Object Oriented Software Estimation

Object Oriented Software Estimation

- After the requirements are gathered, the customer may like to know the cost and time estimates of the project.
- In order to determine cost and time of a project, estimates of size and effort are required. Providing accurate estimates for any software project is an important activity.
- Unfortunately, determining estimates in the early phases of software development is difficult due to lack of information available about the system at such an early stage.
- However, early estimates are highly desirable in order to determine the feasibility of a project or to bid for a tender.

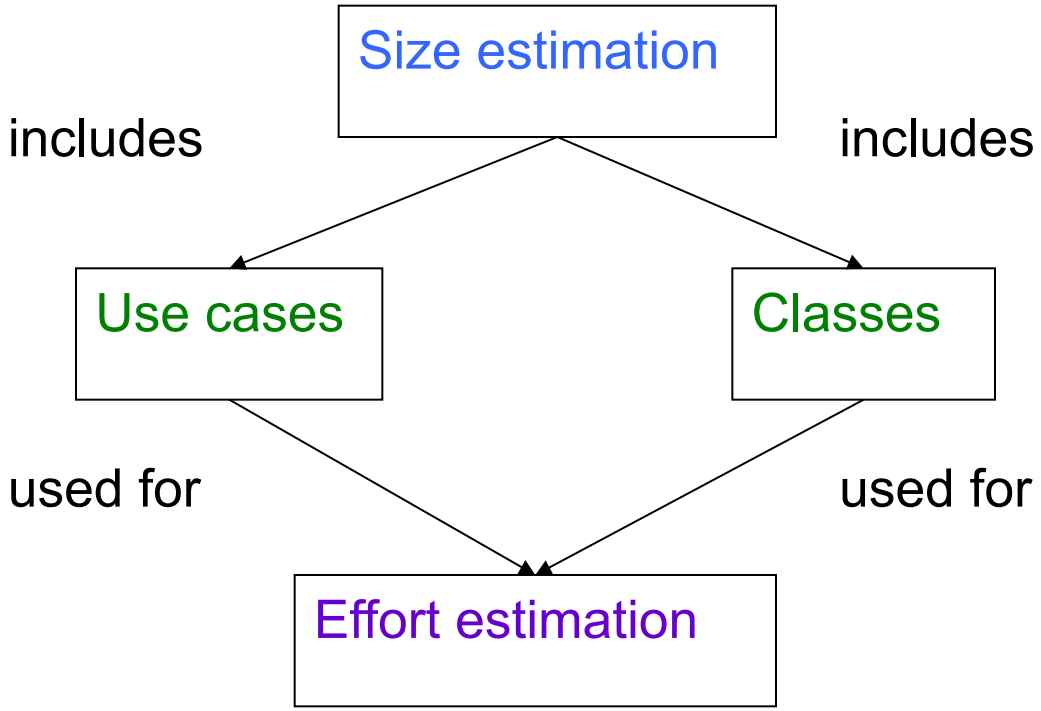
Object Oriented Software Estimation

- In order to conduct effective software estimates, we must identify:
  - Scope/boundaries of the project
  - Size of the project
  - Effort of the project
  - Resources required in the project
  - Risk involved in the project

Need of Object Oriented Software Estimation

- "Is object oriented software estimation different than traditional software estimation?"
  - We would say it is similar activity but the key parameters (for example size) to do an estimation change in case of object oriented software. Object oriented software engineering uses unified modeling language for creating models.
  - One of the important mechanisms to depict the functionality of the system is use cases.
  - The use case diagram may be used to predict the size and hence the effort of the software at an early stage of software development.
  - Classes are also an important element for measuring size in object oriented software.

Need of Object Oriented Software Estimation



Lorenz and Kidd Estimation Method

- The Lorenz and Kidd method for estimation of size and effort is one of the earliest methods developed in 1994. Lorenz and Kidd provides two methods for estimation of number of classes:
- **Use of scenario scripts:** Number of scenario scripts (same as use cases) may be used to estimate size of classes.
  - Number of classes= 17 × number of scenario scripts
  - This method for determining number of classes can be used in early phases of software development life cycle i.e. before the classes have been identified.

Lorenz and Kidd Estimation Method

- **Use of key and support classes:** After elaborating the design phase, the number of classes can be determined.
- Lorenz and Kidd differentiated between key classes and support classes.
  - Key classes are specific to business applications and are ranked with higher priority by the customer. These classes also involve many scenarios.
  - Support classes are common for many applications. These classes include user interface, back end classes and communications.

Lorenz and Kidd Estimation Method

Interface type	Multiplier
No GUI	2.0
Text-based user interface	2.25
Graphical user interface	2.5
Complex Graphical user interface	3.0

Lorenz and Kidd Estimation Method

- Support classes can be calculated as:
  - Support classes= number of key classes × Multiplier
- Finally, the total number of classes are obtained by adding key classes and support classes.
  - Total number of classes=Key classes + Support classes
- According to Lorenz and Kidd each class requires 10 to 20 person days for implementation. Thus effort can be calculated as
  - Effort=Total number of classes × (10 to 20 person days)

Lorenz and Kidd Estimation Method

- **Example 4.1:** An application consists of 15 scenario scripts and requires 15 person days to implement each class. Determine effort of the given application.

Lorenz and Kidd Estimation Method

- **Solution:** Number of classes = 17 × Scenario scripts

  = 17 × 15 =255

  Effort = 255 × 15 =3825 person days

Lorenz and Kidd Estimation Method

- **Example 2:** Consider the database application project with the following characteristics
  - The application has 45 key classes
  - A graphical user interface is required
- Calculate the effort to develop such a project given 20 person days.

Lorenz and Kidd Estimation Method

- **Solution**

Number of key classes = 45

Number of support classes = Number of key classes × multiplier

= 45 × 2.5

= 112.5

Total number of classes = Number of key classes + Number of support classes

= 112.5 + 45

= 157.5

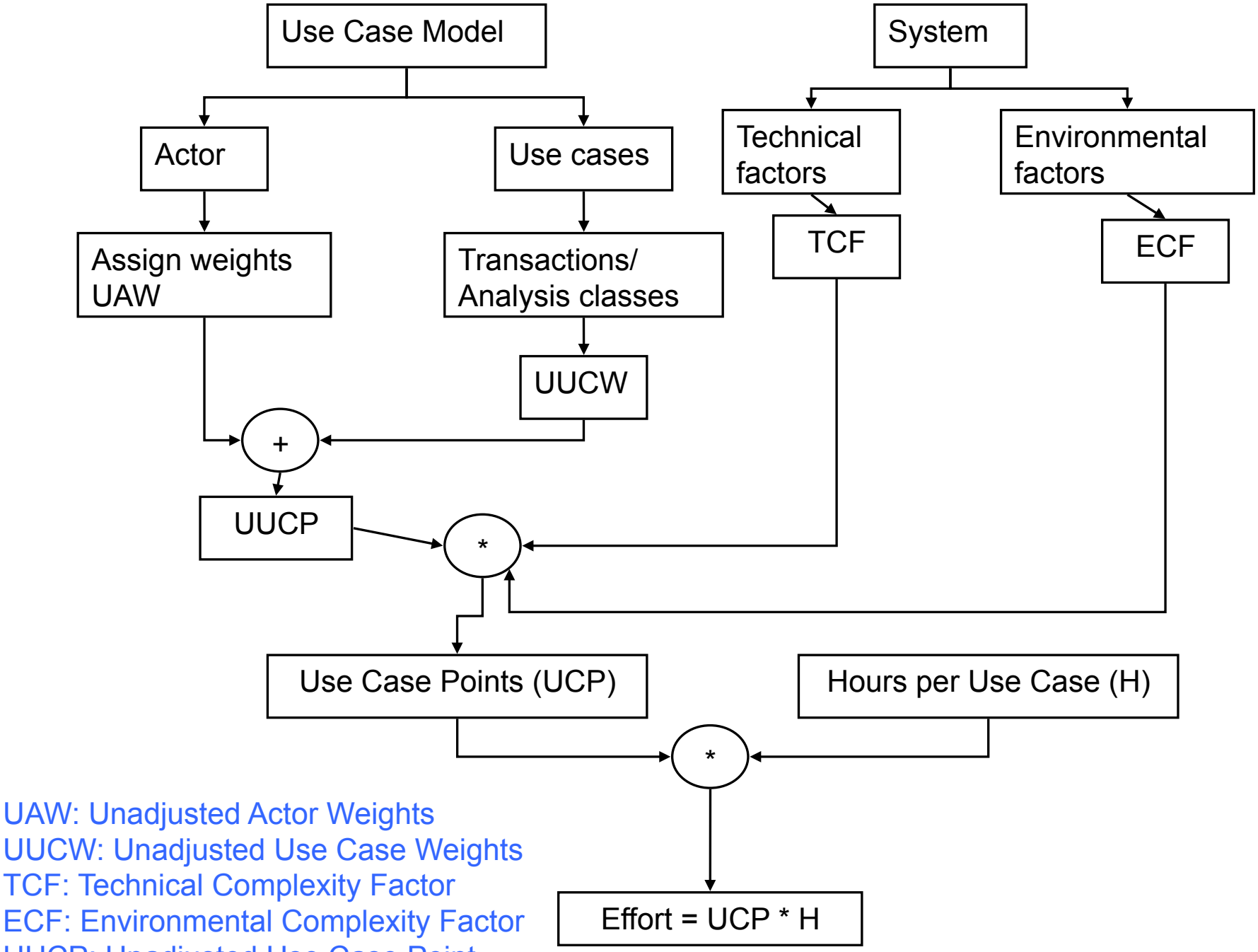
Effort = 157.5 × 20 = 3150 person days

Use Case Points

- The use case points method was developed by Gaustav Karner of Objectory (currently known as IBM Rational Software) in 1993.
- The method is used for estimating size and effort of object oriented projects using use cases.
- The method is an extension of function point analysis technique developed by Alan Albrecht.
- Kerner's work on use case points was written in his diploma thesis titled as "Metrics Objectory".

Use Case Points

- Thus following steps are used in use case points method:
  - **Classification of Actors and Use Cases:** In this step the complexity level of the actors and use cases are identified and their weighted sum is computed.
  - **Computing Unadjusted Use Case Points:** the estimates of unadjusted use case points are made.
  - **Calculating Technical Complexity Factors:** In this step we identify the degree of influence of technical factors on the project.
  - **Calculating Environmental Complexity Factors:** The environmental complexity factors are classified.
  - **Calculating Use Case Points:** In this final step the use case points are calculated on the basis of values obtained from step 1, 2, and 3.



UAW: Unadjusted Actor Weights
UUCW: Unadjusted Use Case Weights
TCF: Technical Complexity Factor
ECF: Environmental Complexity Factor
UUCP: Unadjusted Use Case Point

Use Case Points

Classification of Actors and Use Cases

- The first step involved in calculation of use case points is classification of actors and use cases. The actors are ranked according to their complexity i.e., simple, average or complex.

Actor complexity	Description	Weighting factor
Simple	Represents another system with a defined Application Programming Interface (API)	1
Average	Represents interaction another system has through a protocol such as TCP/IP or a human interaction with a line terminal.	2
Complex	Represents interaction through a graphical user interface	3

Use Case Points

- The Unadjusted Actor Weights (UAW) is the weighted sum of actors.

$$UAW = \sum_{i=1}^{n} a_i \times w_i$$

  - where n is number of actors

Use Case Points

Classification of Actors and Use Cases

- The use cases are ranked according to their complexity: simple, average or complex. There are two methods for classifying use case complexity:
  - A use case may be classified as simple, average or complex based on number of transactions. A transaction can be defined as collection of activities that are counted by counting the use case steps.
  - The other method to classify use case is counting the analysis classes which are counted by determining number of classes that we will need to implement a use case.

Use Case Points

Classification of Actors and Use Cases

Use Case complexity	Description	Weighting factor
Simple	Number of transactions <=3 OR Analysis objects<=5	5
Average	Number of transactions >3 and <7 OR Analysis objects>5 and <10	10
Complex	Number of transactions >= 7 OR Analysis objects >= 10	15

Use Case Points

Classification of Actors and Use Cases

- Each use case is multiplied by its corresponding weighting factors and this sum is added to get Unadjusted Use Case Weights (UUCW).

$$UUCW = \sum_{i=1}^{m} u_i \times w_i$$

- where m is number of use cases

Use Case Points

- **Computing Unadjusted Use Case Points**
- After classifying actors and use cases, the resultant unadjusted use case points (UUCP) is computed by adding UAW and UUCW as shown in mathematical form below:
  - UUCP = UAW + UUCW

Use Case Points

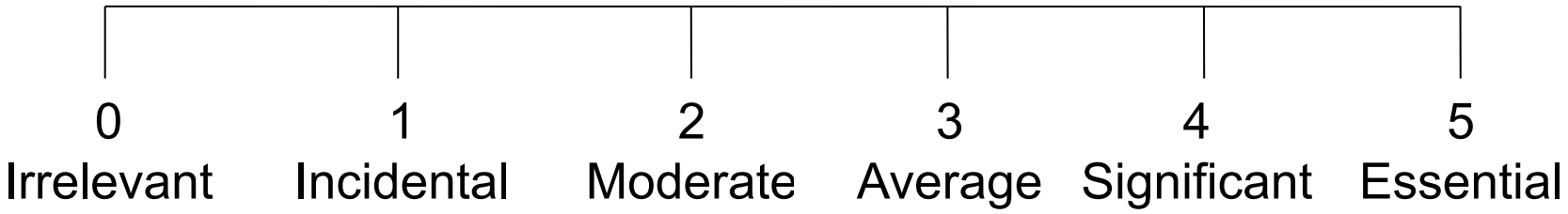
Calculating Technical Complexity Factors

- Technical Complexity Factor (TCF) assess the functionality of the software. TCFs are similar to the ones in function point analysis except some factors are added and some deleted.
- The criterion for technical factor is defined by Symons (1988) as:
  - *"A system requirement other than those concerned with information content intrinsic to and affecting the size of the task, but not arising from the project environment"*

Use Case Points

Calculating Technical Complexity Factors

- TCF vary depending on the difficulty level of the system. Each factor is rated on the scale of 0 to 5



Weighting factors for TCF

$Factor_i$	Contributing factors	W_i
1	Distributed systems	2
2	Application performance, Objectives in either response or throughput	1
3	End user efficiency	1
4	Complex internal processing	1
5	Reusability of source code	1
6	Installation ease	0.5
7	Ease of usability	0.5
8	Portability	2
9	Ease to change	1
10	Concurrency	1
11	Special security issues	1
12	Direct access for third parties	1
13	Special customer training provided	1

Use Case Points

Calculating Technical Complexity Factors

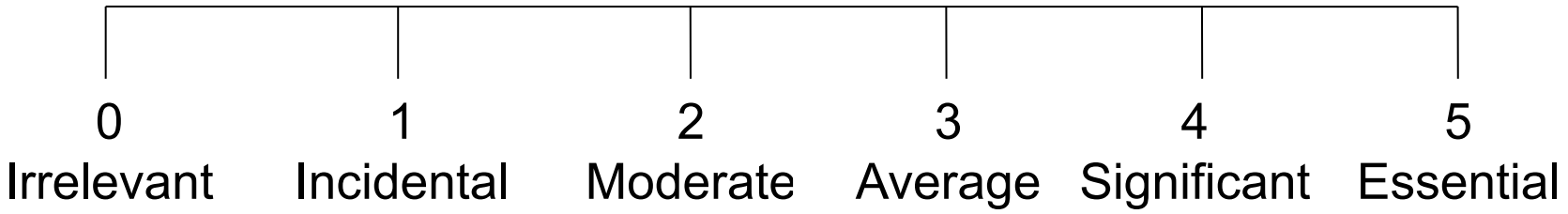
- TCF is obtained by using the following relationship:

$$TCF = 0.6 + 0.01 \sum_{i=1}^{13} F_i \times W_i$$

Use Case Points

Calculating Environmental Complexity Factors

- Environmental Complexity Factor (ECF) helps in estimating the efficiency of the project.
- These factors are calculated based on the early estimations in the project based out the interviews carried out Objectory.



Use Case Points

$Factor_i$	Contributing factors	W_i
1	Familiarity with objectory/ process	1.5
2	Application experience	0.5
3	Analyst capability	0.5
4	Object oriented experience	1
5	Motivation	1
6	Part-time workers	-1
7	Programming language difficulty	-1
8	Stable requirements	2

Use Case Points

Calculating Environmental Complexity Factors

- The ECF are obtained by using the following relationship:

$$ECF = 1.4 + (-0.03)\sum_{i=1}^{8} F_i \times W_i$$

Use Case Points

Calculating Use Case Points

- The final number of use case points is calculated by multiplying UUCP by TCF and ECF.
  - UCP = UUCP × TCF × ECF
- Duration can be measured 20 person hours per use case point.
- The use case points method can be used in early phases of software development in order to estimate the size an effort.

Use Case Points

Example 3

- Consider an airline reservation system where the following information is available:
  - Number of actors: 05
  - Number of use cases: 10
- Assume all complexity factors are average. Compute the use case points for the project.

$$UAW = \sum_{i=1}^{5} a_i \times w_i$$

$$= 5 \times 2 = 10$$

$$UUCW = \sum_{i=1}^{10} u_i \times w_i$$

$$= 10 \times 10 = 100$$

$$UUCP = UAW + UUCW$$

$$= 10 + 100 = 110$$

$$TCF = 0.6 + 0.01 \sum_{i=1}^{13} F_i \times W_i$$

$$= 0.6 + 0.01 \times (3 \times 2 + 3 \times 1 + 3 \times 1 + 3 \times 1 + 3 \times 1 + 3 \times 0.5 + 3 \times 0.5 + 3 \times 2 + 3 \times 2 + 3 \times 1$$

$$+ 3 \times 1 + 3 \times 1 + 3 \times 1)$$

$$= 0.6 + 0.01 \times 42 = 1.02$$

$$ECF = 1.4 + (-0.03) \sum_{i=1}^{8} F_i \times W_i$$

$$= 1.4 + (-0.03) \times (3 \times 1.5 + 3 \times 0.5 + 3 \times 0.5 + 3 \times 1 + 3 \times 1 + 3 \times -1 + 3 \times -1 + 3 \times 2)$$

$$ECF = 0.995$$

$$UCP = UUCP \times TCF \times ECF$$

$$= 110 \times 1.02 \times 0.995$$

$$UCP = 111.639$$

Use Case Points

Example 4 The following information is available for an application

- 2 simple actors, 2 average actors, 1 complex actor, 2 use cases with number of transactions 3, 4 use cases with number of transactions 5 and 2 use cases with number of transactions 15. In addition to the above information, the system requires:
  - Significant user efficiency
  - Essential ease to change
  - Moderate concurrency
  - Essential application experience
  - Significant object oriented experience
  - Essential stable requirements
- Other technical and environmental complexity factors are treated as average. Compute the use case points for the project.

Use Case Points

$$UAW = \sum_{i=1}^{5} a_i \times w_i$$

$$= 2 \times 1 = 2$$

$$2 \times 2 = 4$$

$$1 \times 3 = 3$$

$$UAW = 2 + 4 + 3 = 9$$

$$UUCW = \sum_{i=1}^{8} u_i \times w_i$$

$$= 2 \times 5 + 4 \times 10 + 2 \times 15 = 10 + 40 + 30 = 80$$

$$UUCP = UAW + UUCW$$

$$= 9 + 80 = 89$$

$$TCF = 0.6 + 0.01 \sum_{i=1}^{13} F_i \times W_i$$

$$= 0.6 + 0.01 \times (3 \times 2 + 3 \times 1 + 4 \times 1 + 3 \times 1 + 3 \times 1 + 3 \times 0.5 + 3 \times 0.5 + 3 \times 2 + 5 \times 1 + 2 \times 1$$

$$+ 3 \times 1 + 3 \times 1 + 3 \times 1)$$

$$= 0.6 + 0.01 \times 44 = 1.84$$

Use Case Points

$$ECF = 1.4 + (-0.03)\sum_{i=1}^{8} F_i \times W_i$$

$$= 1.4 + (-0.03) \times (3 \times 1.5 + 3 \times 0.5 + 3 \times 0.5 + 4 \times 1 + 3 \times 1 + 3 \times -1 + 3 \times -1 + 5 \times 2)$$

$$= 1.4 + (-0.03) \times 19.5$$

$$ECF = 0.815$$

$$UCP = UUCP \times TCF \times ECF$$

$$= 89 \times 1.84 \times 0.815$$

$$UCP = 133.46$$

Use Case Points

Example 5

- Consider example 4 and calculate effort for the given application.

Solution:

Effort=UCP×20

= 133.46×20

Effort=2669.29 person hours

Class Point Method

- This method is used to provide system level size estimation of object oriented software.
- The class point method was given by Gennaro Costagliola et al. in 2005.
- This method primarily focuses on classes for the estimation of size.
- The complexity of a class is analyzed through determining number of method in a class, number of attributes in a class and interaction of class with other classes.

Class Point Method

- The steps required for the estimation of size are given below
  - Identification of classes
  - Determination of complexity of classes
  - Calculation of unadjusted class point
  - Calculation of technical complexity factor
  - Calculation of class point

Class Point Method

Identification of Classes

- In class point method four types of classes are identified.

Class type	Description	Example (Library management system)
Problem Domain Type (PDT)	Represents real world entities in the application domain.	Student, book
Human Interface Type (HIT)	Satisfies the need for visualizing information and human-computer interaction.	Loginform, bookdetailform
Data Management Type (DMT)	Includes classes which incorporate data storage and retrieval.	Login
Task Management Type (TMT)	Includes classes which are responsible for tasks	Bookdetailscontroller, studnetdetailslcontroller

Class Point Method

Classifying Class Complexity

- In class point method, two measures CP1 and CP2 are used, in CP1 the initial estimate of size is made and CP2 provides detailed estimates of size. The following measures are used in the calculation of CP1 and CP2 measures:
  - Number of External Methods (NEM): It measures the size of class in terms of methods. It counts the number of public methods in a class.
  - Number of Service Requested (NSR): It measures interaction between system components. It counts the number of services requested by other classes.
  - Number of Attributes (NOA): It measures the complexity of a class. It counts the number of data members in a class.

Class Point Method

Classifying Class Complexity

- In CP1, NEM and NSR are used in order to classify complexity of a class.

	0-4 NEM	5-8 NEM	>=9 NEM
0-1 NSR	Low	Low	Average
2-3 NSR	Low	Average	High
>=4 NSR	Average	High	High

Class Point Method

Classifying Class Complexity

- In CP2 measure, NOA is also considered, thus a detailed insight into the estimate of size is obtained.

		NOA		
	0-2 NSR	0-5	6-9	>=10
NEM	0-4	Low	Low	Average
	5-8	Low	Average	High
	>=9	Average	High	High

Class Point Method

Classifying Class Complexity

		NOA		
	3-4 NSR	0-4	5-8	>=9
NEM	0-3	Low	Low	Average
	4-7	Low	Average	High
	>=8	Average	High	High

		NOA		
	>=5 NSR	0-3	4-7	>=8
NEM	0-2	Low	Low	Average
	3-6	Low	Average	High
	>=7	Average	High	High

Class Point Method

Calculating Unadjusted Class Points

- The Total Unadjusted Class Point (TUCP) is calculated by assigning complexity weights based on the classifications

Class type	Description	Complexity weight		
		Low	Average	High
PDT	Problem Domain Type	3	6	10
HIT	Human Interface Type	4	7	12
DMT	Data Management Type	5	8	13
TMT	Task Management Type	4	6	9

Class Point Method

Calculating Unadjusted Class Points

- After classifying the complexity level of classes the TUCP are calculated as follows:

$$TUCP = \sum_{i=1}^{4} \sum_{j=1}^{3} w_{ij} \times x_{ij}$$

1. Where w_{ij} are complexity weights j (j is low, average, high) assigned to class type i. x_{ij} is number of classes of type i (i is type of class problem domain, human interface, data management, task management).

Class Point Method

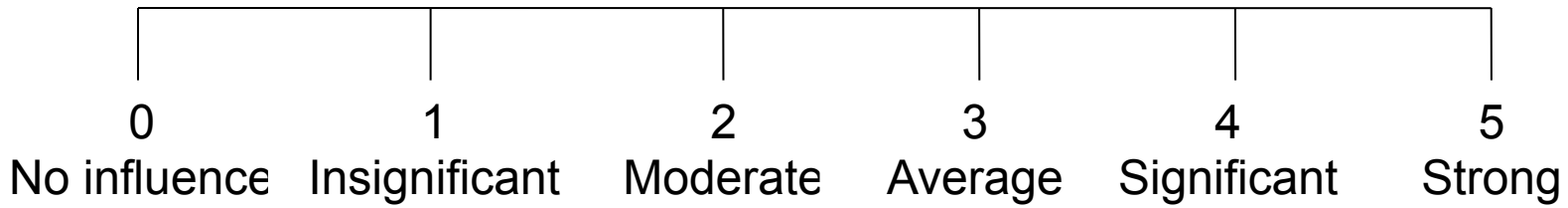
Calculating Technical Complexity Factor

- The factors F1-F18 are the Degree of Influence (DI). Each DI is rated on the scale of 0-5. DI is used to determine Technical Complexity factor (TCF). The TCF is determined by the following mathematical formula:

$$TCF = 0.55 + 0.01 \sum_{i=1}^{18} F_{ii}$$

Class Point Method

Calculating Technical Complexity Factor



Class Point Method

F_i	Description	F_i	Description
1	Data communication	10	Reusability
2	Distributed functions	11	Ease of installation
3	Performance	12	Ease of operation
4	High used configuration	13	Multiple users
5	Transaction rate	14	Facilitation of change
6	Online data entry	15	Adaptability by user
7	End-user efficiency	16	Rapid prototyping
8	Online update	17	Multiuser interaction
9	Compiler processing	18	Multiple interfaces

Class Point Method

Calculating Class Point and Effort

- The final class point is calculated by multiplying total unadjusted class point values with technical factor. The procedure for calculating adjusted Class Point (CP) is given below:
  - $CP = TUCP \times TCF$
- The effort is defined in terms of person hours for both CP1 and CP2 measures as:
  - $Effort = 0.843 \times CP1 + 241.85$
  - $Effort = 0.912 \times CP2 + 239.75$

Class Point Method

Example 7

- Consider a project with the following parameters:
- Assume all the technical complexity factors have average influence. Calculate class points.

Class type	Number of classes	NEM	NSR
PDT	4	6	2
		4	5
		10	15
		5	7
HIT	6	7	10
		5	8
		8	12
		6	9
		2	4
		4	2
DMT	3	2	3
		1	0
		3	4
TMT	2	4	8
		8	12

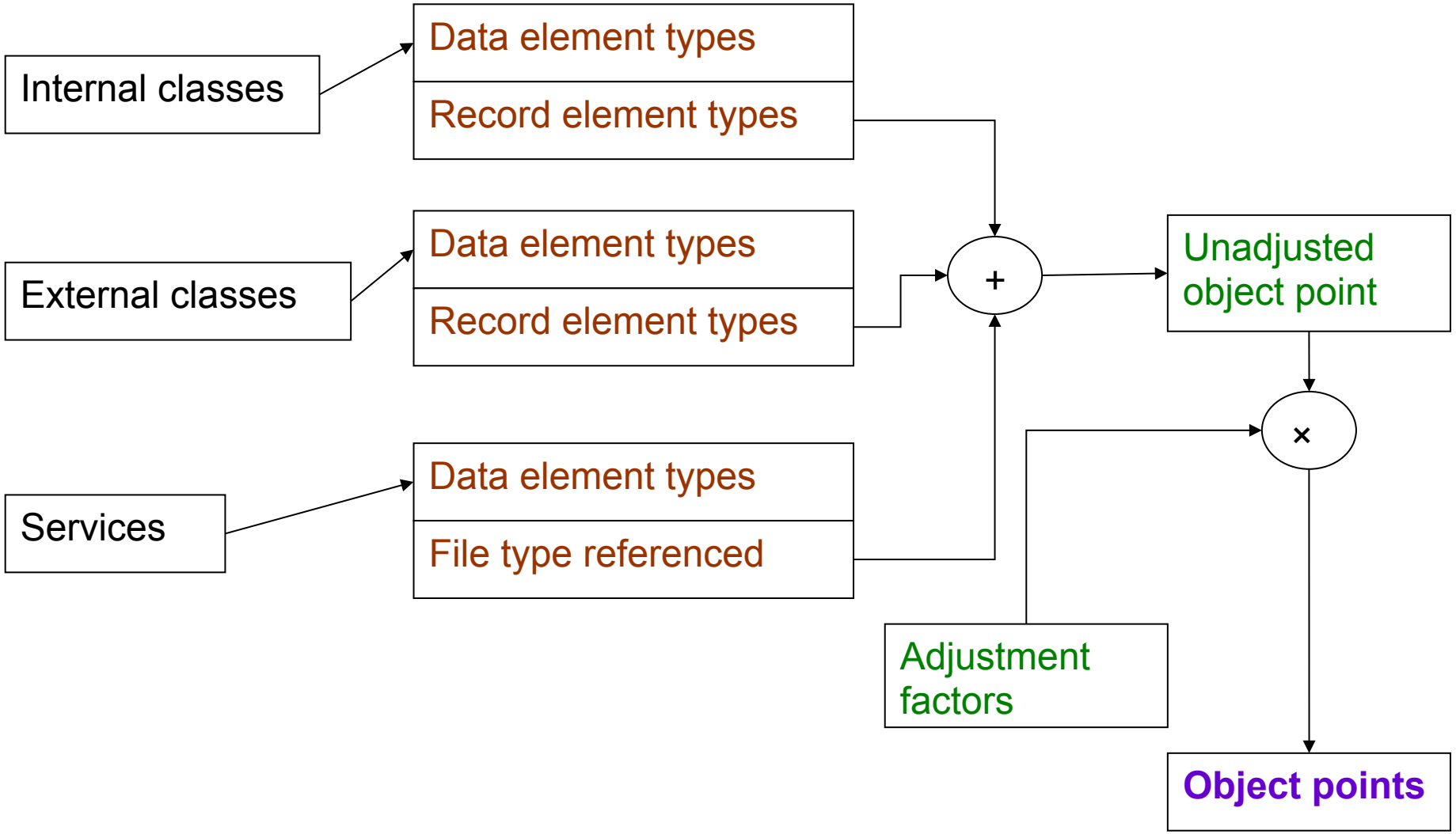
Object Oriented Function Point

- The traditional function point method can be mapped to obtain object points.
- The object point sizing method is also known as object oriented function point method as it is very similar to function point method.
- The object point method does not require much experience, requires less effort for computation and can be calculated quickly.
- It better suites for object oriented systems and is easier to calculate as compared to traditional function point method.

Object Oriented Function Point

- The object point method involves counting of classes and methods (services).
- All this information is obtained from object model of object oriented design.
- In this approach the function point concepts are mapped to object oriented concepts and the ambiguities present in the function point method are removed.

Object Oriented Function Point

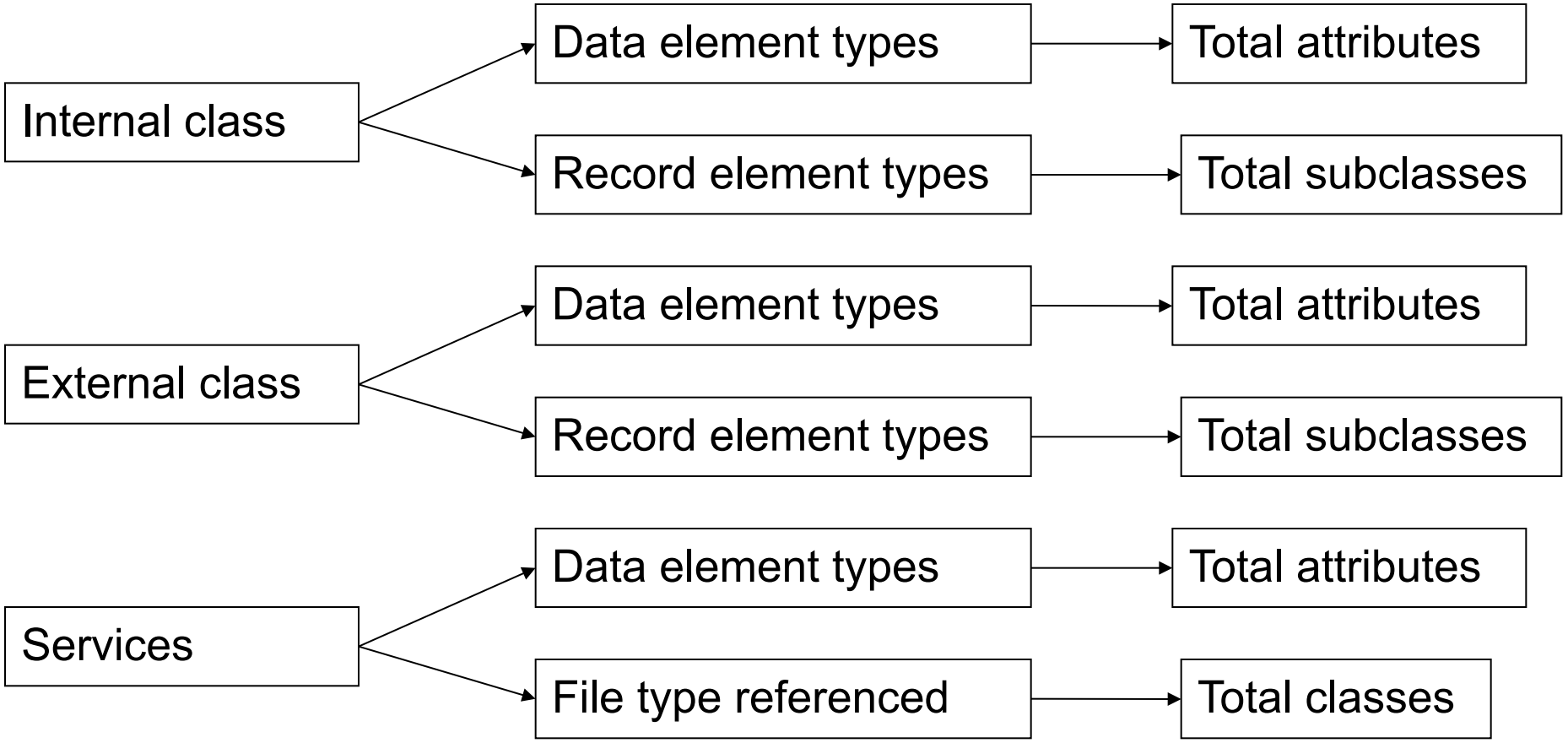


Object Oriented Function Point

- **Relationship between Function Points and Object Points**

Function point concept	Object point concept
Internal logical files	Internal class
External interface files	External class
External inquires	Services (methods)
External outputs	
External inputs	

Object Oriented Function Point



Object Oriented Function Point

- For each external/internal class it is necessary to compute the number of Data Element Types (DETs) and Record Element Types (RETs).
- The DETs correspond to total number of attributes of a class and RETs correspond to total number of subclasses of a class (descendants of a class).
- In inheritance hierarchy the classes that inherit the properties of base class while having their own properties are known as sub classes

Object Oriented Function Point

Counting Internal Classes, External Classes and Services

- An internal class identifies the total number of DETs and RETs in the class. In the IFPUG Manual 4.1, DETs are defined as [IFPU04]:
- *"a unique user recognizable, non repeatable field."*

Object Oriented Function Point

Counting Internal Classes, External Classes and Services

- *DETs are counted by applying the following rules:*
  - *A DET is counted for each simple attribute (integer, string, real) defined in a class. For example, the book accession number (integer type) stored in a class is counted as one DET.*
  - *A DET is counted for each attribute required to communicate with another internal class or external class (association or aggregation relationship; see details in chapter 5).*

Object Oriented Function Point

Counting Internal Classes, External Classes and Services

- *In the IFPUG Manual 4.1, RET is defined as:*
  - *"a unique recognizable subgroups of data elements within an internal logical file or external interface file."*
  - In object oriented systems these subgroups are known as subclasses or descendants. RET is counted for each of a given class. For example if employee is a base class and salary, contract based or hourly employee are three subclasses of class employee, then the RET count for employee class is 3. One of the following rules is applicable to a class while counting RETs.
    - Count a RET for each subclass of the internal/external class.
    - If there are no subclasses, count internal/external classes as one RET.

Object Oriented Function Point

Counting Internal Classes, External Classes and Services

- Each service in the class is examined. For each service, the number of DETs and FTRs need to be counted. The counting of DETs and FTRs involve simple and complex data types referenced by the methods. A simple data type is a compiler defined data type and a complex data type is a user defined data type. The following are the counting rules for DETs and FTRs for each service:
  - A DET is counted for each simple data type referenced as arguments of the service or global variables referenced by the service.
  - A FTR is counted for each complex data type referenced as arguments of the service or returned by the service.

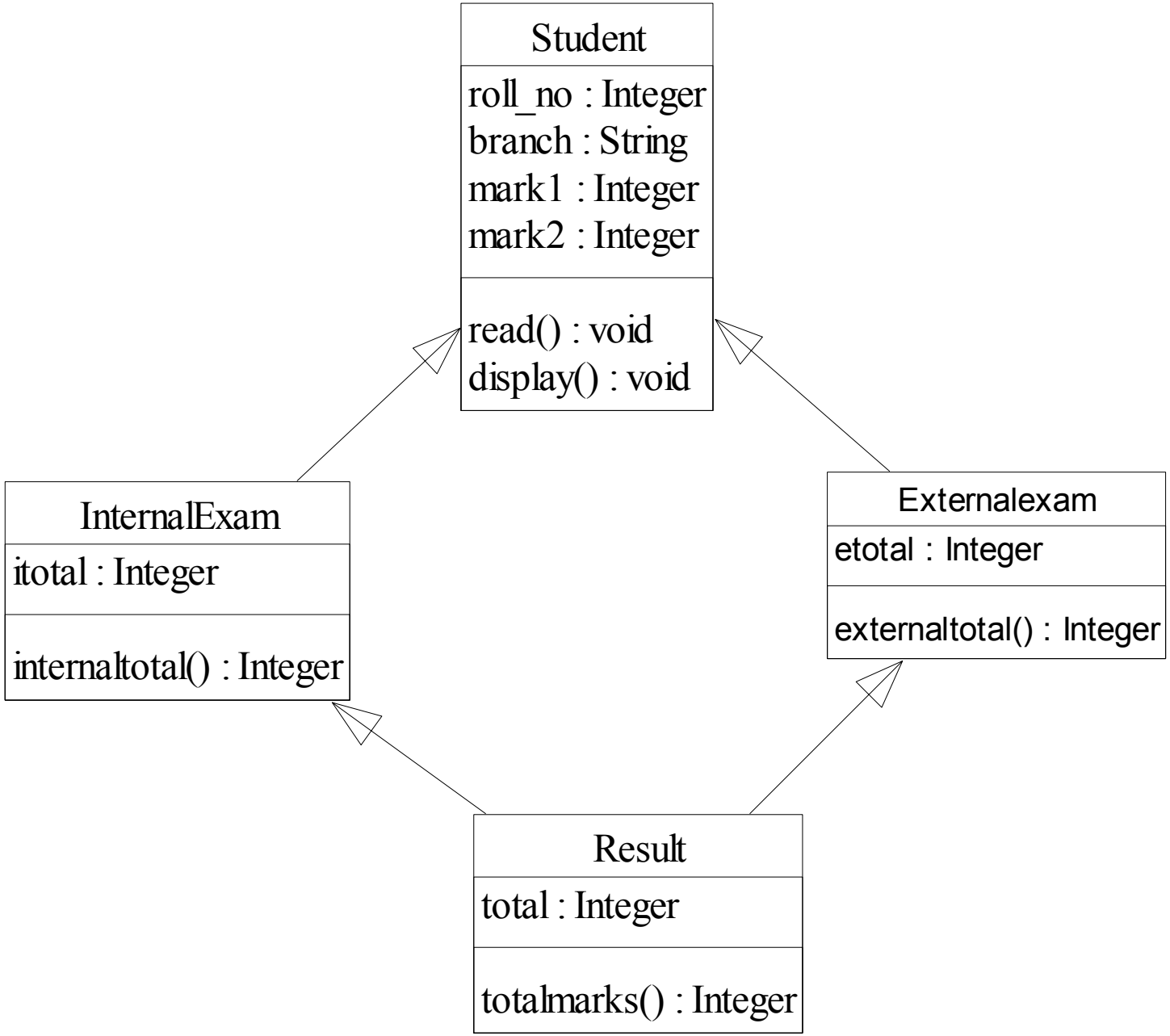
Object Oriented Function Point

	1 to 19 DETs	**20 to 50 DETs**	**51 or more DETs**
0 to 1 RETs	Low	Low	Average
2 to 5 RETs	Low	Average	High
6 or more RETs	Average	High	High

	1 to 19 DETs	**20 to 50 DETs**	**51 or more DETs**
0 to 1 RETs	Low	Low	Average
2 to 5 RETs	Low	Average	High
6 or more RETs	Average	High	High

Object Oriented Function Point

	1 to 4 DETs	5 to 15 DETs	16 or more DETs
0 to 1 FTRs	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High


Student
roll_no : Integer
branch : String
mark1 : Integer
mark2 : Integer
read() : void
display() : void
InternalExam
itotal : Integer
internaltotal() : Integer
Externalexam
etotal : Integer
externaltotal() : Integer
Result
total : Integer
totalmarks() : Integer

Object Oriented Function Point

Class	DET	RET	Complexity	Complexity value
Student	4	3	Low	7
Internal exam	1	1	Low	7
External exam	1	1	Low	7
Result	1	1	Low	7

Function type	Low	Average	High
Internal class (IC)	7	10	15
External class (EC)	5	7	10
Services (S)	3	4	6

Object Oriented Function Point

- After classifying the internal classes, external classes and services, these are multiplied by their complexity values. Finally the results are summed up using the following formulas:

$$IC_{total} = \sum_{i=1}^{n} IC_i$$

$$EC_{total} = \sum_{j=1}^{m} EC_i$$

$$S_{total} = \sum_{k=1}^{o} S_i$$

Object Oriented Function Point

- The classes in the given example are internal, hence the occurrences of internal classes are multiplied with their corresponding weights and summing all the resulting values, we get IC = 4×7=28. The concrete services in the classes are 5.
- Assuming that the methods have average complexity (as their signature is not given in the example) the services are 5 × 4 = 20.

Object Oriented Function Point

Calculating Unadjusted Object Points

- The Unadjusted Object Points (UOP) are obtained as follows:
  - UOP = ICtotal + ECtotal + Stotal
- In the example given in figure above, the UOP = ICtotal + Stotal = 28+20 = 48

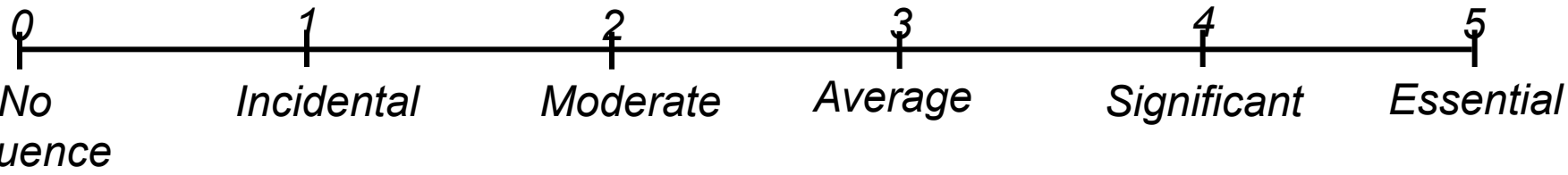
Object Oriented Function Point

- **Adjustment Factors**
  - The Fi (i=1 to 14) are the degree of influence and are based on responses to questions noted in figure shown in next slide. Each factor is rated on a scale of 0 to 5. The Adjustment Factor (AF) is calculated using the following mathematical formula:

$$AF = 0.65 + 0.01\sum_{i=1}^{14} F_i$$

  - Finally the adjusted object points are calculated by multiplying UOP by AF as follows:
    - OP=UOP × AF

Object Oriented Function Point



1. Does the system require reliable backup and recovery?
2. Is data communication required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing heavily utilized operational environment?
6. Does the system require on line data entry?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on line?
9. Is the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Object Oriented Function Point

- **Example 9**
- Consider a project with the following data:

Internal Class	**DET**	**RET**
C1	4	2
C2	5	7
C3	3	1
C4	2	1
External Class	**DET**	**RET**
C5	6	10

- The above classes contain 6 methods with high complexity and the complexities of the adjustments factors is average. Compute object points.

Object Oriented Function Point

- ICtotal = 3 × 7 + 1 × 10=31
- ECtotal = 1 × 7=7
- Stotal = 6 × 6=36
- UOP = ICtotal + ECtotal + Stotal
  - = 31+7+36=74
  - = 0.65 + 0.01 (14×3)
  - = 0.65 + 0.42 = 1.07
- OP=UOP × AI
  - = 74×1.07
  - = 79.18

Object Oriented Function Point

Example 10

- An application has the following:
  - 5 high internal classes, 2 average external classes and 6 average services. Assume the adjustment factor as significant. What are the unadjusted object points and object points?

Object Oriented Function Point

- **Solution**
- ICtotal = 5 × 15=75
- ECtotal = 2 × 7=14
- Stotal = 6 × 4=24
- UOP = ICtotal + ECtotal + Stotal
  - = 75+14+24=113
  - = 0.65 + 0.01 (14×4)
  - = 0.65 + 0.56 = 1.21
- OP=UOP × AI
  - = 113×1.21
  - = 136.73