adb: too many args\n";
136            return usage();
137        }
138        if (argc > 1) {
139            "adb: too many args\n";
140            return usage();
141        }
142        if (argc > 1) {
143            "adb: too many args\n";
144            return usage();
145        }
146        if (argc > 1) {
147            "adb: too many args\n";
148            return usage();
149        }
150        if (argc > 1) {
151            "adb: too many args\n";
152            return usage();
153        }
154        if (argc > 1) {
155            "adb: too many args\n";
156            return usage();
157        }
158        if (argc > 1) {
159            "adb: too many args\n";
160            return usage();
161        }
162        if (argc > 1) {
163            "adb: too many args\n";
164            return usage();
165        }
166        if (argc > 1) {
167            "adb: too many args\n";
168            return usage();
169        }
170        if (argc > 1) {
171            "adb: too many args\n";
172            return usage();
173        }
174        if (argc > 1) {
175            "adb: too many args\n";
176            return usage();
177        }
178        if (argc > 1) {
179            "adb: too many args\n";
180            return usage();
181        }
182        if (argc > 1) {
183            "adb: too many args\n";
184            return usage();
185        }
186        if (argc > 1) {
187            "adb: too many args\n";
188            return usage();
189        }
190        if (argc > 1) {
191            "adb: too many args\n";
192            return usage();
193        }
194        if (argc > 1) {
195            "adb: too many args\n";
196            return usage();
197        }
198        if (argc > 1) {
199            "adb: too many args\n";
200            return usage();
201        }
202        if (argc > 1) {
203            "adb: too many args\n";
204            return usage();
205        }
206        if (argc > 1) {
207            "adb: too many args\n";
208            return usage();
209        }
210        if (argc > 1) {
211            "adb: too many args\n";
212            return usage();
213        }
214        if (argc > 1) {
215            "adb: too many args\n";
216            return usage();
217        }
218        if (argc > 1) {
219            "adb: too many args\n";
220            return usage();
221        }
222        if (argc > 1) {
223            "adb: too many args\n";
224            return usage();
225        }
226        if (argc > 1) {
227            "adb: too many args\n";
228            return usage();
229        }
230        if (argc > 1) {
231            "adb: too many args\n";
232            return usage();
233        }
234        if (argc > 1) {
235            "adb: too many args\n";
236            return usage();
237        }
238        if (argc > 1) {
239            "adb: too many args\n";
240            return usage();
241        }
242        if (argc > 1) {
243            "adb: too many args\n";
244            return usage();
245        }
246        if (argc > 1) {
247            "adb: too many args\n";
248            return usage();
249        }
250        if (argc > 1) {
251            "adb: too many args\n";
252            return usage();
253        }
254        if (argc > 1) {
255            "adb: too many args\n";
256            return usage();
257        }
258        if (argc > 1) {
259            "adb: too many args\n";
260            return usage();
261        }
262        if (argc > 1) {
263            "adb: too many args\n";
264            return usage();
265        }
266        if (argc > 1) {
267            "adb: too many args\n";
268            return usage();
269        }
270        if (argc > 1) {
271            "adb: too many args\n";
272            return usage();
273        }
274        if (argc > 1) {
275            "adb: too many args\n";
276            return usage();
277        }
278        if (argc > 1) {
279            "adb: too many args\n";
280            return usage();
281        }
282        if (argc > 1) {
283            "adb: too many args\n";
284            return usage();
285        }
286        if (argc > 1) {
287            "adb: too many args\n";
288            return usage();
289        }
290        if (argc > 1) {
291            "adb: too many args\n";
292            return usage();
293        }
294        if (argc > 1) {
295            "adb: too many args\n";
296            return usage();
297        }
298        if (argc > 1) {
299            "adb: too many args\n";
300            return usage();
301        }
302        if (argc > 1) {
303            "adb: too many args\n";
304            return usage();
305        }
306        if (argc > 1) {
307            "adb: too many args\n";
308            return usage();
309        }
310        if (argc > 1) {
311            "adb: too many args\n";
312            return usage();
313        }
314        if (argc > 1) {
315            "adb: too many args\n";
316            return usage();
317        }
318        if (argc > 1) {
319            "adb: too many args\n";
320            return usage();
321        }
322        if (argc > 1) {
323            "adb: too many args\n";
324            return usage();
325        }
326        if (argc > 1) {
327            "adb: too many args\n";
328            return usage();
329        }
330        if (argc > 1) {
331            "adb: too many args\n";
332            return usage();
333        }
334        if (argc > 1) {
335            "adb: too many args\n";
336            return usage();
337        }
338        if (argc > 1) {
339            "adb: too many args\n";
340            return usage();
341        }
342        if (argc > 1) {
343            "adb: too many args\n";
344            return usage();
345        }
346        if (argc > 1) {
347            "adb: too many args\n";
348            return usage();
349        }
350        if (argc > 1) {
351            "adb: too many args\n";
352            return usage();
353        }
354        if (argc > 1) {
355            "adb: too many args\n";
356            return usage();
357        }
358        if (argc > 1) {
359            "adb: too many args\n";
360            return usage();
361        }
362        if (argc > 1) {
363            "adb: too many args\n";
364            return usage();
365        }
366        if (argc > 1) {
367            "adb: too many args\n";
368            return usage();
369        }
370        if (argc > 1) {
371            "adb: too many args\n";
372            return usage();
373        }
374        if (argc > 1) {
375            "adb: too many args\n";
376            return usage();
377        }
378        if (argc > 1) {
379            "adb: too many args\n";
380            return usage();
381        }
382        if (argc > 1) {
383            "adb: too many args\n";
384            return usage();
385        }
386        if (argc > 1) {
387            "adb: too many args\n";
388            return usage();
389        }
390        if (argc > 1) {
391            "adb: too many args\n";
392            return usage();
393        }
39
```

Adaptability

```
// printing array values
public void printValues(int numbers[]) {
    for (int num : numbers)
        System.out.println(num);
}
```

```
// printing array values
public void printPositiveValue(int numbers[]) {
    for (int num : numbers)
        if (num >= 0)
            System.out.println(num);
}
```

```
// printing array values
public void printNegativeValue(int numbers[]) {
    for (int num : numbers)
        if (num < 0)
            System.out.println(num);
}
```

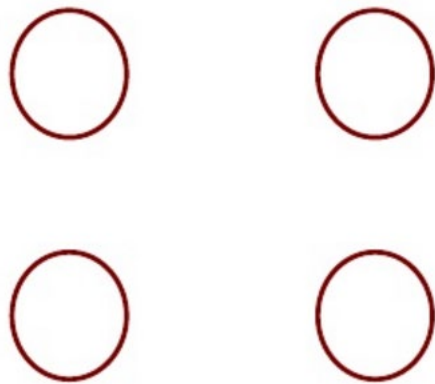
Extensibility

```
public interface Check {  
    public boolean checkCond(int num);  
}  
  
public class isPositive implements Check {  
    public boolean checkCond(int num){  
        return num >= 0;  
    }  
}  
  
public class isEven implements Check {  
    public boolean checkCond(int num){  
        return (num % 2) == 0;  
    }  
}
```

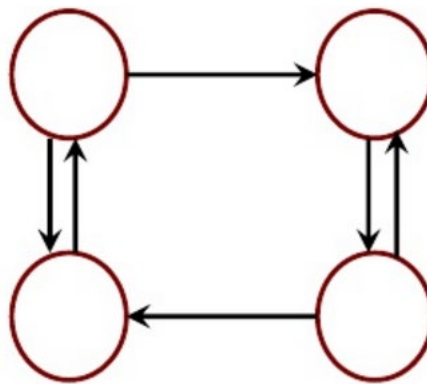
```
public void printValue(int numbers[], Check check) {  
    for (int num : numbers)  
        if (check.checkCond(num))  
            System.out.println(num);  
}
```

Coupling

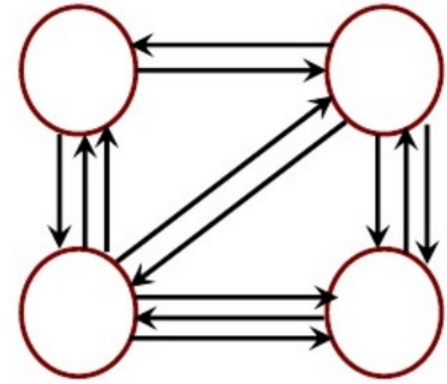
- degree of **interdependency** between software modules



uncoupled
no dependency



loosely coupled
some dependencies



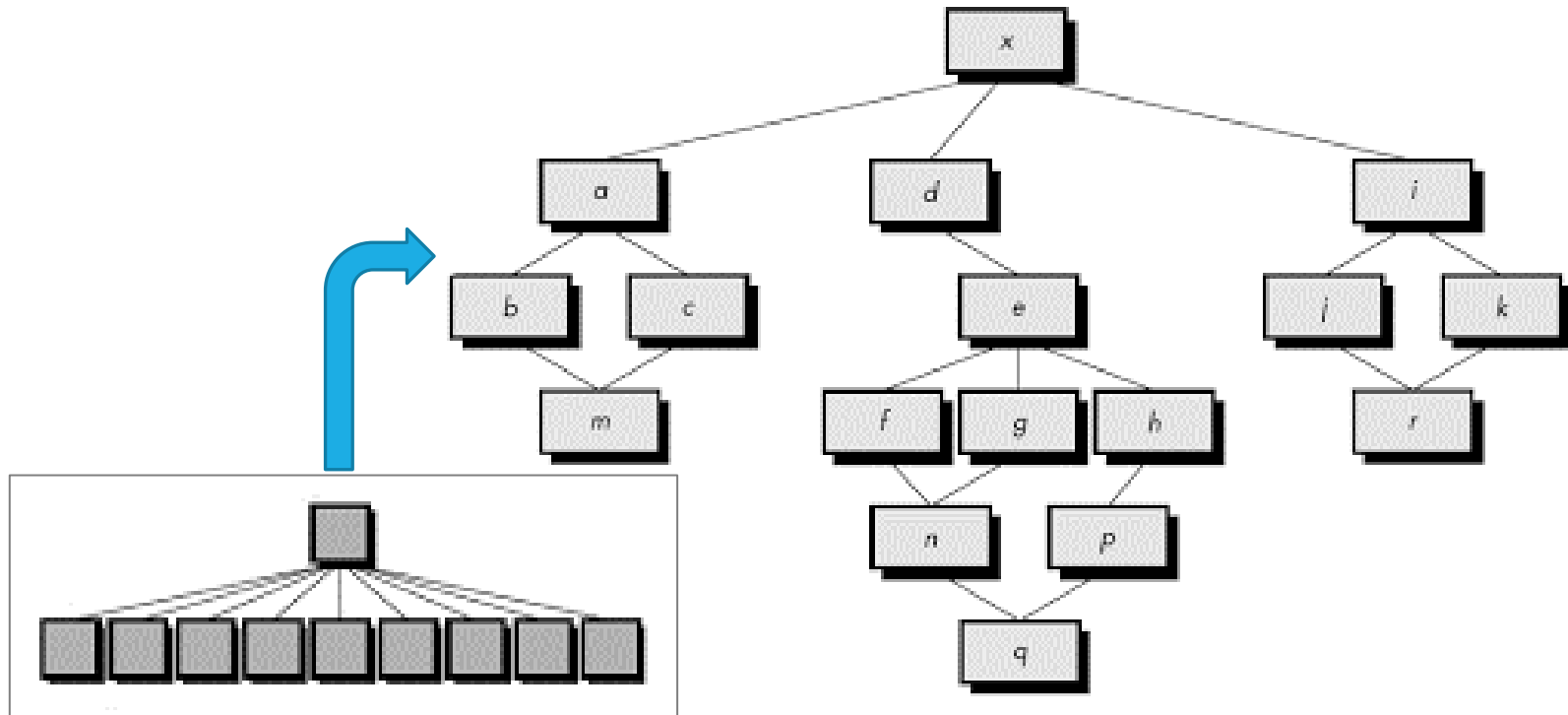
highly coupled
many dependencies

Coupling

- type of coupling
 - no coupling
 - data coupling
 - stamp coupling
 - control coupling
 - external coupling
 - common coupling
 - content coupling

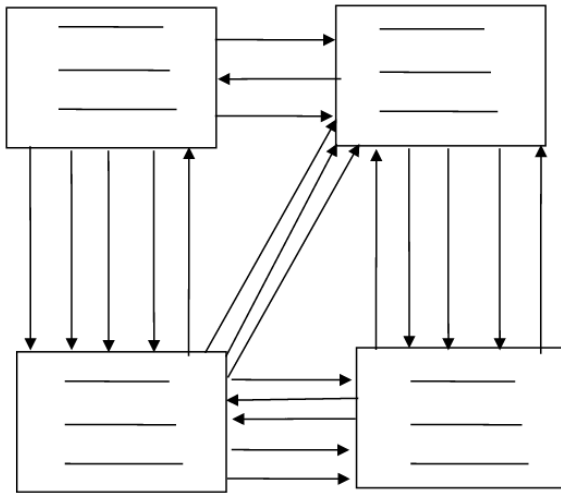
Coupling

- fan-in : number of calling modules
- fan-out : number of called modules
 - high fan-in / out suggests high coupling

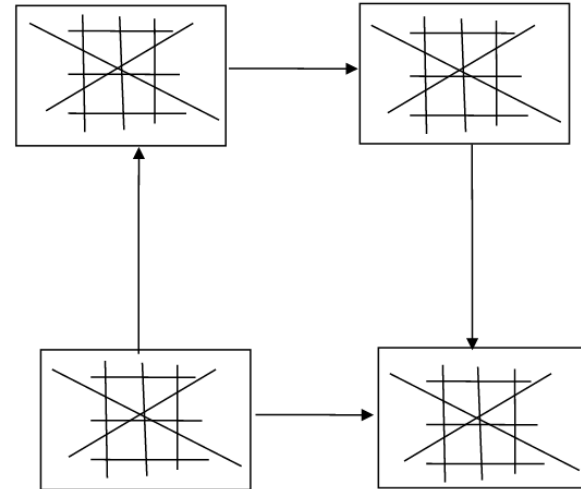


Cohesion

- degree to which the elements of a module are **functionally related** within a module
- cohesion & coupling



high coupling
low cohesion



low coupling
high cohesion

Cohesion

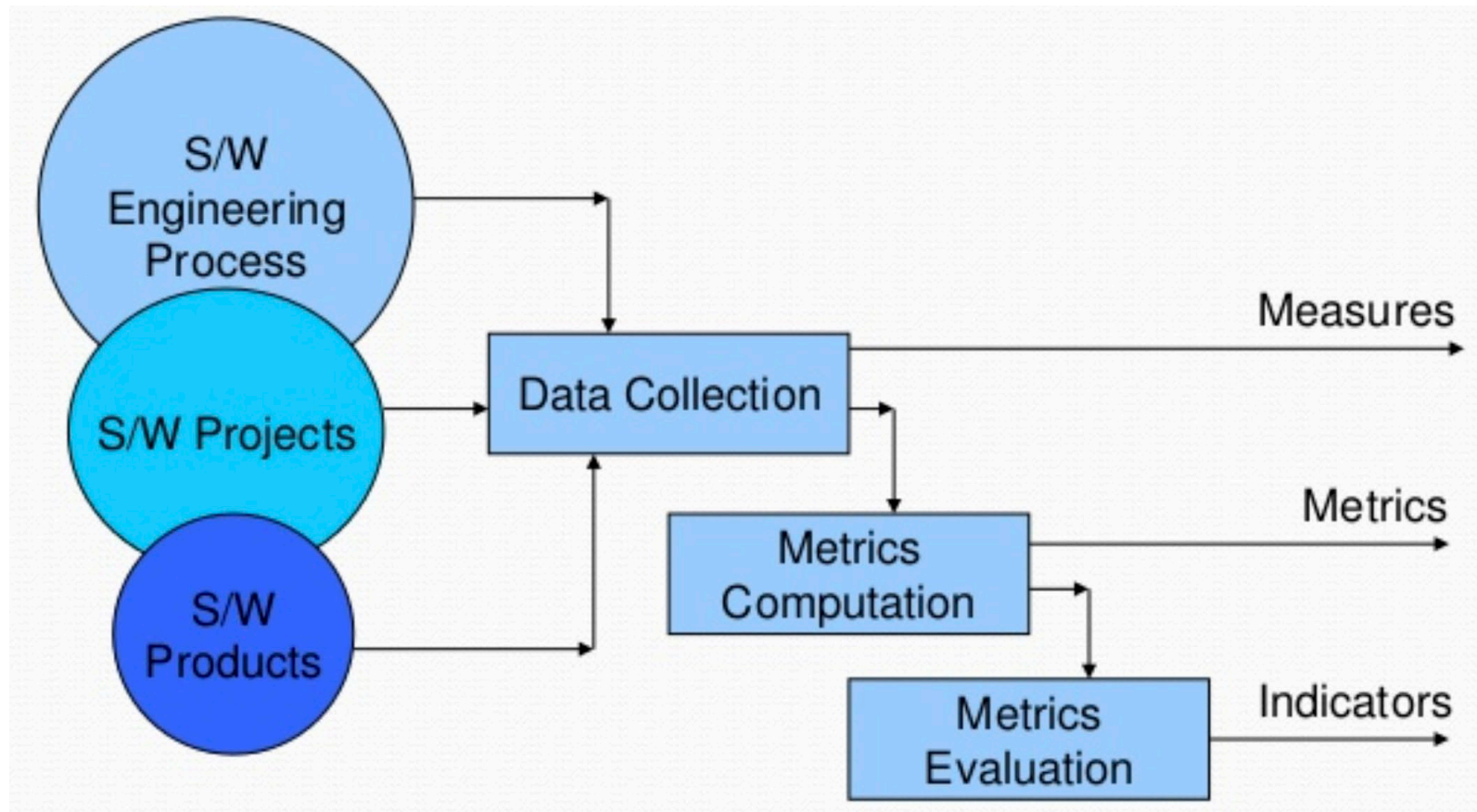
- type of cohesion
 - functional cohesion
 - sequential cohesion
 - communicational cohesion
 - procedural cohesion
 - temporal cohesion
 - logical cohesion
 - coincidental cohesion

Software Metrics

Measure, Metrics, Indicators

- collects **measures** and develops **metrics** to obtain **indicators**
 - a **measures** provides a **quantitative** indication of the extent, amount, dimension, capacity, or size of some attributes of a product or process
 - IEEE defines a **metrics** as “a quantitative measure of the degree to which a system, component, or process possesses a given **attribute**.”
 - IEEE Standard Glossary of SE Terminology
 - **indicator** is a metric or combination of metrics that provide **insight** into the software process, project, or the product itself

Measure, Metrics, Indicators



Cyclomatic Complexity (CC)

- measures complexity of a module / program
 - = *number of decisions + 1*

```
public void foo(){
    boolean jar = false;
    int tree = 9;
    if(tree == 9){
        ;
    }
    else if(jar == true){
        ;
    }
    if(tree != 9){
        ;
    }
}
```

CC =

```
public void foo2(int x){
    if(x > 10){
        ;
    }
    else{
        for(int i = 0; i < x; i++){
            ;
        }
    }
}
```

CC =

Cyclomatic Complexity (CC)

Construct	Decision	Reasoning
If..Then	+1	An If statement is a single decision.
Elseif..Then	+1	Elseif adds a new decision.
Else	0	Else does not cause a new decision. The decision is at the If.
#If..#Elseif..#Else	0	Conditional compilation adds no run-time decisions.
Select Case	0	Select Case initiates the following Case branches, but does not add a decision alone.
Case	+1	Each Case branch adds a new decision.
Case Else	0	Case Else does not cause a new decision. (decisions were made at other Cases)
For [Each] .. Next	+1	There is a decision at the For statement.
Do While Until	+1	There is a decision at the start of the Do..Loop.
Loop While Until	+1	There is a decision at the end of the Do..Loop.
Do..Loop alone	0	There is no decision in an unconditional Do..Loop without While or Until.
While	+1	There is a decision at the start of the While..Wend or While..End While loop.
Catch	+1	Each Catch branch adds a new conditional path of execution. Even though a Catch can be either conditional or unconditional, we treat all of them the same way.
Catch..When	+2	The When condition adds a second decision.

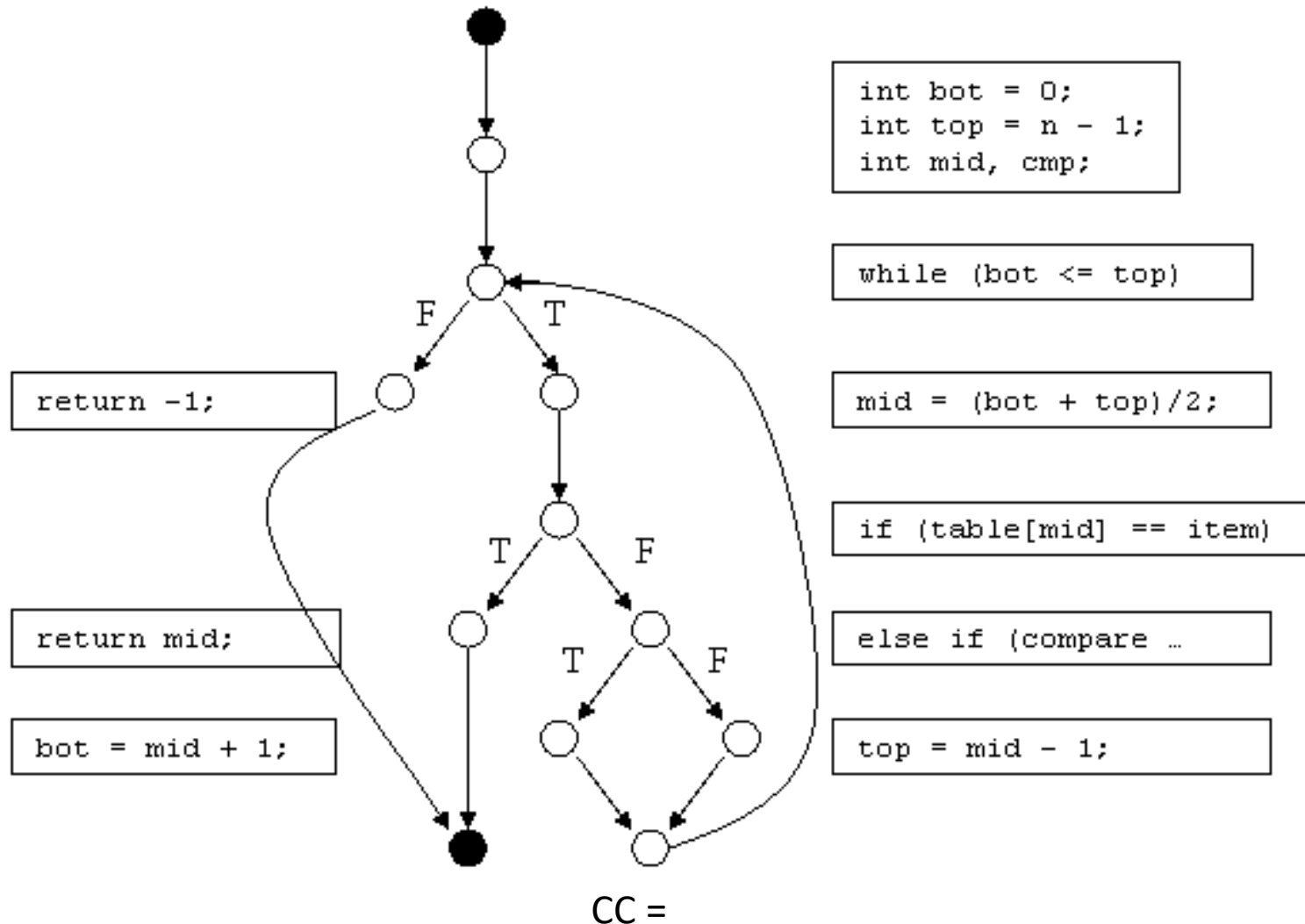
<http://www.aivosto.com/project/help/pm-complexity.html>

Cyclomatic Complexity (CC)

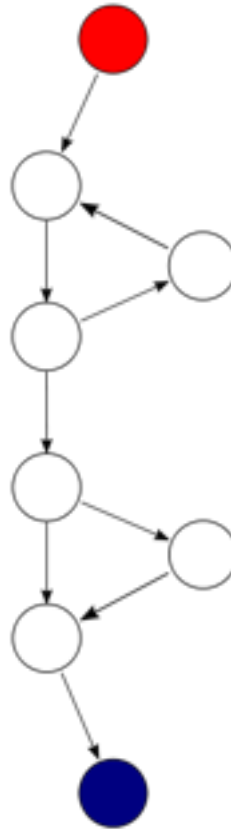
```
int BinSearch (char *item, char *table[], int n) {
    int bot = 0;
    int top = n - 1;
    int mid, cmp;
    while (bot <= top) {
        mid = (bot + top) / 2;
        if (table[mid] == item)
            return mid;
        else if (compare(table[mid], item) < 0)
            top = mid - 1;
        else
            bot = mid + 1;
    }
    return -1; // not found
}
```

CC =

Cyclomatic Complexity (CC)



Cyclomatic Complexity (CC)



CC =

Cyclomatic Complexity (CC)

- Risk detection

CC	Type of procedure	Risk
1 -4	a simple procedure	low
5-10	a well structured and stable procedure	low
11-20	a more complex procedure procedure	moderate
21-50	a complex procedure, alarming	high
>50	An error-prone, extremely troublesome, untestable procedure	very high

Nesting Depth

- number of structuring levels

```
public void foo(int x){  
    if(x > 10){  
        ;  
    }  
}
```

Nesting Depth = 1

```
public void foo2(int x){  
    int y = 0;  
    while(x > 10){  
        if(x > 20){  
            //... y = ~  
        }  
        else{  
            //... y = ~  
        }  
    }  
    if(y > 10) return 1;  
    else return 0;  
}
```

Nesting Depth = 2

Npath

- number of (static) execution path

```
public void foo(int x){  
    if(x > 10){  
        ;  
    }  
}
```

NPath = 2

```
public void foo2(int x){  
    int y = 0;  
    while(x > 10){  
        if(x > 20){  
            //... y = ~  
        }  
        else{  
            //... y = ~  
        }  
    }  
    if(y > 10) return 1;  
    else return 0;  
}
```

NPath = 6

Chidamber & Kemerer Metrics

- Object-Oriented (OO) metrics designed to
 - measure **aspects** of the OO approach
 - measure **complexity** of the design
 - improve the **development** of software
 - identify outlying values; a signal of high complexity / design violations
 - taking managerial decisions
 - re-designing
 - assigning extra or higher skilled resources to develop, test, maintain

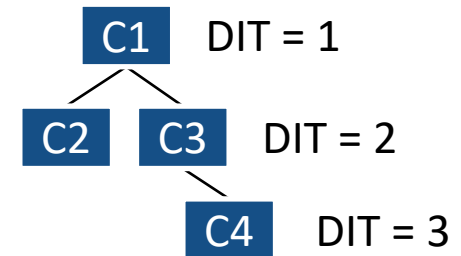
Chidamber & Kemerer Metrics

- **Weighted Methods per Class (WMC)**

- **total complexity** of methods in a class
 - can predict **required time/effort** to develop or maintain class

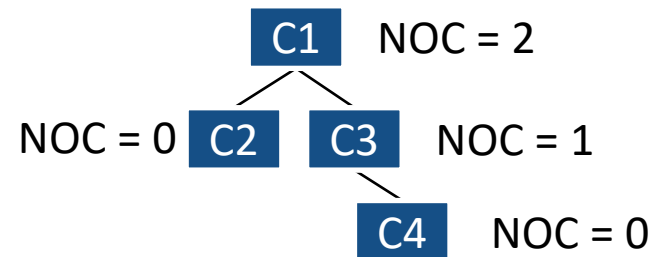
- **Depth of Inheritance Tree (DIT)**

- **depth** of **class** in inheritance tree



- **Number of Children (NOC)**

- **number** of immediate **subclasses**



Chidamber & Kemerer Metrics

- coupling metrics

- **Response For Class (RFC)**

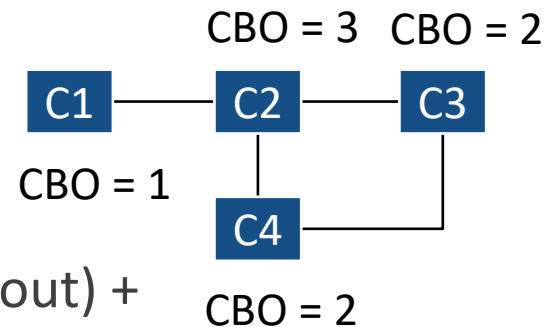
- measures coupling in terms of **method calls**

= no. of methods in class (without inherited methods) +
no. of distinct method calls made by the methods in the class

- **Coupling Between Objects (CBO)**

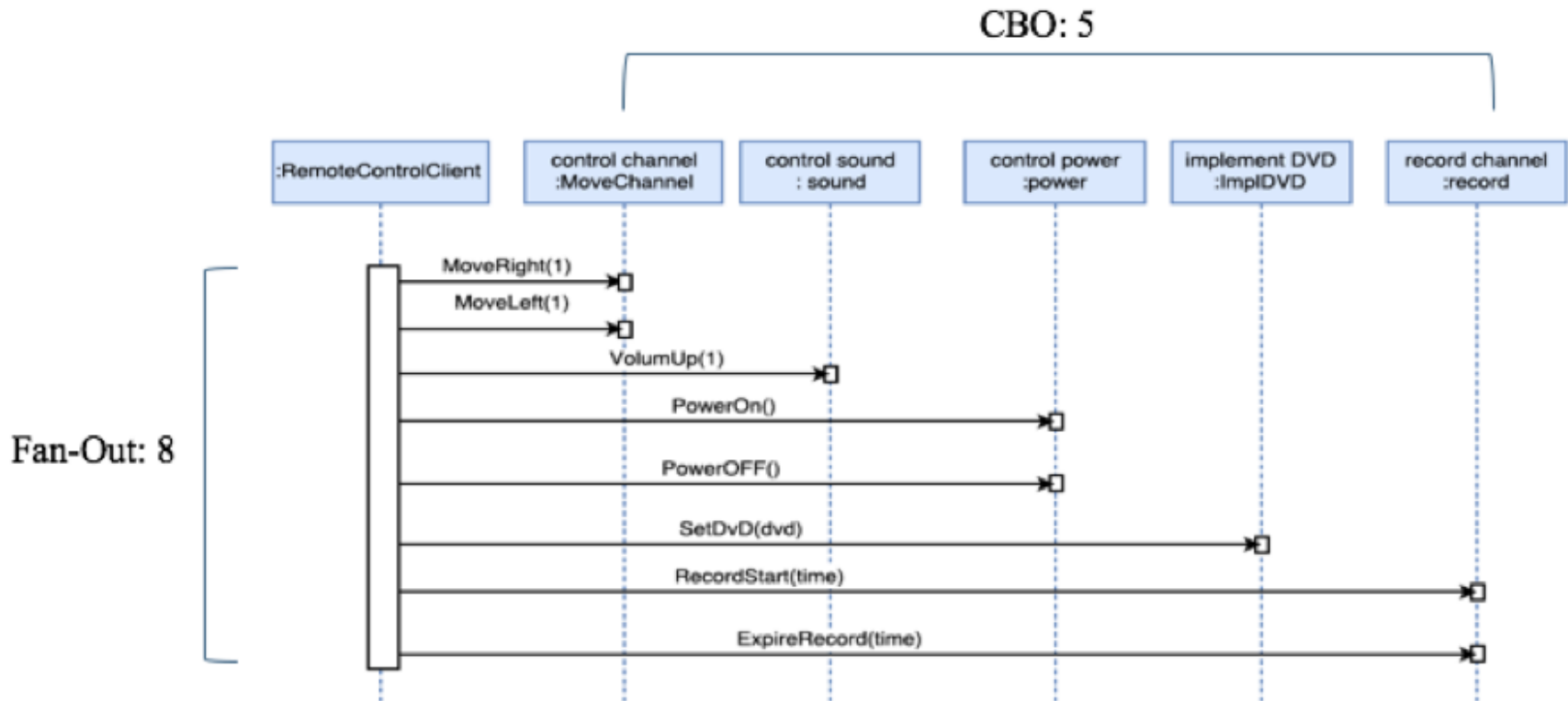
- measures coupling in terms of **classes**

= no. of classes that a class references (fan-out) +
no. of classes that references the class (fan-in)



Chidamber & Kemerer Metrics

Coupling between Object (CBO)



CBO: Coupling Between Object Classes

Chidamber & Kemerer Metrics

- **Lack of Cohesion Method (LCOM)**

- measures **dissimilarity of methods** in a class using instance variable or attribute

= count of no. of method pairs whose similarity is zero

given n methods M_1, M_2, \dots, M_n contained in a class C_1 which also contains a set of instance variables $\{I_i\}$

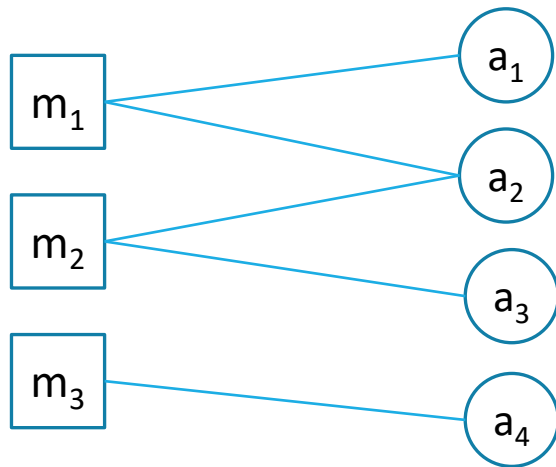
then for any method M_i we can define the partitioned set of

$$P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset\} \quad \& \quad Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset\}$$

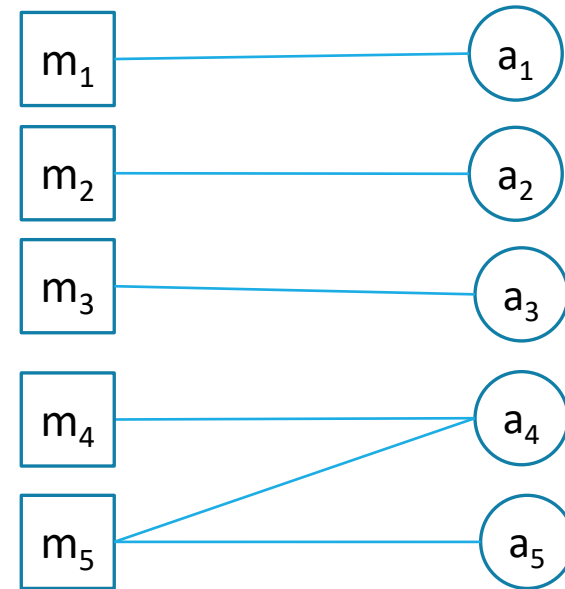
$$\begin{aligned} \text{then } \text{LCOM} &= |P| - |Q|, & \text{if } |P| > |Q| \\ &= 0 \quad \text{otherwise} \end{aligned}$$

Chidamber & Kemerer Metrics

Lack of Cohesion Method (LCOM)



$$\text{LCOM} = 2 - 1$$



$$\text{LCOM} = 9 - 1$$

Chidamber & Kemerer Metrics

- **Lack of Cohesion of Methods (LCOM)**

- another definition : LCOM2
- Henderson-Seller version : LCOM-HS

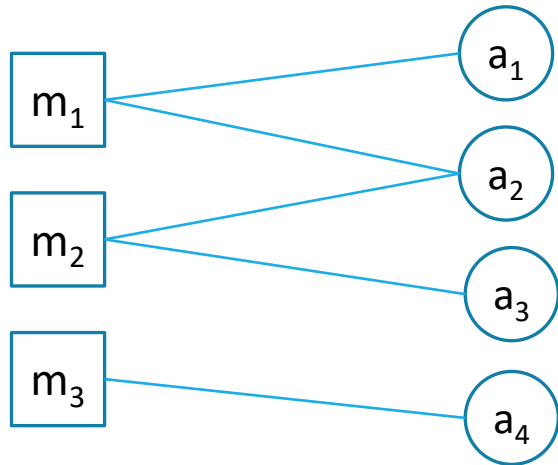
- ***M*** : number of methods in class
- ***A*** : number of variables in class
- ***MA*** : number of methods that access a particular variable
- ***Sum(MA)*** : sum of ***MA*** over all attributes of class

$$\text{then } \text{LCOM2} = 1 - (\text{Sum}(\text{MA}) / (M \times A))$$

$$\text{LCOM-HS} = (M - \text{Sum}(\text{MA}) / A) / (M - 1)$$

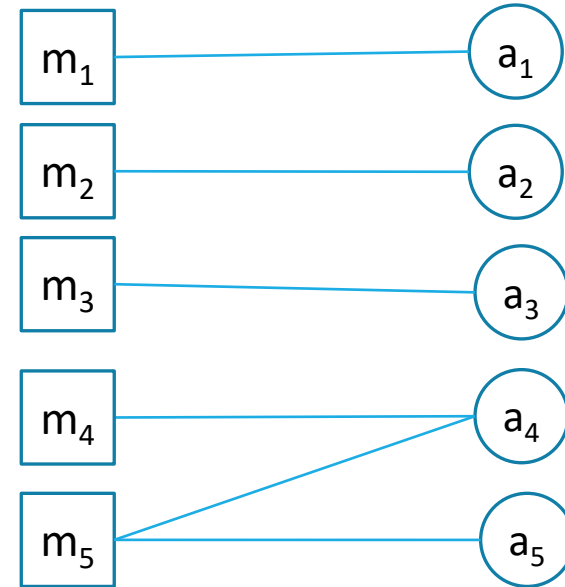
Chidamber & Kemerer Metrics

Lack of Cohesion Method (LCOM2 & LCOM-HS)



$$\text{LCOM2} = 1 - (5 / 12) = 0.583$$

$$\text{LCOM-HS} = (3 - 5 / 4) / 2 = 0.875$$

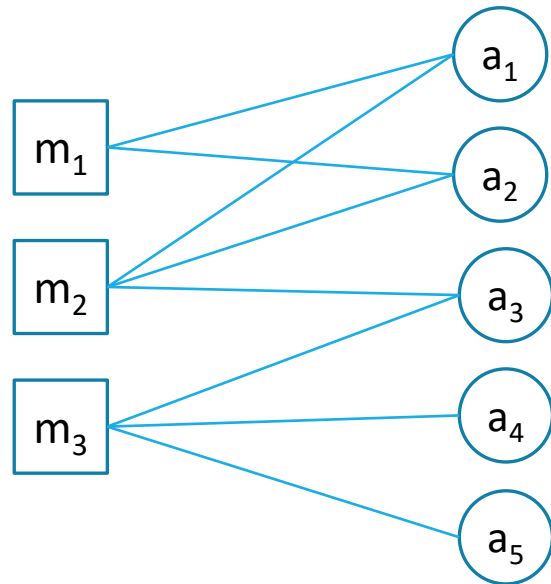


$$\text{LCOM2} = 1 - (6 / 25) = 0.76$$

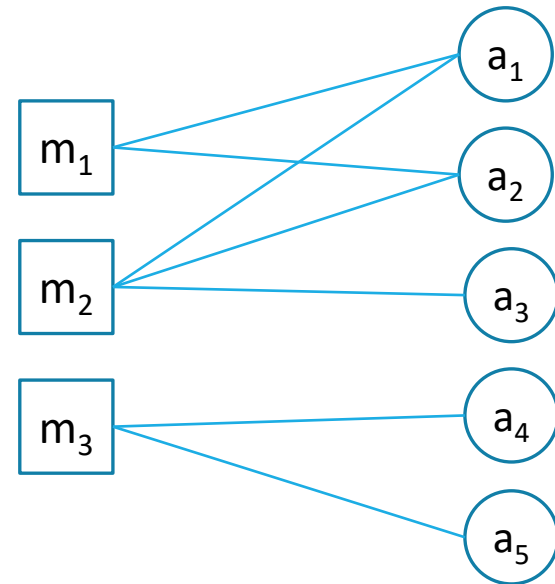
$$\text{LCOM-HS} = (5 - 6 / 5) / 4 = 0.95$$

Chidamber & Kemerer Metrics

Lack of Cohesion Method



LCOM =
LCOM2 =
LCOM-HS =



LCOM =
LCOM2 =
LCOM-HS =

Chidamber & Kemerer Metrics

- summary

Metric	Goal	Level	Complexity (To develop, to test and to maintain)	Re- usability	Encapsulation, Modularity
WMC	Low	▼	▼	▲	
DIT	Trade-off	▼	▼	▼	
		▲	▲	▲	
NOC	Trade-off	▼	▼	▼	
		▲	▲	▲	
CBO	Low	▼	▼		▲
RFC	Low	▼	▼		
LCOM	Low	▼	▼		▲

Robert C. Martin Metrics

- group of OO metrics
 - proposed by “Uncle Bob” in 1994
 - do not represent attributes to access individual OO design
 - focus on **relationship between packages**
 - incoming and outgoing dependencies to determine **instability** of package
 - Efferent **C**oupling (Ce)
 - Afferent **C**oupling (Ca)
 - **I**nstability (I)
 - **A**bstractness (A)
 - Normalized **D**istance from Main Sequence (D)

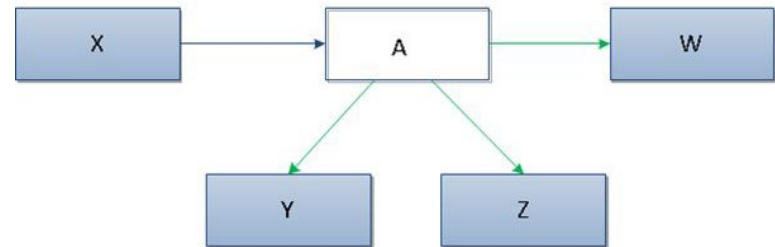
Robert C. Martin Metrics

- **Efferent Coupling (Ce)**

- number of classes in a given package which depend on classes in other packages (number of **outgoing dependencies**)



Ce = 2



Ce = 3

- high Ce (> 20) indicates instability of a package
 - changes in numerous external classes can cause need for changes to the package
 - preferred Ce value : 0 ~ 20

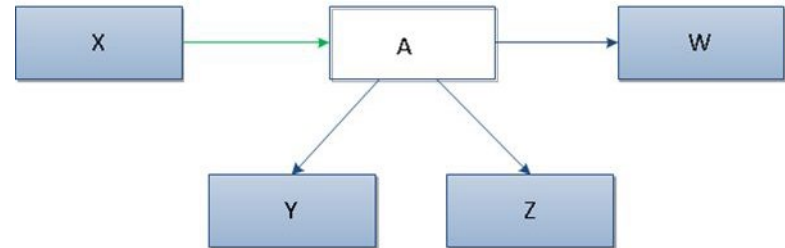
Robert C. Martin Metrics

- **Afferent Coupling (Ca)**

- number of classes in other packages which depend on classes in a given package (number of **incoming dependencies**)



Ca = 3



Ca = 1

- high **Ca** usually suggest high component stability
 - since many other classes depends on the class, it cannot be modified significantly as probability of spreading such changes increase
 - preferred **Ca** value : 0 ~ 500

Robert C. Martin Metrics

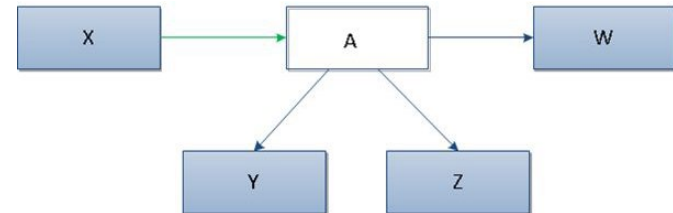
- **Instability (I)**

- measure relative susceptibility of class to changes - potential stability

$$I = \frac{Ce}{Ce + Ca}$$



$$I = 2 / (2 + 3) = 0.4$$



$$I = 3 / (3 + 1) = 0.75$$

- I value ranges between 0 (maximal stability) ~ 1 (maximal instability)
 - close to 1 : unstable due to possibility of easy changes to the package
 - close to 0 : more difficult to modifying due to their greater responsibility
- preferred I value : 0 ~ 0.3 or 0.7 ~ 1

Robert C. Martin Metrics

- **Abstractness (A)**

- measure degree of **abstraction** of package

$$A = \frac{\textit{AbstractClasses}}{\textit{AbstractClasses} + \textit{ConcreteClasses}}$$

AbstractClasses : number of abstract classes in a package

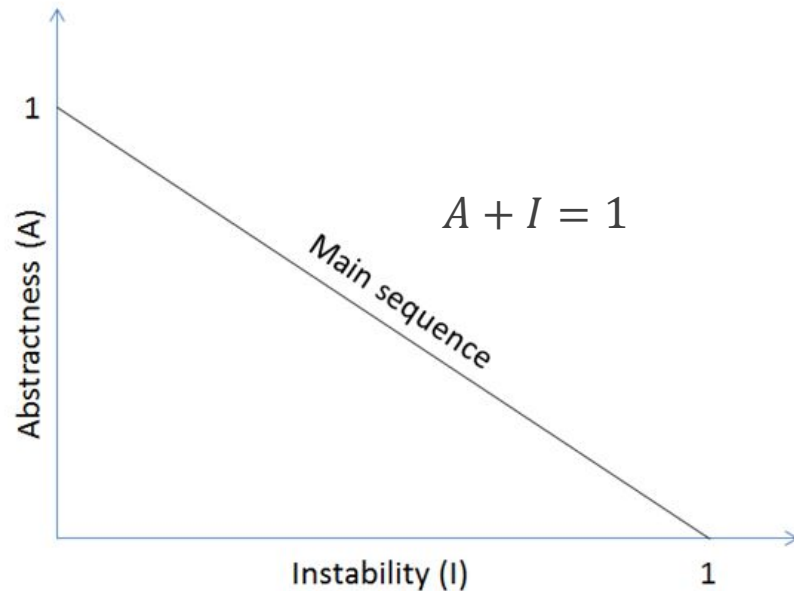
ConcreteClasses : number of concrete classes in a package

- preferred **A** value : extreme values close to 0 or close to 1
 - stable package (**I** close to 0), which are dependent on other packages, should also be abstract (**A** close to 1)
 - unstable package (**I** close to 1) should consists of concrete classes (**A** close to 0)

Robert C. Martin Metrics

- **Main Sequence**

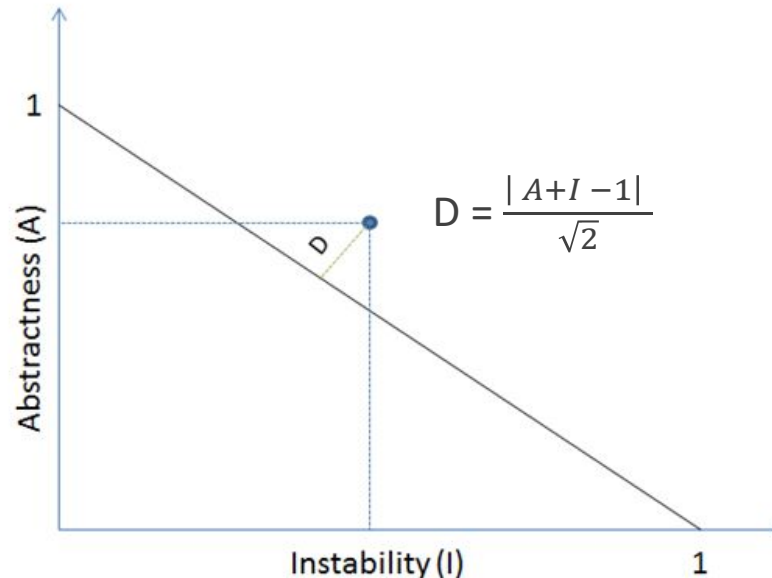
- combining abstractness and stability
 - instability of the class is compensated by its abstractness



- if C_a is small, I will be close to 1 : A should be close to 0
- if C_a is large, I will be close to 0 : A should be close to 1

Robert C. Martin Metrics

- **Normalized Distance from Main Sequence (D)**



- D should be as low as possible to be close to the main sequence
 - $A = 0$ and $I = 0$, package is extremely stable and concrete but undesirable as package is very rigid and cannot be extended
 - $A = 1$ and $I = 1$, impossible completely abstract package must have connection to outside for instance could be created

Design Principles

Key Design Principles

- **S**RP Single Responsibility Principle
- **O**CP Open / Closed Principle
- **L**SP Liskov Substitution Principle
- **I**SP Interface Segregation Principle
- **D**IP Dependency Inversion Principle

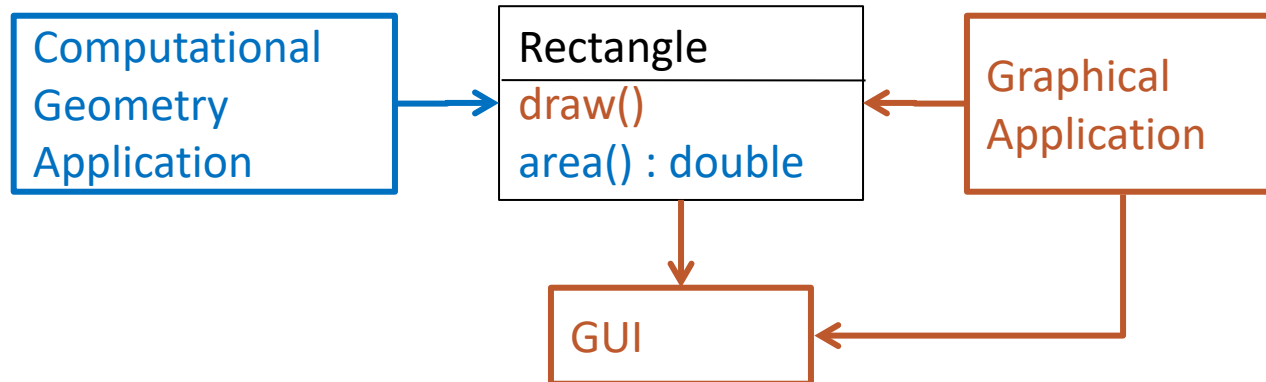
- DRY Don't Repeat Yourself Principle
- SCP Speaking Code Principle

Single Responsibility Principle

- “A class should have one, and only **one reason to change**”
 - Robert C. Martin, 2003
- every object should have a single **responsibility** which should be entirely
 - Wikipedia
- responsibility :
 - a reason to change
 - maps to requirements
 - more requirements leads to more possible change
 - more responsibilities leads to more changes in the code
 - multiple responsibilities in one class causes a possible high coupling
 - higher the coupling, higher the possibility of error on change

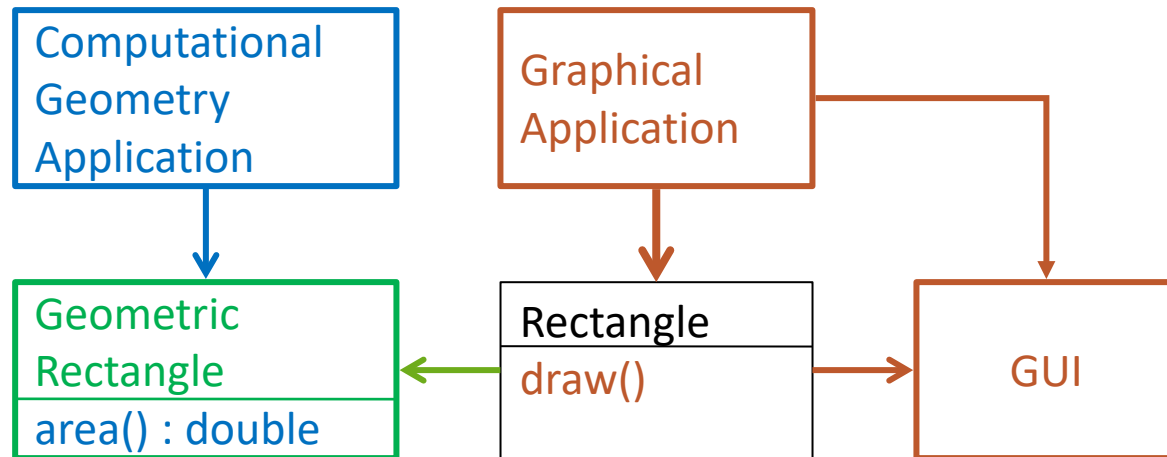
Single Responsibility Principle

- violation of the SRP causes :
 - rigidity of code
 - fragility of code



Single Responsibility Principle

- violation of the SRP causes :
 - rigidity of code
 - fragility of code



Single Responsibility Principle

- multiple responsibilities in a single class can be removed using :
 - multiple small interfaces (ISP)
 - many small classes
 - distinct responsibilities
- to yields :
 - flexible design
 - lower coupling
 - higher cohesion

Single Responsibility Principle

- SRP related bad smells
 - long methods
 - method containing too many lines of code
 - divergent changes
 - making change to a class leads to changing many unrelated methods
- coupling related bad smells
 - message chains
 - existence of series of calls

Open / Closed Principle

- “All systems change during their life cycles. This must be borne in mind when developing systems expected to last longer than the first version”

– Ivar Jacobson

Open / Closed Principle

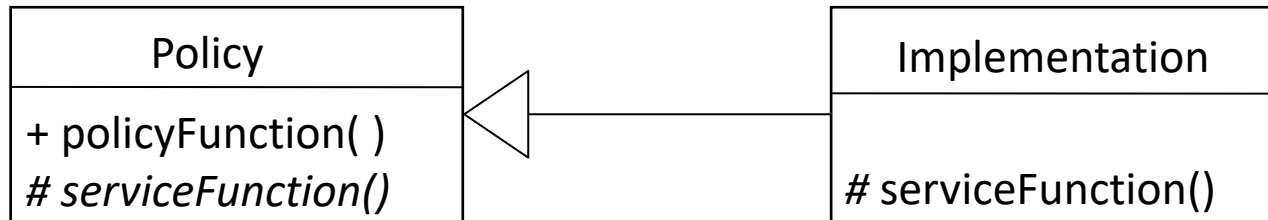
- “**extend** the **behavior** of a system **without** having to **modify** the system”
– Bertrand Meyer, 1988
- “software entities (classes, modules, functions, etc) should be **opened for extension**, but **closed for modification**”
– Wikipedia
 - changing behavior without modifying the code
 - rely on abstractions not on implementations
 - do not limit the variety of implementation
 - by inheritance or abstract classes or parameterization

Open / Close Principle

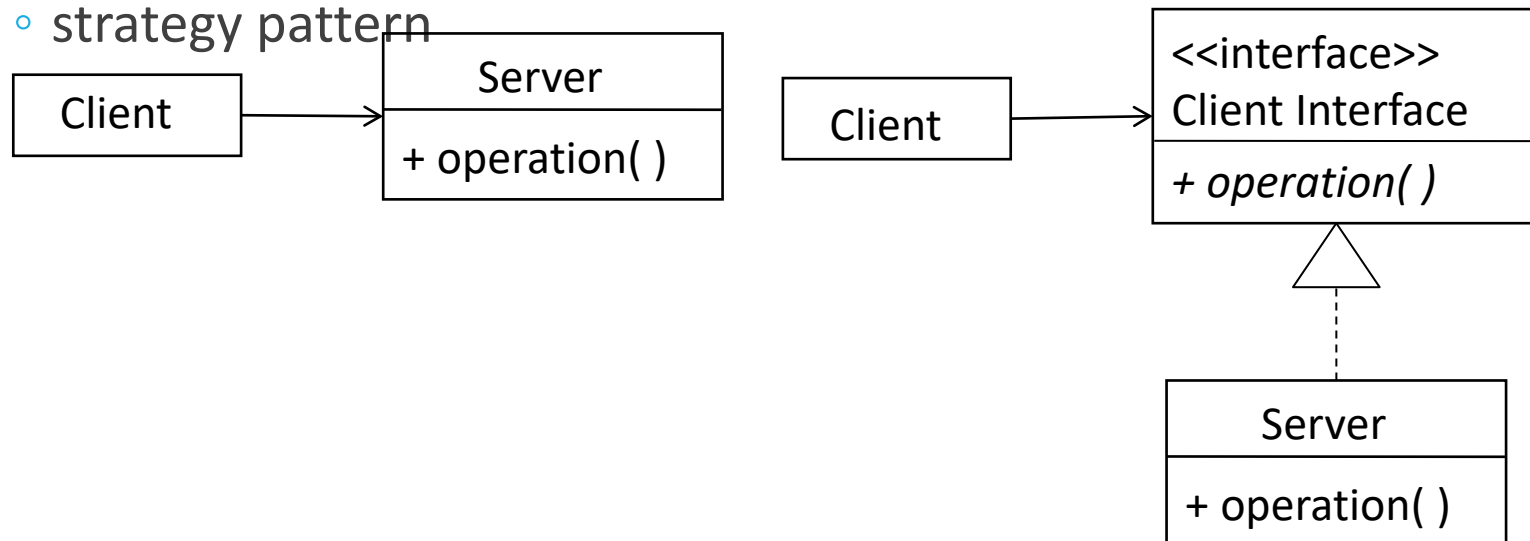
- 3 approaches to implement :
 - parameters
 - pass delegates / callbacks
 - inheritance / template methods pattern
 - child type override behavior of a base class
 - composition / strategy pattern
 - client code depends on abstraction
 - 'plug in' model
- when to apply :
 - experience tells you – “fool me once, shame on you”
- adds complexity to design – TANSTAAFL
- no design can be closed against all changes

Open / Close Principle

- Pattern approaches to implement :
 - template methods pattern



- strategy pattern



Liskov Substitution Principle

- “subtypes must be substitutable for their base type”
 - Robert C. Martin
- (strong) behavior subtyping - Barbara Liskov, 1988
- substitutability : child class must NOT
 - remove parent class behavior
 - violate parent class intent
- normal OOP vs. Liskov Substitution inheritance
 - IS-A relationship vs. IS-SUBSTITUTABLE-FOR relationship
 - e.g. dog is an animal vs. dog is substitutable for animal

Interface Segregation Principle

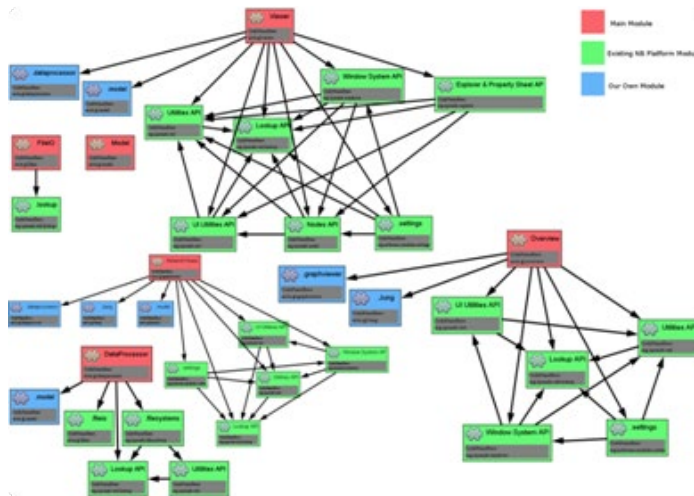
- “clients should not be forced to depend on methods they do not use” – Robert C. Martin
 - prefers small, cohesive interfaces
 - divide ‘fat’ interfaces into smaller ones, as having ‘fat’ interface leads to :
 - classes having methods they do not use
 - increase in coupling
 - reduced flexibility
 - reduced maintainability

Interface Segregation Principle

- applying ISP :
 - if the 'fat' interface is yours, separate it to smaller ones
 - if the 'fat' interface is NOT yours, use 'Adapter' pattern
 - small interfaces
 - cohesive interfaces
 - focused interfaces
 - let the client define interfaces
 - package interfaces with their implementation

Dependency Inversion Principle

- “high-level modules should not depend on low-level modules, both should depend on abstractions”
- “abstractions should not depend on details, details should depend on abstraction” - Robert C. Martin, 2003

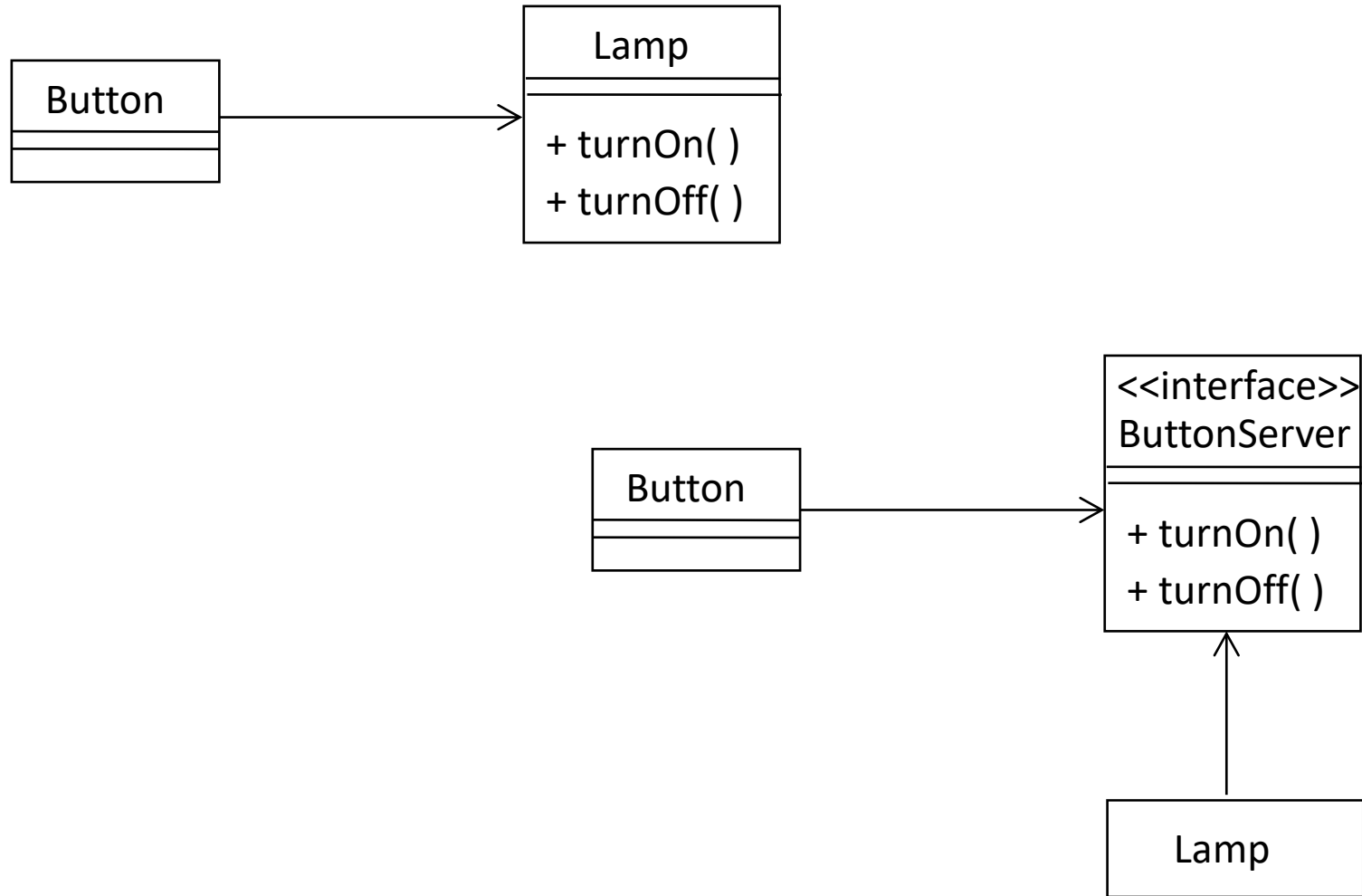


depend on abstractions



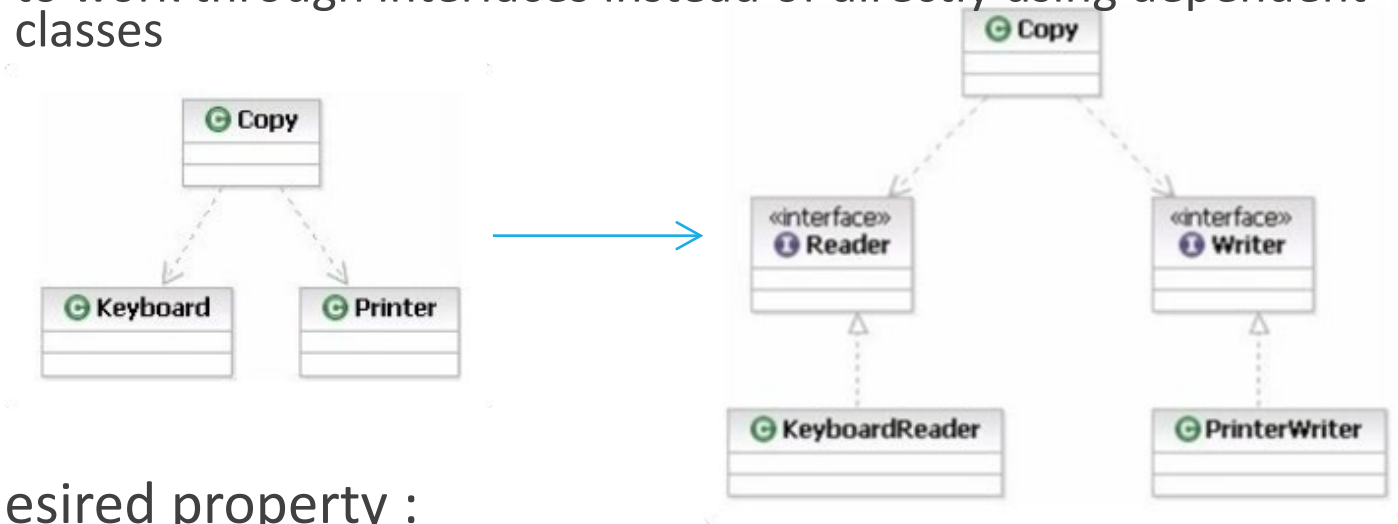
depend directly on other modules

Dependency Inversion Principle



Dependency Inversion Principle

- depend on abstraction
 - to work through interfaces instead of directly using dependent classes



- desired property :
 - classes should declare what they need
 - constructors should require dependencies
 - hidden dependencies should be shown
 - dependencies should be abstractions

Dependency Inversion Principle

- Applying DIP :
 - dependency injection
 - constructor injection
 - property injection
 - parameter injection
 - Hollywood principle : Don't call us, we'll call you!
- DIP related smell
 - inappropriate intimacy
 - a class using the internal fields and methods of another class

Don't Repeat Yourself

- “every piece of knowledge must have a single, unambiguous representation in the system”
 - Andrew Hunt & David Thomas, 1999
- “repetition in logic calls for abstraction, repetition in process calls for automation”
 - variation : Once and Only Once (OOO) & Duplication Is Evil (DIE)
- DRY related smells
 - shotgun surgery
 - any modification requires many small changes to many different classes

Speaking Code Principle

- “Any fool can write code that computer can understand, Good programmers write code that humans can understand” - Martin Fowler, 1999
 - code should communicate its purpose
- SCP related smell
 - comments
 - method filled with explanatory comments

Design Pattern

Design Patterns

- “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice” -

Christopher Alexander (Architect)

- applicable to object-oriented design pattern
- patterns are a solution to a problem in a context

Parts of Design Patterns

- Essential Parts :
 - pattern name
 - brief description of the design problem
 - **common vocabulary** for software designer
 - problem
 - description of the problem that the design pattern is intended to solve
 - solution
 - description of what elements are required to make up the design, their relationship and its context
 - Consequences
 - results and trade offs by applying the design pattern
 - allows comparison between different design patterns to find better fit for the actual problem

Types of Design Patterns

- general description of the type of problem the pattern addresses
- Creational Patterns
 - everything about the creation of objects
 - initializing and configuring classes and objects
- Structural Patterns
 - how classes and objects are composed to form larger structure
 - decoupling the interface and implementation of classes and object
- Behavioral Patterns
 - Algorithms and the assignment of responsibilities between objects
 - dynamic interaction among societies of classes and objects

Types of Design Patterns

- Creational Patterns
 - class : factory method
 - object : abstract factory, builder, prototype, singleton
- Structural Patterns
 - class : adapter
 - object : adapter, bridge, composite, decorator, façade, flyweight, proxy
- Behavioral Patterns
 - class : interpreter, template method
 - object : chain of responsibility, command, iterator, mediator, memento, observer, state, strategy, visitor