COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

CHAPTER 8

REQUIREMENTS ENGINEERING

PROF. DR. KAMRUDDIN NUR

REQUIREMENTS ENGINEERING PHASES

- ❑ Inception
- ❑ Elicitation
- ❑ Analysis and Elaboration
- ❑ Negotiation
- ❑ Specification
- ❑ Validation
- ❑ Requirements Management

INCEPTION

- ❑ Inception—ask a set of questions that establish:
  - basic understanding of the problem
  - the people who want a solution (identify the stakeholder)
  - the nature of the solution that is desired
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
  - what will be the economic benefit of a successful solution

REQUIREMENTS ELICITATION

- Elicitation—elicit requirements from all stakeholders
- Interviewing related stakeholder with pre-determined questionnaire
- meetings are conducted and attended by both software engineers and customers
- Observation and ethnography
- a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used in collecting requirements
- the goal is:
  - to identify the problem
  - propose elements of the solution
  - specify a preliminary set of solution requirements

REQUIREMENTS ELABORATION

Building the Analysis Model:

- ❑ Scenario-based elements
  - Functional—processing narratives for software functions
  - Use-case—descriptions of the interaction between an "actor" and the system
- ❑ Class-based elements
  - Implied by scenarios
- ❑ Behavioral elements
  - State diagram
- ❑ Flow-oriented elements
  - Data flow diagram, Sequence diagram, Activity Diagram

REQUIREMENTS ANALYSIS

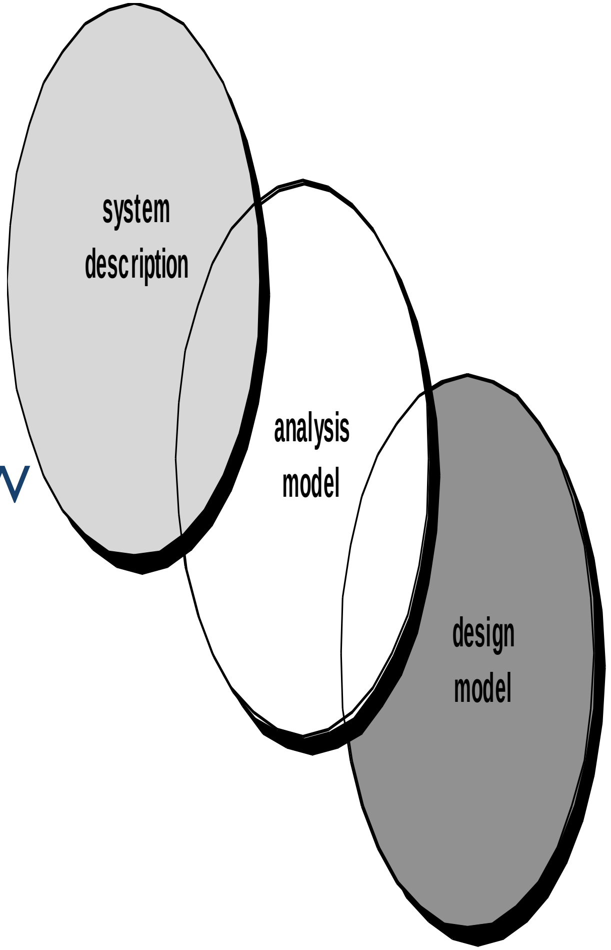
- ❑ Requirements analysis
  - specifies software’s operational characteristics
  - indicates software's interface with other system elements
  - establishes constraints that software must meet

- ❑ Requirements analysis allows the software engineer or requirements analyst to:
  - elaborate on basic requirements established during earlier requirement engineering tasks
  - build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

REQUIREMENTS ANALYSIS

Requirements Analysis Modeling

- Analysis models are build using requirements elicited from the customer
- Analysis modeling results in the first technical representation of the system
- Analysis modeling provides the developer and the customer with the means to access quality once S/W is built
- During modeling, the S/W Engineer should focus on WHAT rather than on HOW

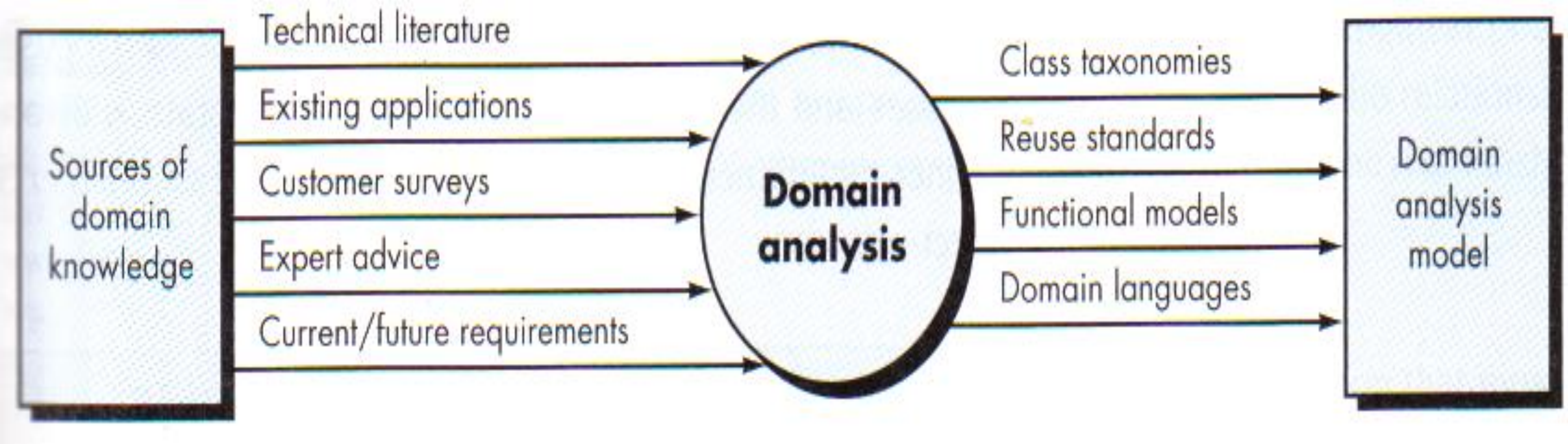


Requirements Analysis Modeling Objectives

- Describe what the Customer requires
- Establish a basis for the creation of a S/W design
- Define a set of requirements that can be validated once the software is built

DOMAIN ANALYSIS

- Define the domain to be investigated
- Collect a representative sample of applications in the domain
- Analyze each application in the sample
- Develop an analysis model for the objects

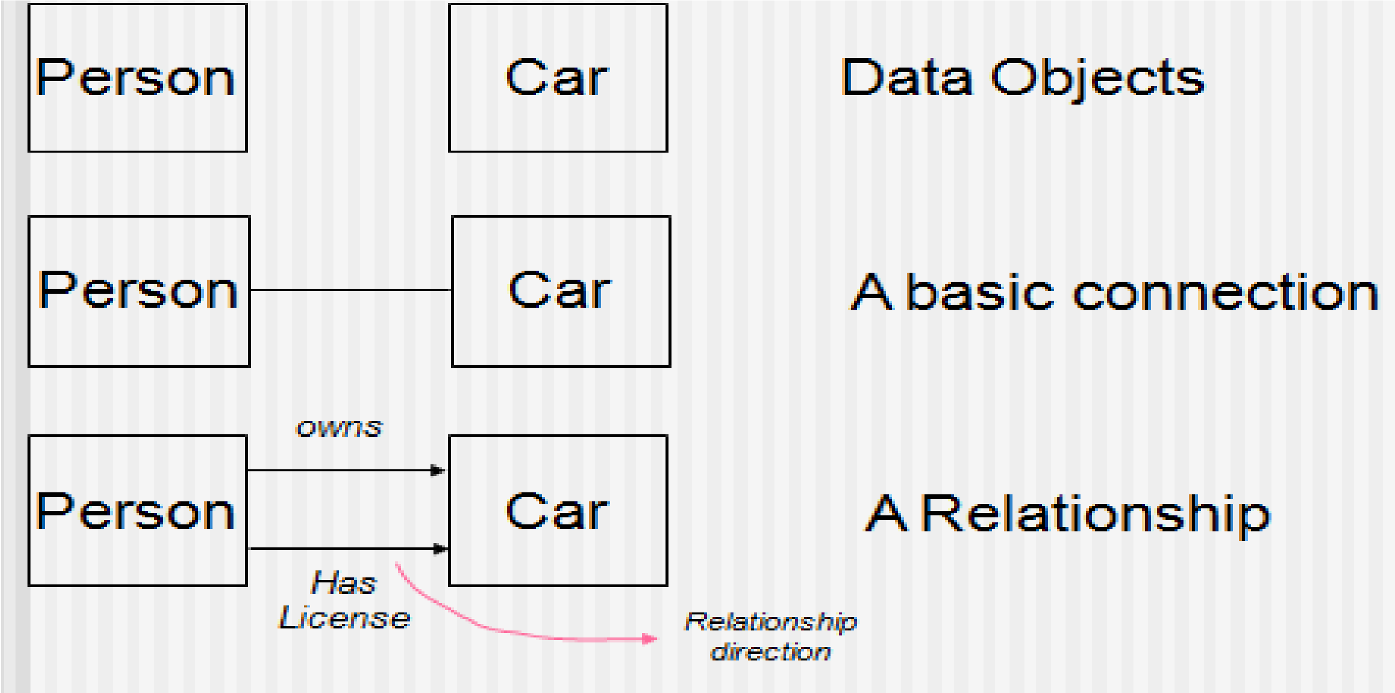


DATA MODELLING

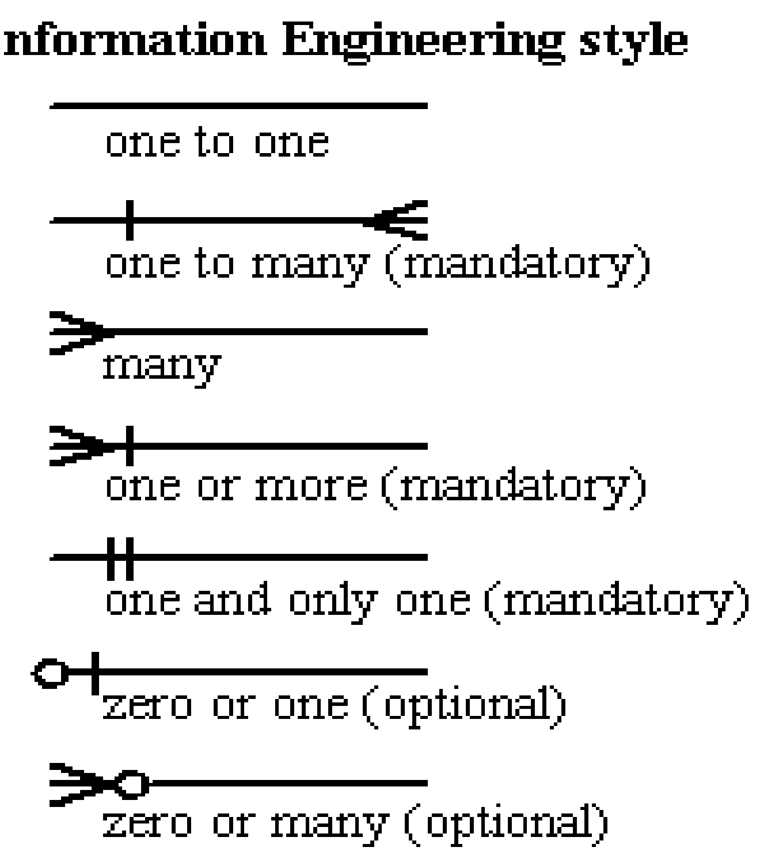
- Indicates how data objects relate to one another
- Data object is a representation of almost any composite information that must be understood by S/W. Composite information means number of different attributes and properties. *Length* or *Breadth* is not a Data Object, *Dimension* is a Data Object (as it is a composition of Length, Breadth & Height)
- *external entities* (e.g., printer, user, sensor)
- *Things* (e.g., reports, displays, signals)
- *occurrences or events* (e.g., interrupt, alarm)
- *roles* (e.g., manager, engineer, salesperson)
- *organizational units* (e.g., division, team)
- *Places* (e.g., manufacturing floor)
- *structures* (e.g., employee record)

DATA OBJECTS & RELATIONSHIPS

- Data objects are connected to one another in different ways.



CARDINALITY – ERD NOTATION



Information Engineering style

one to one

one to many (mandatory)

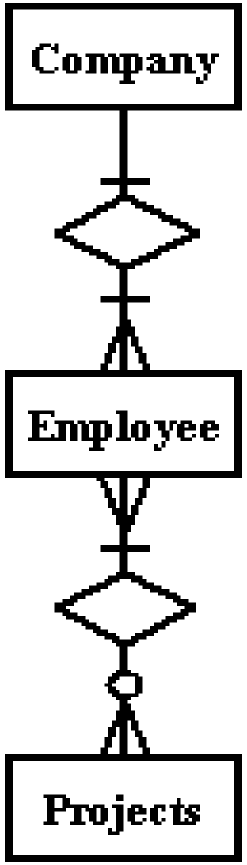
many

one or more (mandatory)

one and only one (mandatory)

zero or one (optional)

zero or many (optional)



CLASSES CATEGORIZATION

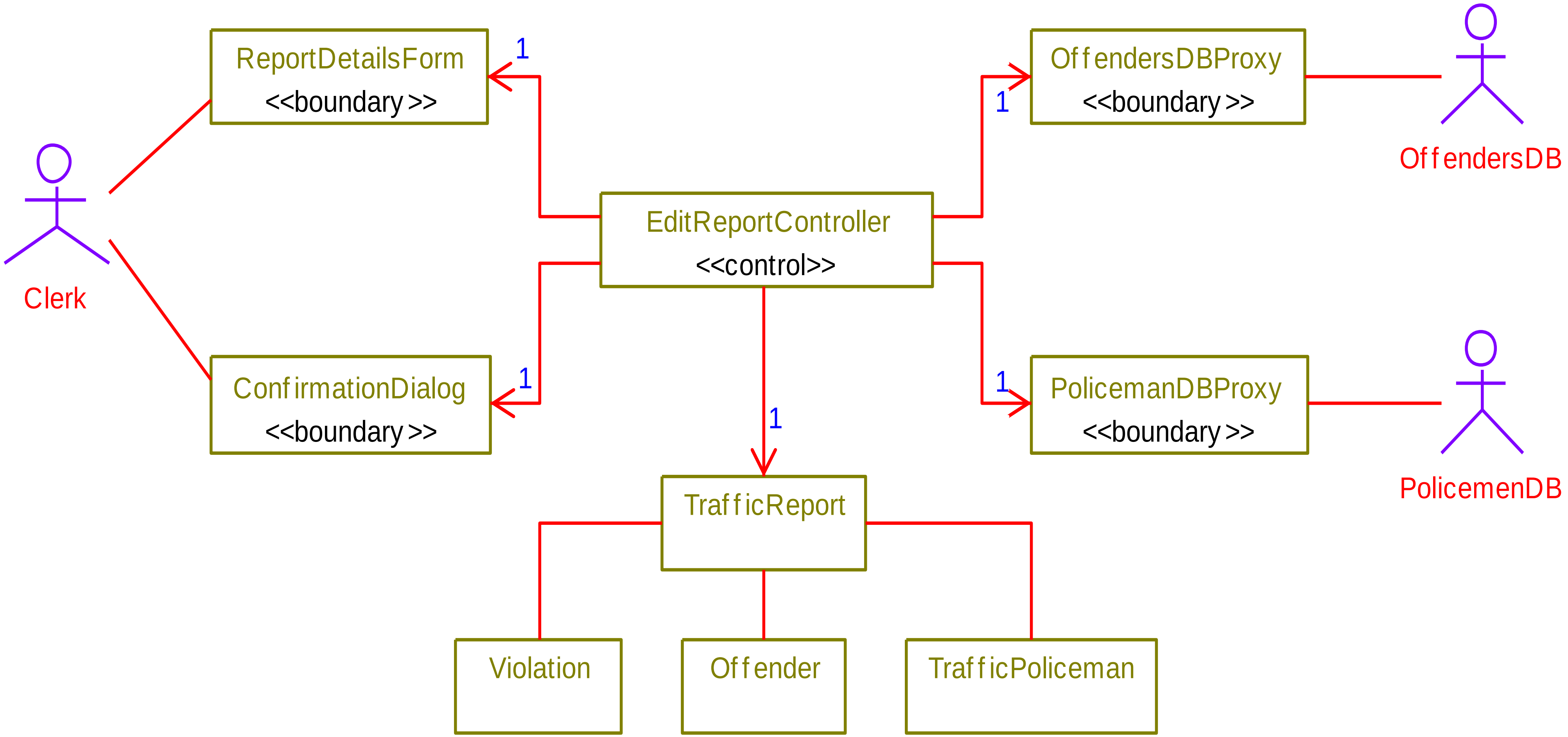
- ❑ Boundary Classes (UI)
  - Models the interaction between the system's surroundings and its inner workings
  - User interface classes, Concentrate on what information is presented to the user, don't concentrate on user interface details
  - System / Device interface classes, concentrate on what protocols must be defined. don't concentrate on how the protocols are implemented
- ❑ Entity Classes
  - Models the key concepts of the system
  - Usually models information that is persistent
  - Contains the logic that solves the system problem
  - Can be used in multiple behaviors

CLASSES CATEGORIZATION

❑ Control Classes

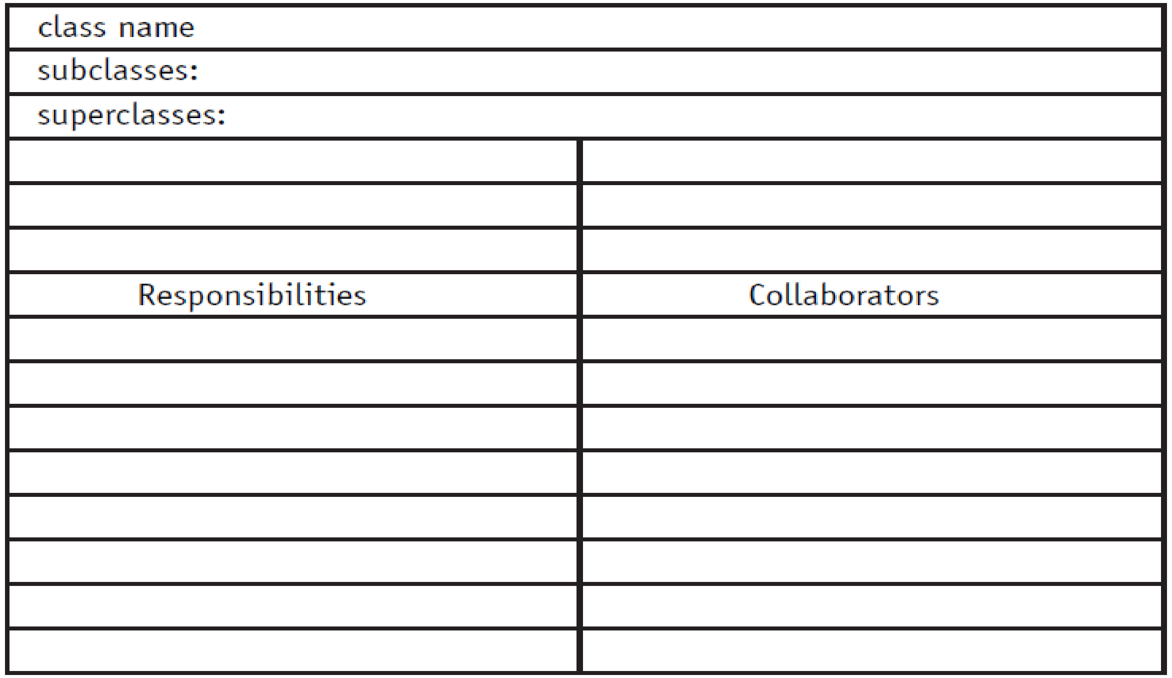
- Controls and coordinates the behavior of the system
- A control class should tell other classes to do something and should never do anything except for delegating (directing) the work to other classes
- Control classes separate boundary and entity classes

CLASSES CATEGORIZATION



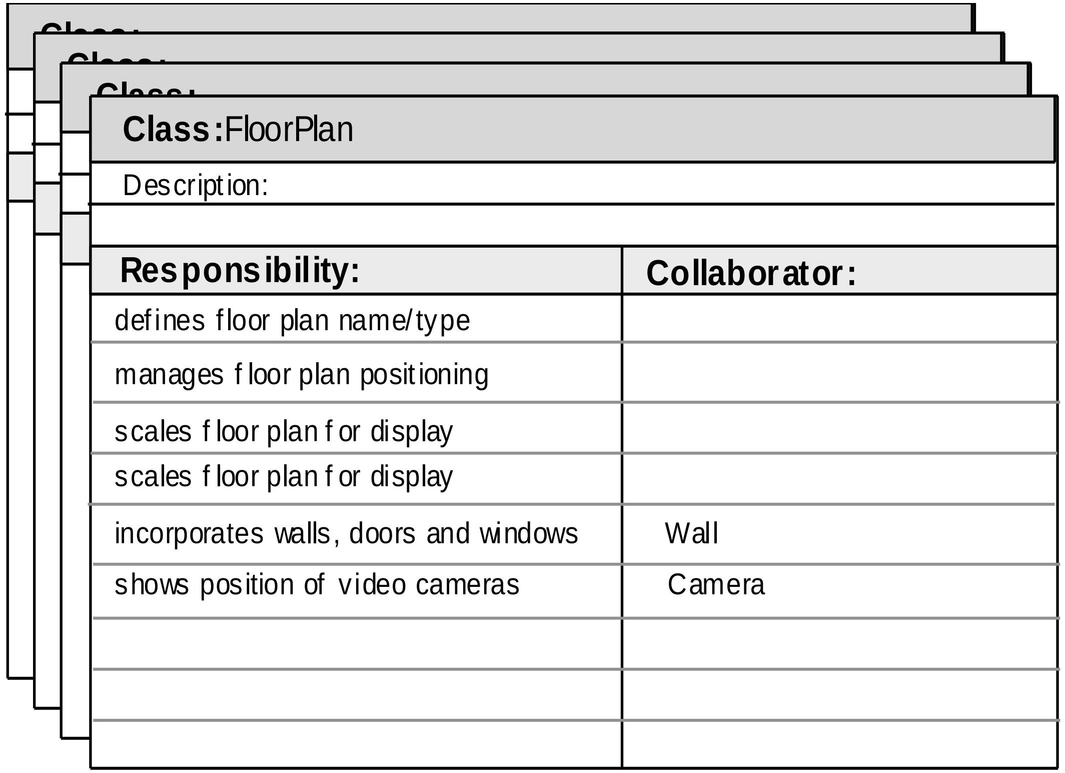
CRC CARD

- **C**lass **R**esponsibility **C**ollaboration
- CRC goals: provide the simplest possible conceptual introduction to OO design



class name	
subclasses:	
superclasses:	
Responsibilities	Collaborators

Figure 2-2 *A CRC card sample*



Class: FloorPlan

Description:

Responsibility:	Collaborator:
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

CRC CARD

- ❑ A CRC card is a 3-x-5" or 4-x-6" lined index card.
- ❑ The physical nature of the cards **emphasizes** the division of responsibility across objects.
- ❑ The physical size of the cards also **helps to establish limits** for the size and complexity of the classes.
- ❑ The CRC card technique does not use the UML, instead it is used to discover information about classes that is then placed into a UML Class diagram.
- ❑ The body of the card is divided in half.
  - The left column/half lists the responsibilities of the class
  - The right column/half lists the other objects that it works with, the collaborators, to fulfill each responsibility.

REQUIREMENTS NEGOTIATION

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders "win conditions"
  - Win conditions are not always obvious
- Negotiate/Prioritization
  - Work toward a set of requirements that lead to "win-win"

REQUIREMENTS VALIDATION

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement unambiguous?
- Do any requirements conflict with other requirements?
- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

THE REQUIREMENTS BASELINE

- **A requirements baseline is a set of requirements that has been reviewed and agreed upon and serves as the basis for further development.**
- A meaningful baselining process gives all the major stakeholders confidence in the following ways:
  - Customer management or marketing is confident that the project scope won't explode out of control, because customers manage the scope change decisions.
  - User representatives have confidence that the development team will work with them to deliver the right solution, even if they didn't think of every requirement before construction began.
  - Development management has confidence because the development team has a business partner who will keep the project focused on achieving its objectives and will work with development to balance schedule, cost, functionality, and quality.
  - Business analysts and project managers are confident that they can manage changes to the project in a way that will keep chaos to a minimum.
  - Quality assurance and test teams can confidently develop their test scripts and be fully prepared for their project activities.

REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering:A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992).An analysis of defect densities found during software inspections.*Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K.,Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation.*IBM Systems Journal*, *33*(1), 182-214.