OBJECT ORIENTED ANALYSIS and DESIGN

Project Management & Inception & Analysis UNIT II

 U2.1

Learning Objectives

Project Management & Inception Phase

Analysis
- Introduction
- The requirements model
- The analysis model

UML: Use case Diagram, Class Diagram, Object Diagram, Activity Diagram, Sequence Diagram

 U2.2

Project Management & Inception phase

 U2.3

A lot of computer science people think:

"I don't want to code for the rest of my life. Maybe I would enjoy managing the project."

What do you think are some of the tasks you would be doing if you were a project manager?

Would you still code?

Would you miss coding ☺?

Is it important the project manager be able to code?

What is Project Management?

Project management encompasses all the activities needed to plan and execute a project:

- Deciding what needs to be done
- Estimating costs
- Ensuring there are suitable people to undertake the project
- Defining responsibilities
- Scheduling
- Making arrangements for the work
- *continued ...*

What is Project Management?

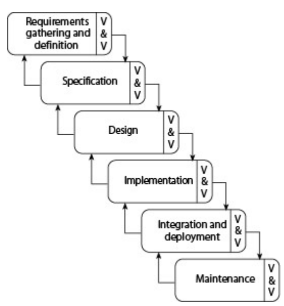
- Directing
- Being a technical leader
- Reviewing and approving decisions made by others
- Building morale and supporting staff
- Monitoring and controlling
- Co-ordinating the work with managers of other projects
- Reporting
- Continually striving to improve the process

Software Process Models

Software process models are general approaches for organizing a project into activities.

- Help the project manager and his or her team to decide:
  - ✓What work should be done;
  - ✓In what sequence to perform the work.
- The models should be seen as *aids to thinking*, not rigid prescriptions of the way to do things.
- Each project ends up with its own unique plan.

The waterfall model



The waterfall model

The classic way of looking at S.E. that accounts for the importance of requirements, design and quality assurance.

- The model suggests that software engineers should work in a series of stages.
- Before completing each stage, they should perform quality assurance (verification and validation).
- The waterfall model also recognizes, to a limited extent, that you sometimes have to step back to earlier stages.
- **QUESTION: What is wrong with getting all the requirements completed upfront (like I have done for you with our project)?**

Limitations of the waterfall model

- The model implies that you should attempt to complete a given stage before moving on to the next stage
  - ✓Does not account for the fact that requirements constantly change.
  - ✓It also means that customers can not use anything until the entire system is complete.
- The model makes no allowances for prototyping.
- It implies that you can get the requirements right by simply writing them down and reviewing them.
- The model implies that once the product is finished, everything else is maintenance.

Reengineering

Periodically project managers should set aside some time to re-engineer part or all of the system

- The extent of this work can vary considerably:
  - ✓Cleaning up the code to make it more readable.
  - ✓Completely replacing a layer.
  - ✓*Re-factoring* part of the design.
- In general, the objective of a re-engineering activity is to increase maintainability**.**

Cost estimation

To estimate how much software-engineering time will be required to do some work.

- *Elapsed time*
  - ✓The difference in time from the start date to the end date of a task or project.
- *Development effort*
  - ✓The amount of labour used in *person-months* or *person-days*.
  - ✓To convert an estimate of development effort to an amount of money:
    - You multiply it by the *weighted average cost* (*burdened* cost) of employing a software engineer for a month (or a day).

Question

Assume that I gave your group the task of figuring out how much time you needed to code your project because you were going sell it online.

What are some techniques/ideas/concerns/thoughts you have for estimating the timing of a large project?

i.e. How do you decide/figure out how long it takes you to do an assignment?

Principles of effective cost estimation

Principle 1: Divide and conquer.

- To make a better estimate, you should divide the project up into individual subsystems.
- Then divide each subsystem further into the activities that will be required to develop it.
- Next, you make a series of detailed estimates for each individual activity.
- And sum the results to arrive at the grand total estimate for the project.

Principles of effective cost estimation

Principle 2: Include all activities when making estimates.

- The time required for *all* development activities must be taken into account.
- Including:
  - Prototyping
  - Design
  - Inspecting
  - Testing
  - Debugging
  - Writing user documentation
  - Deployment.

Principles of effective cost estimation

Principle 3: Base your estimates on past experience combined with knowledge of the current project.

- If you are developing a project that has many similarities with a past project:
  - ✓ You can expect it to take a similar amount of work.
- Base your estimates on the *personal judgement* of your experts

  or
- Use *algorithmic models* developed in the software industry as a whole by analyzing a wide range of projects.
  - ✓ They take into account various aspects of a project's size and complexity, and provide formulas to compute anticipated cost.

Algorithmic models

Allow you to systematically estimate development effort.

- Based on an estimate of some other factor that you can measure, or that is easier to estimate:
  - ✓ The number of use cases
  - ✓ The number of distinct requirements
  - ✓ The number of classes in the domain model
  - ✓ The number of widgets in the prototype user interface
  - ✓ An estimate of the number of lines of code

Algorithmic models

- A typical algorithmic model uses a formula like the following:
  - ✓ COCOMO:

$$E = a + bN^c$$

  - ✓ Functions Points:

$$S = W_1F_1 + W_2F_2 + W_3F_3 + \dots$$

Principles of effective cost estimation

Principle 4: Be sure to account for *differences* when extrapolating from other projects.

- Different software developers
- Different development processes and maturity levels
- Different types of customers and users
- Different schedule demands
- Different technology
- Different technical complexity of the requirements
- Different domains
- Different levels of requirement stability

Principles of effective cost estimation

Principle 5: Anticipate the worst case and plan for contingencies.

- Develop the most critical use cases first
  - ✓ If the project runs into difficulty, then the critical features are more likely to have been completed
- Make three estimates:
  - ✓ Optimistic (O)
    - Imagining a everything going perfectly
  - ✓ Likely (L)
    - Allowing for typical things going wrong
  - ✓ Pessimistic
    - Accounting for everything that could go wrong

Principles of effective cost estimation

Principle 6: Combine multiple independent estimates.

- Use several different techniques and compare the results.
- If there are discrepancies, analyze your calculations to discover what factors causing the differences.
- Use the Delphi technique.
  - ✓ Several individuals initially make cost estimates in private.
  - ✓ They then share their estimates to discover the discrepancies.
  - ✓ Each individual repeatedly adjusts his or her estimates until a consensus is reached.

Principles of effective cost estimation

Principle 7: Revise and refine estimates as work progresses

- As you add detail.
- As the requirements change.
- As the risk management process uncovers problems.

Skills needed on a team

- Architect
- Project manager
- Configuration management and build specialist
- User interface specialist
- Technology specialist
- Hardware and third-party software specialist
- User documentation specialist
- Tester

Project Scheduling and Tracking

- *Scheduling* is the process of deciding:
  - ✓In what sequence a set of activities will be performed.
  - ✓When they should start and be completed.
- *Tracking* is the process of determining how well you are sticking to the cost estimate and schedule.

Some Basic Project Management Terminology

- *Deliverable:* some concrete thing which is to be delivered, to the client or internally to the development team; e.g.
  - Specifications reports
  - Executable program
  - Source code
- *Task/Activity*: something we have to do during the project; e.g.
  - Defining user requirements
  - Coding a module
  - Doing system testing
- Each task or activity will take some length of time
  - Referred to as ***duration*** of task
  - Sometimes measured in days, weeks, etc.
  - Sometimes measured in person-days, person-weeks, etc.
  - ***Person-day*** = number of people X number of days
    - ✓ Example: 12 person days for writing all code could mean 1 person 12 days or 4 people 3 days
    - ✓ **Note**: not always true that a task that takes 1 programmer 12 days would take 12 programmers 1 day

Dependencies and Milestones

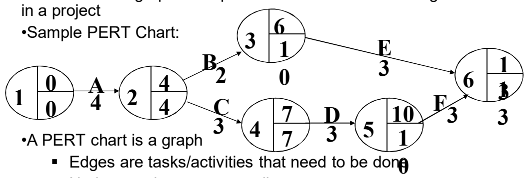
- For a given task or activity, may be impossible to start it without some other task(s) or activity(ies) having been completed; e.g.
  - Cannot start coding without completing design
  - Cannot start system testing without completing code integration and test plan
- If task B cannot start without A being completed, we say
  - B depends on A
  - There is a dependency between A and B
- *Milestone*: some achievement which must be made during the project; e.g.
  - Delivering some deliverable
  - Completing some task
- Note, delivering a deliverable may be a milestone, but not all milestones are associated with deliverables

Setting and Making Deadlines

- *Deadline* time by which milestone has to be met
  - Some deadlines are set by the client
  - Others are set by us on project to make sure project stays on track
- To set a deadline for completing task T, we must consider how long it will take to:
  - Complete the tasks that task T depends on
  - Complete task T itself
- If we miss a deadline, we say (euphemistically) "the deadline has slipped"
  - This is virtually inevitable
- Important tasks for project managers
  - Monitor whether past deadlines have slipped
  - Monitor whether future deadlines are going to slip
  - Allocate or reallocate resources to help make deadlines
- PERT chart and Gantt charts help project managers do these things (among others)

PERT Charts

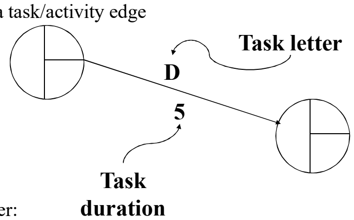
- PERT = Project Evaluation and Review Technique
- PERT chart = graphical representation of the scheduling of events in a project
- Sample PERT Chart:



- A PERT chart is a graph
  - Edges are tasks/activities that need to be done
  - Nodes are the events or milestones
- Task edge T from event node E1 to event node E2 signifies:
  - Until event E1 happens, task T cannot be started
  - Until task T finishes, event E2 cannot happen
- Events often simply represent completion of tasks associated with arrows entering it

PERT Chart Task Edges

- Parts of a task/activity edge



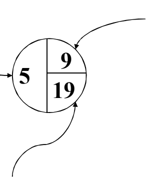
- Task letter:
  - Often keyed to a legend to tell which task it represents
- Task duration = how long (e.g. days, hours) task will take

PERT Chart Event Nodes

Event Number:

Sequence number assigned

Only task edges indicate dependencies



Earliest Completion Time (ECT):

Earliest time this event can be achieved, given durations and dependencies

Latest Completion Time (LCT):

Latest time that this event could be safely achieved

Building a PERT Chart

Steps:

1. Make a list of all project tasks (and events if possible).
2. Find interrelated task dependencies (what task has to be completed before other tasks)
3. Draw initial PERT without durations, ECTs or LCTs
4. Estimate duration of each task
5. Fill in durations
6. Calculate ECTs and LCTs

- **We will do this for an example system:**
  - **→ Generic software system with 3 modules**

Example: Generic Software Project

TASK ID	Task Description
A	Specification
B	High Level Design
C	Detailed Design
D	Code/Test Main module
E	Code/Test DB module
F	Code/Test UI module
G	Write test plan
H	Integrate/System Test
I	Write User Manual
J	Typeset User Manual

- **To start PERT chart: identify dependencies between tasks**

Dummy Tasks

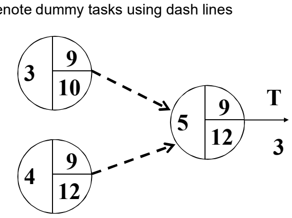
Sometimes it is necessary to use *dummy tasks*:

- Shows the dependency between 2 events where no activity is performed

Example:

- Events 3, 4 signify the compilation of separate modules.
- Create an event 5 to signify "all modules compiled together".

Denote dummy tasks using dash lines

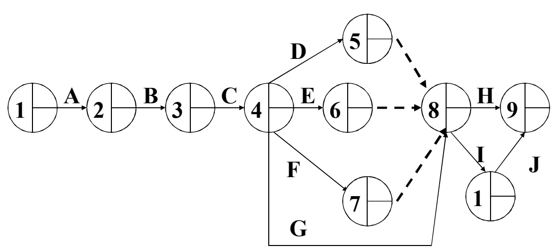


Example: Tasks with Dependencies

To start the PERT, identify the dependencies amongst tasks

TASK ID	Task Description	Preceed ID	Succ. ID
A	Specification	1	2
B	High Level Design	2	3
C	Detailed Design	3	4
D	Code/Test Main	4	5
E	Code/Test DB	4	6
F	Code/Test UI	4	7
G	Write test plan	4	8
	Dummy Task	5	8
	Dummy Task	6	8
	Dummy Task	7	8
H	Integrate/System Test	8	9
I	Write User Manual	8	10
J	Typeset User Manual	10	9

Software Example: Skeleton PERT Chart



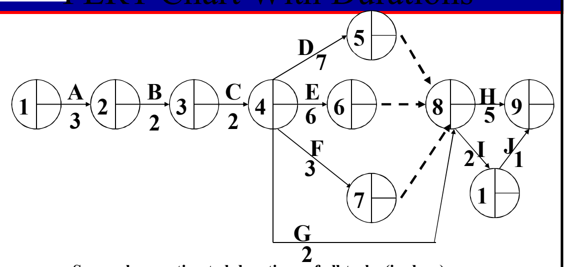
Note: dummy tasks connecting events 5, 6 and 7 to 8

Estimating Durations

Suggestions for estimating durations of tasks:
- Don't just make up a number
- Look at previous similar tasks from other projects and use those as guidelines
- Try to identify factors such as difficulty, skill level
  - ✓ Each weighting factor will help you make a better estimate

Factors to consider:
- Difficulty of task
- Size of team
- Experience of team
- Number, attitude and availability of end users
- Management commitment
- Other projects in progress

PERT Chart With Durations



- Say we have estimated durations of all tasks (in days)
- New PERT chart, with durations filled in:
- Note, dummy tasks (dashed lines) always have a duration of zero

Calculating ECTs

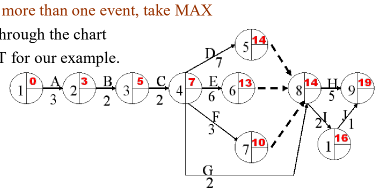
ECT = earliest time event can be completed

To calculate:

- For an event not depending on others: ECT = 0
  - ✓ Usually this is the first event
- For an event E depending on one or more others:
  - ✓ Calculate ECTs of event(s) that E depends on
  - ✓ Add duration(s) of task(s) leading to E
  - ✓ If E depends on more than one event, take MAX

Proceed left to right (→) through the chart

Exercise: calculate the ECT for our example.



Calculating LCT

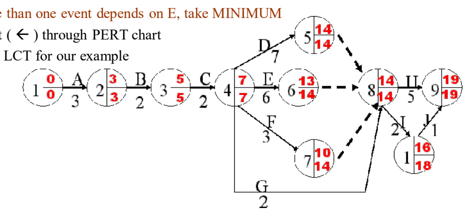
LCT = latest time event can be completed, while still finishing last ask at indicated time

To calculate:

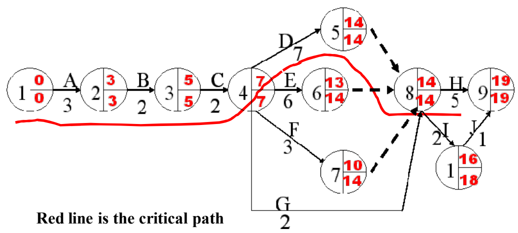
- For an event which no other events depend on: LCT = ECT
  - ✓ Generally there will only be one such event
- For an event E which one or more others depend on:
  - ✓ Calculate LCTs of event(s) that depend on E
  - ✓ Subtract duration(s) of task(s) leading from E
  - ✓ If more than one event depends on E, take MINIMUM

Proceed right to left (←) through PERT chart

Exercise: calculate LCT for our example



Critical Path



Red line is the critical path

What does it represent?

 U2.40

Uses of PERT Charts

We can use PERT charts for:

- Determining the estimated time to complete a project
- Deriving actual project dates
- Allocating resources
- Identifying potential and current problems (is one task behind schedule?, can we shuffle people?)

Critical Path: Path through chart such that if any deadline slips, the final deadline slips (where all events have ECT = LCT (usually there is only one)

In software example:

- Task I is not on the critical path: even if we don't finish it until time 18, we're still okay
- Task D is on the critical path: if we don't finish it until for example, time 16, then:
  - ✓ We can't start task H (duration 3) until time 16
  - ✓ So we can't complete task H until time 21

We can use PERT charts for

- Identifying the critical path
- Reallocating resources, e.g. from non-critical to critical tasks.

 U2.41

PERT Chart Exercise

Task	Prec Tasks	Description	Time(hrs)
A	none	decide on date for party	1
B	A	book bouncy castle	1
C	A	send invitations	4
D	C	receive replies	7
E	D	buy toys and balloons	1
F	D	buy food	3
G	E	blow up balloons	2
H	F	make food	1
I	H, G	decorate	1
J	B	get bouncy castle	1
K	J, I	have party	1
L	K	clean up	4
M	K	send back bouncy castle	1
N	L	send thank you letters	3
O	M	donate unwanted gifts	3

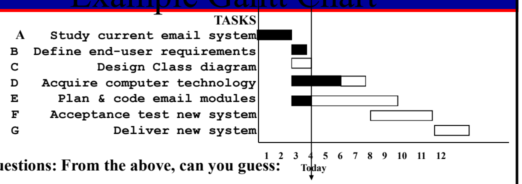
 U2.42

Gantt Charts

- •Graphical Representation of a schedule
- •Helps to plan, coordinate and track specific tasks in a project
- •Named after Henry Gantt who invented them in 1917
- •Depicts some of the same information as on a PERT chart
- •Also depicts new information

 U2.43

Example Gantt Chart



Questions: From the above, can you guess:

- **•Which, if any, tasks should have been completed by today and aren't even started? ______**
- **•Which, if any, tasks have been completed? ______**
- **•Which, if any, tasks have been completed ahead of schedule:? ______**
- **•Which, if any, tasks are on or ahead of schedule? _________**
- **•Which, if any, tasks are behind schedule? ________**

 U2.44

Building and Using a Gantt Chart

Steps for building a Gantt Chart

1. Identify the tasks to be scheduled
2. Determine the durations of each task
3. List each task down the vertical axis of chart
   1. In general, list tasks to be performed first at the top and then move downward as the tasks will happen
4. Use horizontal axis for the dates
5. Determine start and finish dates for activities
   1. Consider which tasks must be completed or partially completed before the next task

To use the Gantt chart to report progress:

- If the task has been completed, completely shade in the bar corresponding to the task
- If the task has been partially completed, shade in the percentage of the bar that represents the percentage of the task that has been completed
- Unshaded bars represents tasks that have not been started.

 U2.45

Gantt Chart: Exercise

Task	Prec Tasks	Description	Time(hrs)
A	none	decide on date for party	1
B	A	book bouncy castle	1
C	A	send invitations	4
D	C	receive replies	7
E	D	buy toys and balloons	1
F	D	buy food	3
G	E	blow up balloons	2
H	F	make food	1
I	H, G	decorate	1
J	B	get bouncy castle	1
K	J, I	have party	1
L	K	clean up	4
M	K	send back bouncy castle	1
N	L	send thank you letters	3
O	M	donate unwanted gifts	3

Gantt Chart: Exercise

Draw the Gantt chart using the following criteria:

- label hours 0 to 30 across the horizontal axis
- Mark a review stage at hour 14 to monitor the progress
- Assume and illustrate that tasks A, B, C and D have been completed at hour 14
- State which tasks are ahead and which tasks are behind schedule

- NOTE: if you are using MS Project and want a different unit of time, just type 2 hours (instead of 2 days). ALSO, if you want to have a milestone, like Handing in Group Assignment 1, then give it a ZERO duration.

PERT vs. Gantt

PERT chart
- All us to show dependencies explicitly
- Allow us to calculate critical path
- Can tell us how one task falling behind affects other tasks

Gantt charts
- Allow us to record progress of project
- Allow us to see what tasks are falling behind
- Allow us to represent overlapping tasks

Project Management Tools, e.g. MS Project
- Allow us to specify tasks, dependencies, etc
- Allow us to specify progress on tasks, etc
- Can generate either PERT or Gantt charts (whichever we want) from data entered

Difficulties and Risks in Project Management

- **Accurately estimating costs is a constant challenge**
  - ✓ *Follow the cost estimation guidelines.*
- **It is very difficult to measure progress and meet deadlines**
  - ✓ *Improve your cost estimation skills so as to account for the kinds of problems that may occur.*
  - ✓ *Develop a closer relationship with other members of the team.*
  - ✓ *Be realistic in initial requirements gathering, and follow an iterative approach.*
  - ✓ *Use earned value charts to monitor progress.*

 U2.49

Difficulties and Risks in Project Management

- **It is difficult to deal with lack of human resources or technology needed to successfully run a project**
  - ✓ *When determining the requirements and the project plan, take into consideration the resources available.*
  - ✓ *If you cannot find skilled people or suitable technology then you must limit the scope of your project.*

 U2.50

Difficulties and Risks in Project Management

- **Communicating effectively in a large project is hard**
  - ✓ *Take courses in communication, both written and oral.*
  - ✓ *Learn how to run effective meetings.*
  - ✓ *Review what information everybody should have, and make sure they have it.*
  - ✓ *Make sure that project information is readily available.*
  - ✓ *Use 'groupware' technology to help people exchange the information they need to know*

 U2.51

Difficulties and Risks in Project Management

- **It is hard to obtain agreement and commitment from others**
  - *Take courses in negotiating skills and leadership.*
  - *Ensure that everybody understands*
    - *The position of everybody else.*
    - *The costs and benefits of each alternative.*
    - *The rationale behind any compromises.*
  - *Ensure that everybody's proposed responsibility is clearly expressed.*
  - *Listen to everybody's opinion, but take assertive action, when needed, to ensure progress occurs.*

 U2.52

Review

Draw a PERT Chart for the following activities:

Activity	Description	Predecessor	Estimated Time
A	Drive home	None	0.5
B	Wash Clothes	A	4.0
C	Pack	B	0.5
D	Go to bank	A	1.0
E	Pay bill	D	0.5
F	Pack car	C,E	0.5
G	Drive to bus	F	0.5

 U2.53

OOSE Analysis Models

- Object-oriented software engineering (OOSE) proposes two analysis models for understanding the **problem domain**
  - Requirements Model
  - Analysis Model

The **requirements model** serves two main **purposes**
- To delimit the **system**
- To define the **system functionality**

 U2.54

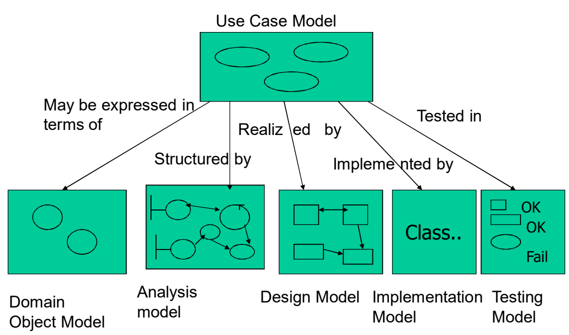
Requirement Model

- **Conceptual model** of the system is developed using:
  - **Problem domain objects**
  - **Specific interface descriptions** of the system (if meaningful to the system being developed)

⊗ The system is described as a *number of use cases* that are performed by a *number of actors*

- **Actors** constitute the **entities** in the **environment** of the system
- **Use cases** describe what takes places **within the system**
- A use case is a **specific way** of **using** the system by performing **some part** of the system functionality

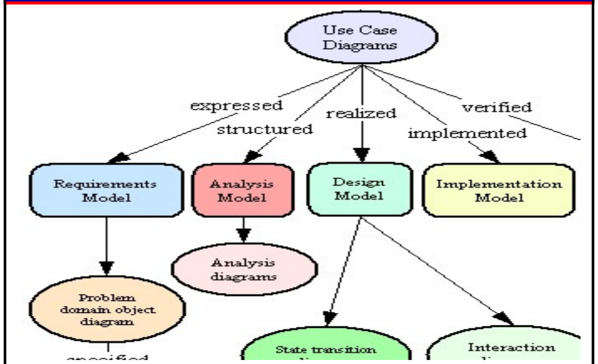
Requirement Model



Requirement Model

- The requirements model for the will comprise three main models of representation:
  - The use case model
  - The problem domain model
  - User interface descriptions

Requirement Model



Actors

- In order to identify use cases to be performed in the system, we need to first **identify system users**
- The **system users** are referred to as **actors**.
- Actors model the **prospective** '***users***' of the system.
- An actor is a **user type** or **category**. When an actor does something, the actor acts as an **occurrence** of that **type**.
- An actor may represent a **person** or **another system interacting** with the **intended system**

Actors and Role Play

- One person can **instantiate** (play the roles of) several different actors
- Actors **define** the **roles** that **users** can play
- Actors model anything that needs to **exchange information** with the **system**.
- Actors can **model human users** but they can also model other systems communicating with the intended system

Identifying Actors

- Actors constitute anything **external** to the system
- Identifying all the relevant actors for a system may require several **iterations**
- **General guidelines** include the following:
  - Ask yourself why the system is been developed
  - Who are people the system is intended to help?
  - What other systems are likely to interface with new system?

Primary and Secondary Actors

- Actors who ***use the system directly*** (or in their daily work) are known as **Primary actors**
- Primary actors are **associated** with one or more of the **main tasks** of the system
- Primary actors **govern** the **system structure**. Thus when ***identifying use cases***, we first start with the primary actors
- Actors who are concerned with **supervising** and **maintaining** the system are called **secondary actors**
- The **distinction** between the primary and secondary actors has a *bearing on the system structuring*

Use Cases

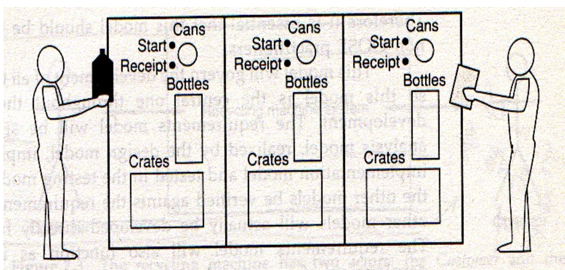
- After the actors have been identified the next step is to ***define the functionality of the system***. This is done by **specifying use cases.**
- Actors are a **major tool** in finding use cases. Each actor will perform a number of **use cases** in the system.
- Each use case constitutes a *complete course of events* initiated by an actor and specifies the **interaction** that takes place between the **actor** and the **system**
- A use case is a **special sequence** of **related transactions** performed by an **actor** and the **system** in dialogue.
- The *collective use cases* should specify all the **existing ways** of **using the system**

Recycling Machine Example

Throughout the discussion of the analysis and the construction activities, we will show how the different concepts are used in practice, by developing a system. The system controls a recycling machine for returnable bottles, cans and crates (used in Europe to hold several bottles). The machine can be used by several customers at the same time, and each customer can return all three types of items on the same occasion,

Since there may be different types and sizes of bottles and cans, the system has to check, for each item, what type was turned in. The system will register how many items each customer returns, and when the customer asks for a receipt, the system will print out what he turned in, the value of the returned items and the total return sum that will be paid to the customer.

Recycling Machine Example

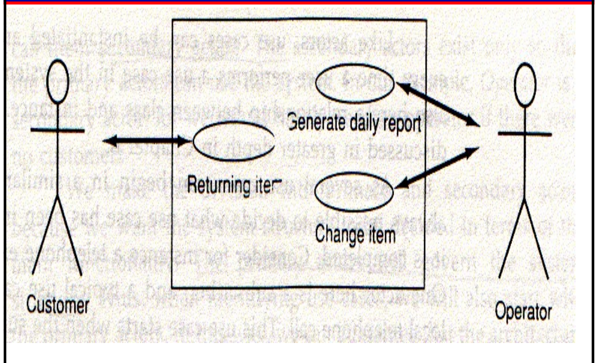


The recycling machine can receive several different types of returnable items from several customers at the same time.

Recycling Machine Example

The system is also used by an operator. The operator wants to know how many items of each type have been turned in during the day. At the end of the day, the operator asks for a printout of the total number of items that have been turned in to the system on that particular day. The operator should also be able to change information in the system, such as the deposit values of the items. If there is something amiss, for instance if a can gets stuck, or if the receipt roll is finished, the operator will be called by a special alarm signal.

Use Case Diagram



 U2.67

Use Case Description

Returning Item is started by *Customer* when he wants to return cans, bottles or crates. With each item that *Customer* places in the recycling machine, the system will increase the received number of items from *Customer* as well as the daily total of this particular type. When *Customer* has turned in all his items, he will press the receipt button to get a receipt where the returned items have been printed as well as a total return sum.

Generate Daily Report is started by *Operator* when he wants to print out information about the returned deposit items of the day. The system will print out how many of each deposit item type have been received this day, as well as the overall total for the day. The total number will be reset to zero to start a new daily report.

Change Item is used by *Operator* to change information in the system. The return value as well as the size of each returnable item can be changed, and new types of items can be added.

 U2.68

Case Study

Consider a **telephone exchange**:

- One actor is a subscriber and a typical use case to make a local telephone call. This use starts when the subscriber lifts his telephone receiver.
- Another use case is to order a wake-up call.
- Both use cases start when the subscriber lifts the telephone.
- However, when the subscriber lifts his telephone, its not obvious which use case he would like to perform.
- Thus uses cases may begin in a similar manner but we may not know which use case is to be carried out until its over.
- The actor should be viewed as someone who initiates a course of events that eventually results in a complete use case. Rather than someone who demands that a use case be performed.

 U2.69

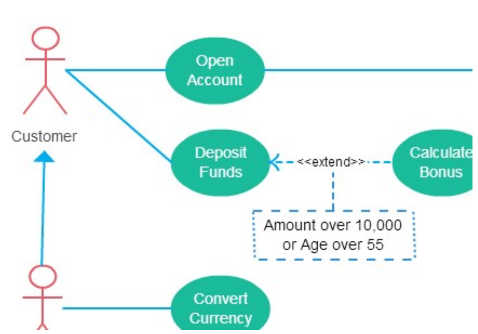
Use Case Relationships- Include

- **Use case include** is a directed relationship between two **use cases** which is used to show that behaviour of the **included** use case (the addition) is inserted into the **behaviour** of the **including** (the base) use case.
- The **include** relationship could be used:
  - To **simplify large use case** by splitting it into several use cases,
  - To **extract common parts** of the behaviours of two or more use cases.
- A large use case could have some behaviours which might be **detached** into distinct smaller use cases to be **included back** into the base use case using the UML **include** relationship.
- The **purpose** of this action is **modularization of behaviours**, making them more **manageable**.

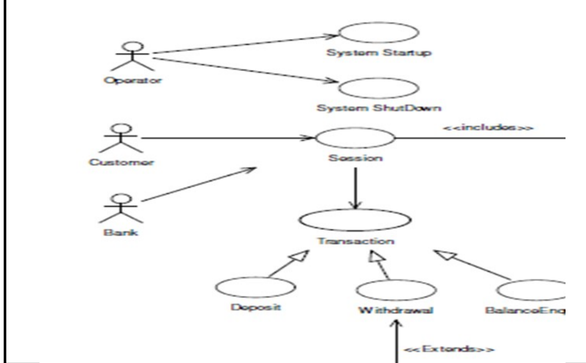
Use Case Relationships- Extend

- **Extend** is a **directed relationship** that specifies how and when the behaviour defined in usually supplementary (optional) **extending use case** can be inserted into the **behaviour** defined in the **extended use case**.
- **Extended** use case is meaningful on its own, it is **independent** of the extending use case.
- **Extending** use case typically defines **optional behaviour** that is not necessarily meaningful by itself.
- The extension takes place at one or more **extension points** defined in the **extended use case**.
- **Extend** relationship is shown as a dashed line with an open arrowhead directed from the **extending use case** to the **extended (base) use case**.

Use Case Relationships



Banking System Use Case Diagram



A Comparative Study

Generalization	Extend	
Bank ATM Transaction ◁— Withdraw Cash	Bank ATM Transaction ←«extend»--- Help	Bank ATM Transaction «i
Base use case could be **abstract use case** (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incom
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case requ
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion l

Requirement Engineering

Requirements

- Goal
  - To understand the problem
- Necessary to Understand Requirements
  - Organization
  - Existing Systems
  - Processes
  - Improvements
- Once you have all this information, now what?

Requirement Elicitation

✓ Techniques
- Interview / Meeting
- Survey / Questionnaire
- Observation
- Ethnography / Temporary Assignment
- Business Plans
- Review Internal / External Documents
- Review Software

Requirement Analysis

Requirements Analysis

- Goal
  - To bridge the gap between the problem domain and the technical domain
- Tasks
  - Problem Recognition
  - Evaluation and synthesis
  - Modeling
  - Specification
  - Review

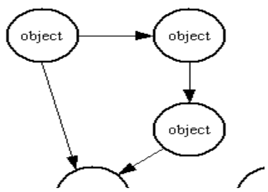
Requirement Analysis Principles

Requirements Analysis Principles

- Information domain of a problem must be represented and understood
- Models that depict system information, function, and behavior should be developed
- Models must be partitioned in a manner that uncovers detail in a layered fashion
- Analysis process should move from essential information toward implementation detail

Problem Domain Object Model

Provides a logical view of the system, which is used to specify the use cases for use case diagrams

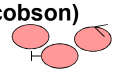


Object Oriented Analysis

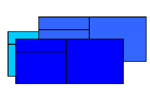
- **Identifying objects**: Using concepts, CRC cards, stereotypes, etc.
- **Organising the objects**: classifying the objects identified, so similar objects can later be defined in the same class.
- **Identifying relationships between objects**: this helps to determine inputs and outputs of an object.
- **Defining operations of the objects**: the way of processing data within an object.
- **Defining objects internally**: information held within the objects.

Object Oriented Analysis Approaches

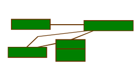
1. Analysis model with stereotypes (Jacobson)
   - **Boundaries, entities, control.**



2. CRC cards (Beck, Cunningham)
   - **Index cards and role playing.**



3. Conceptual model (Larman)
   - **Produce a "light" class diagram.**



A good analyst knows more than one strategy and even may mix strategies

The Analysis Phase

- Begins with a **problem statements** generated during **system conception.**
- In software engineering, analysis is the process of **converting** the user requirements to system specification (system means the software to be developed).
- System specification, also known as the **logic structure**, is the developer's view of the system.
- **Function-oriented analysis**
  - Concentrating on the **decomposition** of complex functions to simply ones.
- **Object-oriented analysis**
  - Identifying **objects** and the **relationship** between objects.

 U2.82

Analysis Model

⊗ The analysis model gives a **conceptual configuration** of the system.

It consists of:

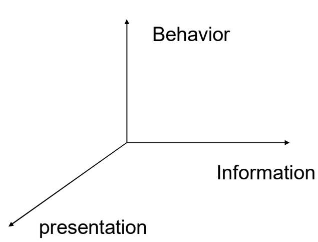
- The entity objects
- Control objects
- Interface objects

⊗ The analysis model forms the **initial transition** to **object-oriented design**

 U2.83

Dimensions of Analysis Model

Dimensions of the analysis model



 U2.84

Analysis Model- *Objects*

Entity object
- Information about an entity object is stored even after a use case is completed.

Control object
- A control object shows functionality that is not contained in any other object in the system

Interface object
- Interface objects interact directly with the environment

 U2.85

Requirement Model Structured in Analysis Model

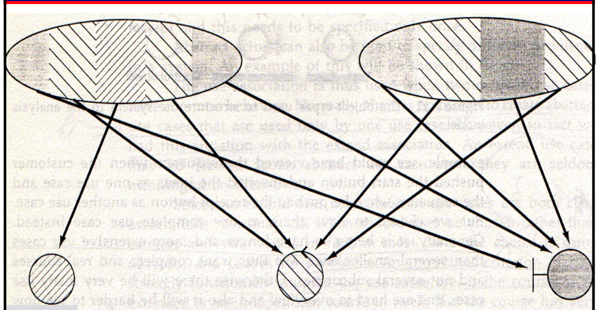


Figure 7.13 Each use case is distributed among analysis objects. Several use cases can have objects in common.

 U2.86

Design Model

- Developed based on the analysis model
  - Implementation environment is taken into consideration
- The considered **environment factors** includes
  - Platform
  - Language
  - DBMS
  - Constraints
  - Reusable Components
  - Libraries
  - so on..

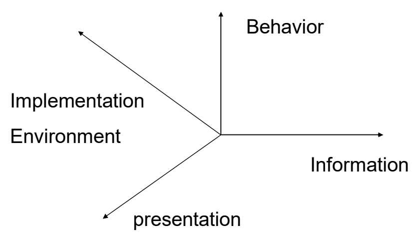
 U2.87

Design Model

- Design objects are different from analysis objects
- **Models**
  - Design object interactions
  - Design object interface
  - Design object semantics
    - ✓ **(i.e., algorithms of design objects' operations)**
- More closer to the **actual source code**

Design Model Dimensions

Dimensions of the Design model



Design Model

- Use **block term** in place of object
- Sent from one block to another to trigger an execution
- A typical **block** is **mapped** to **one file**
- To manage system **abstractly** subsystem concept is introduced
- **Analysis Model** is viewed as **conceptual** and **logical model,** whereas the **design model** should take as closer to the **actual source code**
- Consist of explained source code
- OO language is desirable since all fundamentals concepts can easily be mapped onto **language constructs**
- Strongly desirable to have an easy match between a **block** and the **actual code module**

Implementation Model

- Consists of **annotated source code.**
- Object oriented language is desirable since all **fundamental concepts** can be easily **mapped** onto **language constructs**.
- Strongly desirable to have an easy **match** between a **block** and the **actual code module.**

Test Model

Fundamental concepts are **test specifications** and the **test results**

Analysis

Learning Objectives

- The Analysis Phase
- Analysis Model
- Meta Model of Analysis Model
- Analysis workflow details
- Analysis model-rules of thumb
- Object Oriented Analysis
- Three ways to do Object Oriented Analysis
- Conceptual Model – Overview
- The Concept Category List
- Finding Concepts with Noun Phrase Identification

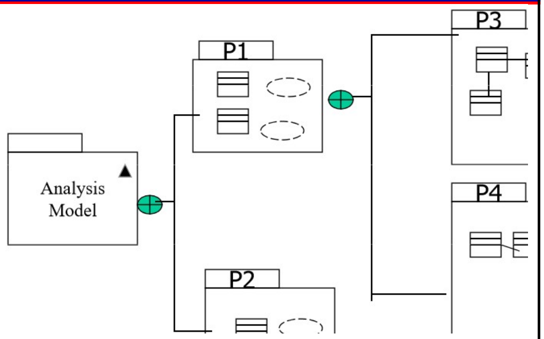
Learning Objectives

- Exercise
- How to make a conceptual model
- Drawing of Concepts
- Adding Associations
- Adding Attributes
- The Object Oriented Analysis Model (Jacobson)
- Subsystem
- Good Analysis class
- Bad Analysis class

Analysis Model

- The analysis model is intended to define a **first structure** of the system to be **designed** in the form of a **class** and/or an **object diagram**.
- The analysis model, gives the system its ***initial structure*** which is subject to further refinement in later development steps.
- According to the **Unified Process,** the development of the analysis model has to occur on the basis of the **use case specifications.**
- The analysis model has to be capable to fulfill the **functional requirements** stated in the **use case descriptions**.

Meta Model of Analysis Model

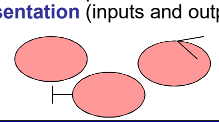


Analysis Model- Rule of Thumb

- 50 to 100 analysis classes in analysis model in an average system
- Include classes from the vocabulary of the problem domain
- Do not make implementation decisions
- Focus on classes and associations- minimize coupling
- Use inheritance whenever needed
- Keep it simple

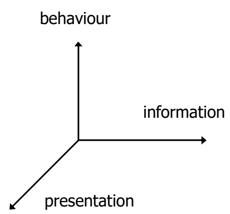
The Object Oriented Analysis Model (Jacobson)

- An **analysis model** is used to **represent** the **system specification.**
- To achieve **robustness** and **stability** the model must be **implementation environment independent.**
- Any change in the implementation environment will not affect the **logical structure** of the system.
- The model must be able to capture **information, behaviour** (operations) and **presentation** (inputs and outputs).



Behaviour - Information - Presentation

- The model is defined in information - behaviour - presentation **space**.

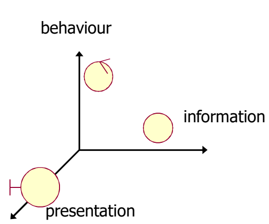


OO Analysis Model Syntax

- Within an use case, we employ three types of **objects**
- In Rational Rose, known as three types of **entities** or **stereotypes**

- Entity — **On information – behaviour plane and incline to information axis**
- Boundary / Interface — **On the presentation axis**
- Control — **On information – behaviour plane but incline towards behaviour axis**

Entity, Control, Interface



Pragmatics of OO Analysis Model

Identifying interface objects
- functions directly related to **actors**.

Identifying entity objects
- information used in an **use case** and functions of processing the **information**.

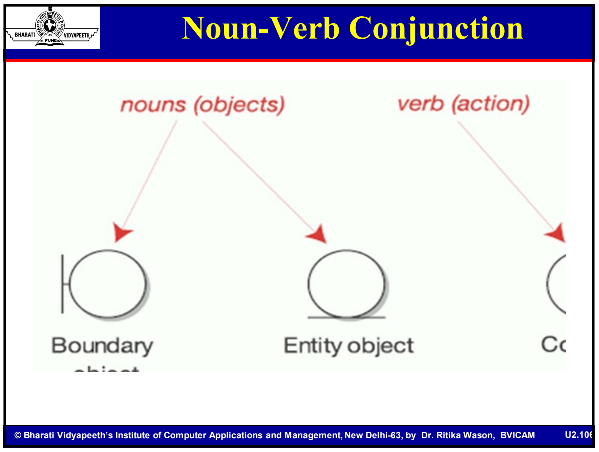
Identifying control objects
- functions that **link interface objects** and **entity objects**

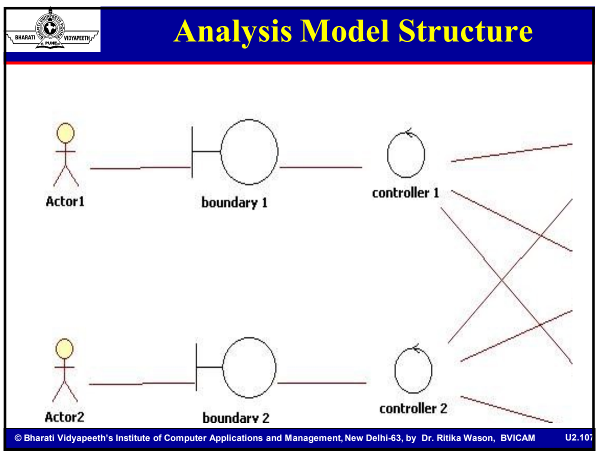
Semantics of OO Analysis Model

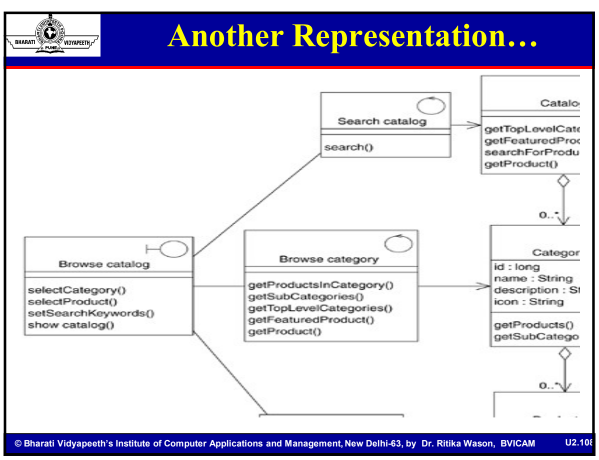
- An **entity object** models **information** that shows the state of a system.
  - ✓ This information is often used to **record** the *effects of operations*
  - ✓ Related to the **behaviour** of the **system**.
- A **boundary/interface object** models **inputs** and **outputs** and **operations** that process them.
- A **control object** models **functionality/operations** regarding to validate and decide whether to process and pass information from the interface object to the entity object or the way around.

Semantics of OO Analysis Model

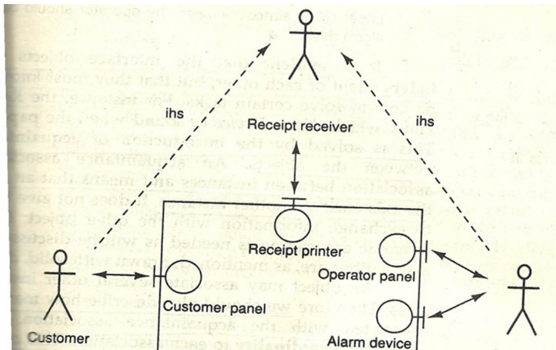
- **Interface objects:** **Mediate** the **communication** with the **actors**. They directly correspond to the actor/system interfaces.
- **Entity objects:** are objects to hold **information**. They often correspond to the **objects** in reality and can be found by ***conceptual domain modeling***.
- **Control objects:** are those objects which **coordinate** and **allocate work** between the different **objects** in fulfillment of a particular **use case.**


Noun-Verb Conjunction
nouns (objects)
verb (action)
Boundary
Entity object
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, BVICAM


Analysis Model Structure
Actor1
boundary 1
controller 1
Actor2
boundary 2
controller 2
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, BVICAM


Another Representation…
Search catalog
search()
Browse catalog
selectCategory()
selectProduct()
setSearchKeywords()
show catalog()
Browse category
getProductsInCategory()
getSubCategories()
getTopLevelCategories()
getFeaturedProduct()
getProduct()
id : long
name : String
icon : String
getProducts()
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, BVICAM

Recycling Machine Case Study



Recycling Machine Case Study

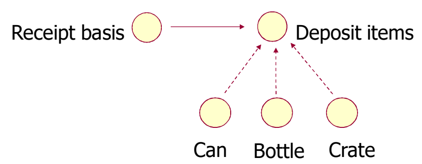
Identifying interface objects
- Printer, Customer Panel

Identifying entity objects
- Long term information: Crate, Bottle, Can
- Superclass: Deposit item
- Short term information: Receipt basis

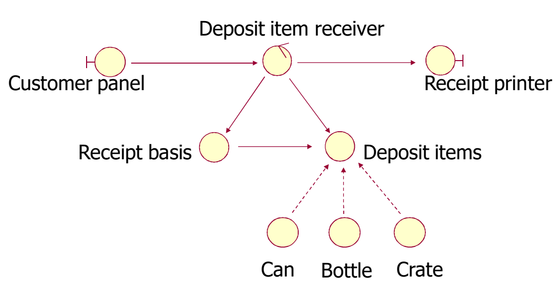
Identifying control objects
- Deposit item receiver

Recycling machine: Interface Objects

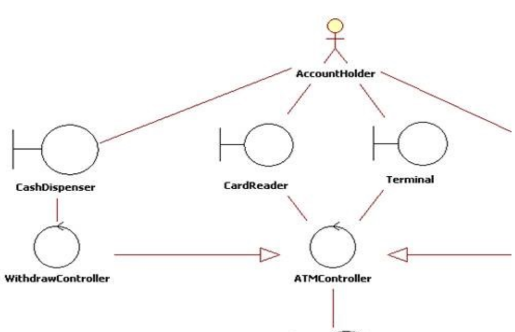
Recycling machine: Entity Objects



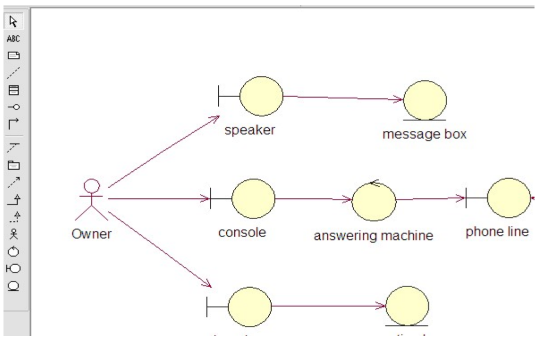
Link Interface and Entity Objects by a Control Object



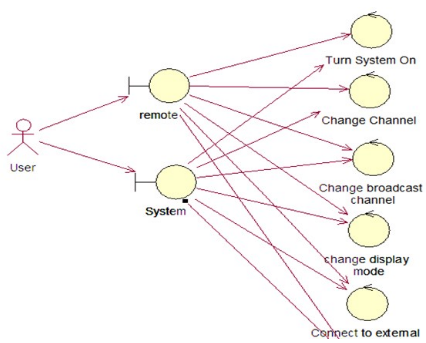
ATM Machine Analysis Model



Telephonic System Analysis Model



Television Control Analysis Model



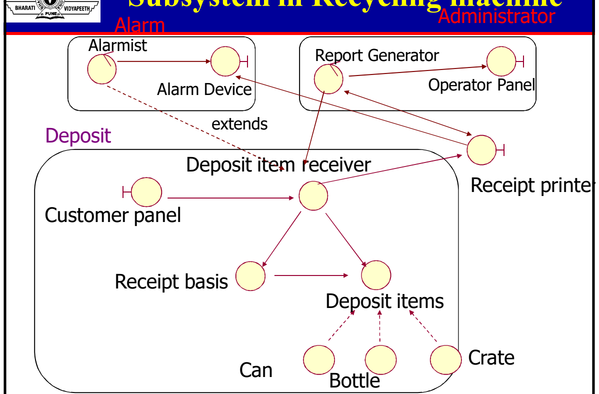
Subsystem in Analysis Model

- **Package** the **objects** so that **complexity** is **reduced**
- Lowest level is **service package**
- **Little communication** between different **subsystem** as possible

Subsystem in Analysis Model

- The aim is to have **strong functional coupling within** the **subsystem** and a **weak coupling between subsytem**
  - Whether two objects are strongly functionally?
  - Will changes in one object lead to changes in the other object?
  - Do they communicate with the same actor?
  - Are both of them dependent on a third object, such as an interface object or an entity object?
  - Does one object perform several operations on the other?

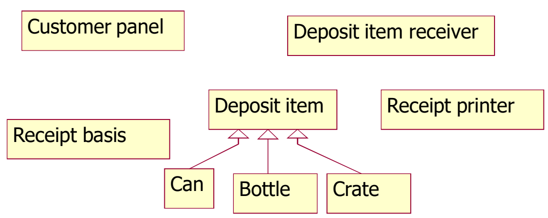
Subsystem in Recycling machine



Good Analysis Class Characteristic

- **Name** reflects its goal
- **Hard abstraction** that models one **specific element** of the problem domain
- **Maps** to a **clearly identifiable feature** of the problem domain
- Has a **small**, **well-defined** set of responsibilities

Drawing of Concepts

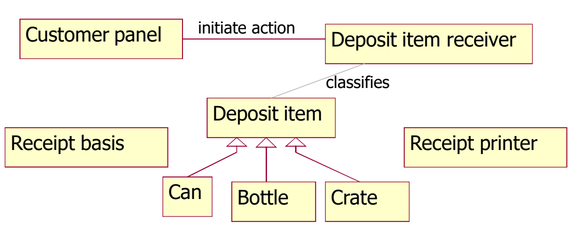


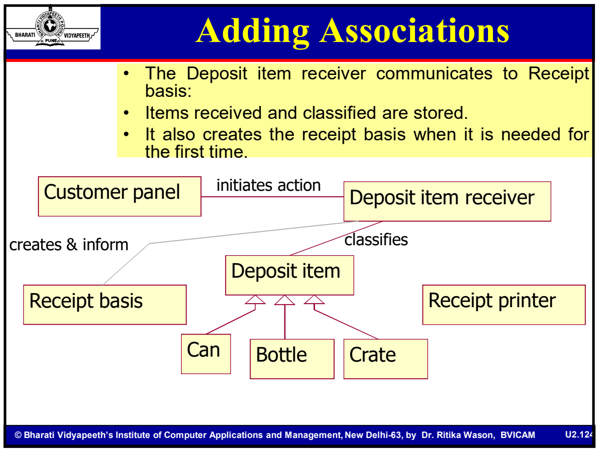
Adding Associations

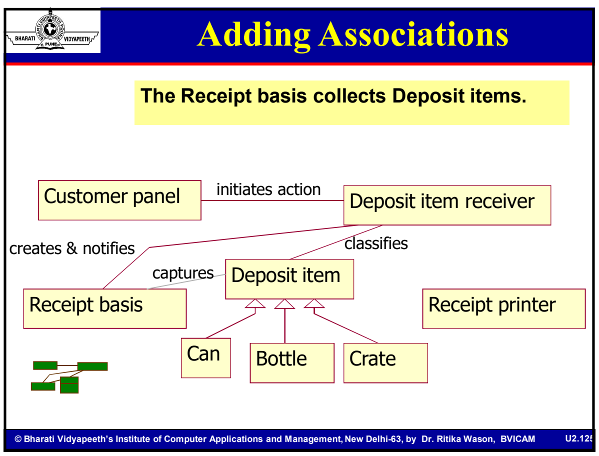
- If one concept needs to know of a another concept for some duration they should be linked by an association.
- Also use the Common Association list in order to identify associations.

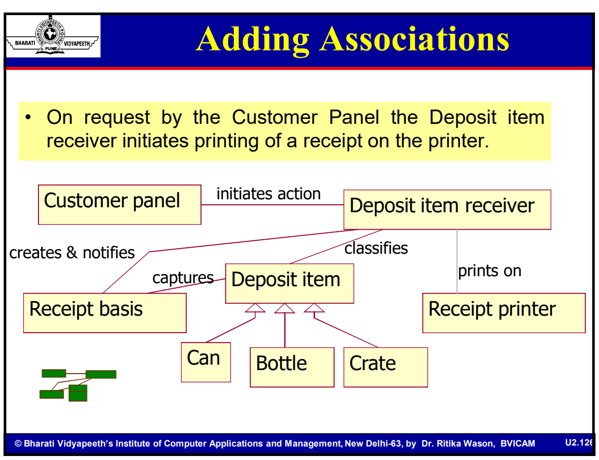
Adding Associations

The Deposit item receiver manages Deposit items:
The items are classified.



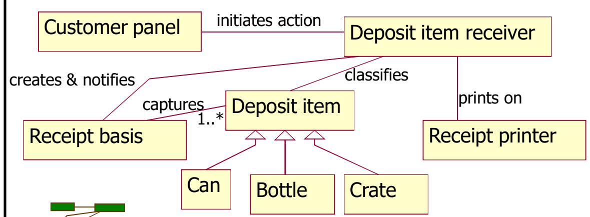

Adding Associations
• The Deposit item receiver communicates to Receipt basis:
• Items received and classified are stored.
• It also creates the receipt basis when it is needed for the first time.
Customer panel
initiates action
Deposit item receiver
creates & inform
classifies
Deposit item
Receipt basis
Receipt printer
Can
Bottle
Crate
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, BVICAM
U2.124


Adding Associations
The Receipt basis collects Deposit items.
Customer panel
initiates action
Deposit item receiver
creates & notifies
classifies
captures
Deposit item
Receipt basis
Receipt printer
Can
Bottle
Crate
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, BVICAM
U2.125


Adding Associations
• On request by the Customer Panel the Deposit item receiver initiates printing of a receipt on the printer.
Customer panel
initiates action
Deposit item receiver
creates & notifies
classifies
prints on
captures
Deposit item
Receipt basis
Receipt printer
Can
Bottle
Crate
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, BVICAM
U2.126

Adding Associations

- Adding multiplicities
- Only one association here is a 1 to many relationship
- All others are 1 to 1.

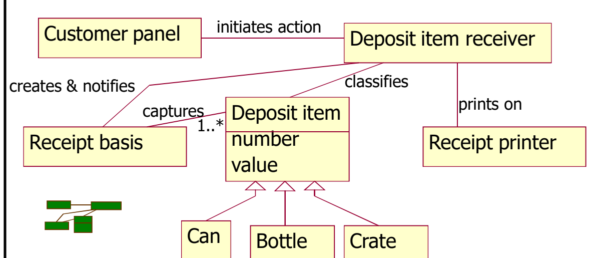


Adding Attributes

- An attribute is a **logical data value** of an **object**.
- Attributes in a conceptual model are **simple data values** as
  - Boolean, Date, Number, String (Text), Time, Address, Colour, Price, Phone Numbers, Product Codes, etc.
- Sometimes it is difficult to distinguish between attributed and concepts
  - E.g. Concept "Car" vs. attribute "Reg. Number".

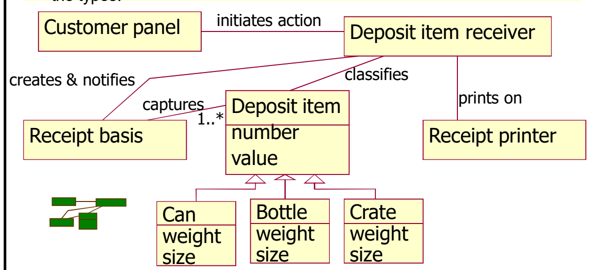
Adding Attributes

- The Deposit item has a value.
- Also it will be assigned a number that shows later on the receipt.



Adding Attributes

- In order to be classified by the Deposit item receiver each item has also a weight and a size.
- However this is the same for each type of item, but different *between* the types.



Conceptual Model

- Representation of concepts in a problem domain.
- **In UML it is basically a "class diagram" without operations.**
- It may show
  - ✓Concepts
  - ✓Associations
  - ✓Attributes

The Concept Category List

- physical or tangible objects
- specifications, designs, descriptions of things
- places
- transactions
- transaction line items
- roles of people
- abstract noun concepts
- organisations
- events
- processes
- rules and policies
- catalogues
- records
- services
- manuals, books

Noun Phrase Identification

- Identify the **noun** and **noun expression** in **textual descriptions** of a **problem domain**
- Consider them as **concepts** and **attributes**.
- Mechanical *noun-to-concept mapping* isn't possible
- Words in usual languages are ambiguous (especially English).

The "return item" Use case-Exercise: Find the Nouns

- The system controls a recycling machine for **returnable bottles, cans and crates.** The machine can be used by several customers at the same time and each customer can return all three types of item on the same occasion. The system has to check, for each item, what type has been returned.
- The **system will register how many items each customer returns** and when the customer asks for a receipt, the system will print out what was deposited , the value of the returned items and the total **return sum that will be paid to the customer.**
- An operator also … (not in "return item" Use Case)

Case Study: Nouns found in the description

- recycling machine
- bottles, cans, and crates
- customers, customer
- types of item, item, type, returned items
- system
- receipt
- return sum

Case Study: Discussion of "recycling machine".

recycling machine
bottles, cans and crates
machine
customers, customer
types of item, item, type, returned items
system
receipt
return sum

- This concept is the "overall system"
- As we consider only one single use case,
- better to name this concept in the context of this use case, e.g.
  - **Deposit item receiver**

Case Study: Discussion of "bottles, cans, and crates".

deposit item receiver
bottles, cans, and crates
machine
customers, customer
types of item, item, type returned items
system
receipt
return sum

- Usually better to use singular and multiplicities instead of plural.
- As **bottle**, **can** and **crate** have much in common (they are processed as items),
- they could be generalised to an "item". We should remember this for later (**inheritance**).

Case Study: Discussion of "machine" and "system"

deposit item receiver
bottle, can, crate
machine
customers, customer
types of item, item, type, returned items
system
receipt
return sum

"Machine" and "System" mean here the same, namely the "Recycling machine", i.e. the
- **Deposit item receiver**

Case Study: Discussion of "customers" and "customer".

deposit item receiver
bottle, can, crate

customers, customer
types of item, item, type, returned items

receipt
return sum

- The customer has already been identified as an actor.
- They are outside of the system.
- We establish a concept, that interfaces with the customer (and is inside the system):
  - **Customer panel**

Case Study: Discussion of "item" (etc.)

deposit item receiver
bottle, can, crate

customer panel
types of item, item, type, returned items

receipt
return sum

- The items that are inserted in the machine.
- Good candidate as superclass for bottle, can, crate.
- Let's call it
  - **Deposit item**

Case Study: Discussion of "receipt"

deposit item receiver
bottle, can, crate

customer panel
deposit item

receipt
return sum

- The concept that "remembers" all items inserted in the machine.
- To distinguish it from the piece of paper returned to the customer, call it
  - **Receipt basis**

Case Study: Discussion of "return sum"

deposit item receiver
bottle, can, crate

customer panel
deposit item

receipt basis
return sum

- The sum that it is returned to the customer is actually computed by adding up all values of the items stored in the receipt basis.
- The sum itself is only a **primitive data value,** and may therefore not be considered as a concept.

 U2.142

Case Study: Discussion of Other Concepts

deposit item receiver
bottle, can, crate

customer panel
deposit item

receipt basis

- These are the concepts identified by nouns. Did we forget something?
- Check the "Concept Category List" !
- The system "**interfaces**" with the physical object "printer", so we add an interface concept
  - **Receipt printer**

 U2.143

Case Study: Summary of Concepts Identified in the Analysis

deposit item receiver
bottle, can, crate

customer panel
deposit item

receipt basis

receipt printer

- So far we have identified:
  - Concepts
  - A generalisation relationship.
- **Next step: Finding associations.**

 U2.144

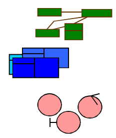
Summary - Object Oriented Analysis

The main task is identifying the objects.

Also: Relationships between objects.

Three strategies:

- Conceptual Model (concepts as objects)
- CRC cards (index cards as objects)
- Analysis Model (Stereotypes as objects)



Next step: Design.

Objective Questions

Q1. Define Architecture.

Q2. Justify the statement System Development is a Model Building.

Q3. At what time, you will decide to start System Development.

Q4. In what way specifications can be used?

Q5. Define Conceptual modeling.

Q6. Define block design.

Q7. Define requirement model.

Q8. Define analysis model.

Q9. Define design model

Q10. Define Implementation Model.

Q11. Define Test Model.

Short Questions

Q1. Suggest some heuristics for identifying objects during object oriented analysis of problem.

Q2. Differentiate between analysis objects with examples.

Q3. Consider air ticket reservation system. Identify entity, control and interface objects.

Q4. Write short note on Architecture.

Q5. Differentiate Method and Process

Q6. What are the five different models for system development, as per the Jacobson approach?

Q7. How models are tightly coupled to the architecture? Discuss.

Long Questions

Q1. What are the features of analysis model and design? Explain with examples.

Q2. For a library management system make analysis model, design model and construction model.

Q3. Justify the statement "System development is model building".

Q4. "The goal if analysis model is to develop a model of what the system will do." Explain the statement with the help of the steps that an analyst will follow throughout the analysis.

Q5. Describe what is done in Analysis with example?

Q6. Describe the system development process with model building.

Research Problems

Q1.Many people invest their money in a number of securities (shares). Generally, an investor has multiple portfolios of investments, each portfolio having investments in many securities. From time to time an investor sells or buys some securities and gets dividends for the securities. There is a current value of each security-many sites give this current value. It is proposed to build a personal investment management system (PIMS) to help investors keep track of their investments as well as on the overall portfolios. The system should also allow an investor to determine the net-worth of the portfolios.

- Discuss the problem analysis for the PIMS problem statement/
- Provide the use case based requirement analysis and specification
- Identify the conceptual objects and draw the Analysis model for PIMS

References

1. Ivar Jacobson, "Object Oriented Software Engineering", Pearson, 2004.
2. Grady Booch, James Runbaugh, Ivar Jacobson, "The UML User Guide", Pearson, 2004
3. R. Fairley, "Software Engineering Concepts", Tata McGraw Hill, 1997.
4. P. Jalote, "An Integrated approach to Software Engineering", Narosa, 1991.
5. Stephen R. Schach, "Classical & Object Oriented Software Engineering", IRWIN, 1996.
6. James Peter, W Pedrycz, "Software Engineering", John Wiley & Sons
7. Sommerville, "Software Engineering", Addison Wesley, 1999.
8. http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/userguide.html
9. http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/statemachinediagram.html
10. http://www.developer.com/design/article.php/2238131/State-Diagram-in-UML.htm