# Using Metrics To Manage Software Risks

## 1. Introduction

***Definition***

*Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world so as to describe them according to specified rules.*

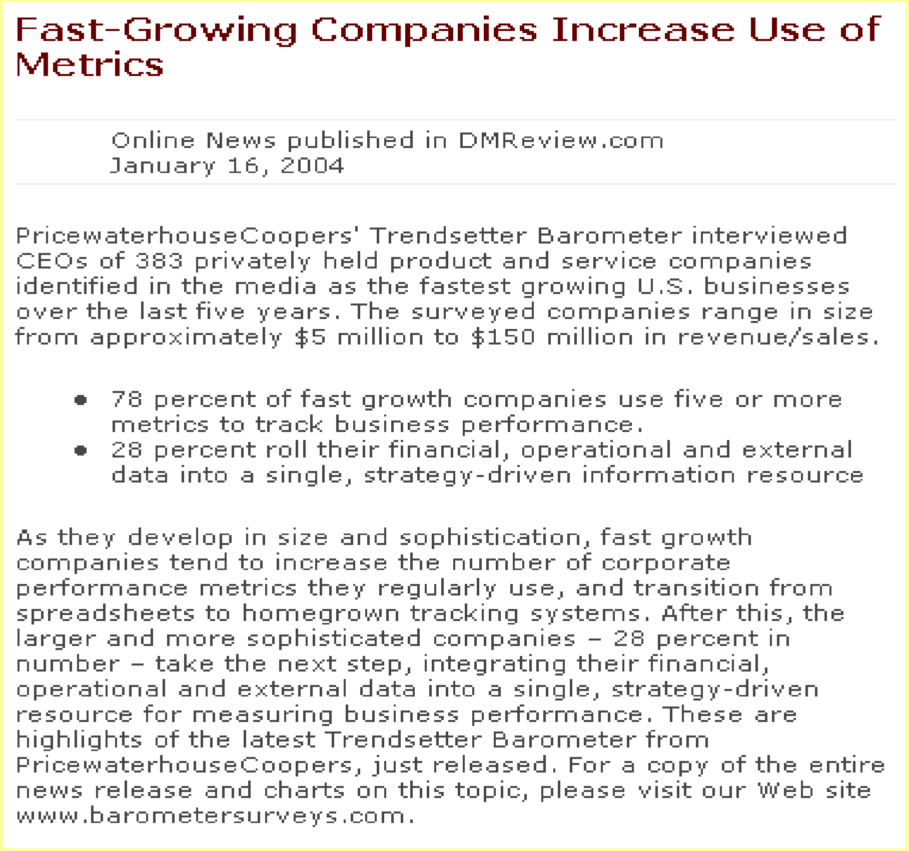
Measurement gives a snapshot of specific parameters represented by numbers, weights or binary statements. Taking measurements over time and comparing them with specific baseline generate metrics.

-There are two broad use of measurement: assessment and prediction:

- Predictive measurement of some attribute A relies on a mathematical model relating A to some existing measures of attributes $A_1, \ldots, A_n$.
- Assessment is more straightforward and relies on the current status of the attribute.

# Rationale for Software Metrics

- **Why use metrics in software development?**
  - Improve software development process through the ability to measure it.
  - Manage the process by assessing or predicting software quality.



Fast-Growing Companies Increase Use of Metrics

Online News published in DMReview.com
January 16, 2004

PricewaterhouseCoopers' Trendsetter Barometer interviewed CEOs of 383 privately held product and service companies identified in the media as the fastest growing U.S. businesses over the last five years. The surveyed companies range in size from approximately $5 million to $150 million in revenue/sales.

- 78 percent of fast growth companies use five or more metrics to track business performance.
- 28 percent roll their financial, operational and external data into a single, strategy-driven information resource

As they develop in size and sophistication, fast growth companies tend to increase the number of corporate performance metrics they regularly use, and transition from spreadsheets to homegrown tracking systems. After this, the larger and more sophisticated companies – 28 percent in number – take the next step, integrating their financial, operational and external data into a single, strategy-driven resource for measuring business performance. These are highlights of the latest Trendsetter Barometer from PricewaterhouseCoopers, just released. For a copy of the entire news release and charts on this topic, please visit our Web site www.barometersurveys.com.

- As a matter of fact, software metrics have been successfully developed for a broad range of quality attributes including reliability, performance, and maintainability.

## 2. Software Metrics

### *Overview*

-Software metrics provide a ***quantitative vehicle for evaluating and managing quality factors and risks*** related to a given software product.

-The software artifacts concerned by metrics include ***analysis***, and ***design models***, as well as ***program code***.

-Metrics can be used at early stages as leading quality indicators of the software architecture design. They can also be used to ***drive an iterative design process*** (such as the Rational Unified Process).

-Metrics may be collected either dynamically or statically.

-*Dynamic* metrics require execution of the software system, which restrict their applicability to later phase of the development.

-*Static* metrics, in contrast can be collected and used at early stages of the design.

## ***Software Measurement***

-Examples of software measures:

- product cost
- development effort
- development time
- product size
- productivity
- defect count
- scalability
- reliability

-There are 3 classes of entities of interest in software measurement: *processes, products, and resources.*

- *processes:* Are any software related activities which take place over time.
- *products:* are any artifacts, deliverables or documents which arise out of the processes.
- *resources:* are the items which are inputs to processes

-Measurement always targets specific software attribute or concept:

- Examples: complexity, cohesion, coupling, size, length, time, effort, maintainability etc.

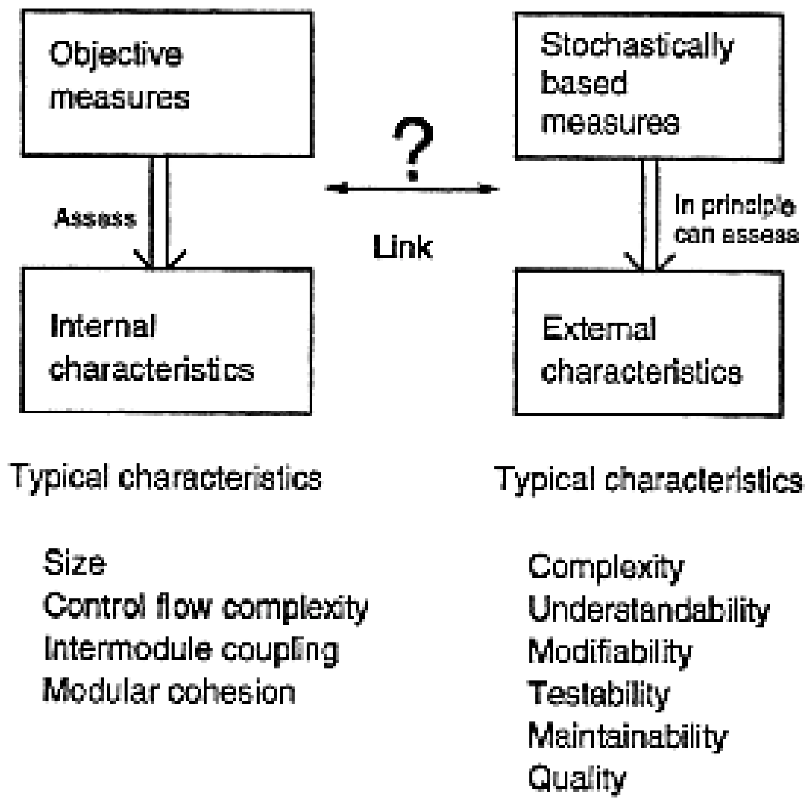
-In software measurement, a distinction is made between internal and external attributes:

- ***Internal attributes***: are those which can be measured purely in terms of the product, process, or resource itself. Example: size for product and elapsed time for process.
- ***External attributes***: are those which can only be measured with respect to how the product, process, or resource relates to other entities in its environment. Example: reliability for product and productivity for resource (e.g. people).

-Software Managers and Users would like to measure and predict external attributes.

- External attributes are easy to interpret but hard to measure **directly**, while internal attributes are hard to interpret but easy to collect directly.

-In practice, measurement of external attributes are derived indirectly from internal (attributes) measures, through correlation or statistical analysis such as regression or Bayesian probabilistic models.



•Example:

*Product Cost = f(effort, time); Effort (person/month)= g(size)*

## 3. Case Study: Measuring Maintainability

-Important aspects of maintainability include product simplicity, understandability, flexibility, and reusability.

- Complex code is difficult to understand, and thereby to maintain and evolve. Complex code increases the cost of testing, because the likelihood of faults is higher.

-Complexity is mastered by applying the principle of ***"divide and conquer"***, which typically underlies another common design principle, namely ***modular design***.

-Good modular design requires ***high cohesion*** of modules, and ***less coupling*** between them.

- Less cohesion means more complexity.
- Strong coupling means reduced reusability.

-Several software product metrics have been proposed to evaluate the complexity factors that affect the ***creation***, ***comprehension***, ***modification***, and ***maintenance*** of a piece of software.

| Metrics | Available at Design | Constructs/ Concepts |
|---|---|---|
| **Cyclomatic complexity (CC)** | **N** | Method/ Complexity |
| **Lines of Code (LOC)** | **N** | Method/ Size, complexity |
| **Comment percentage (CP)** | **N** | Method/ Complexity |
| **Weighted methods per class (WMC)** | ***Y*** | Class,Method/ Complexity |
| **Response for a class (RFC)** | **N** | Class, Method/ Complexity |
| **Lack of cohesion of methods (LCOM)** | **N** | Class/Cohesion |
| **Coupling between objects classes (CBO)** | ***Y*** | Class/Coupling |
| **Depth of inheritance tree (DIT)** | ***Y*** | Inheritance/ Complexity |
| **Number of children (NOC)** | ***Y*** | Inheritance/ Complexity |

## *Cyclomatic Complexity (CC)*

-Also called McCabe complexity metric

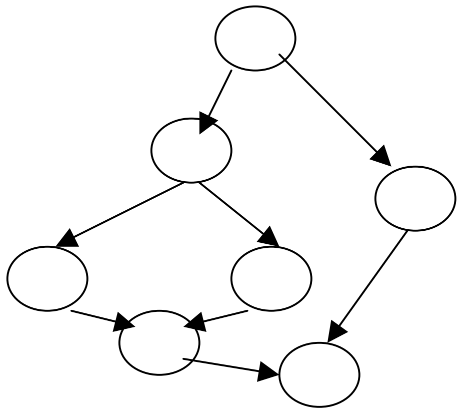
-Evaluate the ***complexity of algorithms*** involved in a method.

-Give a count of the number of test cases needed to test a method comprehensively;

-Use a control flow graph (CFG) to describe the software module or piece of code under study:

- ***Each node*** correspond to a block of sequential code.
- ***Each edge*** corresponds to a path created by a decision.

-CC is defined as the number of edges minus the number of nodes plus 2: ***CC= edges - nodes + 2***

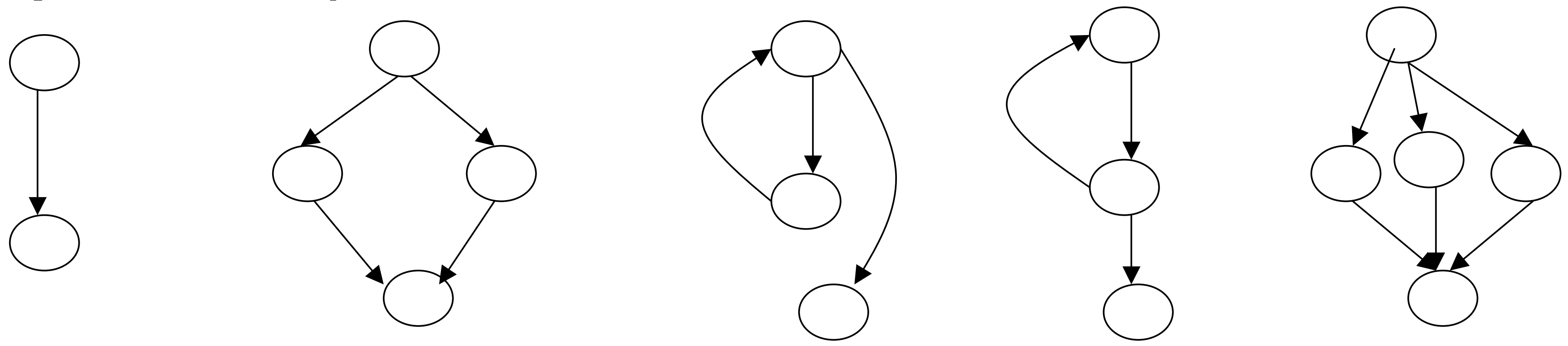


***CC= e- n + 2 = 8 – 7 + 2 = 3***

***Low CC means reduced testing, and better understandability.***

# *Primitive Operations of Structured Programming*

***sequence*** ***if/then/else*** ***while*** ***repeat*** ***case***



***sequence***

```
y=2+x;
```

**CC=1-2+2 = 1**

***if/then/else***

```
if (x>2)  y=2x;
else y=2;
```

**CC = 4 – 4 + 2 = 2**

***while***

```
while (x>2)
   y=2x;
```

**CC=3-3+2=2**

***repeat***

```
for(int i=0;i<5;
                  i++)
  x=x+i;
```

**CC=3-3+2=2**

***case***

```
switch(c) {
  case 0:…;
  case 1: …;
  default:…;
}
```

**CC=6-5+2=3**

## *Size*

-The size of a piece of code can be measured using different metrics.

- ***Lines of code (LOC)*** count all lines, including comments;
- ***Non-comment non-blank (NCNB)*** counts all lines except comments and blanks.
- ***Executable statements (EXEC)*** count the number of executable statements.

High size ***decreases understandability***, and therefore ***increases risk and faults***.

### *Examples:*

```
if x>2
then y=x+z;
```

***LOC=2, NCNB=2, EXEC=1***

```
/*evaluates…*/
if x>2
then y=x+z;

x=2z;
```

***LOC=5, NCNB=3, EXEC=2***

## *Comment Percentage (CP)*

-Is obtained by the total number of comments divided by the total number of lines of code less the number of blank lines.

Higher comment percentage means ***better understandability and maintainability***.

***Example:***

```
/*evaluates…*/
if x>2
then y=x+z;

x=2z;

/*computes…*/
 z=x*x-y;
```

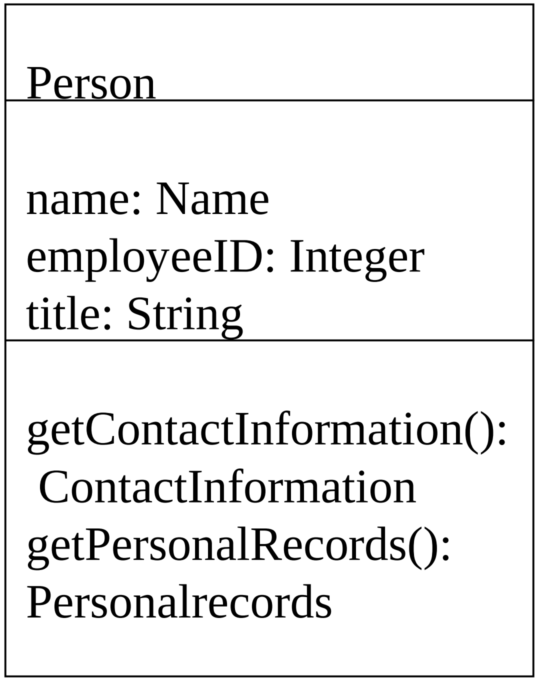
***CP = 2/(8-2)=33%***

## *Weighted Methods per Class (WMC)*

-Is measured either by counting the number of methods associated with a class, or by summing the complexities (CC) of the methods.

$$\mathrm{WMC} = \sum_{i=1}^{n} c_i,\ c_i = CC_i$$

***Example:***


Person
name: Name
employeeID: Integer
title: String
getContactInformation(): ContactInformation
getPersonalRecords(): Personalrecords

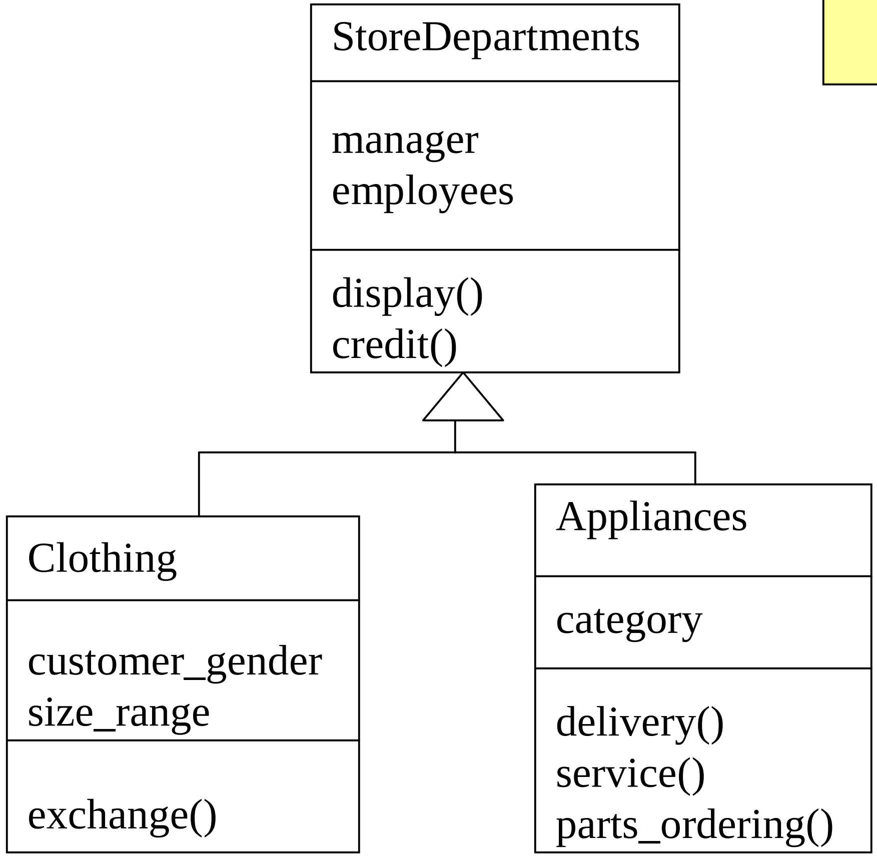
***High WMC value is a sign of high complexity, and less reusability.***

***WMC=2***

## *Response For a Class (RFC)*

-Measures the number of methods that can be invoked in response to a message to an object of the class or by some methods in the class; this includes all the methods accessible in the class hierarchy.

***Example:***



***Higher RFC value is a predictor of larger number of communications with other classes, so more complexity.***

***RFC (StoreDepartments) =2+1+3=6***

***RFC(Clothing) = 2+1=3***

***RFC(Appliances)=2+3=5***

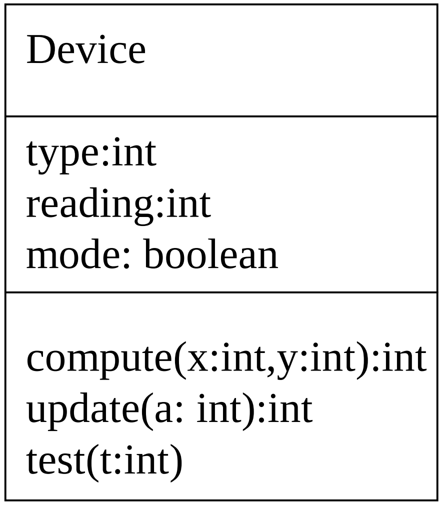
## *Lack of Cohesion (LCOM)*

-Measures the cohesion or lack of a class; evaluate the dissimilarity of methods in a class by instance variables or attributes.

-LCOM is measured by *counting the number of pairs of methods that have no attributes in common, minus the number of methods that do*. A negative difference corresponds to LCOM value of zero.

***Low cohesion is a sign of high complexity, and shows that the class can be subdivided. High cohesion indicates simplicity and high potential for reuse.***

### *Example:*



```
Class Device {
   int  reading, type;
   boolean mode=false;

   public int update (int a) {return a + reading; }
   public int  compute(int x, int y) {return x*y*type - reading;}
   public void test (int t) { if  t ==1   mode=true;}
}
```

***LCOM(Device)= 2 – 1 = 1***

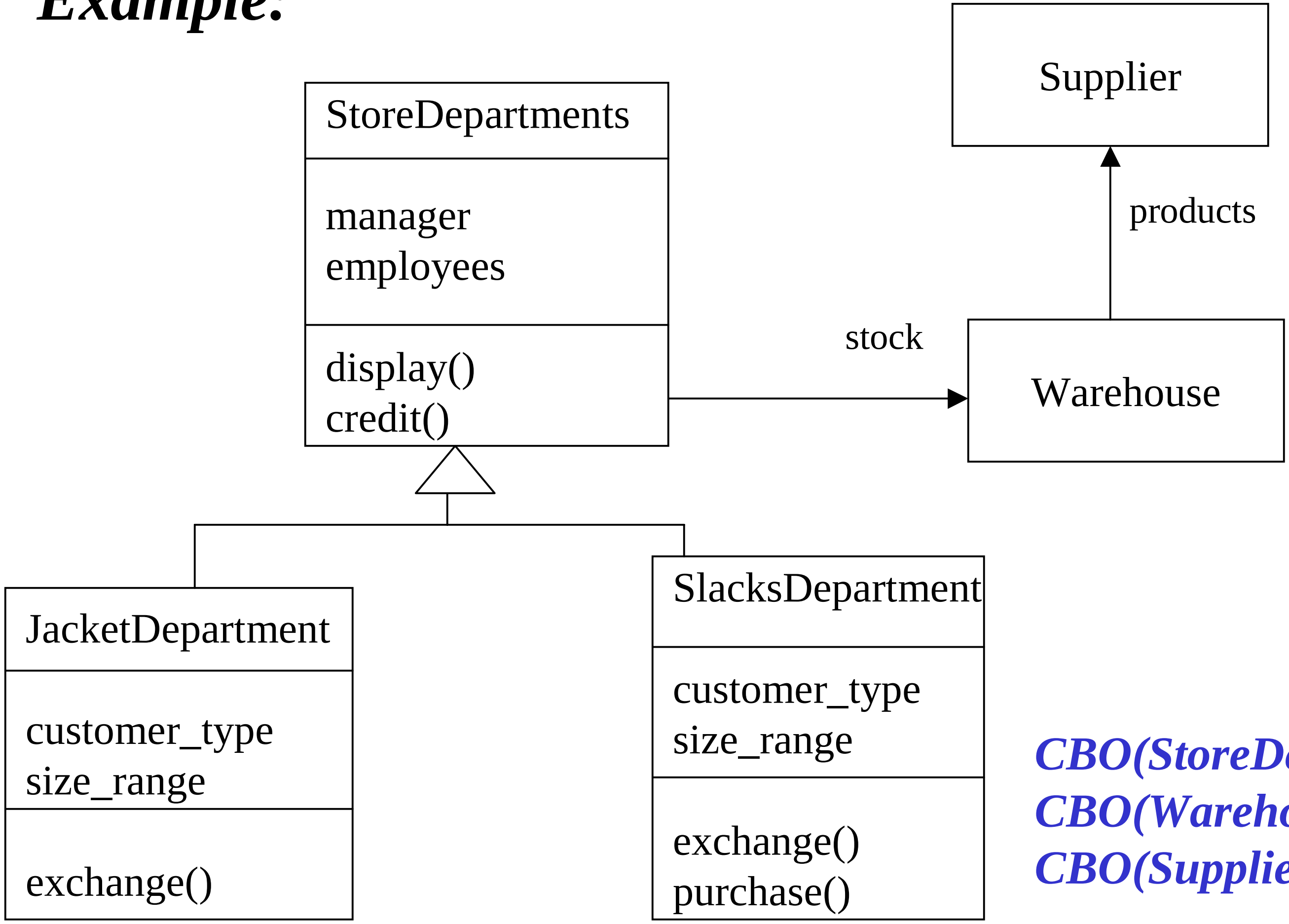
## *Coupling Between Object Classes (CBO)*

-Measures the number of classes to which a class is coupled.

-A class *A* is coupled to class *B* iff *A* uses *B*'s methods or instance variables.

-Coupling Is calculated by counting the number of distinct non-inheritance related class hierarchies on which a class depends.

***Example:***



***High coupling means increased dependency among the classes; this restricts reusability.***

***Useful for determining reusability***

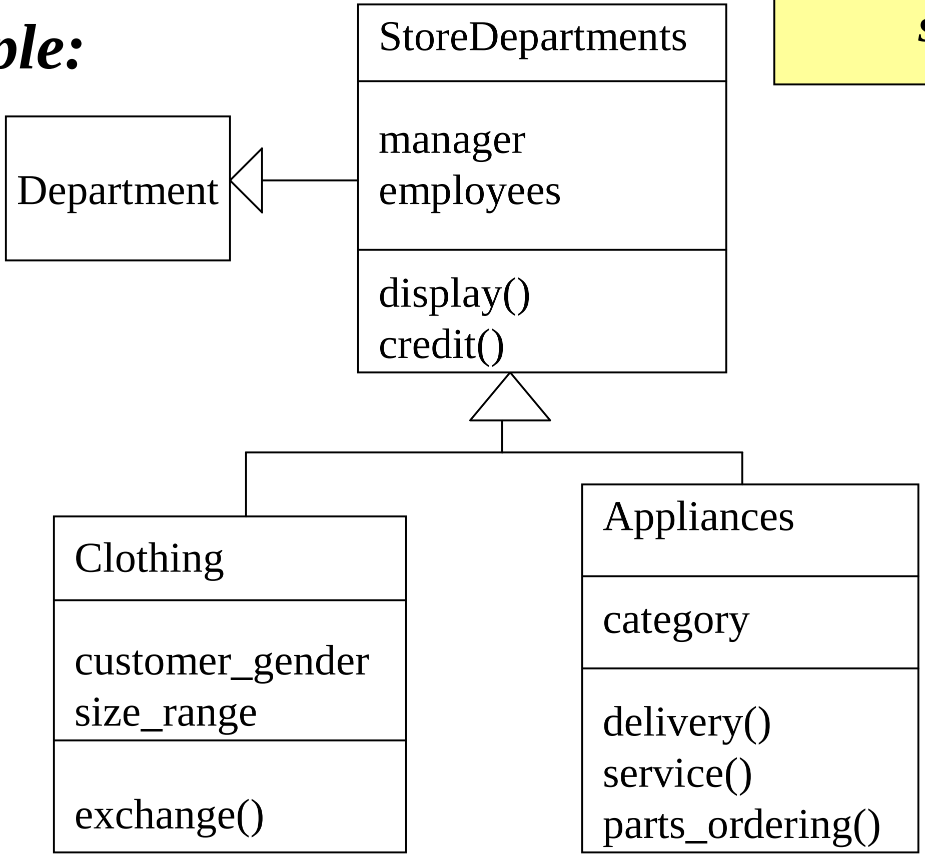
***CBO(StoreDepartments)=1***
***CBO(Warehouse)=1***
***CBO(Supplier)=0***

## *Depth of Inheritance Tree (DIT)*

-Measures the number of ancestor classes of a given class involved in an inheritance relation.

***Greater value of DIT means more methods to be inherited, so increased complexity; but at the same time that means increased reusability; so a trade-off must be made here.***

***Example:***



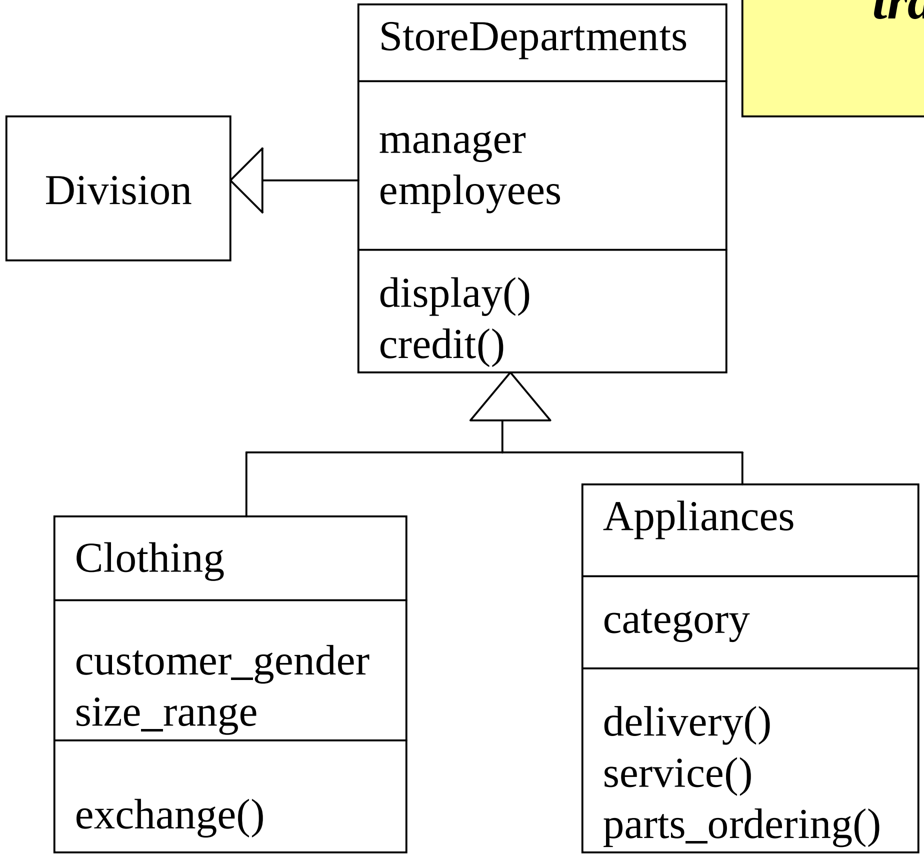
***DIT (Appliances) = 2***
***DIT(StoreDpartments)=1***
***DIT(Department)=0***

# *Number of Children (NOC)*

-Measures the number of immediate subclasses of a class in an inheritance hierarchy.

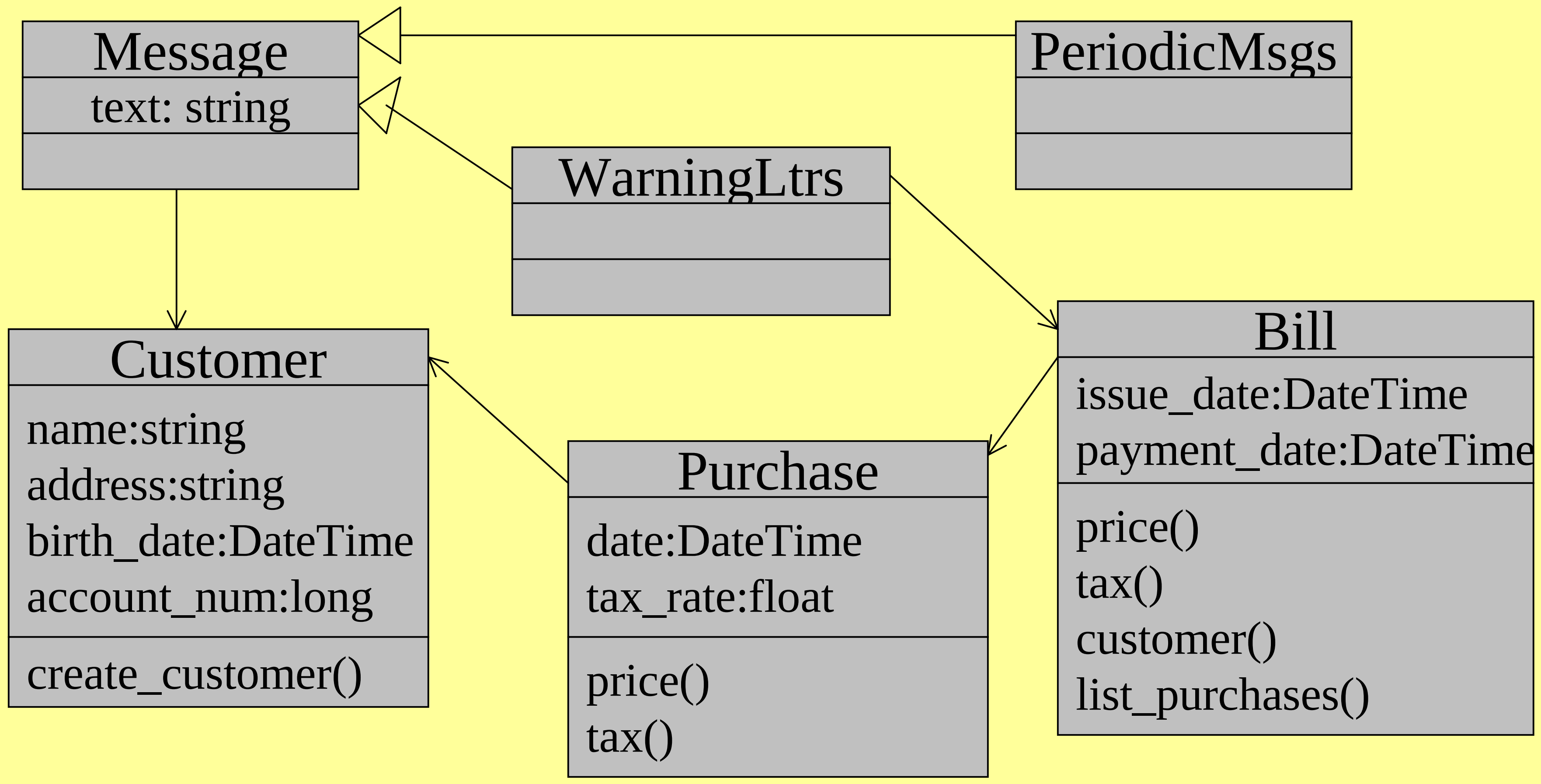
***High NOC means high reuse, but may also be the sign of improper abstraction or misuse of inheritance. High NOC may also be the sign of increased complexity. So a trade-off must be made for this metric.***

## *Example:*



*NOC(Division) =1*
*NOC(StoreDepartments)=2*
*NOC(Appliances)=0*

EXAMPLE:


Message
text: string
PeriodicMsgs
WarningLtrs
Bill
issue_date:DateTime
payment_date:DateTime
price()
tax()
customer()
list_purchases()
Customer
name:string
address:string
birth_date:DateTime
account_num:long
create_customer()
Purchase
date:DateTime
tax_rate:float
price()
tax()

| Metric | Bill | Purchase | Warning Ltrs | Periodic Msgs | Message | Customer |
|---|---|---|---|---|---|---|
| Weighted Methods/Class | 4 | 2 | 0 | 0 | 0 | 1 |
| Number of Children | 0 | 0 | 0 | 0 | 2 | 0 |
| Depth of Inheritance Tree | 0 | 0 | 1 | 1 | 0 | 0 |
| Response for a Class | - | - | - | - | - | - |
| Coupling Between Objects | 1 | 1 | 2 | 1 | 1 | 0 |
| Lack of Cohesion in Methods | - | - | - | - | - | - |

## 4. Metrics and Quality

***Metrics can be useful indicators of unhealthy code and design, pointing out areas where problems are likely to occur, by focusing on specific quality attributes.***

| **Metric** | **Source** | **OO Construct** | **Objectives** | **Quality Attribute** |
|---|---|---|---|---|
| CC | Traditional | Method | Low | Testability<br>Understandability |
| LOC | Traditional | Method | Low | Understandability<br>Reusability<br>Maintainability |
| CP | Traditional | Method | ~20-30% | Understandability<br>Maintainability |
| WMC | New OO | Class/<br>Method | Low | Testability<br>Reusability |
| DIT | New OO | Inheritance | Low<br>(Trade-off) | Reuse<br>Understandability<br>Maintainability |
| NOC | New OO | Inheritance | Low<br>(Trade-off) | Reusability<br>Testability |
| CBO | New OO | Coupling | Low | Usability<br>Maintainability<br>Reusability |
| RFC | New OO | Class/<br>Method | Low | Usability<br>Reusability<br>Testability |
| LCOM | New OO | Class/<br>Cohesion | Low High | Complexity<br>Reusability |