Chap 4. Using Metrics To Manage Software Risks

1. Introduction
2. Software Measurement Concepts
3. Case Study: Measuring Maintainability
4. Metrics and Quality

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.

-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, …, A_n.

÷Assessment is more straightforward and relies on the current status of the attribute.

-There are 3 classes of software metrics: *process*, *product*, *and project*

÷*Process metrics:* measure process effectiveness; Example: defect removal effectiveness.

÷*Product metrics:* measure product characteristics such as size, cost, defect count etc.

÷*Project metrics:* used to keep track of project execution; Examples: development time, development effort, productivity etc.

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

2. Software Measurement Concepts

-Measurement always targets specific software attribute or concept:

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

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

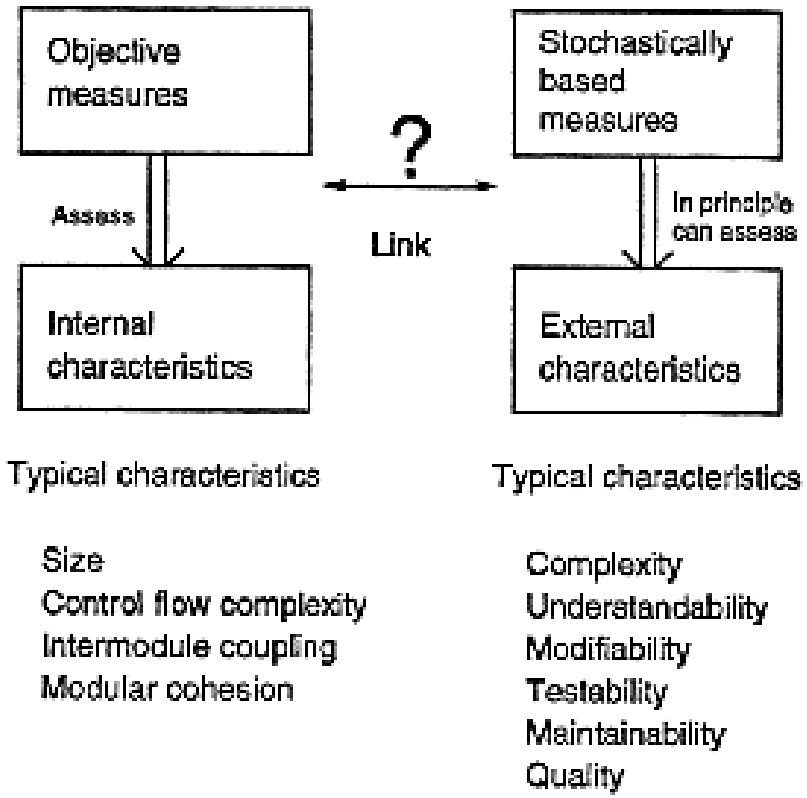
÷***Internal attributes***: are those which can be measured purely in terms of the product, process, or project itself. Examples: size for product and elapsed time for process.

÷***External attributes***: are those which can only be measured with respect to how the product, process, or project relates to other entities in its environment. Examples: reliability for product and productivity for project (e.g., people).

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

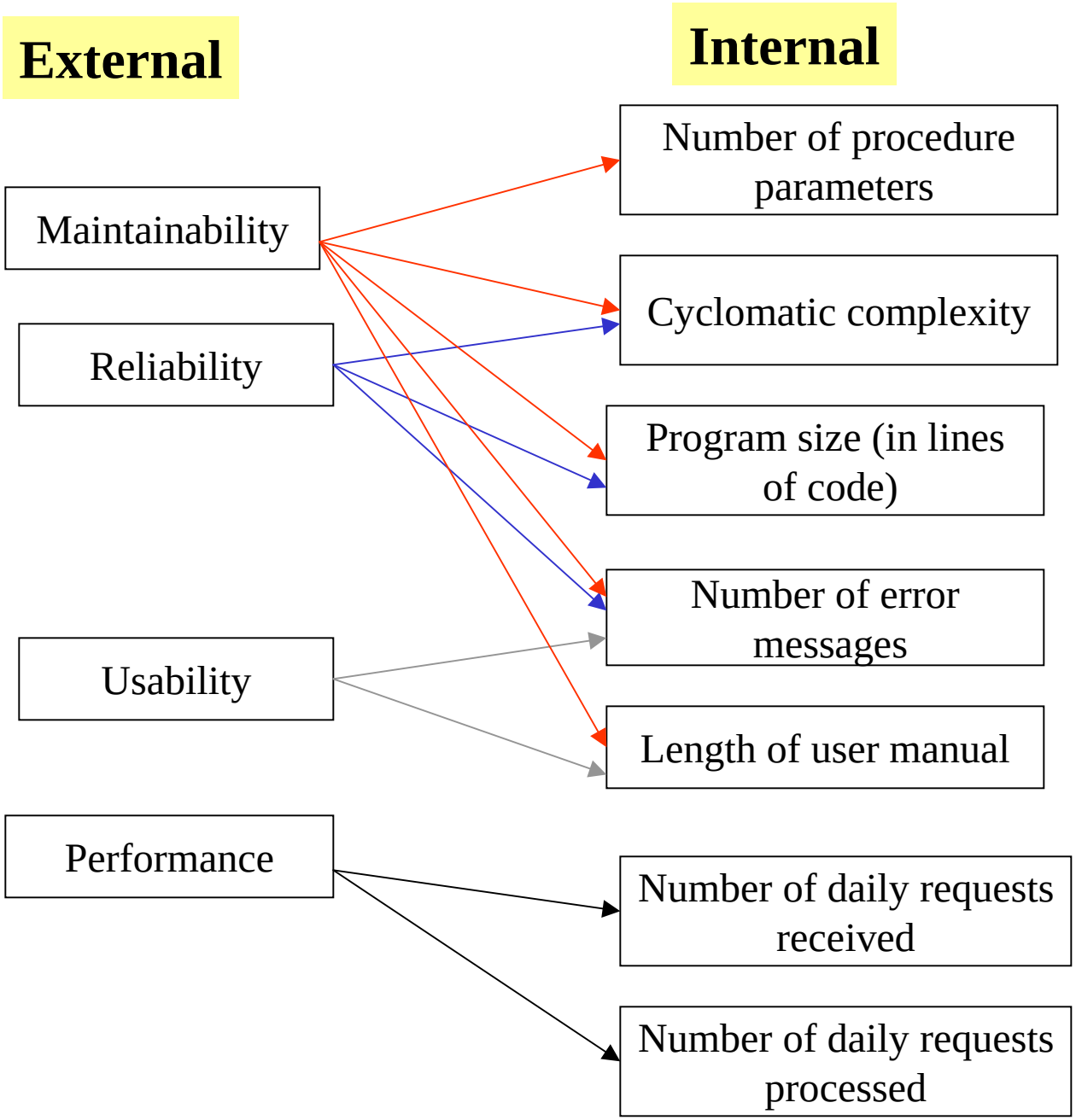
÷External attributes are easy to interpret but hard to measure **directly**, while internal attributes are hard to interpret but relatively easy to collect directly.

-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)


External
Internal
Maintainability
Reliability
Usability
Performance
Number of procedure parameters
Cyclomatic complexity
Program size (in lines of code)
Number of error messages
Length of user manual
Number of daily requests received
Number of daily requests processed

3. Case Study: Measuring Maintainability

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

÷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

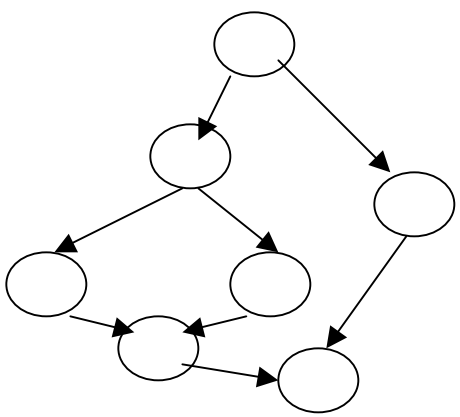
-Evaluate the ***complexity of algorithms*** involved in a method.

-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*** corresponds 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***

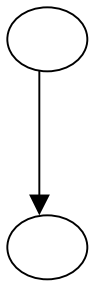


CC= e- n + 2 = 8 – 7 + 2 = 3

Low CC means reduced testing, and better understandability.

Primitive Operations of Structured Programming

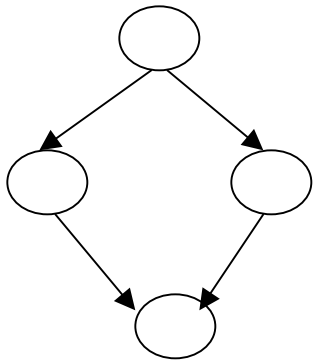
sequence



y=2+x;

CC=1-2+2 = 1

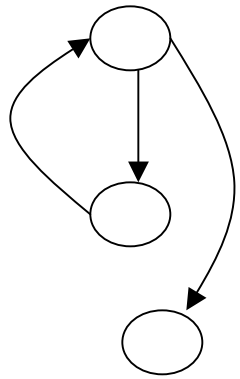
if/then/else



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

CC = 4 – 4 + 2 = 2

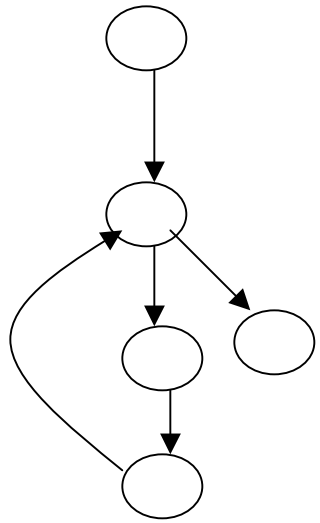
while



while (x>2)
y=2x;

CC=3-3+2=2

for loop



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

CC=5-5+2=2

Size

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

÷***(Basic) 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.

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

Example:

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

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

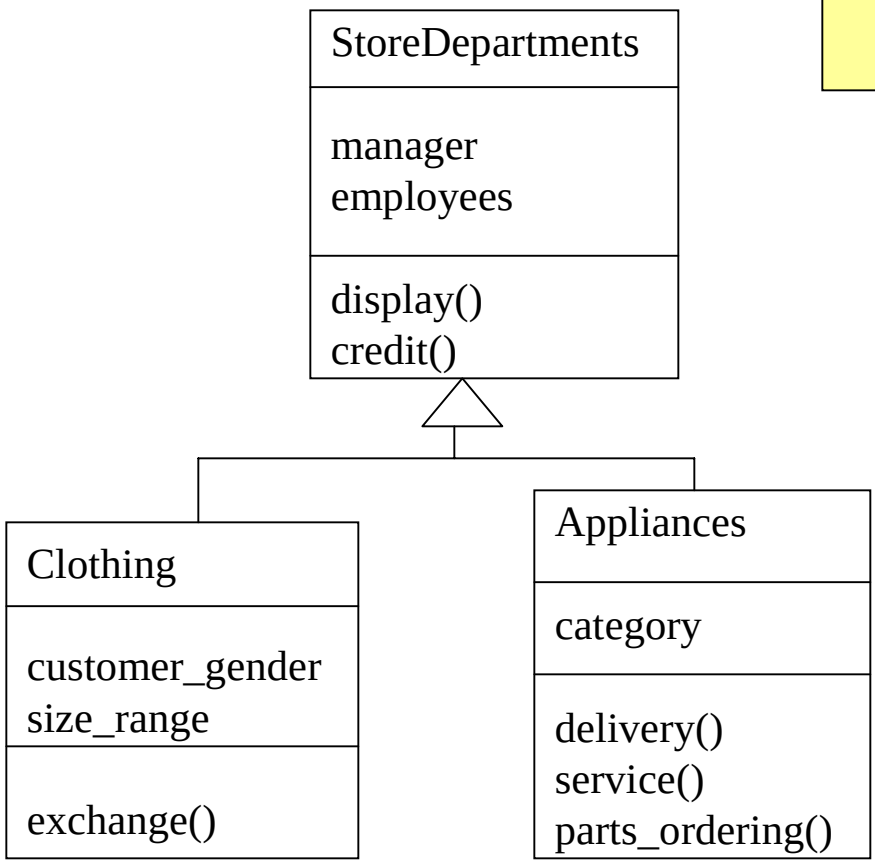
WMC=2

Response For a Class (RFC)

-Measure 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.

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

Example:



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

RFC(Clothing) =1+2=3

RFC(Appliances)=3+2=5

Lack of Cohesion (LCOM)

-Measure 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:

Device
type:int reading:int mode: boolean
compute(x:int,y:int):int update(a: int):int test(t:int)

```
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

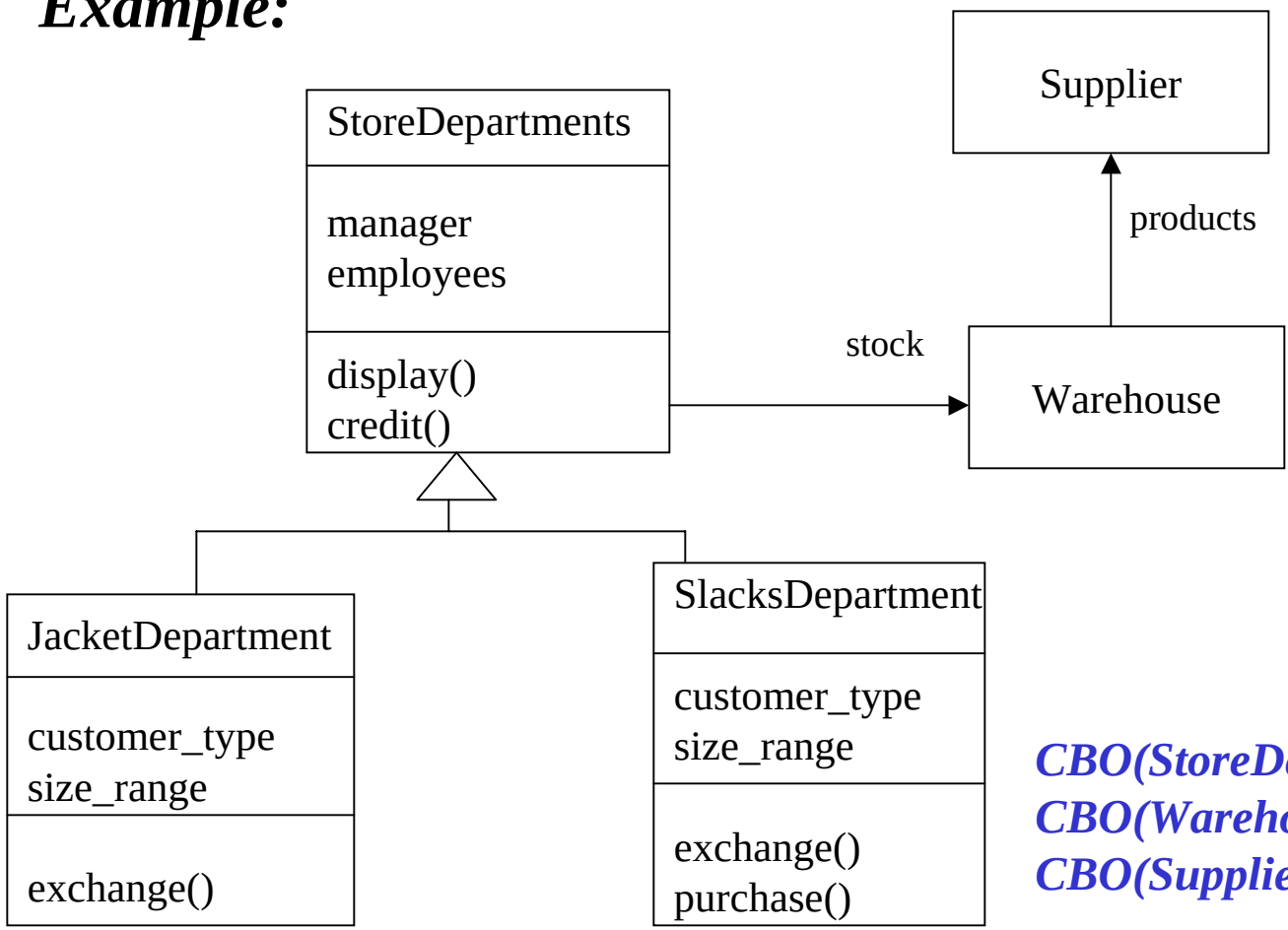
Coupling Between Object Classes (CBO)

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

-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

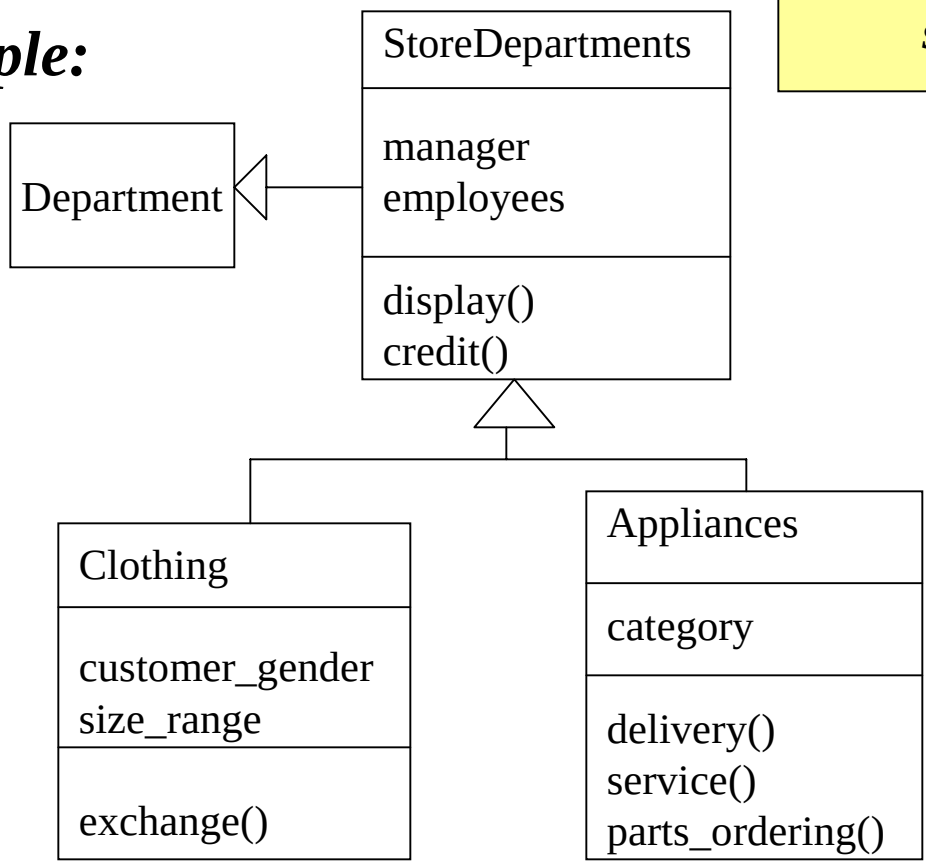
CBO(StoreDepartments) =1
CBO(Warehouse) =1
CBO(Supplier) =0

Depth of Inheritance Tree (DIT)

-Measure 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:



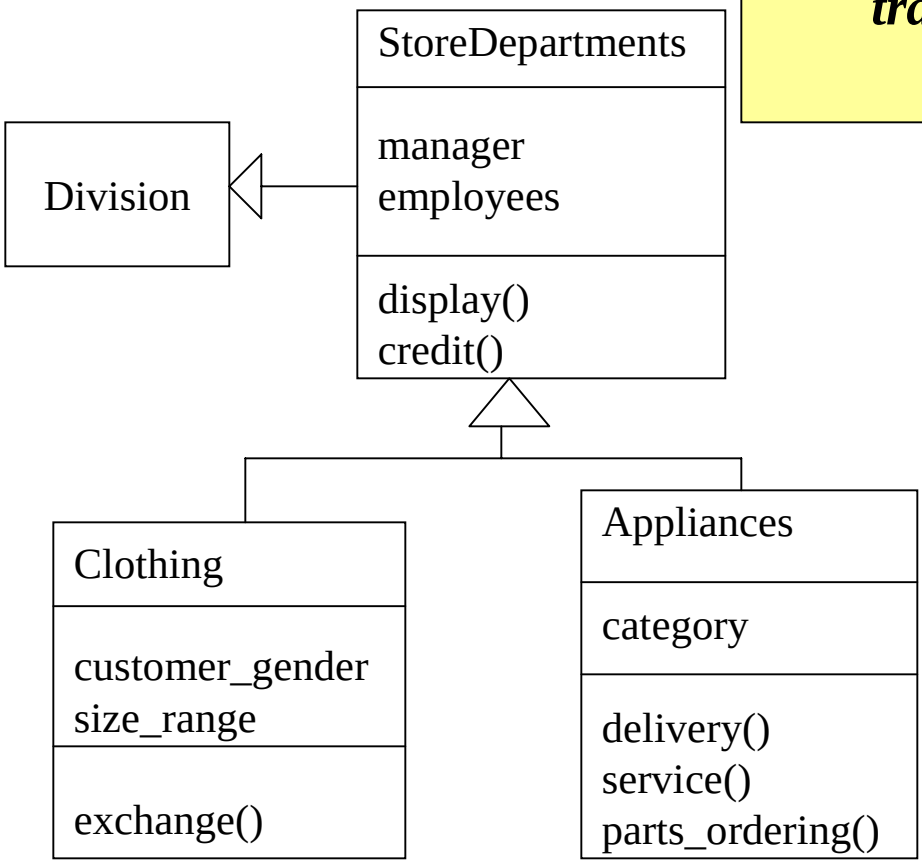
DIT (Appliances) =2
DIT(StoreDepartments)=1
DIT(Department)=0

Number of Children (NOC)

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

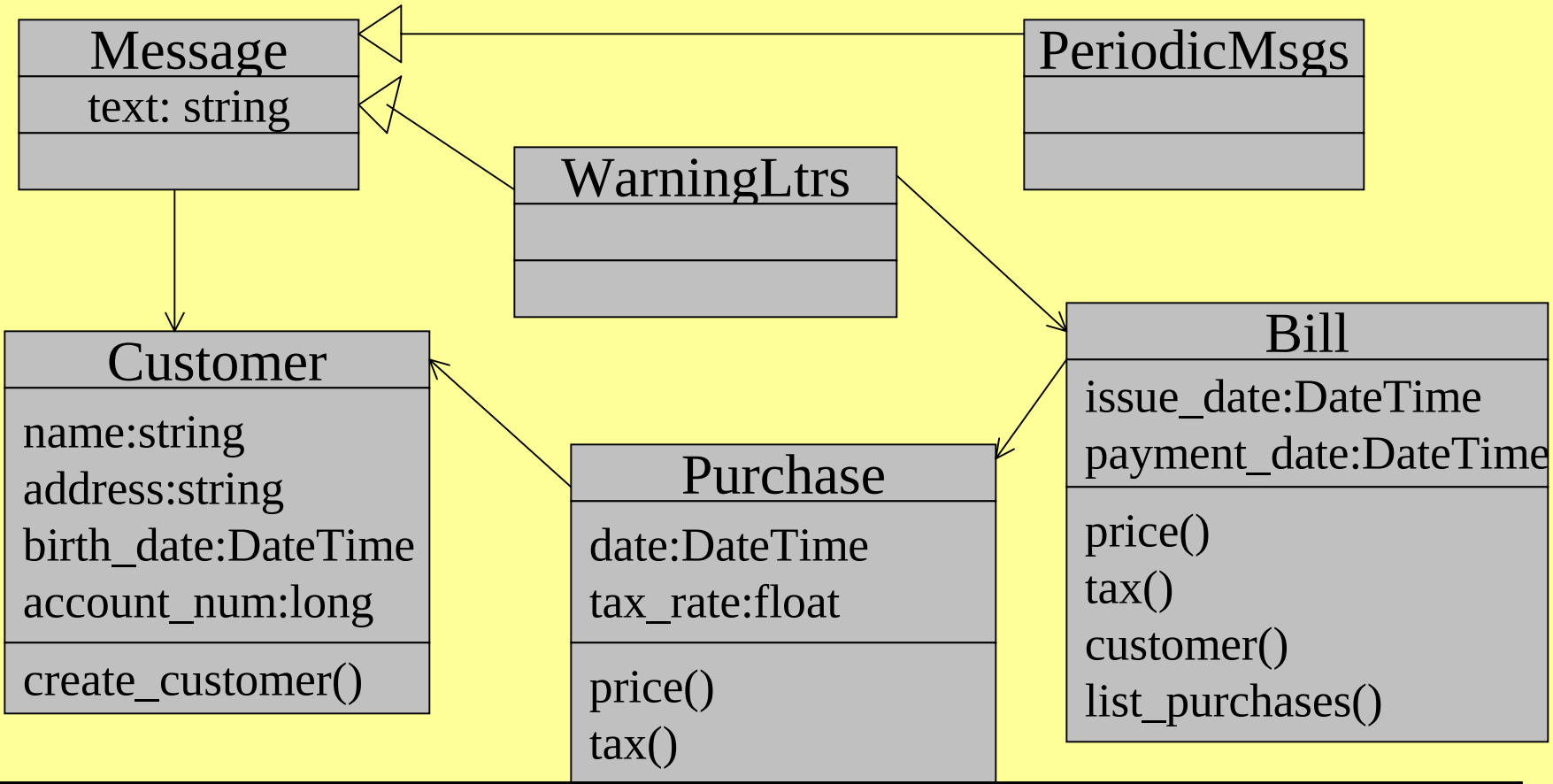
High NOC means high reuse, but may also be the sign of improper abstraction or misuse of inheritance. High NOC may also be the sign of increased complexity. So a trade-off must be made for this metric.

Example:



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

EXAMPLE: compute relevant CK metrics



Metric	Bill	Purchase	Warning Ltrs	Periodic Msgs	Message	Customer
Coupling Between Objects	1	1	2	1	1	0

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 Understandability
LOC	Traditional	Method	Low	Understandability Reusability Maintainability
CP	Traditional	Method	~20-30%	Understandability Maintainability
WMC	New OO	Class/ Method	Low	Testability Reusability
DIT	New OO	Inheritance	Low (Trade-off)	Reuse Understandability Maintainability
NOC	New OO	Inheritance	Low (Trade-off)	Reusability Testability
CBO	New OO	Coupling	Low	Usability Maintainability Reusability
RFC	New OO	Class/ Method	Low	Usability Reusability Testability
LCOM	New OO	Class/ Cohesion	Low High	Complexity Reusability