

Software Process Model 2021

Introduction



Session outcomes

- Introduction to Module
- SPM- Introduction



Academic Integrity Policy

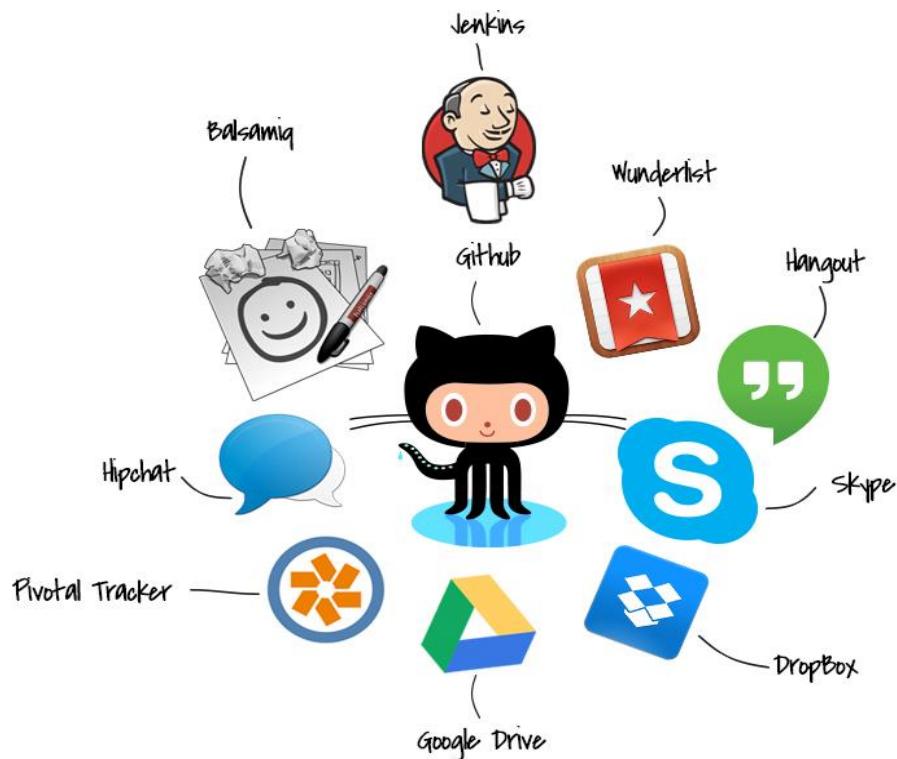
- Are you aware that following are not accepted in SLIIT???
 - **Plagiarism** - using work and ideas of other individuals intentionally or unintentionally
 - **Collusion** - preparing individual assignments together and submitting similar work for assessment.
 - **Cheating** - obtaining or giving assistance during the course of an examination or assessment without approval
 - **Falsification** – providing fabricated information or making use of such materials
- Committing above offenses come with serious consequences !
- See General support section of Courseweb for full information.

Enrollment Key :IT1060 for 2021 Feb

Progression Criteria

- In order to progress from one academic year to the next, you must maintain the following minimum academic standard.
 - From year 1 to year 2 – No more than **5 failed or incomplete** modules
 - From year 2 to year 3 – No more than **3 failed or incomplete** modules
 - From year 3 to year 4 – No more than **2 failed or incomplete** modules
- If you do not meet the above criteria, you will not be able to progress to the next year.

MODULE INTRODUCTION



Module contents

- Course web - SPM-IT1060
 - IT1060 [2021/FEB] General
 - Module outline
 - Notices
 - Marks
- Weekly updates
 - Lecture
 - Lab
 - Tutorial
 - Additional Reading/Recordings

Learning outcomes

Differentiate the characteristics and effects of different types of software engineering processes.

Describe the requirement engineering process and components of a formal requirements document for a software project.

Apply the knowledge of UML to model and represent system requirements.

Describe software design strategies and the importance of design models.

Apply the knowledge of software implementation and testing to write test cases.

Apply Agile development methodology.

Assessment Criteria

Mid Term Examination	30%	LO1-LO4
Assignment I	10%	LO3-LO5
Assignment II	10%	LO4-LO5
Final Examination	50%	LO1-LO9

To pass this module, students need to obtain a pass mark in both “Continuous Assessments” and “End of the Semester Examination” components which would result in an overall mark that would qualify for a “C” grade or above.

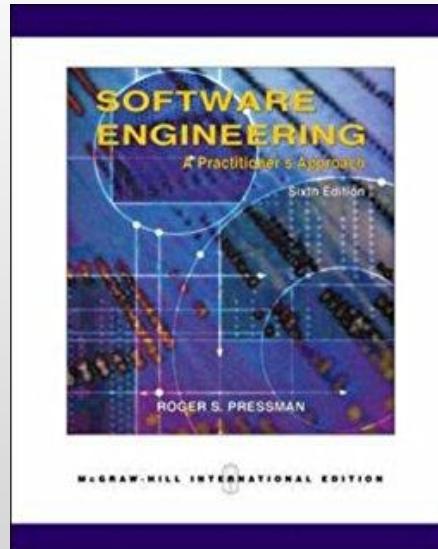
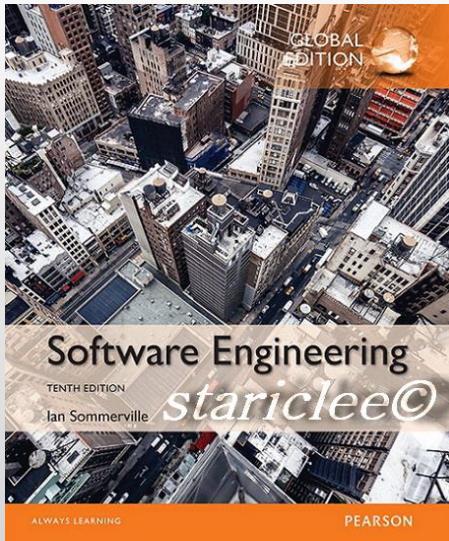
Assignments

- **Five members** in one group
- Randomly chosen case study
- Two submissions
 - Before Mid term – Week 7
 - Based on requirements engineering and use case diagrams
 - After Mid term – Week 12
 - Based on Activity Diagram



Recommend Texts

- Ian Sommerville, “Software Engineering”, Pearson Education Limited, 10th edition, 2016
- R. Pressman, “Software Engineering: a practitioner’s approach”, McGraw-Hill Education; 8th edition, 2014
- K.S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, Addison-Wesley, 2012
- SWEBOK, Guide to the Software Engineering Body of Knowledge, 2014



Lab Schedule

Time	Lectures	Labs/Tutorials
Week 01	Introduction to Software Engineering	No Tutes /No Labs
Week 02	SDLC	Tutorial 01 -Introduction
Week 03	SDLC	Tutorial 02 -SDLC
Week 04	Requirement Engineering	Tutorial 03 -SDLC
Week 05	UseCase Diagram	Tutorial 03 - Requirement Eng.
Week 06	UseCase Diagram	Lab 01- Use Case
Week 07	Activity Diagram	Lab 01- Use Case
Week 08	Mid-term Examination	
Week 09	Software Design	Lab 02- Activity Diagram
Week 10	Implementation and testing	Lab 02- Activity Diagram
Week 11	Implementation and testing	Tutorial 04
Week 12	Modern Software Development Methodology	Lab 03 -Agile
Week 13	Modern Software Development Methodology	Lab 03 -Agile
Week 14	Revision	

SPM- INTRODUCTION

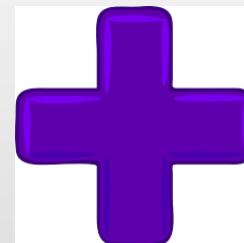


Session Outcomes

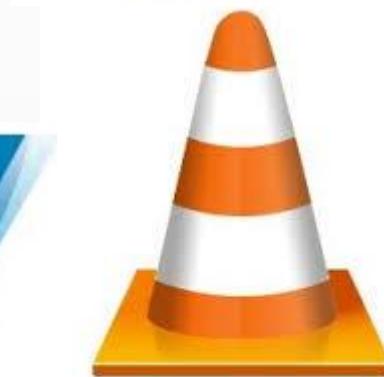
1. What is a Software
2. What is Software Engineering
3. Software Process
4. Software Process Activities
5. Software process model
6. Software Development Life Cycle
7. Software Engineering Ethics

What is Software?

Software is **not only** the computer programs, but also associated documentation and configuration files, needed to make the programs operate correctly.



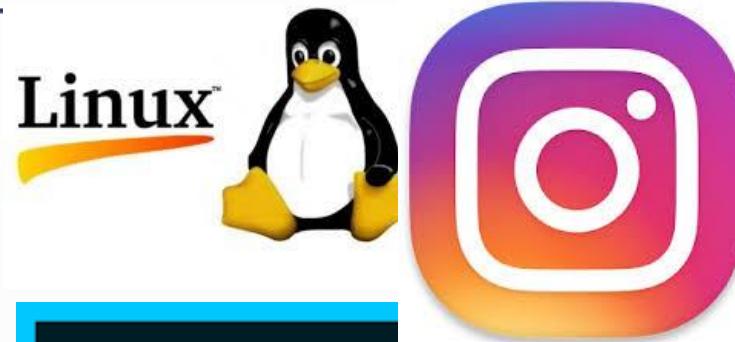
Popular Software



Microsoft®
.NET



MySQL®

The MySQL logo, featuring a blue dolphin leaping out of water next to the word "MySQL".

nero
Burning ROM

1st Year 1st Semester IP

```
/* adding two numbers*/
#include <stdio.h>

int main(void)
{
    int no1, no2;
    int sum;

    no1 = 25; // assign value to no1 variable
    no2 = 12; // assign value to no2 variable

    sum = no1 + no2; // add numbers

    printf(" Sum is %d\n", sum); // print sum
    return 0;
} // end of main function
```

```
/* adding two numbers*/
#include <stdio.h>

int main(void)
{
    int no1, no2;
    int sum;

    printf("Enter first number: "); /* prompt */
    scanf("%d", &no1); /* read the value */

    printf("Enter second number: "); /* prompt */
    scanf("%d", &no2); /* read the value */

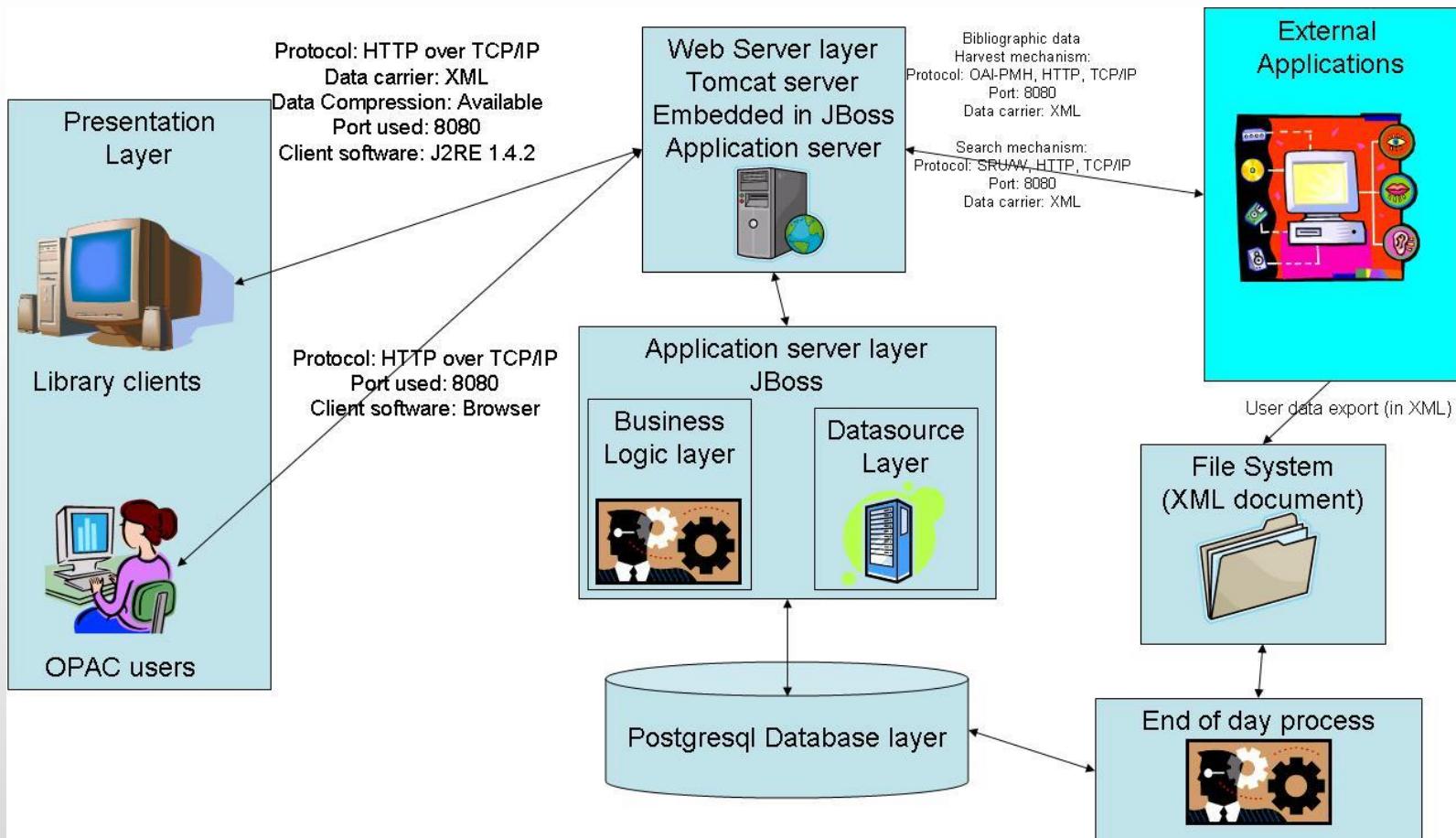
    sum = no1 + no2; /* assign total to sum */

    printf(" Sum is %d\n", sum); /* print sum */

    return 0;
} // end of main function
```

- Are these Software ?
- What are things that you need to do to develop Software?

Library Software



<http://www.verussolutions.biz/technology.php>

Programs Vs. Software Products

Program

- Small
- Single developer
- Small in size
- Limited Functionality
- Single user (author)
- Simple user interface
- Sparse documentation
- No user manual
- Ad hoc development

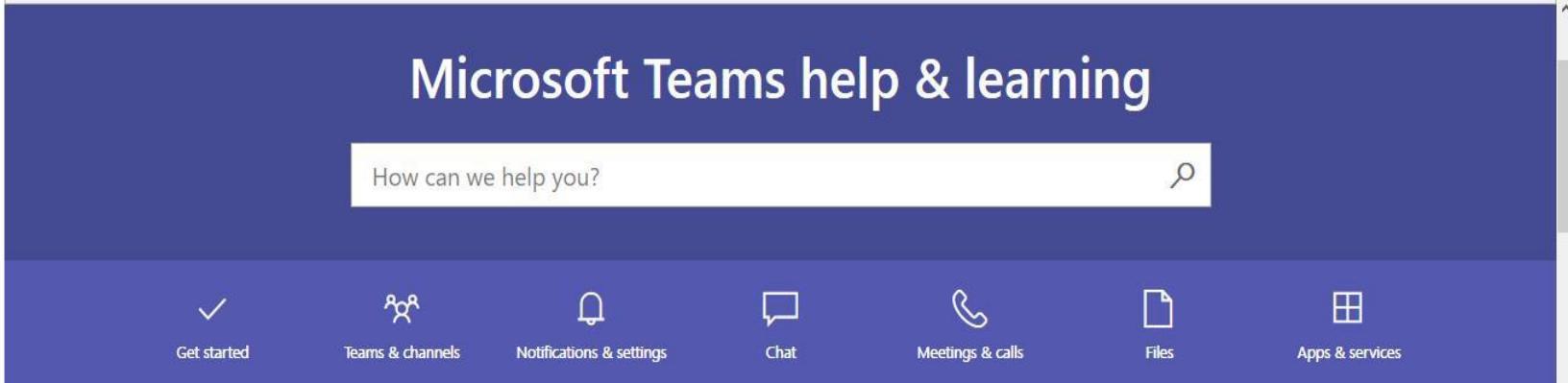
My o

Software Product

- Large
- Team of developers
- Multiple users (customer)
- Complex user interfaces
- Detailed documentation
- User manual
- Systematic development

Standard Procedure

MS Teams



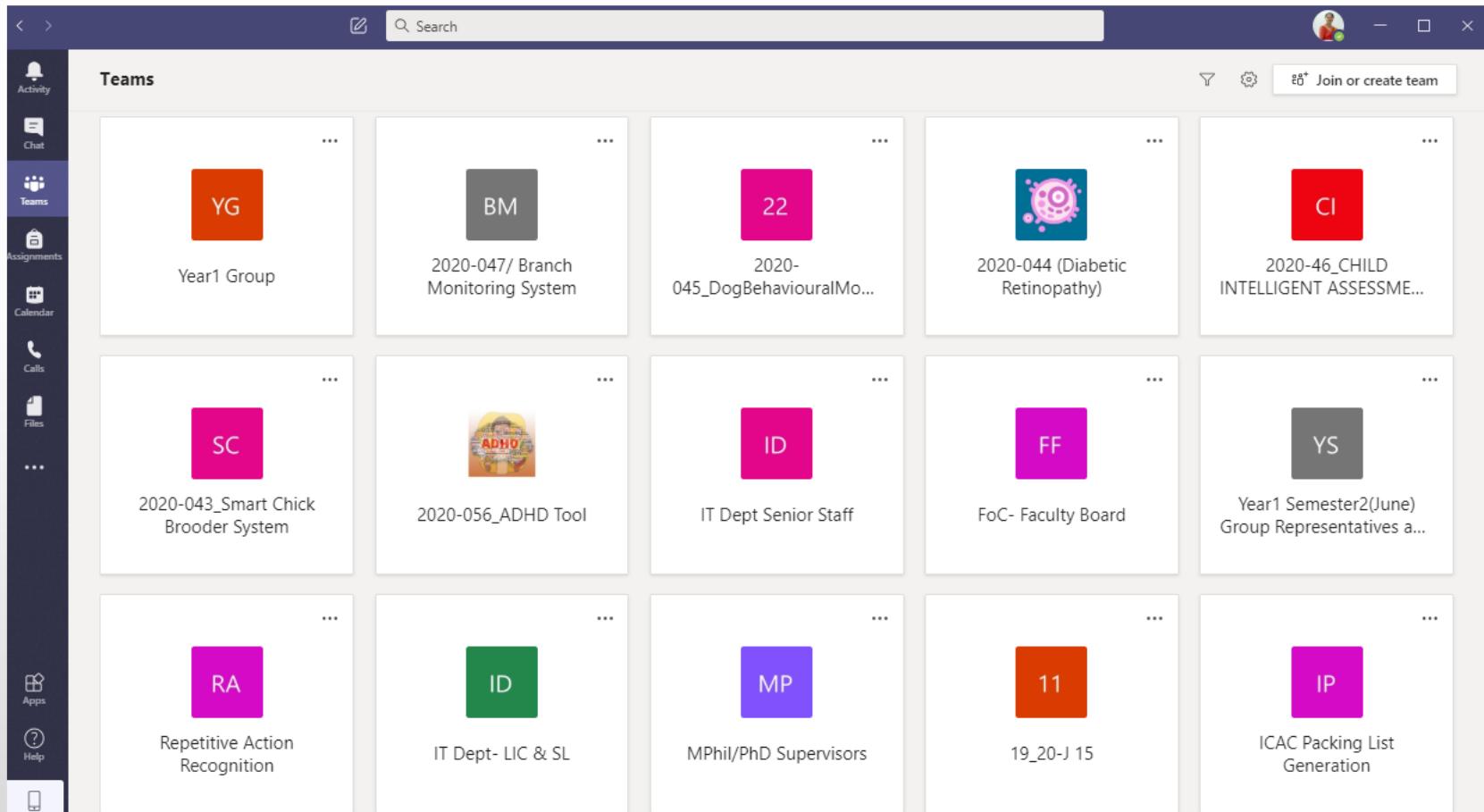
The image shows the Microsoft Teams help & learning interface. At the top, a dark blue header features the text "Microsoft Teams help & learning". Below it is a white search bar with the placeholder "How can we help you?" and a magnifying glass icon. The main area has a light blue background with a grid of nine video preview cards showing various team members. Above the grid is a purple navigation bar with icons for "Get started", "Teams & channels", "Notifications & settings", "Chat", "Meetings & calls", "Files", and "Apps & services".

Video conferencing with Teams

From custom backgrounds to more video feeds per meeting, Teams video meetings help you and your team feel connected.

[LEARN MORE >](#)

MS Teams



The screenshot shows the Microsoft Teams application interface. On the left is a vertical navigation bar with icons for Activity, Chat, Teams (selected), Assignments, Calendar, Calls, Files, Apps, and Help. The main area displays a grid of 15 team tiles:

Team Name	Description
YG	Year1 Group
BM	2020-047/ Branch Monitoring System
22	2020-045_DogBehaviouralMo...
CI	2020-46_CHILD INTELLIGENT ASSESSME...
SC	2020-043_Smart Chick Brooder System
ADHD	2020-056_ADHD Tool
ID	IT Dept Senior Staff
FF	FoC- Faculty Board
YS	Year1 Semester2(June) Group Representatives a...
RA	Repetitive Action Recognition
ID	IT Dept- LIC & SL
MP	MPhil/PhD Supervisors
11	19_20-J 15
IP	ICAC Packing List Generation

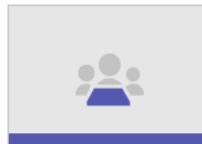
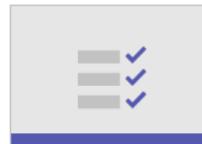
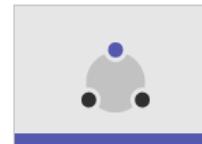
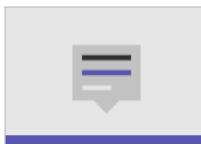
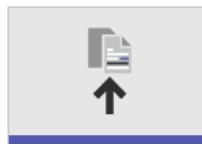
Teams Documentation

Microsoft | Support Microsoft 365 Office Windows Surface Xbox Deals Buy Microsoft 365 All Microsoft ▾   

Office support Products ▾ Devices ▾ What's new Install Office Account & billing Templates More support ▾

Keep in touch and stay productive with Teams and Microsoft 365, even when you're working remotely. [Learn more](#)

Microsoft Teams video training

 Quick start	 Intro to Microsoft Teams	 Set up and customize your team	 Collaborate in teams and channels
 Work with posts and files	 Upload and find files	 Start chats and calls	 Manage meetings

Software products can be

- Generic
 - These are stand alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them.
- Customized
 - These are systems that are developed for a particular customer requirements

How do we develop a real software?

- There will be a real user (Customer) who would need to use the software.
 1. Feasibly study (whether it is technical feasible and financially worthwhile)
 2. You have to find out what the customer wants (Requirements Gathering)
 3. Analyze the problem
 4. Develop a solution (Design)
 5. Code the solution
 6. Test and Debug
 7. Maintenance



Suggest Something Innovative?

- Suggest your dream software
 - Do not think about technical barriers
 - You can think beyond of the reality

“New Ideas will lead you to highest point of the Software Engineering”

Gods Eye

Software Engineering

- IEEE Definition of Software Engineering:

The application of a systematic, disciplined,
quantifiable approach to the development, operation,
and maintenance of software;

that is, the application of engineering to software.

IEEE Standard 610.12-1990, 1993.

Software Engineering Cont.

- Engineering discipline

make things work by applying theories, methods and tools where these are appropriate and also try to discover solutions to problems even when there's no proper theories/methods.

- All aspects of software production

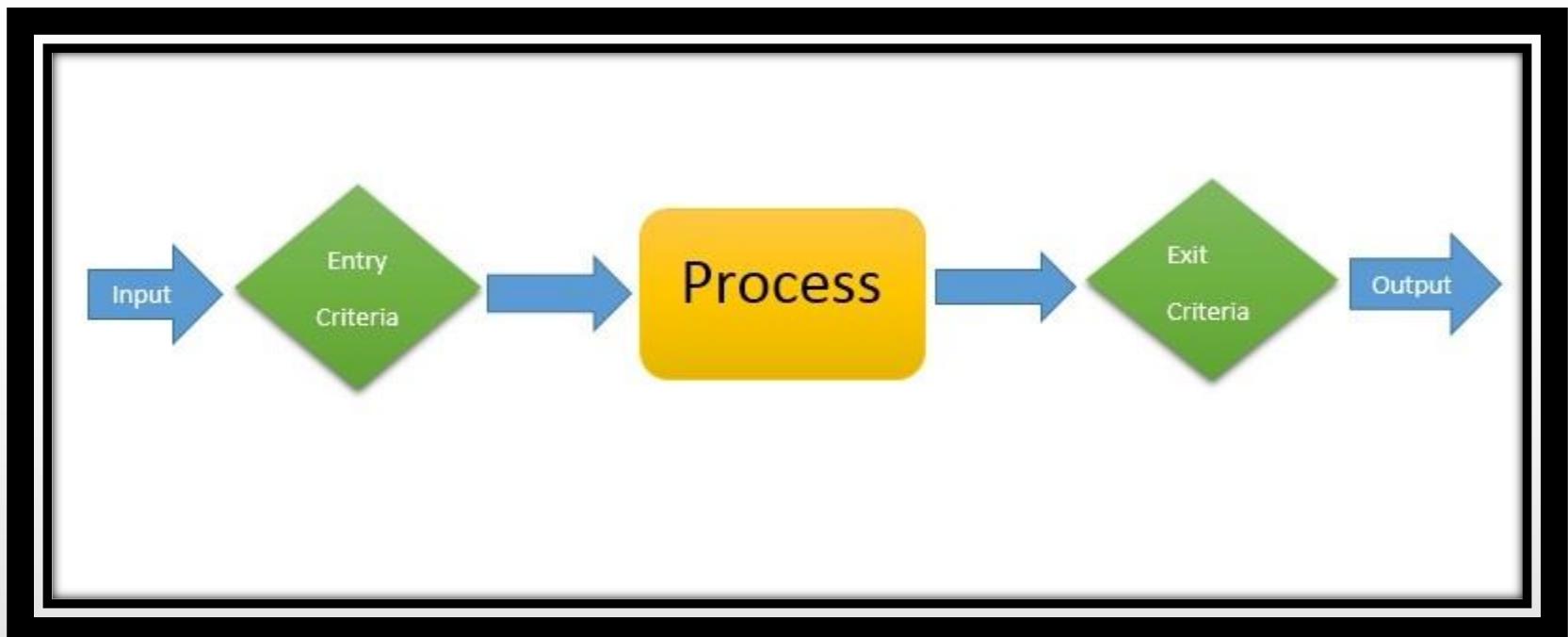
Not only technical processes of software development, but also project management and development of tools, methods and theories to support S/W production.

Key Challenges

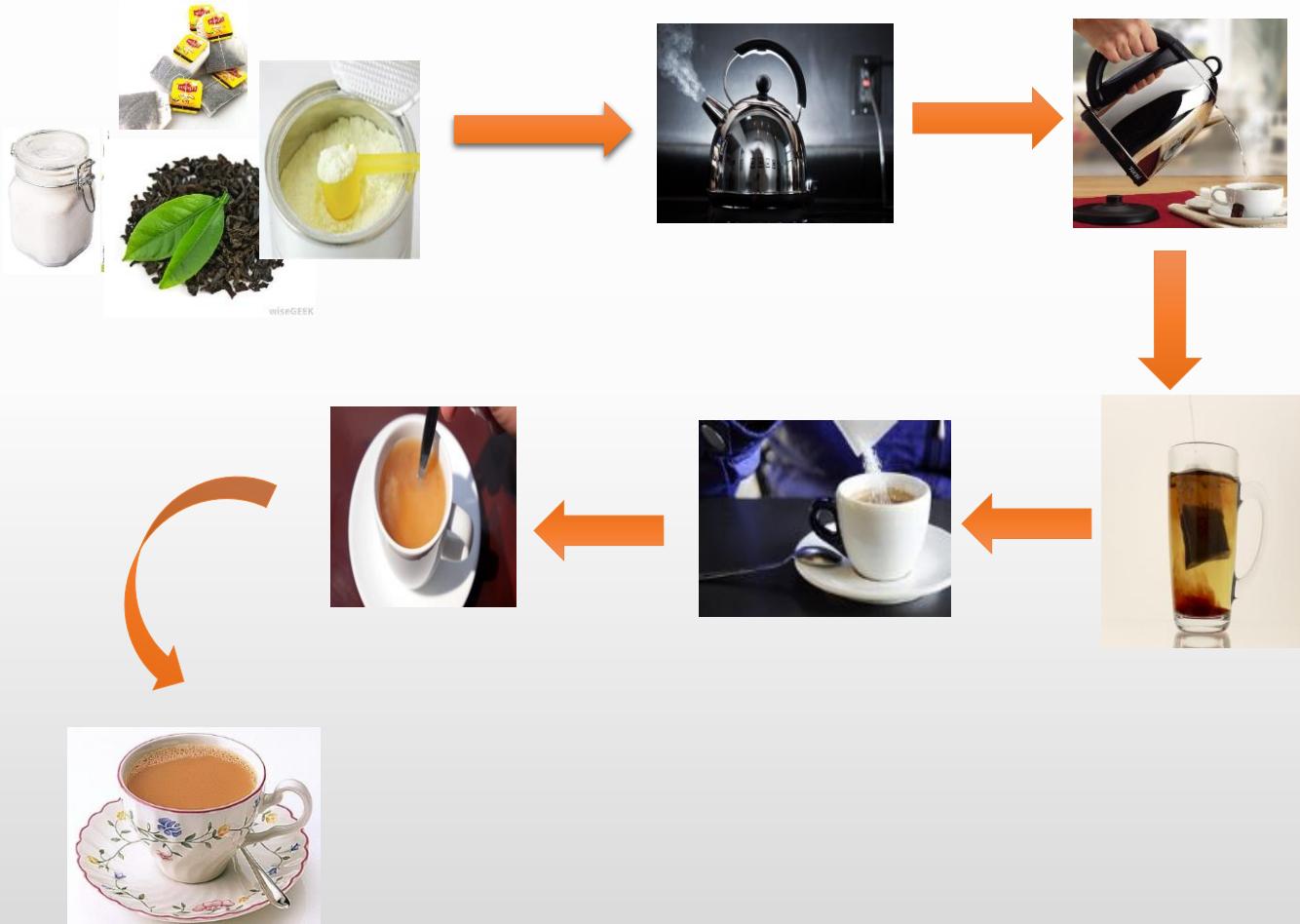
- Deliver Quality software to the customer at the agreed time
- The product is intangible → Can't touch
- Software processes are available and organization/product specific
- Keep overall costs within budget

End for this week

Process



Making A Cup of Tea



Making A Cup of Tea

- Ingredients : Tea Leaves, Sugar, Milk Powder, Boiled Water
- Process
 - Boil the water
 - Pour boiled water into cup
 - Put a tea bag inside a cup
 - Leave it few minutes
 - Put Sugar and Milk (if necessary)
 - Stir few seconds
 - Arrange it nicely
- Output: Tea

Software Process

- A software process is a set of interrelated activities and tasks that transform input work products into output work products. (SWEBOk V3 – Chapter 8)

Software Process Activities

- Software Specification
- Software Development
- Software Validation
- Software Evolution

Software Process Activities

- Software Specification
 - The functionality of the software and constraints
- Software Development
 - The software is designed and programmed.
- Software Validation
 - The software must be validated
- Software Evolution
 - The software must evolve

Software Processes

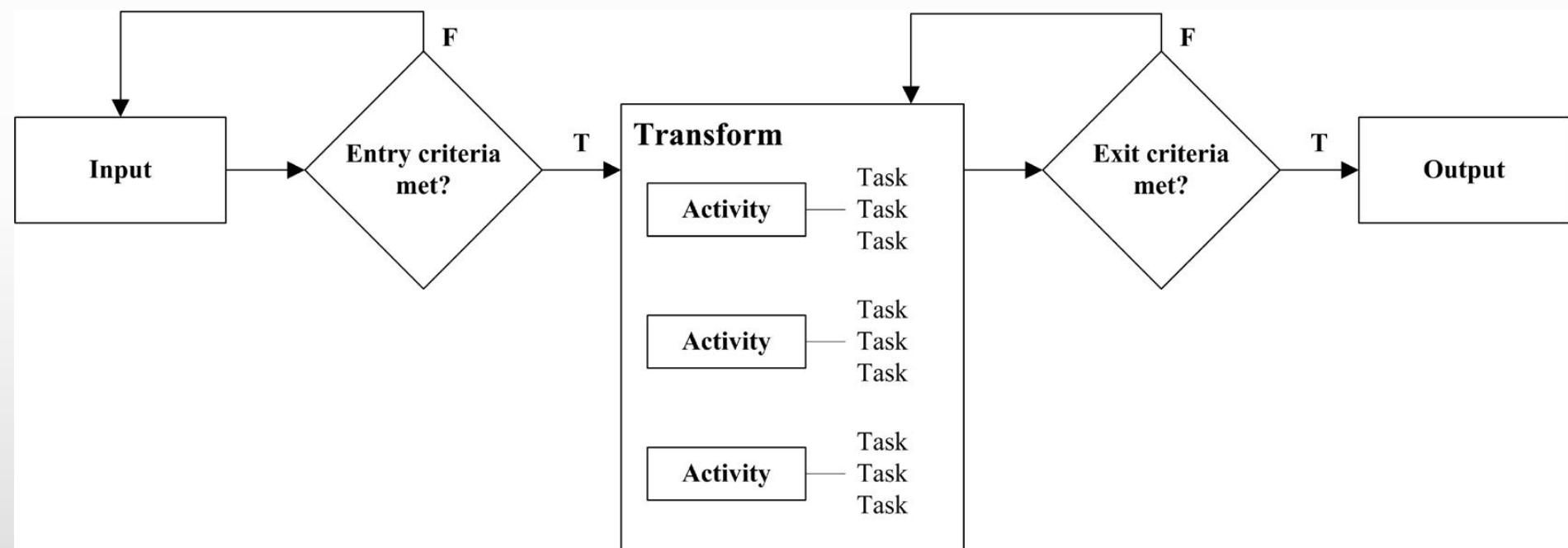
“There is no universal process that is right for all kinds of software”

Ex:

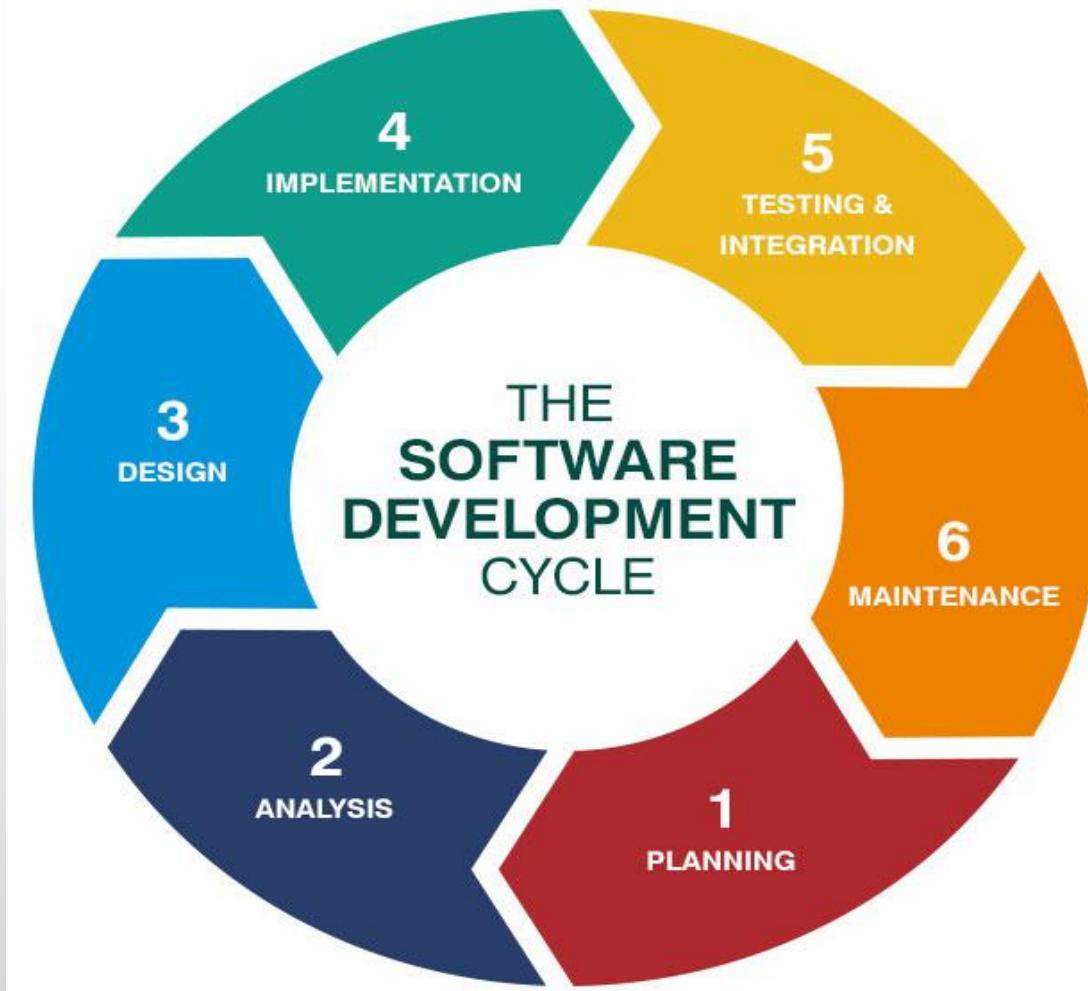
- For safety-critical systems, a very structured development process is required where detailed records are maintained.
- For business systems, with rapidly changing requirements, a more flexible ,agile process is likely to be better

Software process model

- It is a simplified representation of software process.



Software Development Life Cycle



Software Development Life Cycle Models

- A Software Development Life Cycle Model
 - has a series of stages that a software product undergoes during its life time.
 - is a descriptive and diagrammatic representation of the software life cycle.
 - is often referred as software process model.
 - maps the basic development activities to phases in different ways

General Software Process Models

- Waterfall Model
 - Classic
 - Iterative
- Prototyping
- Evolutionary Model
 - Incremental
 - Spiral
- Agile development.

Software Engineering Ethics

As a Professional Software Engineer,

- You should accept that your work involves wider responsibility than simply application of technical skills
- You should behave in an ethical way and morally responsible way
- You should not use your skills and abilities to behave in a dishonest way that will bring disrepute to the software engineering profession

Software Engineering Ethics Con.

Standards

- Confidentiality
- Competence
- Intellectual Property rights
- Computer misuse

Case Studies

- Library Management System

Library Management System

- Sri Lanka Institute of Information Technology (SLIIT) is the largest degree awarding institute in Sri Lanka with degree programs diversified to computing, business and engineering. In order to cater to its growing need of knowledge the institute maintains a Library Information System connecting Malabe, Metropolitan and Matara campuses. Each holds a latest collection of books and periodicals, particularly in the field of Information Technology, business management, engineering, general English, architecture and quantity surveying. The library of the Malabe Campus acts as the main resource center through which all library development activities are coordinated. SLIIT libraries are open to SLIIT students daily including weekends from 7.30 AM to 7.00 PM.

Tasks carried out at the library

- Add library materials
- Manage Library membership
- borrow books
- return books
- Pay fine on overdue materials
- Refund library deposit
- Replace lost library material
- Search library materials
- Generate reports

Next Lecture

Software Development Life Cycle Models

SOFTWARE PROCESS MODELING

Software Development Life Cycles

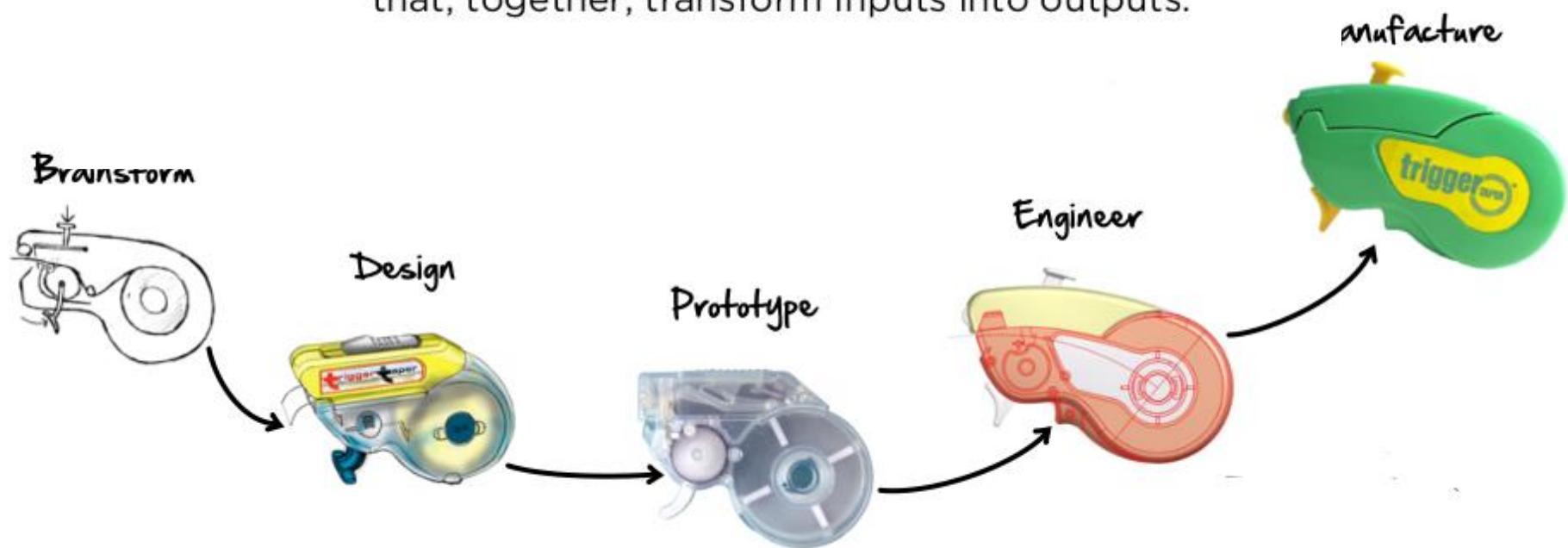
Session Outcomes

- What is a
 - Software Process
 - SDLC
 - Life Cycle Model
- Life Cycle Models
 - Waterfall Model
 - Prototyping Model
 - Incremental Model
 - Spiral Model
- Comparison and Selection

What is an Engineering Process?

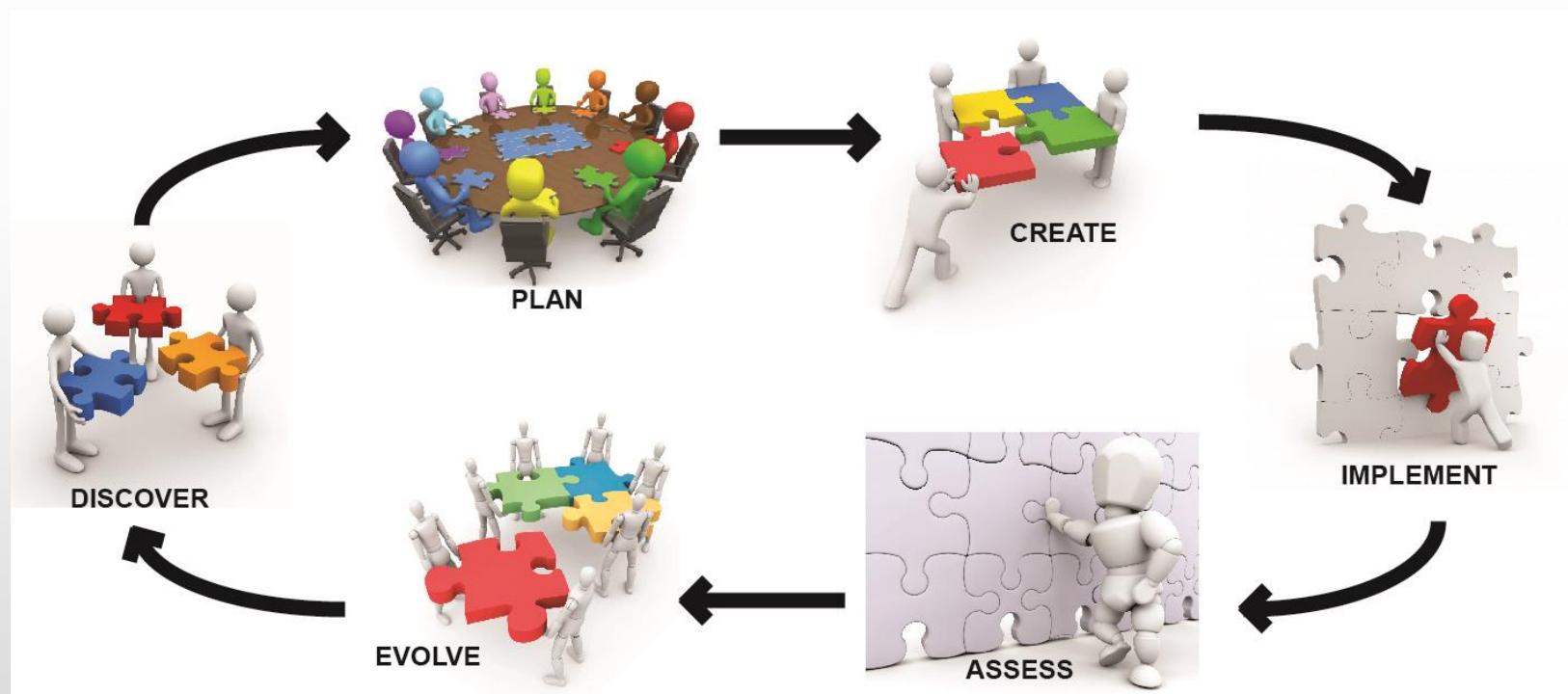


In engineering a process is a set of interrelated tasks that, together, transform inputs into outputs.



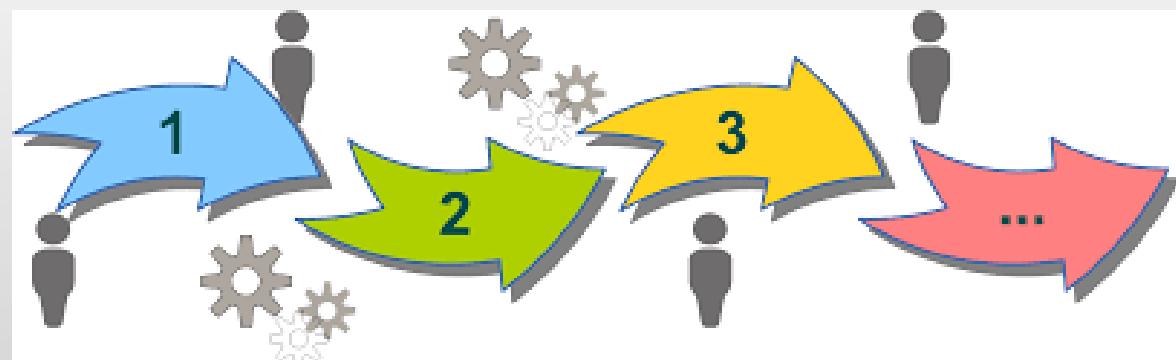
What is a Software Development Process?

- A set of Activities and Associated Results that produce a Software.



Fundamental Process Activities

1. Software Specification
2. Software Development
3. Software Validation
4. Software Evolution



1. Software Specification

- Specification involves clearly documenting the expectations on the system to be built
- Lays out requirements, and may include written and diagrammatic description of the services that the future system must provide.



2. Software Development

- Software development involves designing and implementing the system according to the software specification



3. Software Validation

- Software validation involves checking and verifying whether the system fulfills the requirements.



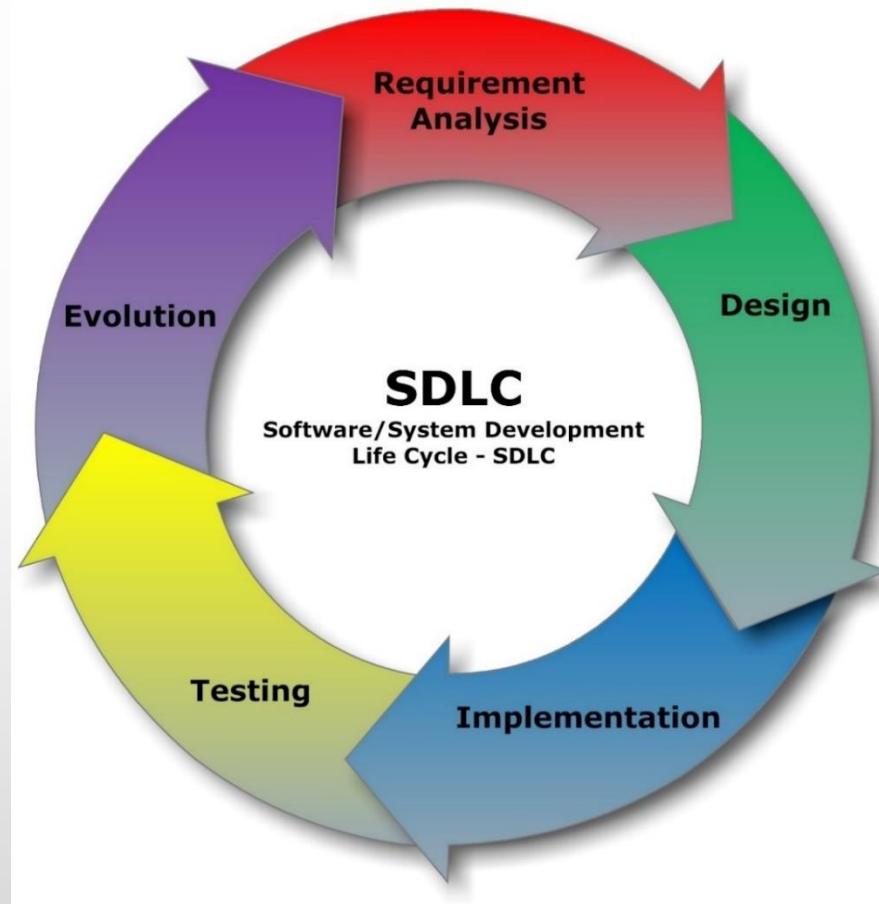
4. Software Evolution

- Software needs to be changed and upgraded with time.



What is a SDLC?

- The Software Development Life Cycle (SDLC) is a framework that defines activities performed throughout the software development process



Life Cycle Model (Process Model)

- A software life cycle (process) model:
 - is a descriptive and **diagrammatic model** of the life cycle of a software product;
 - identifies all the **activities and phases** necessary for software development;
 - establishes a **precedence ordering** among the different activities.
- Life cycle models encourage systematic and disciplined software development.

Life Cycle Models

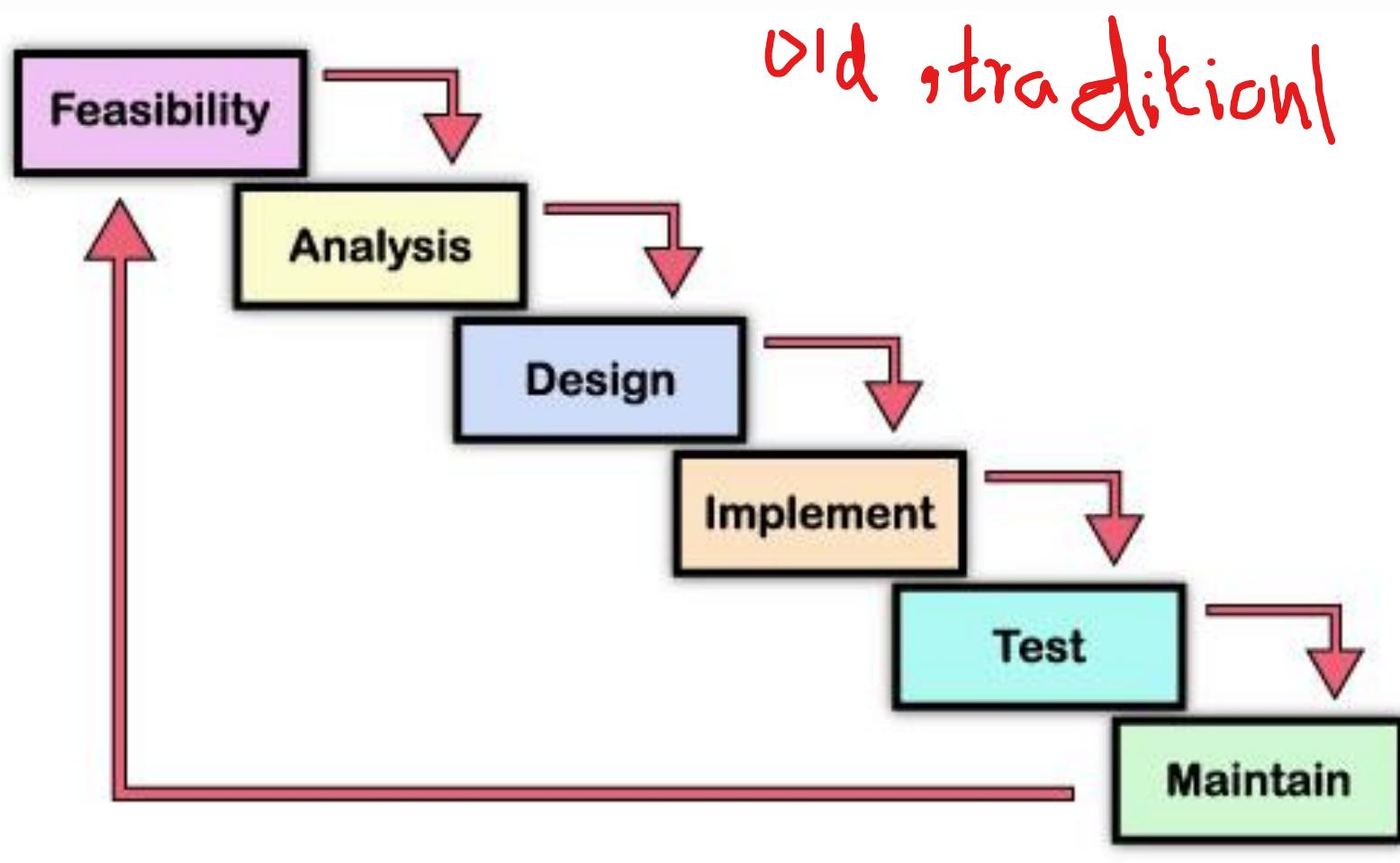
- Traditional Approaches
 1. Waterfall Model
 2. Incremental Model
 3. Prototyping Model
 4. Spiral Model
 5. Unified Process
- Modern Approaches
 - Agile Methods (XP, Scrum)
 - Secure Software Development

Life Cycle Models

1

Waterfall Model

Waterfall Model



Waterfall model

- Waterfall model is the most well known software lifecycle development model.
- It is very simple to understand and use.
- Each phase begins only after the previous phase is over.
- Also called **Linear Model**
- This model specifies what the system is supposed to do (i.e. define the requirements) before building the system (i.e. designing)

1. Feasibility Study

- This is the first phase of any SDLC model.
- The project objective is determined during this phase.
- The client and company developing the software decide if they should ;
 - Keep the existing system as is, or
 - Build a new software

Why do a feasibility study?

- To provide management with enough information to know:
 - Whether the project can be done
 - Whether the final product will benefit its users
 - What the alternative solutions are
 - Whether there is a preferred alternative



Example – Library System

- What are the feasibility criteria for a ‘Library System’?



2. The Requirements Phase

- Aim: to understand the customer's requirements:
- A customer may be a single person, a group, a department, or an entire organization:
- This phase involves two distinct activities:
 - 1. Requirements Gathering and Requirements Analysis**
 - 2. Requirements Specification**

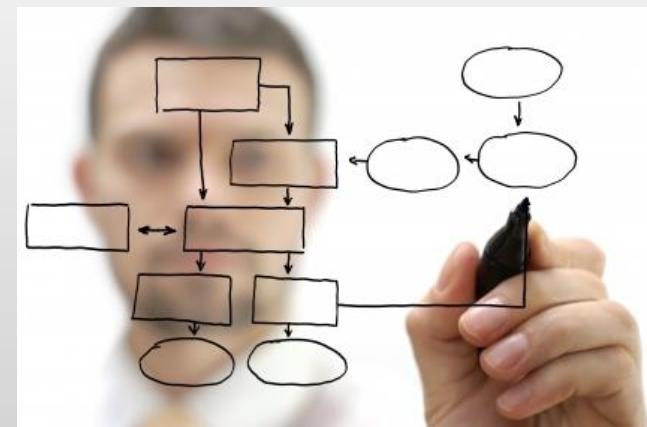
2a.1 Requirement Gathering

- The goal of this phase is for the stakeholders to find out the ‘what to be done’.
- Questions answered during this phase include:
 - Who will use the system?
 - How will they use the system?
 - What will be the input for the system?
 - What will be the output from the system?
- Requirement Gathering involve collecting information through **meetings, interviews and discussions**

2a.2 Requirements Analysis

- **Aim:** To understand exactly what the customer needs.. which may not be what they ask for:
 - data to be **input** to the system;
 - **processing** to be performed on these data;
 - data to be **output** from the system;
 - **characteristics** of the system as a whole;
 - **constraints** on the system/project.

- WHAT, not HOW!

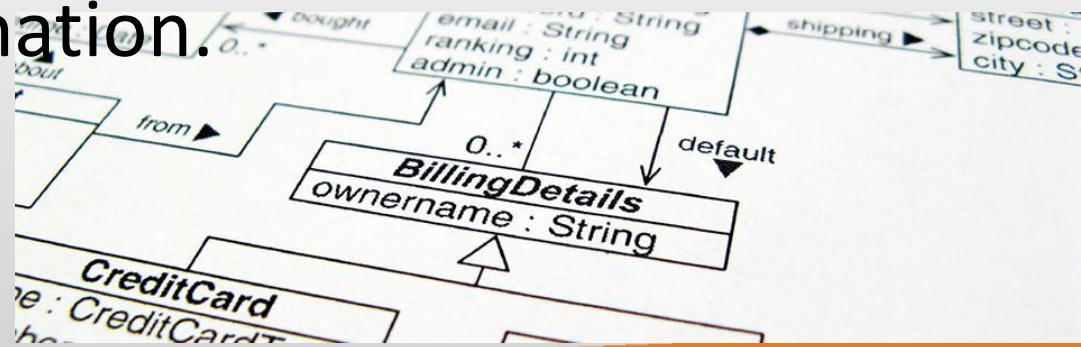


* 2b Requirements Specification

- Requirements are documented in a Software Requirements Specification (SRS).
- The SRS forms the basis of a **legal contract** with the customer:
- Software Engineers who specialize in requirements gathering, analysis, and specification are called (Systems/ Business / Requirement) Analysts.

3. Design

- Architects and Designers craft a high-level and low level design of the software.
 - Architectural Design
 - Low level Design
- Decisions are made about hardware, software and the system architecture.
- A design specification document (DSD) records this information.



4. Development

- On receiving system design documents, the work is divided in modules.
- A set of developers code the software as per the established design specification, using a chosen programming language
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process



5. Testing

- The testing phase ensures that the software requirements are in place and that the software works as expected.
- When a defect is identified, testers inform the developers.
- If the defect is valid, developers resolve it and create a new version of the software which then repeats the testing phase.
- The cycle continues until all defects are mitigated and the software is ready for deployment into the production environment.

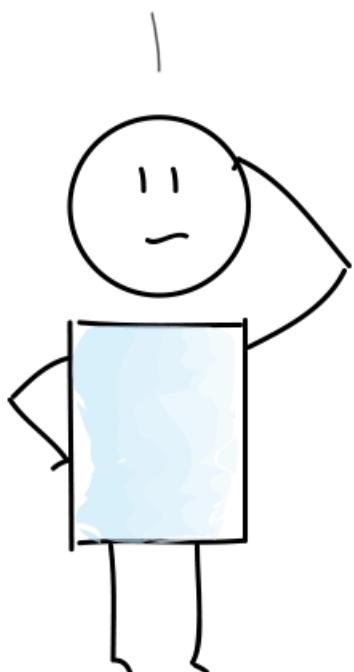
6. Deployment and Maintenance.

- Once the software is error free, it is deployed into the operating environment.
- While the customers are using the software, any issues will be brought to the attention of the maintenance team
- They work to resolve them immediately.

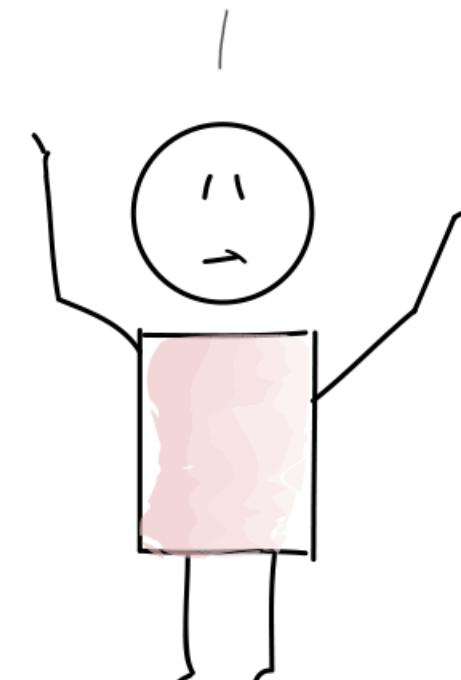
Waterfall in practise

THE MARKET HAS CHANGED.

LET'S ADD THIS
FEATURE TO THE APP.



SCOPE IS ALREADY
SET IN STONE.



Waterfall Model - Strengths

- Simple and easy to manage— each phase has specific deliverables.
- Milestones are better understood
- Sets requirements stability
- Works well for smaller projects where requirements are very well understood.
- A schedule can be set with deadlines.

Waterfall Model - Weaknesses

- No working software is produced until end.
- High uncertainty.
- Delays discovery of serious errors.
- After requirements phase, there is no formal way to make changes to the requirements.
- Not a good model for
 - complex projects
 - projects where requirements are at a moderate to high risk of changing

When to use Waterfall Model

- Software requirements clearly defined and known
- Product definition is stable
- New version of the existing software system is created
- Software development technologies and tools are well known
- Ample resources with required expertise are available

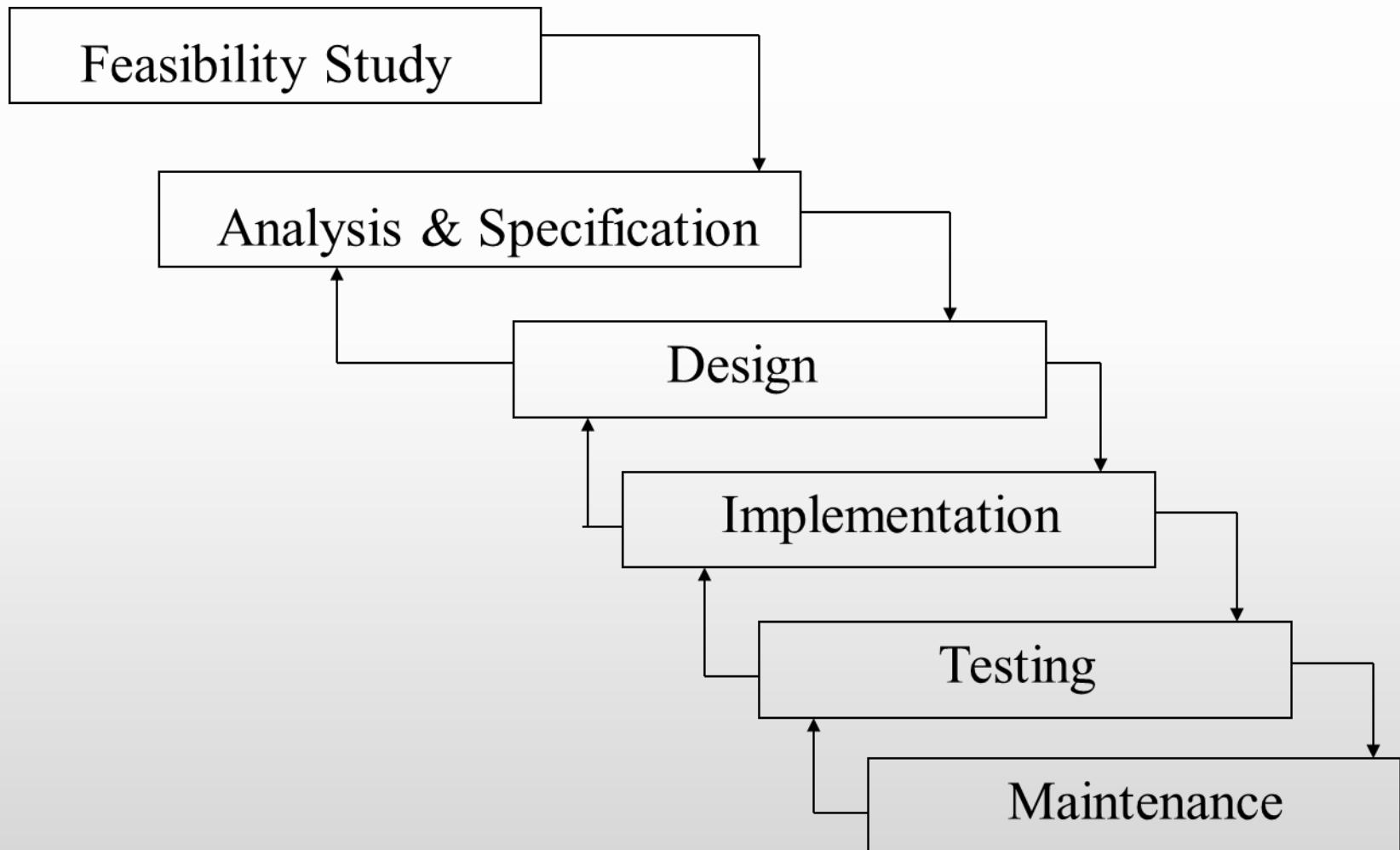
2

Iterative Model

Iterative Waterfall Model

- The classical waterfall model is idealistic:
 - It assumes that no defects are introduced during any of the development phases.
- In practice, defects are introduced during every phase of the software life cycle:
 - Hence feedback paths must be added to the classical waterfall model.
- The resulting Iterative Waterfall Model is one of the most widely used process models....

Iterative Waterfall Mode



Iterative Waterfall Model

- Strengths
 - Defects are detected and fixed early through the feedback path
- Weaknesses
 - Limited customer interactions
 - Difficult to incorporate change requests

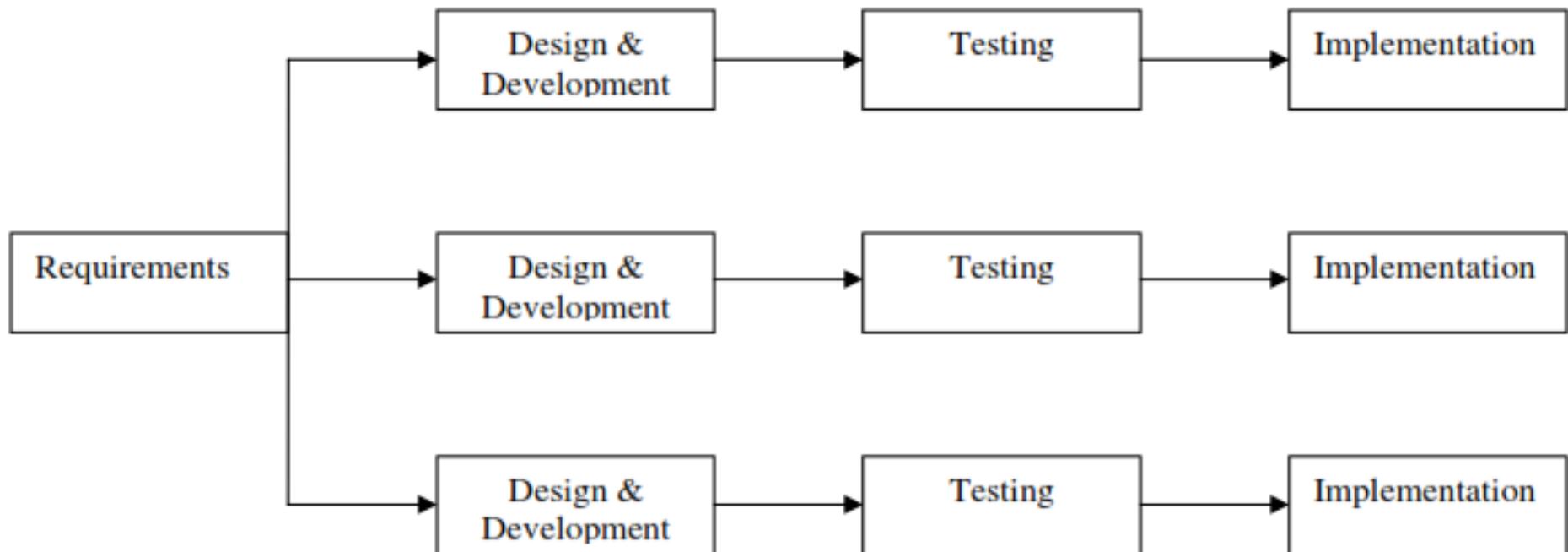
3

Incremental Model

Incremental model

- Incremental model is an evolution of waterfall model.
 - The product is designed, implemented, integrated and tested as a series of incremental builds.
- The incremental model prioritizes requirements of the system and then implements them in groups.
- It is the process of constructing a partial implementation of a total system and slowly adding increased functionality or performance.

Incremental model



Incremental model - Strengths

- Generates working software quickly and early during the software life cycle.
- More flexible - less costly to change scope and requirements.
- Easier to test and debug.
- Easier to manage risk.
- Lowers initial delivery cost.
- Less stress for the development team.

Incremental model - Weaknesses

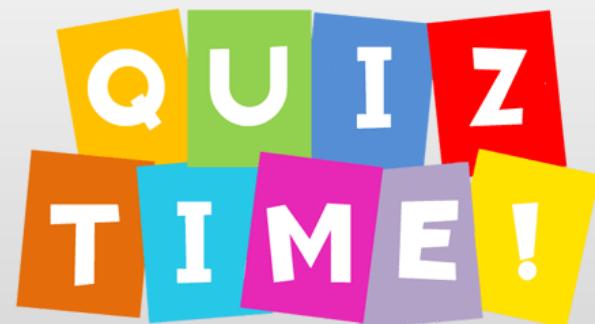
- Requires good planning and design
- Demarcation of increments can be difficult in a practical application.
- Each phase of an iteration is rigid and do not overlap each other.
- Total cost of the system might not be lower.
- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.

When to use Incremental model

- On projects which have lengthy development schedules
- A need to get basic functionality to the market early
- Most of the requirements are known up-front but are expected to evolve over time
- On a project with new technology

Mini-Case 1

- The project is to develop an inventory control system for a new super market in town. It should have all the regular functionalities such as adding new stocks, getting reports such as inventory re-order report and daily sales report etc. A project team has previous experience developing inventory systems for other clients.



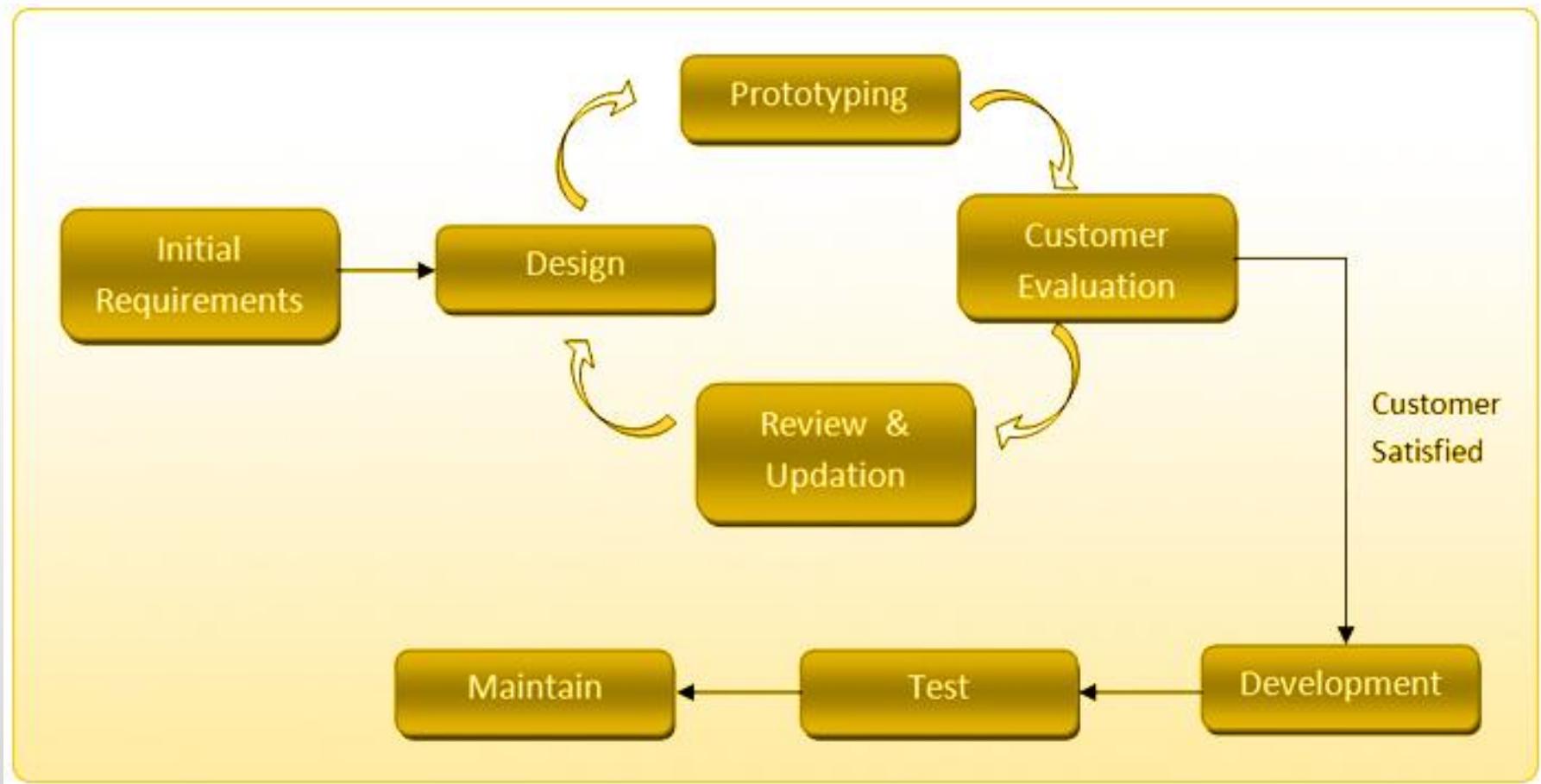
4

Prototyping Model

Prototype Model

- Instead of freezing the requirements before coding or design, a prototype is built to clearly understand the requirements.
- This prototype is built based on the current requirements.
- Through examining this prototype, the client gets a better understanding of the features of the final product.
- Requirements may be changed with the client feedback on the prototype.

Prototype Model



Prototyping Model - Strengths

- Ability to clarify user's expectations for the system to be developed
- Prototype stimulates awareness of additional needed functionality
- Better user satisfaction
- Early user feedback

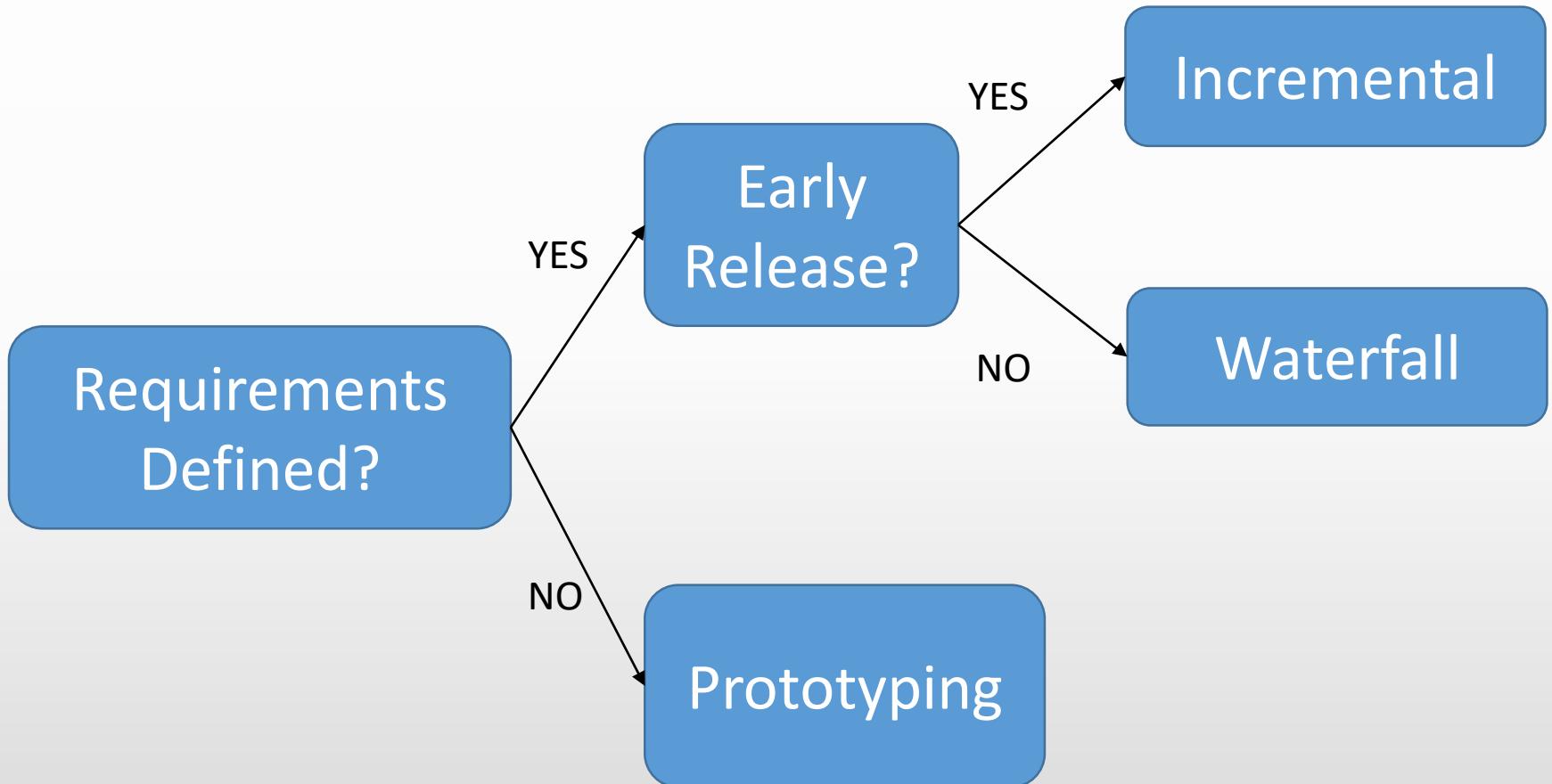
Prototyping Model - Weaknesses

- Scope Creep - The system scope may expand beyond original plans.
- Overall maintainability may be overlooked.
- The customer may want the prototype be delivered.
- Process may continue forever

When to Use Prototyping

- Requirements are unstable or have to be clarified
- Many user interfaces
- New technology
- New, original development
- Developers are not familiar with the technical and development tools

Selection of Approach



Question

- A company is developing an advanced version of their current software available in the market, what model approach would they prefer ?
-
- a) Waterfall
 - b) Incremental
 - c) Prototyping



Question

Selection of a model is based on...?

- a) Requirements
- b) Development team
- c) Users
- d) Project type and associated risk
- e) All of the mentioned



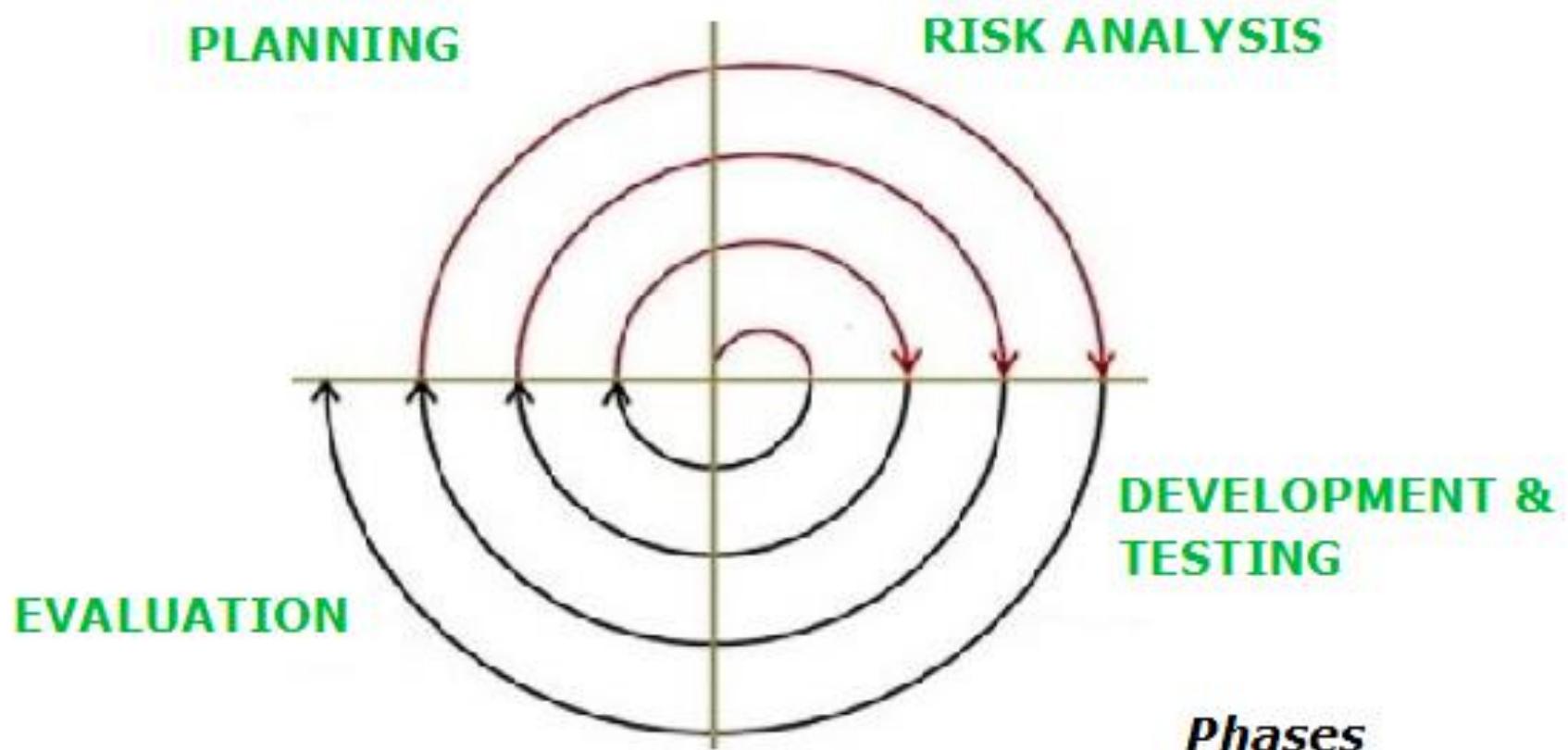
5

Spiral Model

Spiral Model

- Meta Model - combines iterative and prototype development with the systematic, controlled aspects of the waterfall model.
- Allows for incremental releases
- Introduced by Barry Boehm in 1986.
- Allows elements of the product to be added in when they become available or known.
- Emphasises on risk management

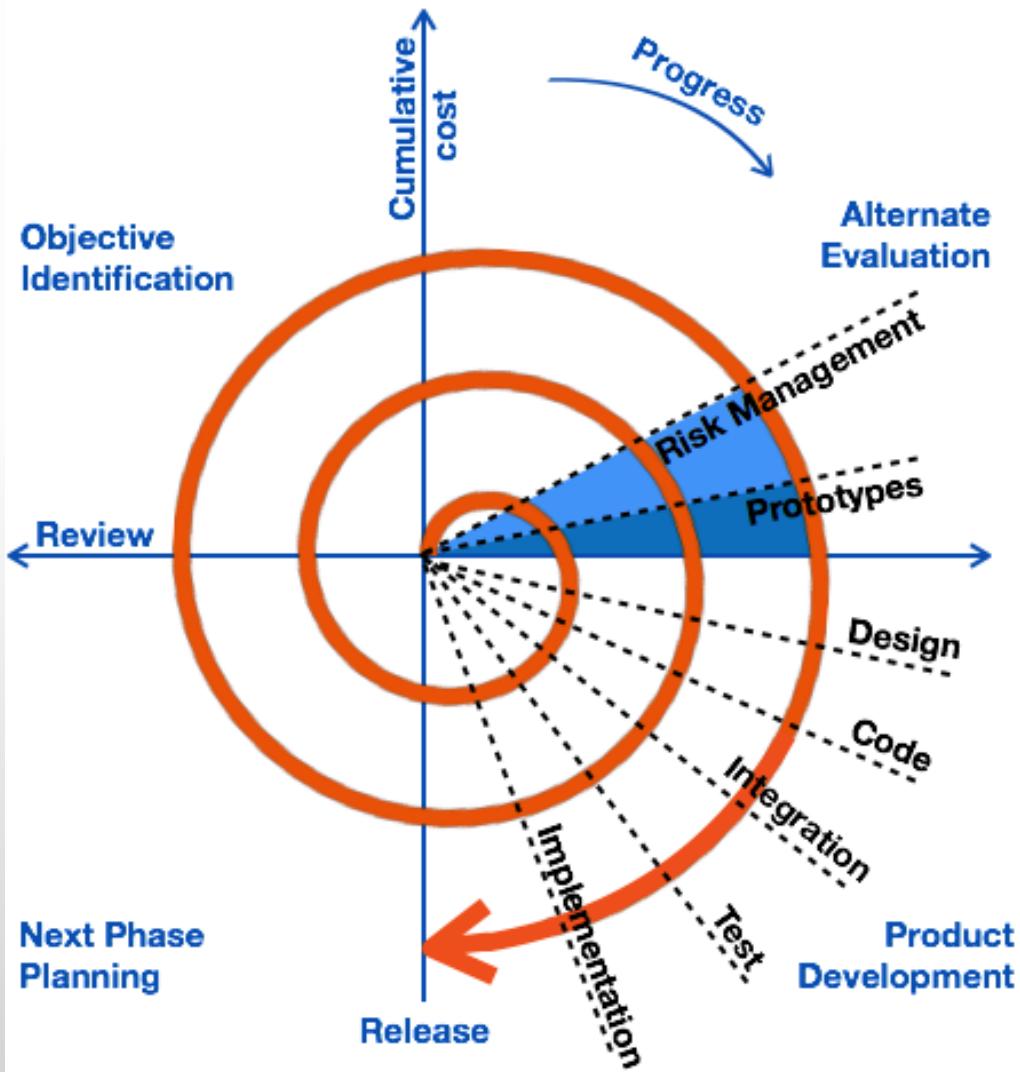
Spiral Model



Spiral Model

- Quadrant 1: Planning
 - Determine objectives, alternatives, and constraints.
- Quadrant 2: Risk Analysis
 - Evaluate alternatives, identify, resolve risks.
- Quadrant 3: Development & Test
 - Develop, verify, next-level product.
- Quadrant 4: Evaluation
 - Analyse feedback and plan next phases.

Spiral Model



Question

- Which of the following models will not be able to give the desired outcome if user's participation is not involved?
 - a) Waterfall
 - b) Spiral
 - c) Prototyping



Spiral Model - Strengths

- Focus on risk analysis.
- Good for large and mission critical projects
- A working software is produced early
- The design does not have to be perfect
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Spiral Model - Weaknesses

- Can be a costly model to use.
- Risk analysis requires expertise.
- Success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral Model

- For medium to high-risk projects
- New technology to be used
- Complex, constantly changing and continuous Requirements
- Significant changes are expected (research and exploration)
- Users are unsure of their needs

Question

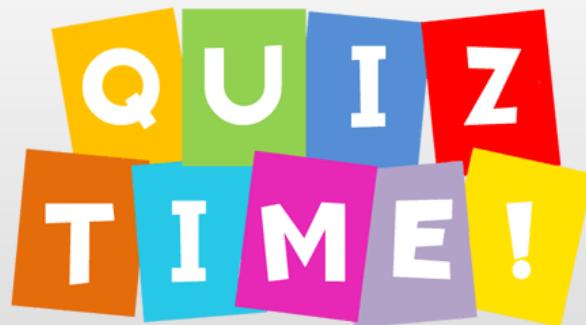
Which of the following life cycle model can be chosen if the development team has less experience on similar projects?

- a) Spiral
- b) Waterfall
- c) Iterative



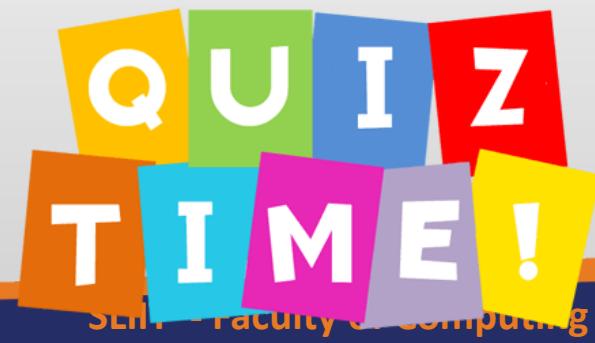
Activities

Which life cycle is suitable?



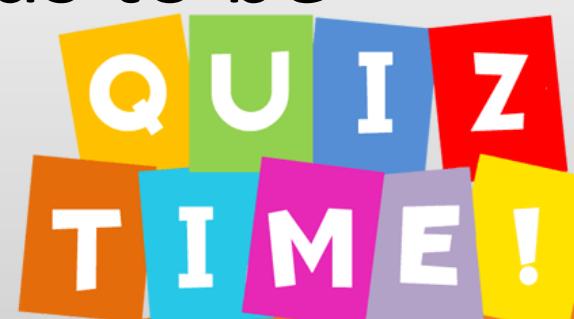
Mini-Case 2

- A library management system is required for a National Library with a number of branches in many cities. System should link to a number of other online libraries, databases, journals and university libraries through web and manage different user subscriptions. The project team has little experience in developing library systems before. However there is not much pressure on time.



Mini-Case 3

- A Project is to develop a complete system for a new bank. System will have many users, interrelationships, and functions. The project has few risks related to requirements definition. These risks needs to be reduced.



	Waterfall	V-Shaped	Evolutionary Prototyping	Spiral	Iterative and Incremental	Agile
Unclear User Requirement	Poor	Poor	Good	Excellent	Good	Excellent
Unfamiliar Technology	Poor	Poor	Excellent	Excellent	Good	Poor
Complex System	Good	Good	Excellent	Excellent	Good	Poor
Reliable system	Good	Good	Poor	Excellent	Good	Good
Short Time Schedule	Poor	Poor	Good	Poor	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent	Excellent	Excellent
Cost limitation	Poor	Poor	Poor	Poor	Excellent	Excellent
Skills limitation	Good	Good	Poor	Poor	Good	Excellent
Documentation	Excellent	Excellent	Good	Good	Excellent	Poor
Component reusability	Excellent	Excellent	Poor	Poor	Excellent	Poor

References

- I. Sommerville, Software Engineering, 10th ed., Addison-Wesley, 2011. – Chapter 2
- *Roger Pressman, Software Engineering: A Practitioner's Approach* – Chapter 2

SOFTWARE PROCESS MODELING

Requirements Engineering

Session Outcomes

- Why we need Requirements Engineering
- Requirement levels and types
- Requirements Engineering process
 - Requirements elicitation and analysis
 - Requirements Specification
 - Requirements Validation

Requirements Engineering

- Addresses two main problems
 - What do we want to build?
 - How do we write this down?

Library Management System

- What do you have to build?
- What do you want to build?
- How does it differ from the existing system?

What is RE?

- Requirements engineering is a process of establishing
 - the functions and attributes
 - that a customer requires from a system
 - the constraints
 - under which it operates and is developed.

Why RE?

- Trouble in understanding what customer really wants
- Record requirements in a disorganized manner
- Spend far too little time verifying what we do record
- Fail to establish a solid foundation for the system or software that the user wants built

System stakeholders

- Any person or organization who is affected by the system in some way and so who has a legitimate interest
- Stakeholder types
 - End users
 - System managers
 - System owners
 - External stakeholders

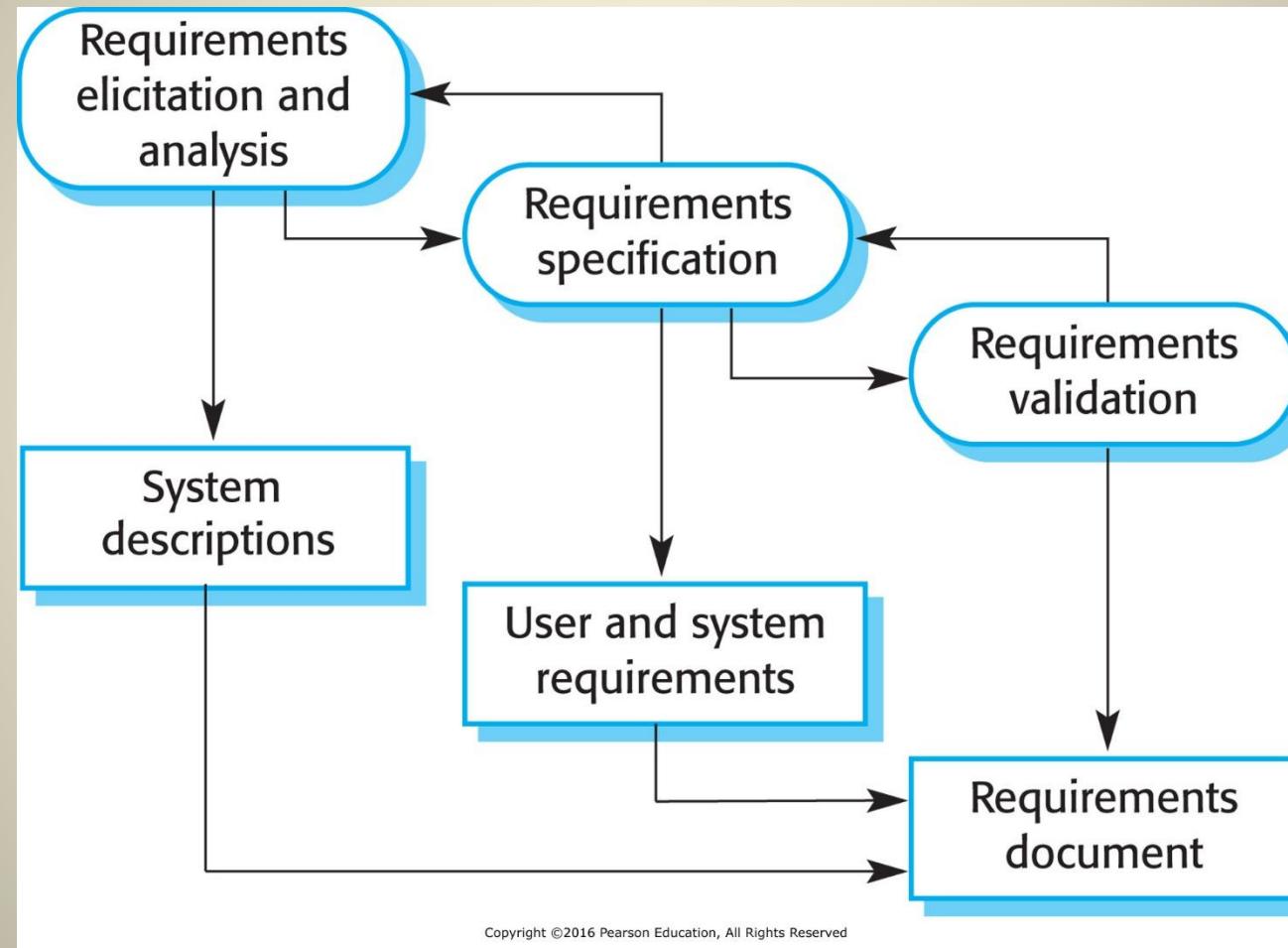


Activity

- Identify Stakeholders in the Library Management System



Requirements Engineering process

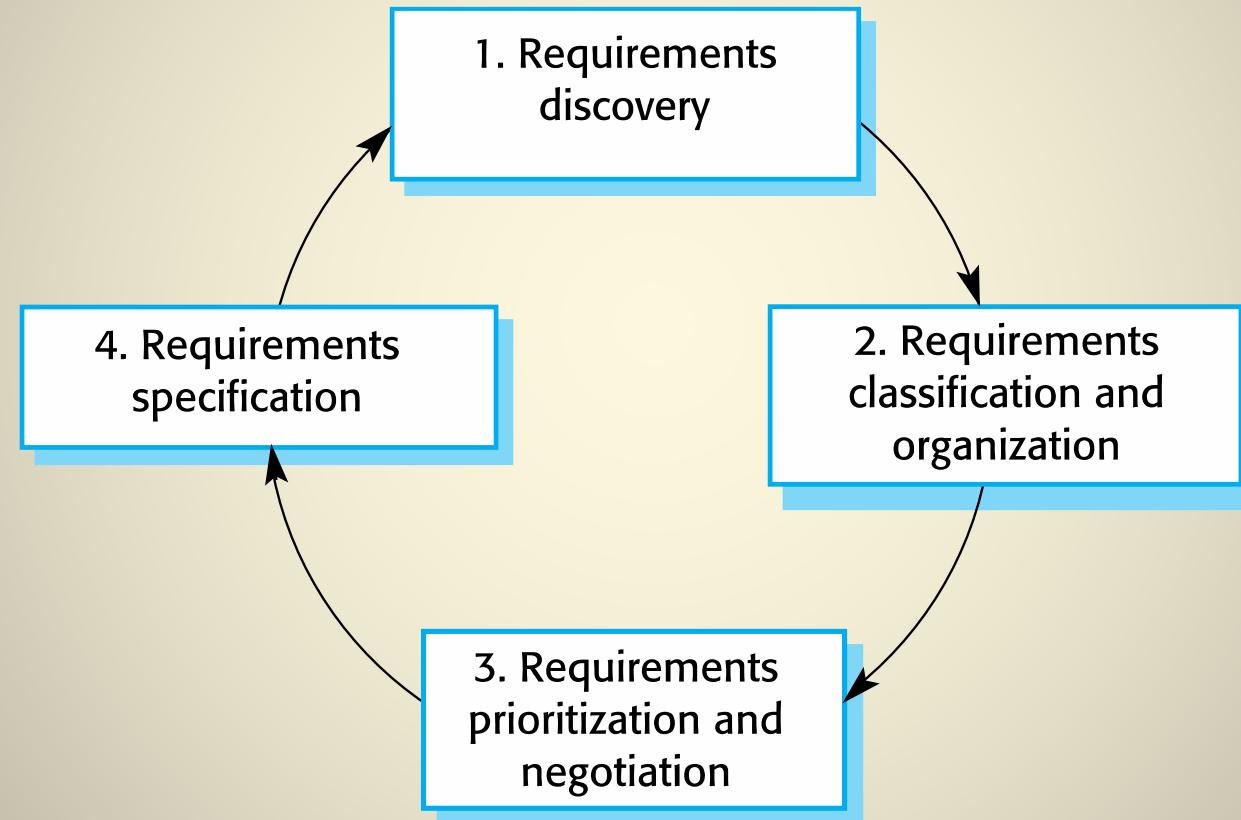


Requirements elicitation and analysis

Requirements elicitation

- Aim – Understand the work stakeholders do and how a new system would support that work.
- Stages include:
 - Requirements discovery,
 - Requirements classification and organization,
 - Requirements prioritization and negotiation,
 - Requirements specification.

The requirements elicitation and analysis process



Requirements Elicitation and Analysis

1. Requirements discovery
 - Interacting with stakeholders to discover their requirements.
2. Requirements classification and organisation
 - Groups related requirements and organises them into coherent clusters.
3. Prioritisation and negotiation
 - Prioritising requirements and resolving requirements conflicts.
4. Requirements specification
 - Requirements are documented and input into the next round.

1. Requirements discovery

- The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- Requirements elicitation techniques
 - Interviewing
 - Closed or Open
 - Observation/Ethnography
 - Observing and analysing how people actually work.



"I'll go talk to the stakeholders and find out their requirements... in the meantime, you guys start coding."

Activity

- Visit a library where a Library System is not used. Observe their behavior.
 - Note the differences in the case study and actual system.
 - Note down the additional requirements you identified

2. Requirements Classification

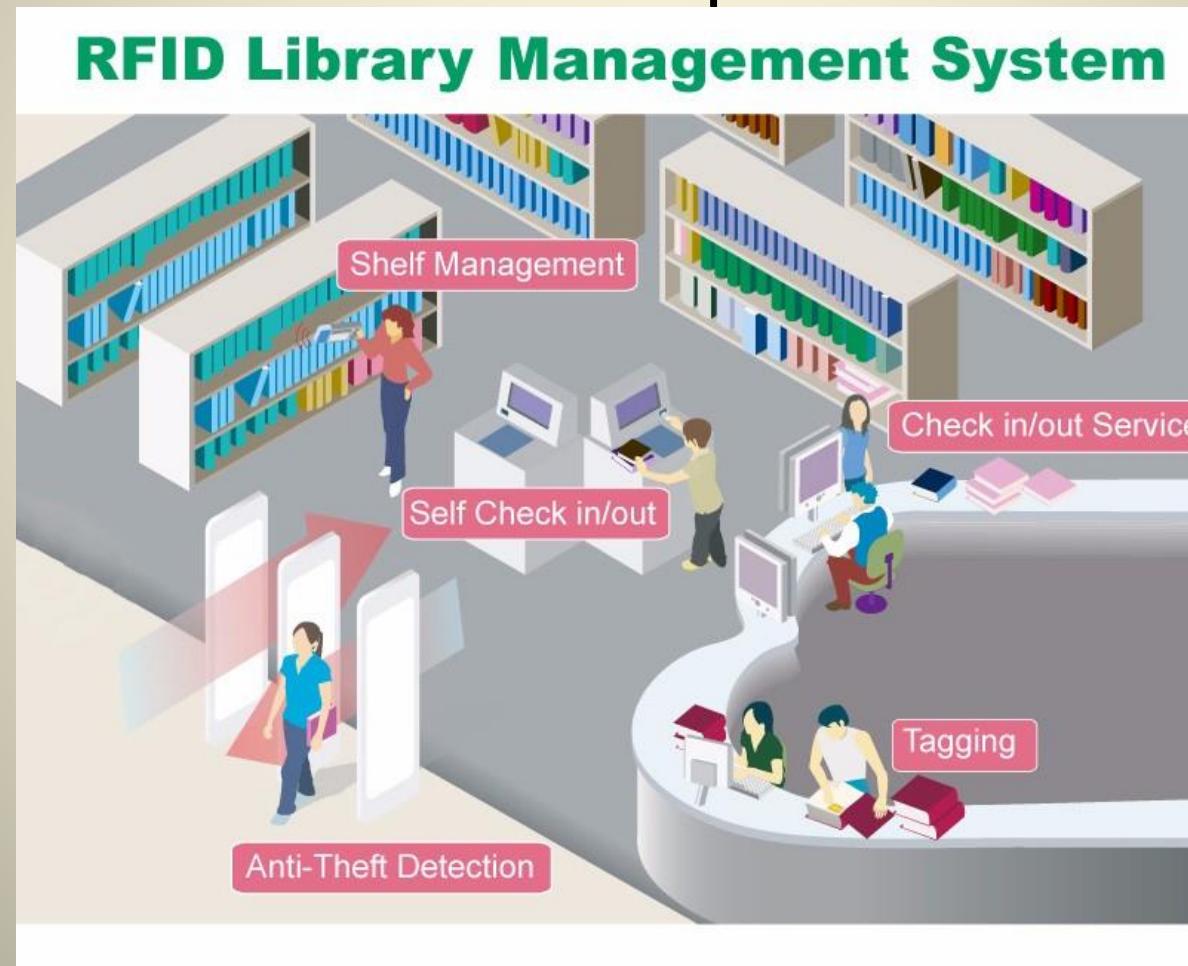
- Functional requirements
 - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- Non-functional requirements
 - Requirements that are not directly concerned with specific functionality
- Constraints

2.1 Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail

Activity

- Write down two Functional Requirements for the Library System



2.2 Non-functional requirements

- Often apply to the system as a whole rather than individual features or services.
- These define system properties
- Are also called **Quality Attributes**
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Non-functional - examples

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Activity

- Write down three non-functional Requirements for the Library System
- Choose the most important among them

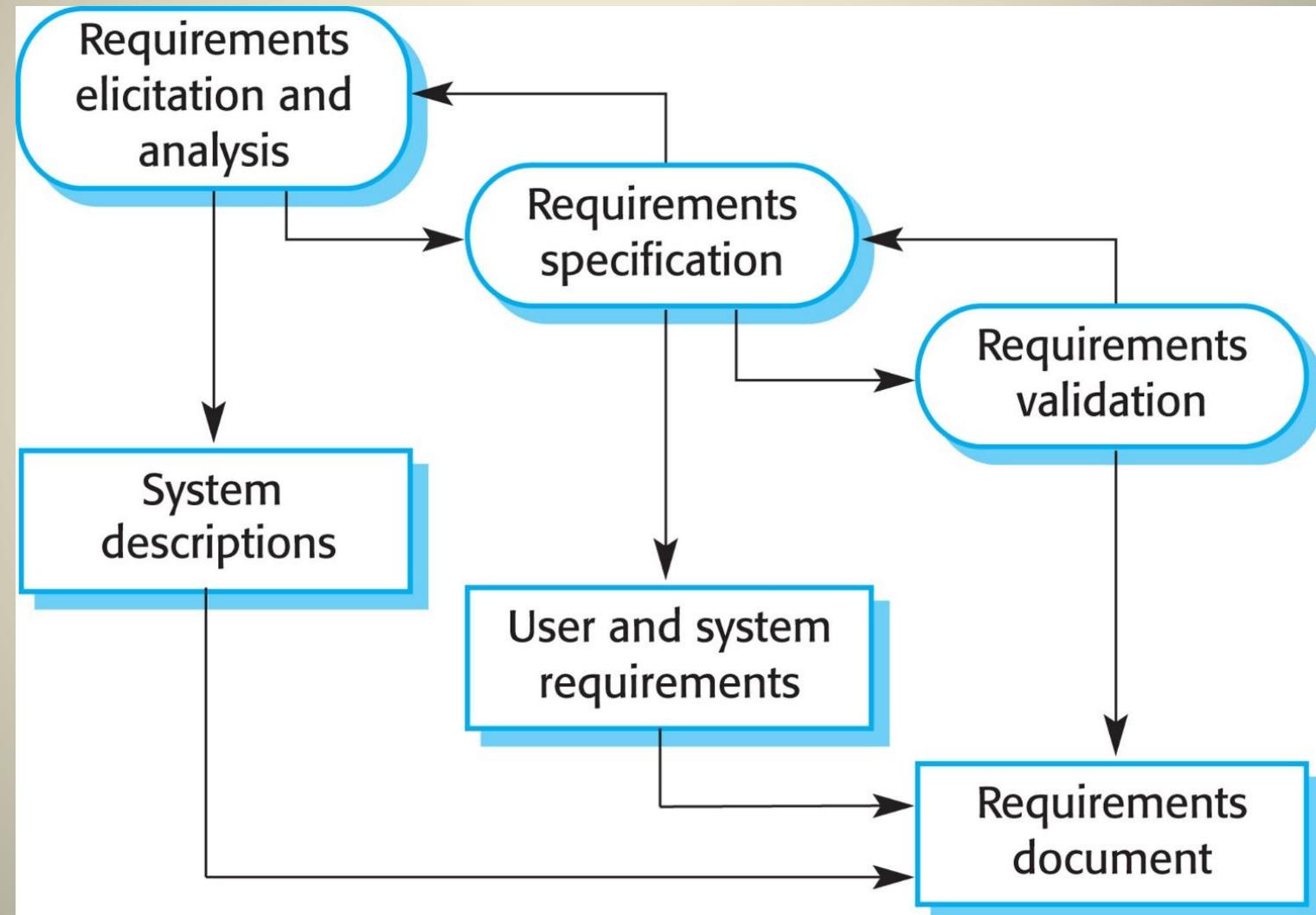
Requirements imprecision

- inconsistent Requirements
 - the customer cannot decide what problem they really want solved.
- ambiguous Requirements
 - it is not possible to determine what the requirements mean.
- Incomplete Requirements
 - there is insufficient information to allow a system to be built.

Activity

- Write down two imprecise Requirements for the Library System.

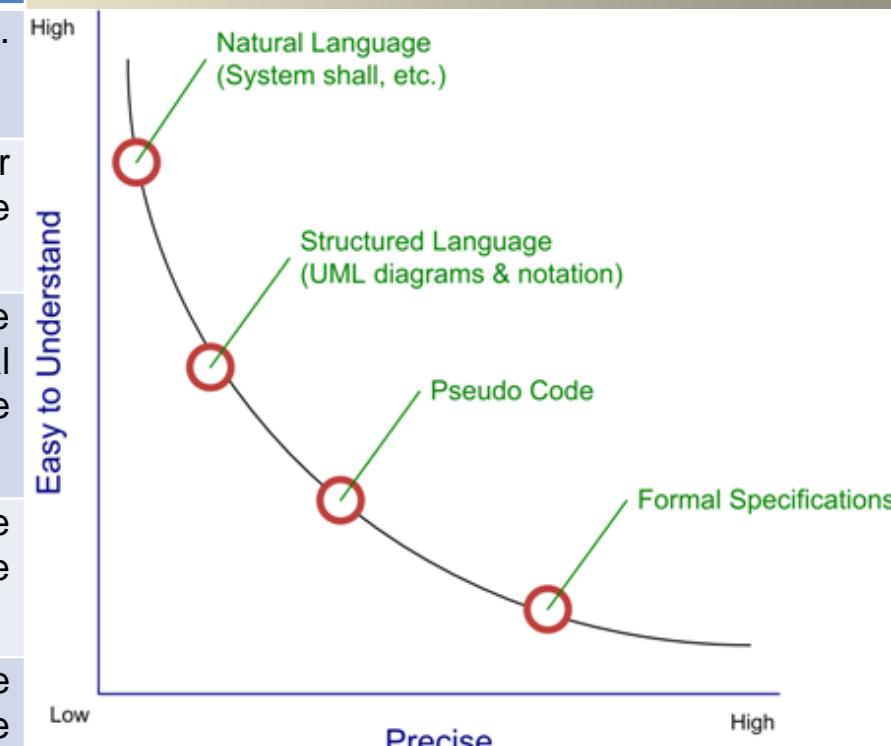
Requirements Engineering process



Requirements Specification

System requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract



Graphical Notations

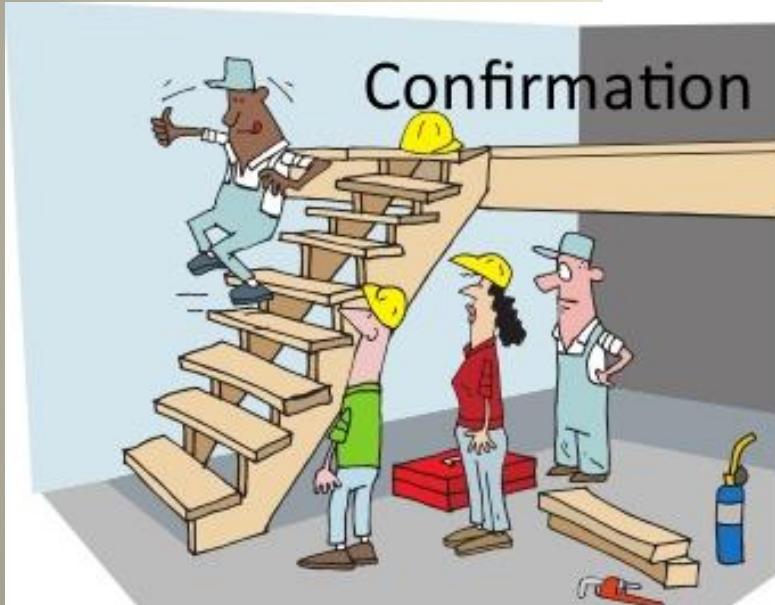
- UML
 - Use case Diagrams
 - Activity Diagrams
- User Stories
 - <https://www.youtube.com/watch?v=LGeDZmrWwsW> (Agile)
 - <https://www.youtube.com/watch?v=6q5-cVeNjCE> (Agile)
 - <https://youtu.be/502ILHjX9EE>(*Agile)
 - <http://www.agileacademy.com.au/agile/knowledgehub>

User Story

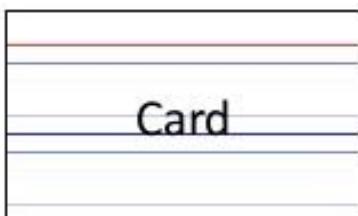
A User Story

Is a concise description of a functionality that will be valuable to a user of a system.





Confirmation Card Conversation



A large vertical title "Confirmation Card Conversation" is displayed. To the left of the main text is a small icon of a lined card with the word "Card" written on it.



Card

- Size
- Format



As a <user role>

I want <goal>

so that <benefit>.

As **who** I want
what so that
why

Example – Online Banking System

- As a Customer I want to view account summary online so that I do not have to wait till the month end to view the statement.
- As an Employee I want to add new customers online so that it saves my time.
- As a User I want to update profile details so that my details are up-to-date

Activity

- Write two user stories for the Library System
 - Be creative
 - Think about the role

Compare

A	B
As a recruiter I want to review resumes from applicants to one of her ads.	As a recruiter I want to manage the ads she has placed.

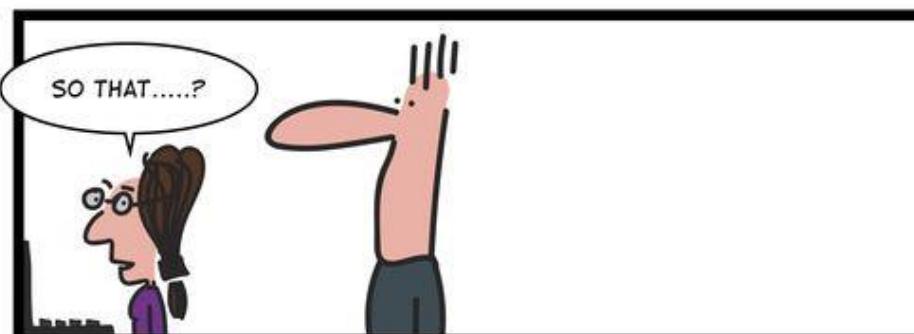
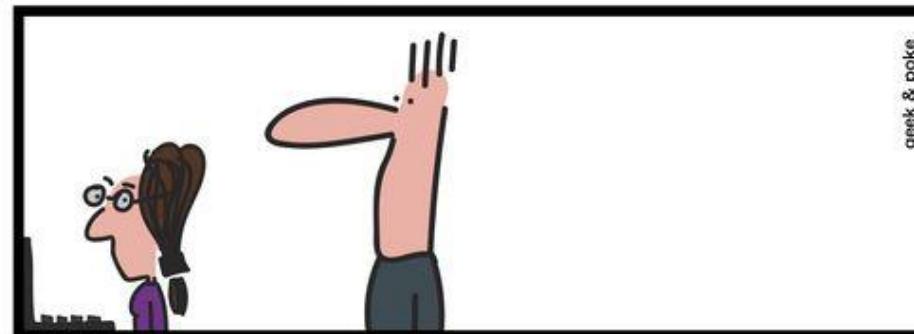
Compare

A	B
As a driver I want to find the store with the shortest drive time so I can get there quickly.	As a driver I want to find directions to a store in Google Maps so I can get there quickly.

Compare

A	B
As a user I want to have my previous orders stored in the database so they will be there permanently	As a repeat customer I want to access old orders so that I can quickly purchase the same order again.

Everyday User Stories

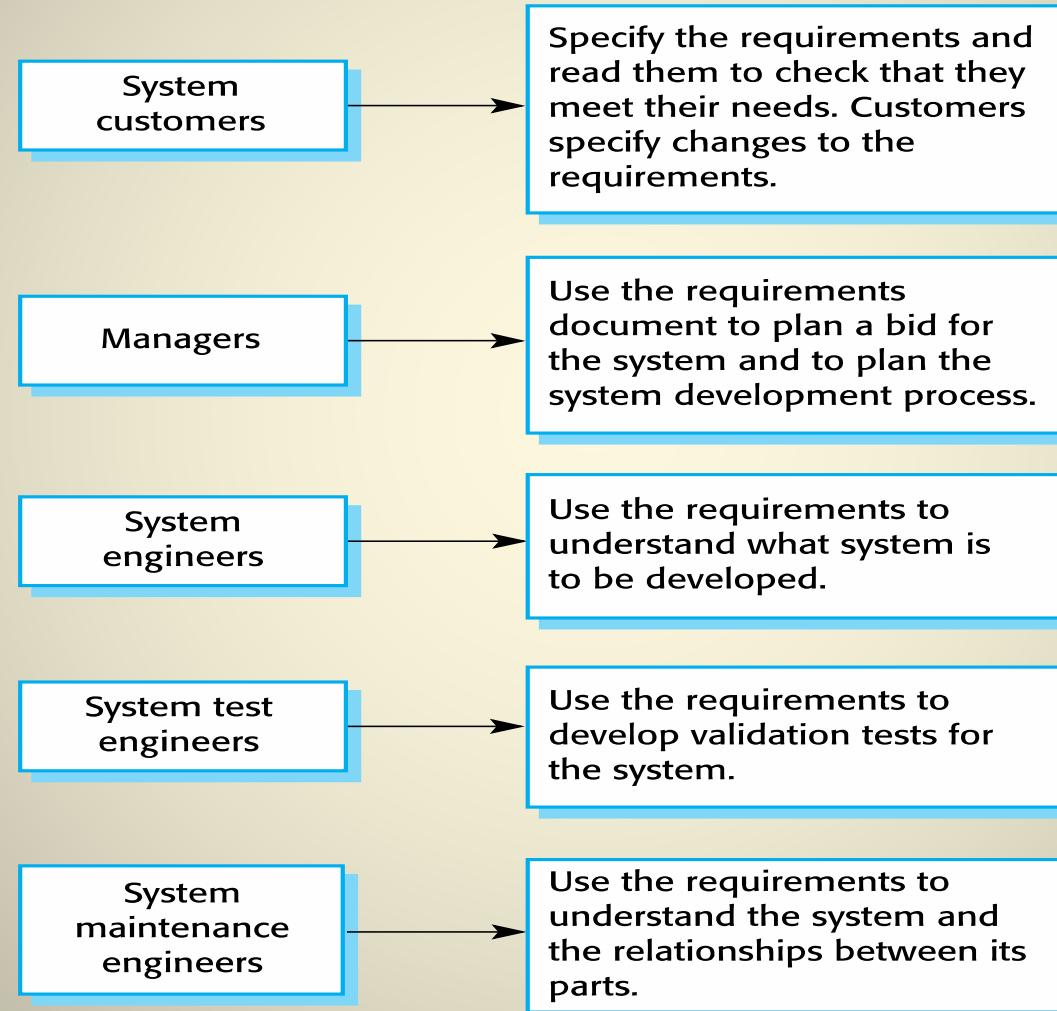


MAKE SURE YOUR USER STORY IS CORRECTLY PHRASED

SRS

- Software Requirements Specification
- The software requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of **WHAT** the system should do rather than **HOW** it should do it.

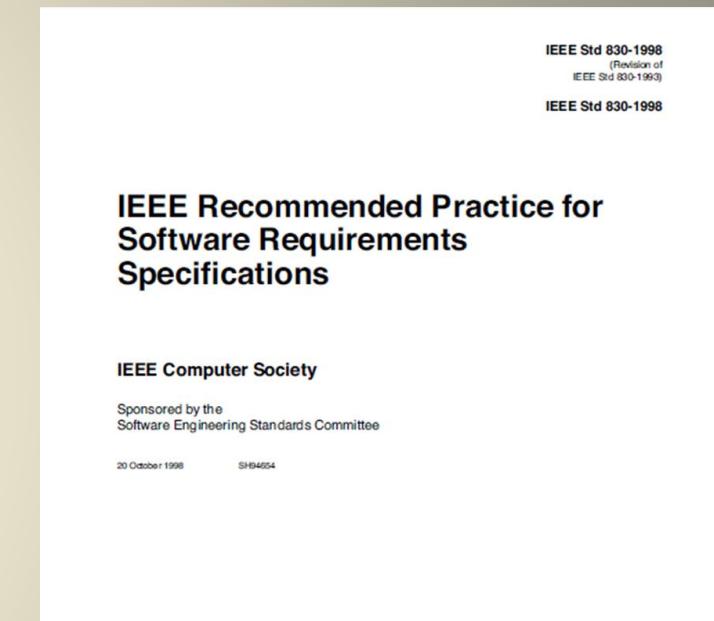
Users of SRS



IEEE SRS Template.pdf

Role of SRS

1. Should correctly define all of the software requirements.
2. Should not describe any design or implementation details.
3. Should not impose additional constraints on the software.



SRS Template

- Introduction
 - Purpose, Scope , Overview
- General Description
 - Product Perspective, User Characteristics
- Specific Requirements
 - Functional Requirements
 - Non-functional
 - Constraints

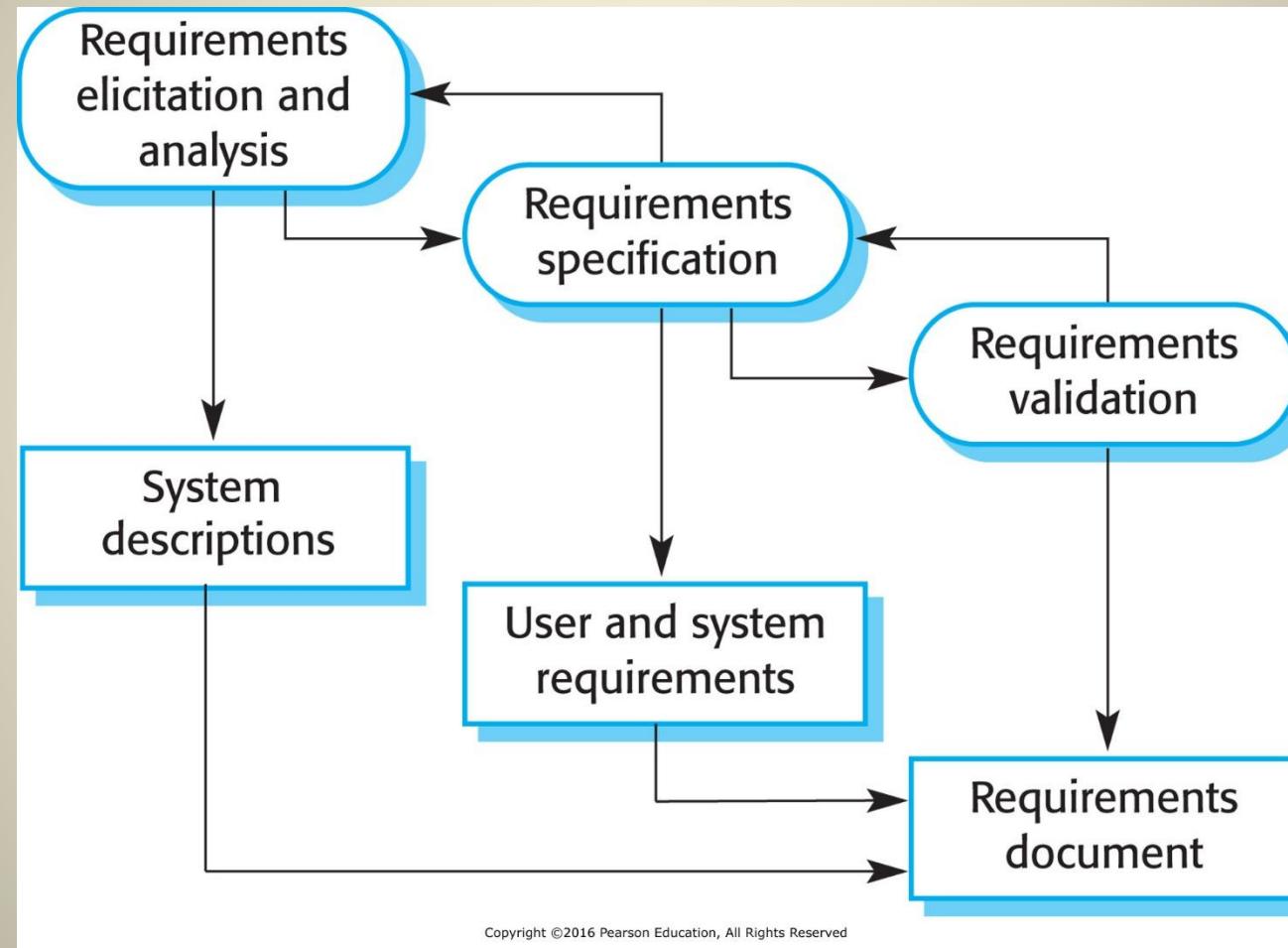
SRS for Library Management System

- Introduction
 - Library Management System (LMS) project aims to develop a computerized system to maintain all the daily work of the SLIIT library. This system will help the users to manage the library daily activities in electronic format.
- The General Description
 - LMS is an online system which would give SLIIT staff members as well as students easy access to the Library contents.

SRS for Library Management System

- Functional Requirements
 - Story cards, Use Case Diagram, Activity Diagrams
- Non-functional Requirements
- Constraints
 - Budget constraints
 - Technology Constraints

Requirements Engineering process



Requirements Validation

Requirements validation

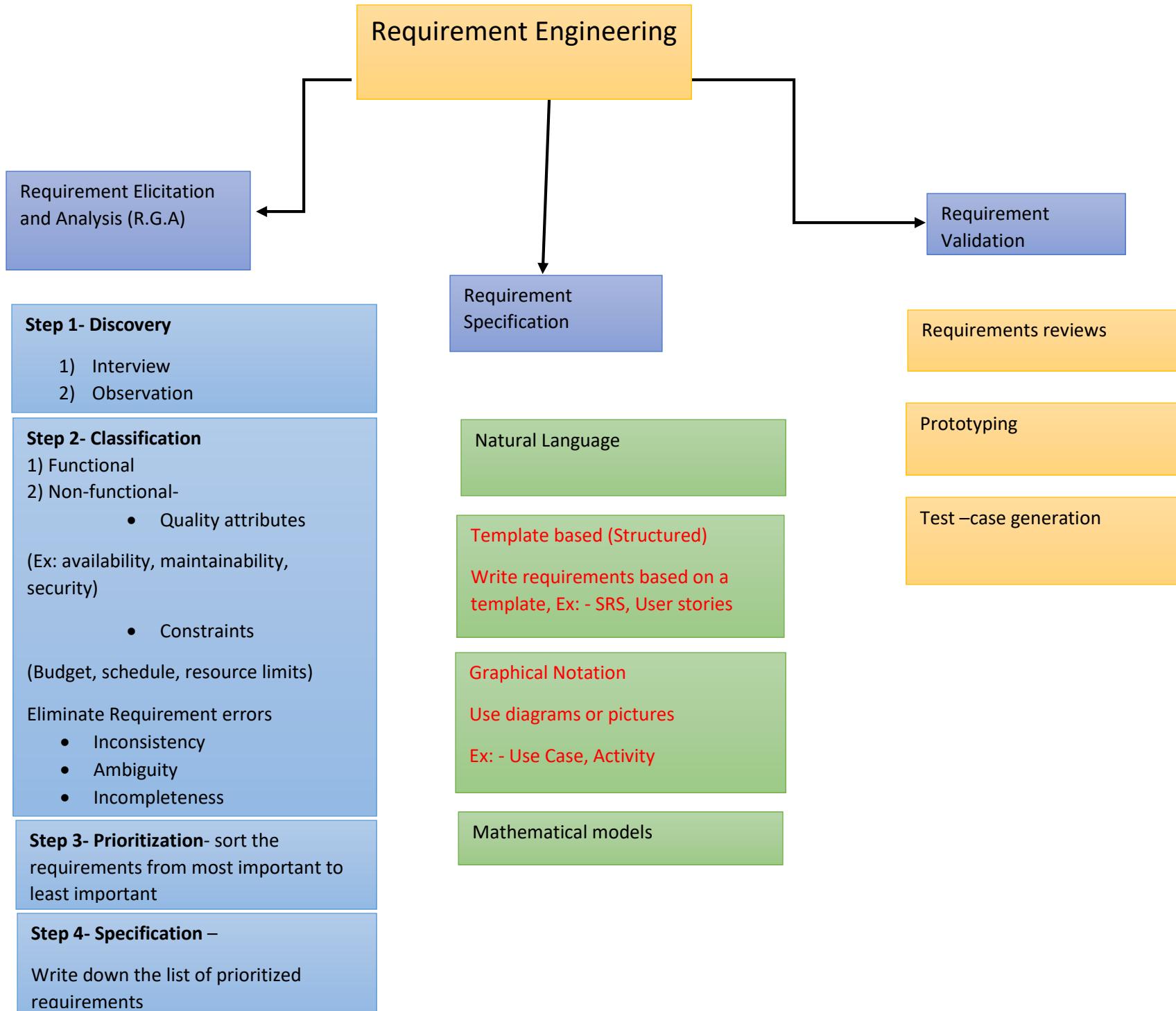
- Validate whether the elicited requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important.
- Requirements checking
 - Validity
 - Precise requirements
 - Realism
 - Verifiability

Requirements validation techniques

- Requirements reviews
 - Systematic manual analysis of the requirements.
- Prototyping
 - Using an executable model of the system to check requirements.
- Test-case generation
 - Developing tests for requirements to check testability.
 - Will be discussed later

References

- Software Engineering – 10th Edition by Ian Sommerville, Chapter 4
- <http://iansommerville.com/systems-software-and-technology/>
- <http://iansommerville.com/software-engineering-book/>
- IEEE Recommended Practice for Software Requirements Specifications - IEEE Std 830-1998



Use Case Diagrams

Software Process Modeling

Session Outcomes

- Introduction
- Components of a Use case diagram
 - System
 - Actors
 - Use cases
 - Relationships
- Applying use case diagrams in real world applications
- Use case scenarios

Requirements Specification

- Structured Natural Language
 - User Stories
- Mathematical Specifications
 - Decision Trees
 - Decision Tables
- Graphical Notations
 - **Use Case Diagrams and Use Case Scenarios**
 - Activity Diagrams

What is a Use Case Diagram?

- Use Case Model;
 - Graphically represent the proposed functionality of the new system.
 - Use Case Model captures the functional requirements of a system.
 - Help to demonstrate the high-level behavior of the proposed system to the clients

Use Cases for Requirements Engineering

- Use case modelling support **requirements elicitation**
- Use cases act as a means of **communicating with stakeholders** about what the system is intended to do.
 - It is an excellent way to communicate to management, customers, and other non-development people:
 - WHAT** a system will do when it is completed.
 - But....it does not go into detail of **HOW** a system will do anything.

Components of a Use Case Diagram

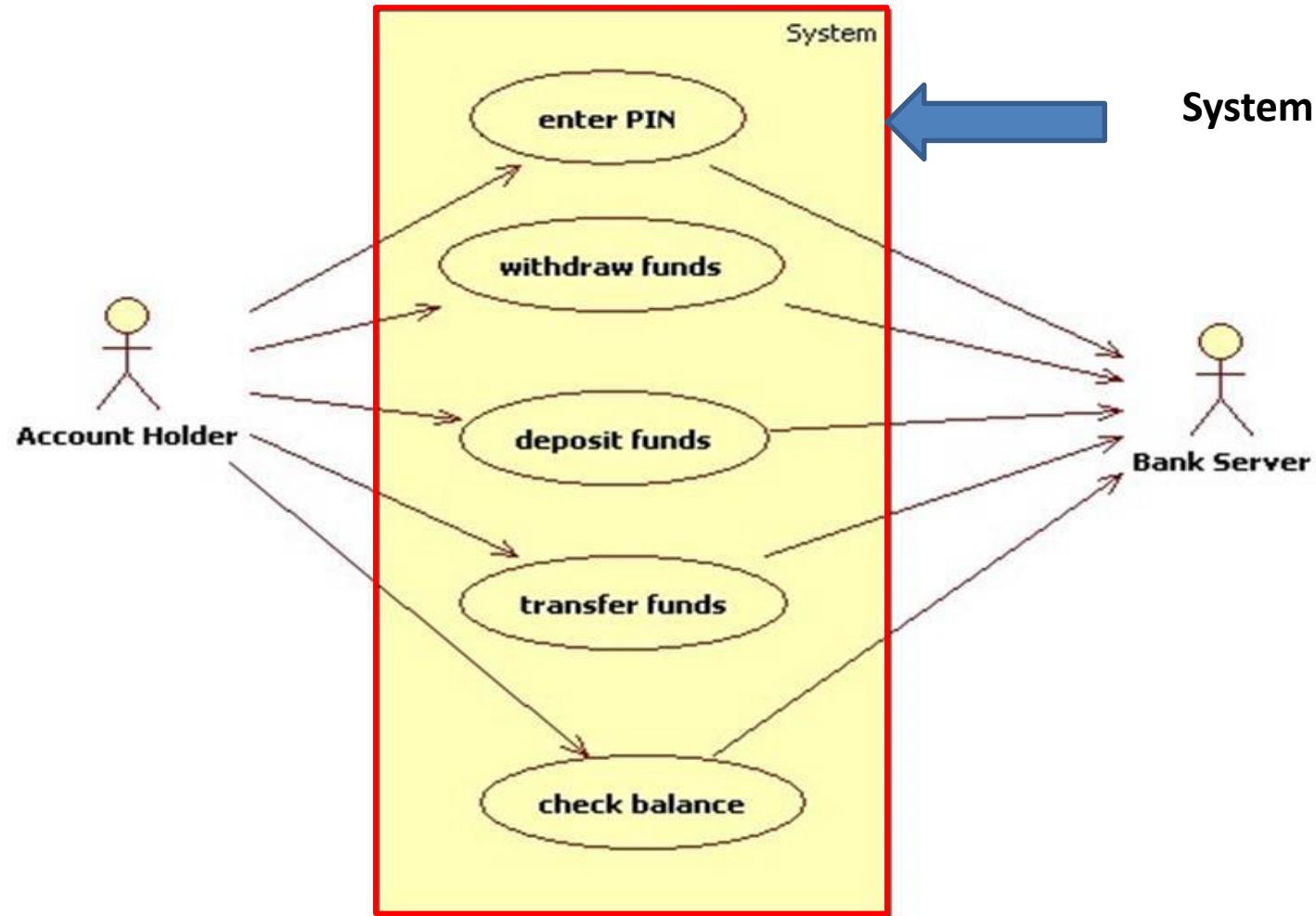
- To construct a Use Case diagram, there are **FOUR** basic components.
 - System**: something that performs function(s).
 - Actors**: the roles adopted by those participating.
 - Use Cases**: high level activities to be supported by the system.
 - Relationships / Links**: which actors are involved in which use cases (dependency, generalization, and association).

1) System

- **System** is something which perform function(s).
- **System Boundary** Represents the boundary between the (physical) system and the actors who interact with the (physical) system.



System - example

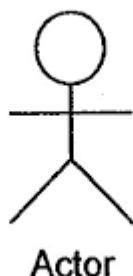


System Boundary

2) Actors

- A Use Case Diagram shows the interaction between the system and entities **external** to the system. These external entities are referred to as **Actors**.
- Actors represent **roles** which may include **human users**, **external hardware** or **other systems**.
- Actors have **direct interactions** with the system

- Notation →



Actor

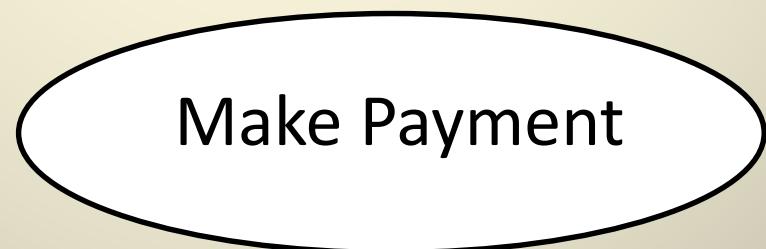
Activity

- Identify Actors of the SLIIT Library system.



3) Use Case

- A Use Case is a **unit of behavior** in the proposed system
- It represents a unit of interaction between a user and the proposed system.
- Use case name typically has a verb-noun phrase
- Notation →



How to Identify a Use case?

- Consider what each actor requires of the system.
- For each actor, human or not, ask yourself the following questions in order to figure out the relevant use cases.
 - What are the primary tasks the actor wants the system to perform?
 - Will the actor create, store, change, remove, or read data in the system?
 - Will the actor need to inform the system about sudden, external changes?
 - Does the actor need to be informed about certain occurrences in the system?
 - Will the actor perform a system start-up or shutdown?

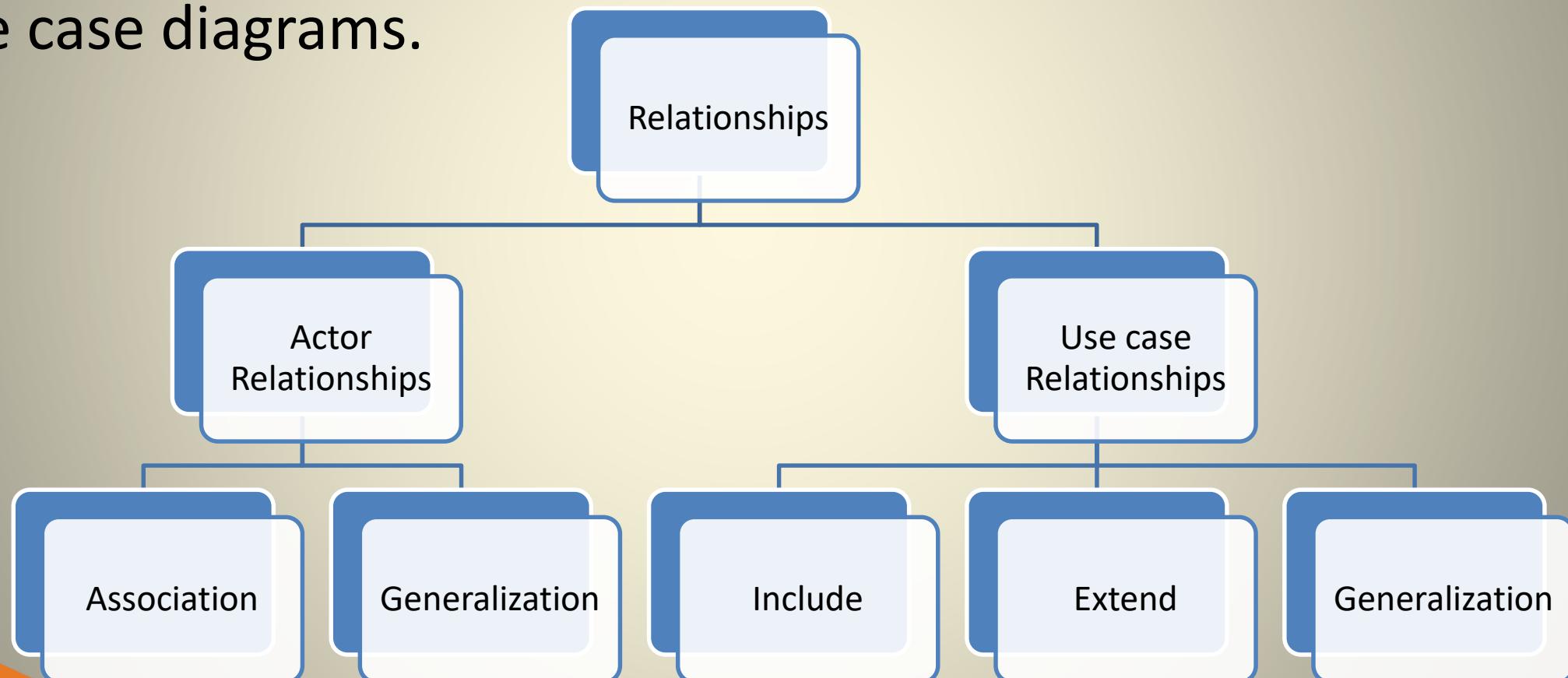
Activity

- Identify use cases for each of the actors in the SLIIT Library System.



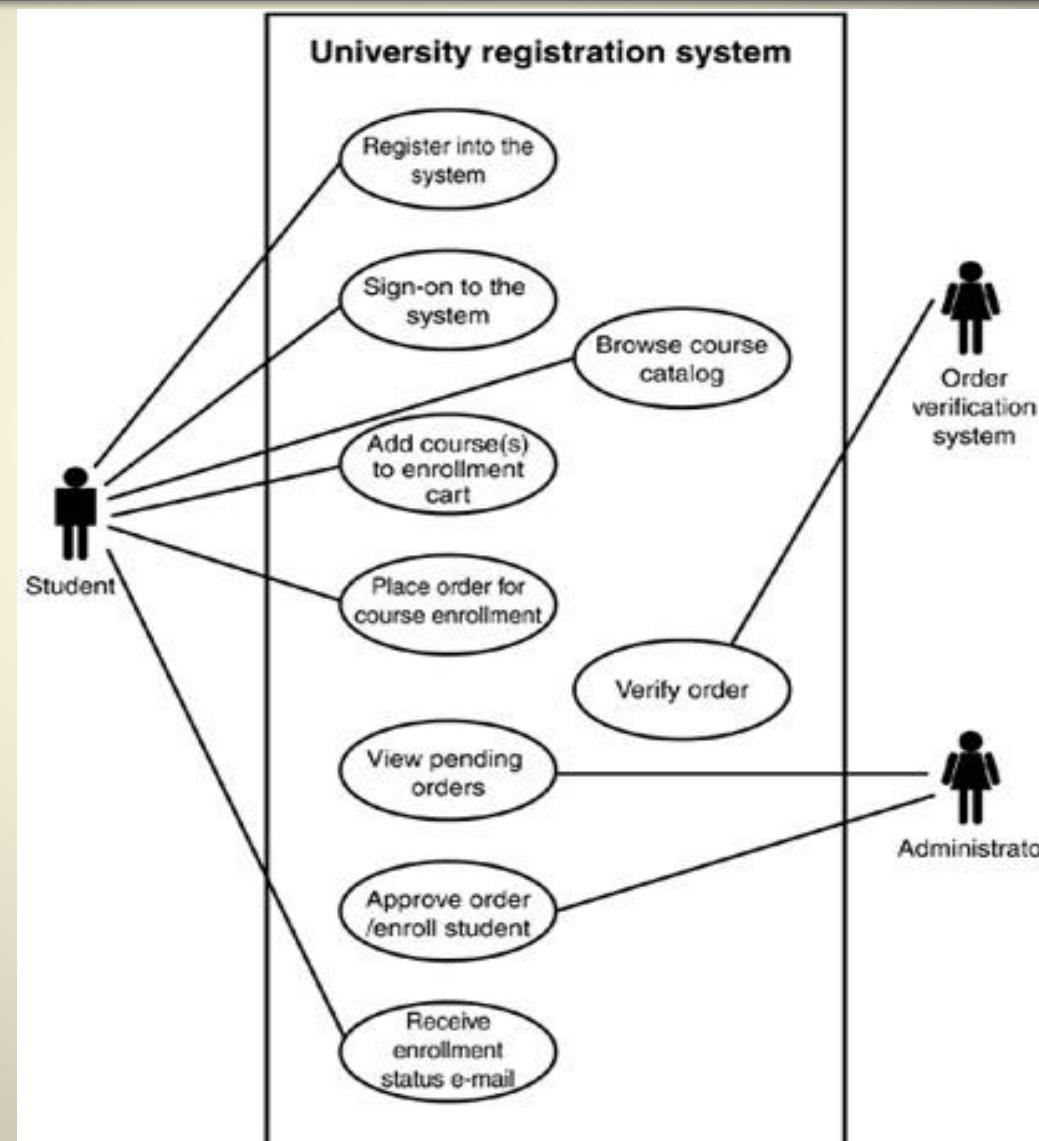
4) Relationships

- Below mentioned are the main types of relationships used in use case diagrams.



Association.

- indicates that an actor participates in (i.e. communicates with) the use case.



Activity

- Draw the Actors and Associations for the SLIIT Library System

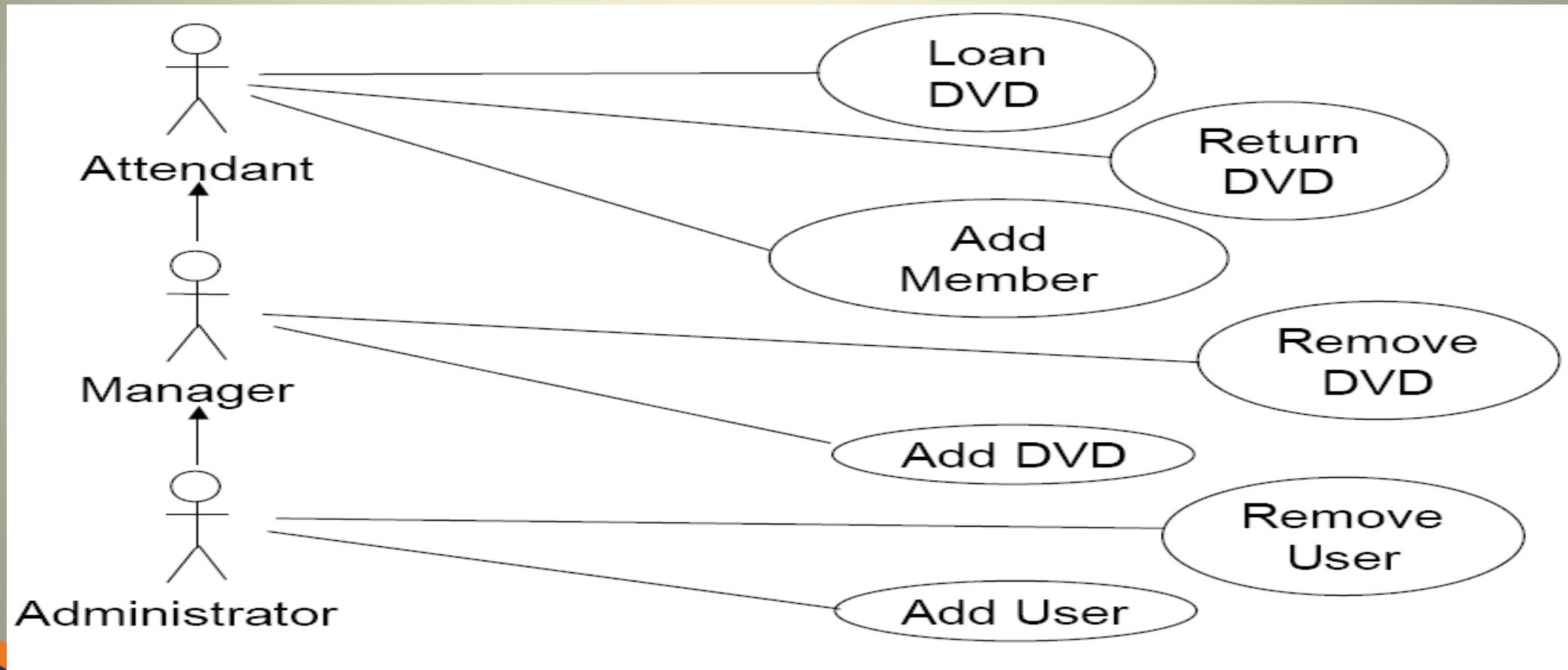


Actor to Actor Relationships

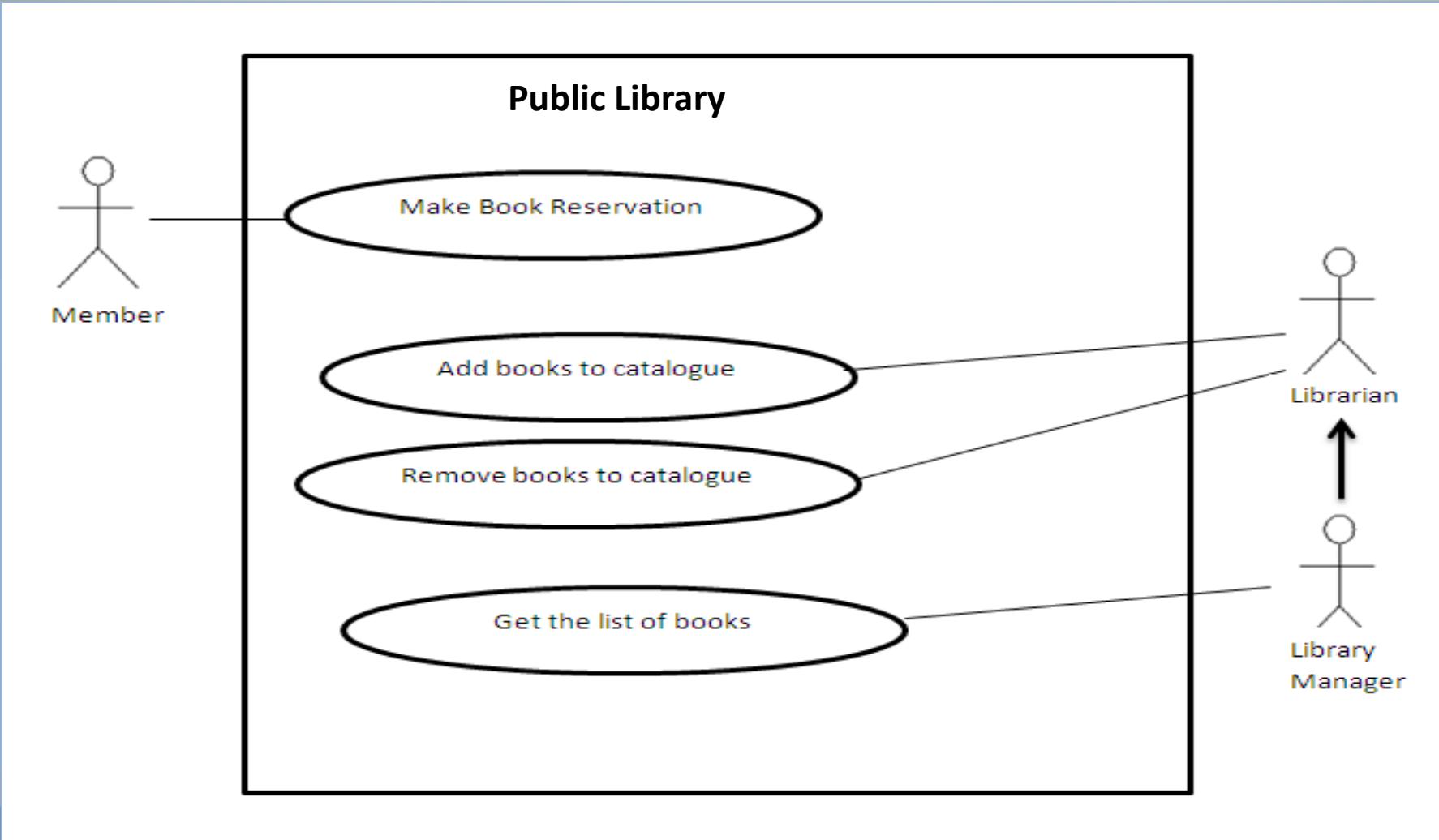
Generalization.

- Actor Generalization is drawn from the concept of inheritance in Object Oriented Programming.
- A **child actor** Inherits all of the characteristics and behavior of the **parent actor**.
- Can add , modify, or ignore any of the characteristics and behaviors of the parent actor.

Who has the most rights in the system?



Example – Public Library



Activity

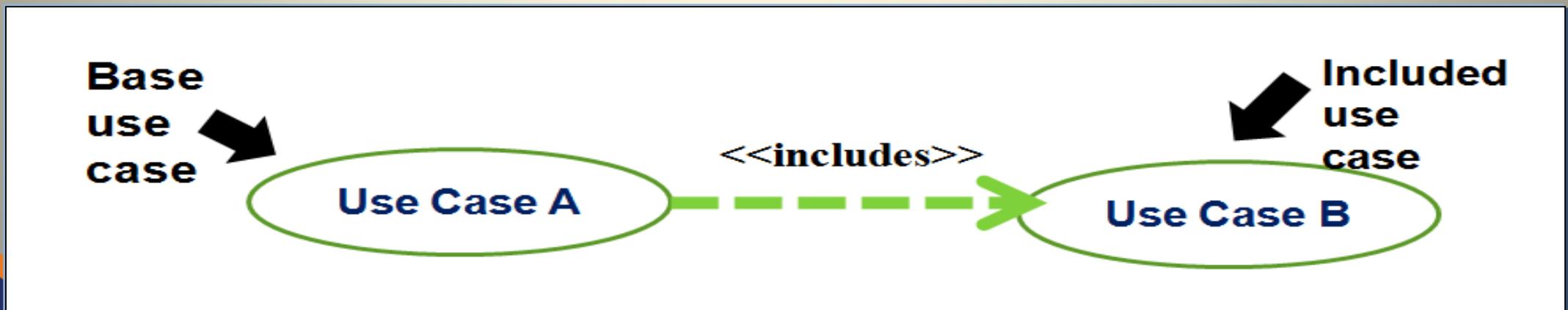
- Draw the actor to actor relationships for the SLIIT Library System



Include Relationship

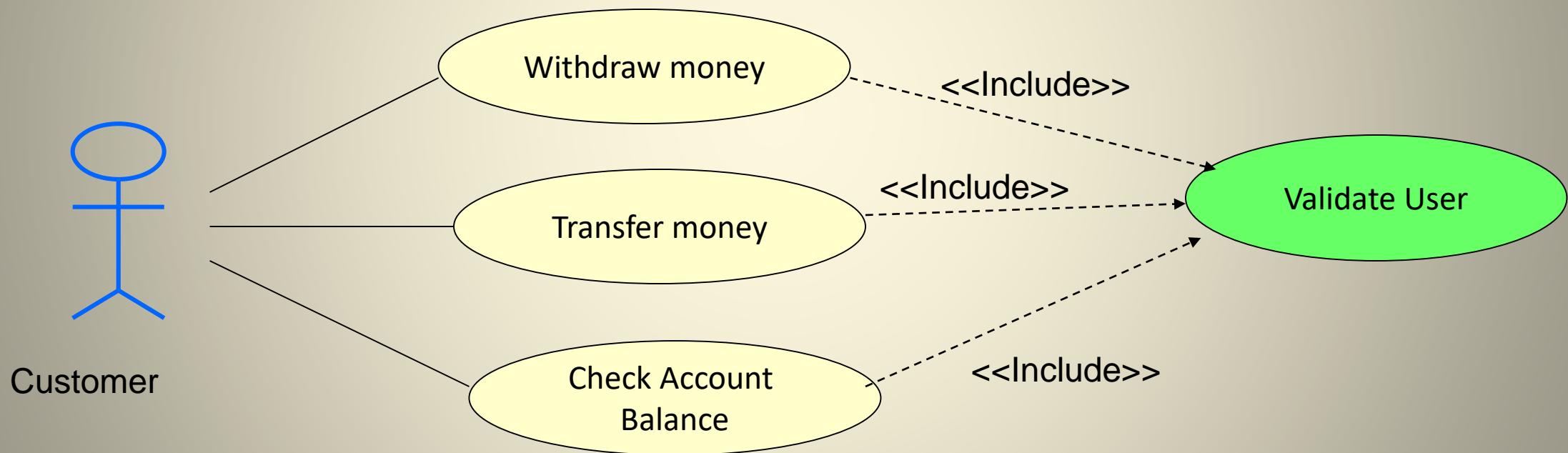
1) Include

- The base use case explicitly incorporates the behavior of another use case at a location specified in the base.
- The included use case never stands alone. It only occurs as a part of some larger base that includes it.



Include Relationship

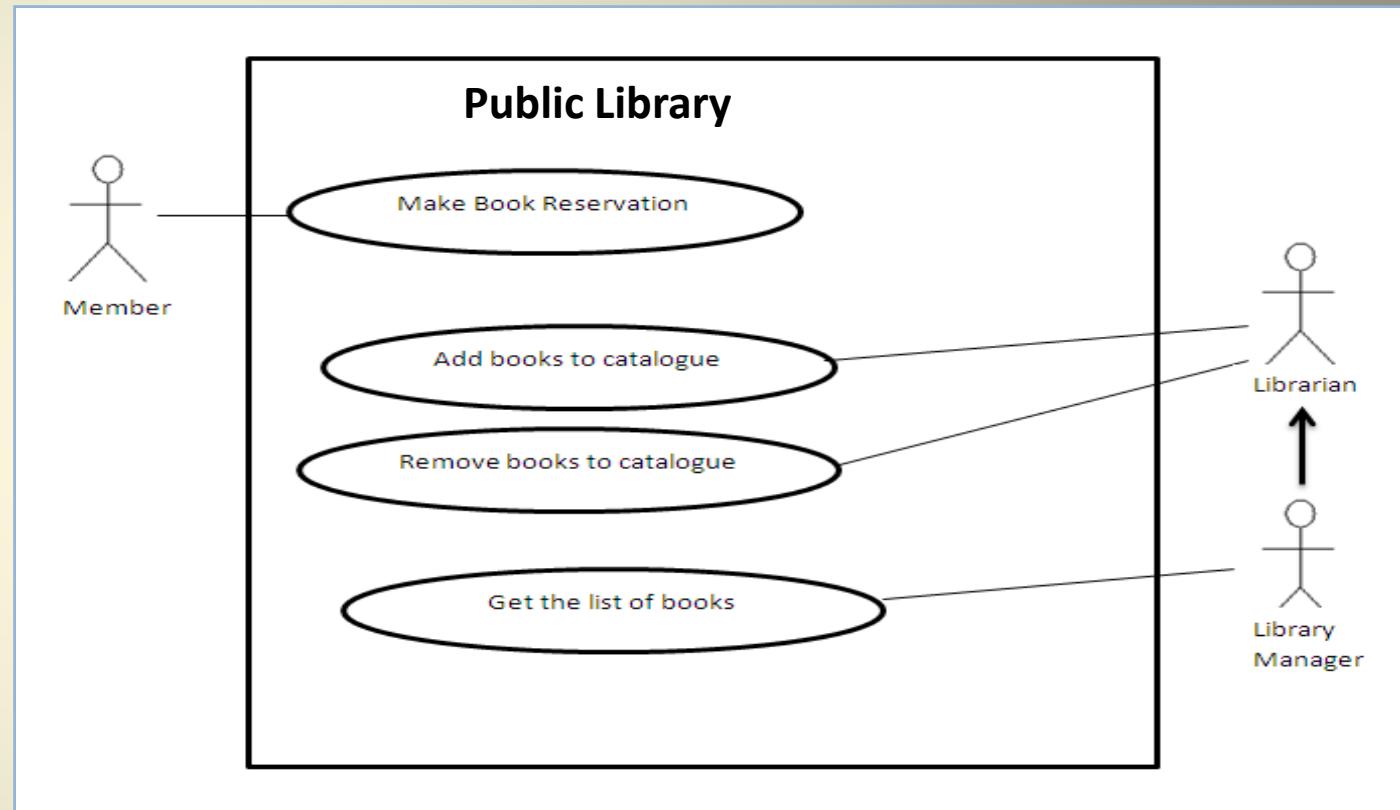
- Enables us to avoid describing the same flow of events several times by putting the common behavior in a use case of its own.



Activity

Update the use case diagram of the Public Library for the below given criteria.

When member is reserving the books he/she has to login to the system.



Activity

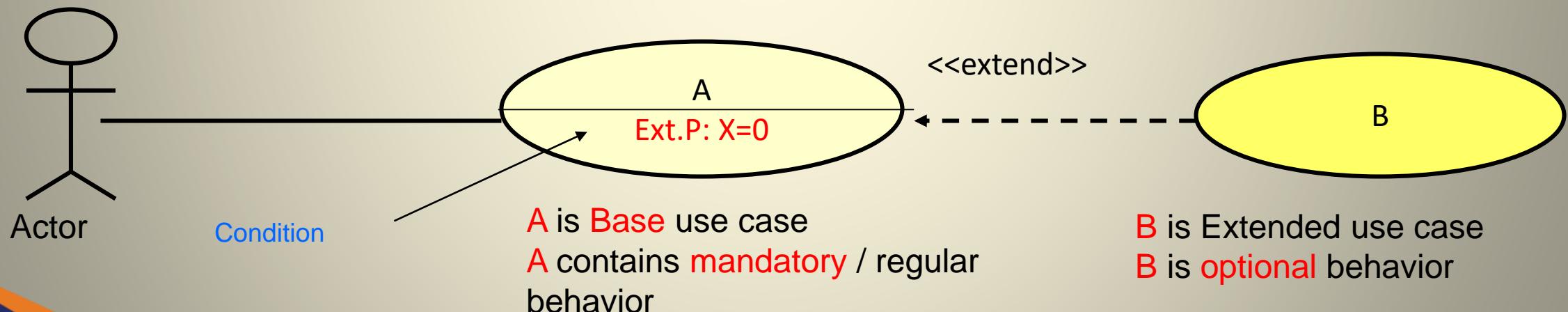
- Draw the **include** relationships between the use cases for the SLIIT Library System



Extend Relationship

2) Extend

- The base use case implicitly incorporates the behavior of another use case at certain points called extension points.
- The base use case may stand alone, but under certain conditions its behavior may be extended by the behavior of another use case.



Extend Relationship

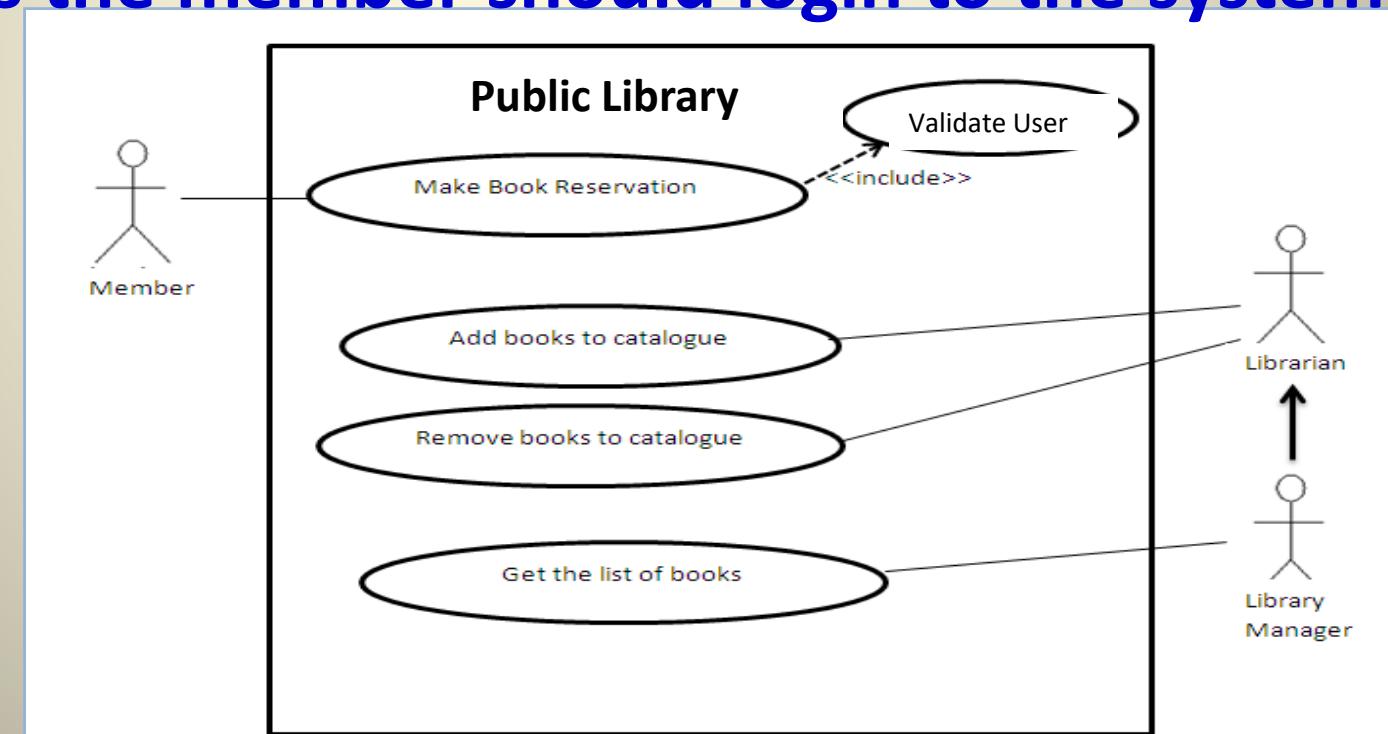
- Eg:- When a student get enrolls in the university they perform a visa check if he/she is a foreign student.



Activity

Update the Public Library for the below given criteria.

Member can renew the books he/she has borrowed. When renewing if book has exceeded the loan period a fine will be calculated. For renewing purposes the member should login to the system.



Activity

- Draw the extends relationships for the SLIIT Library System



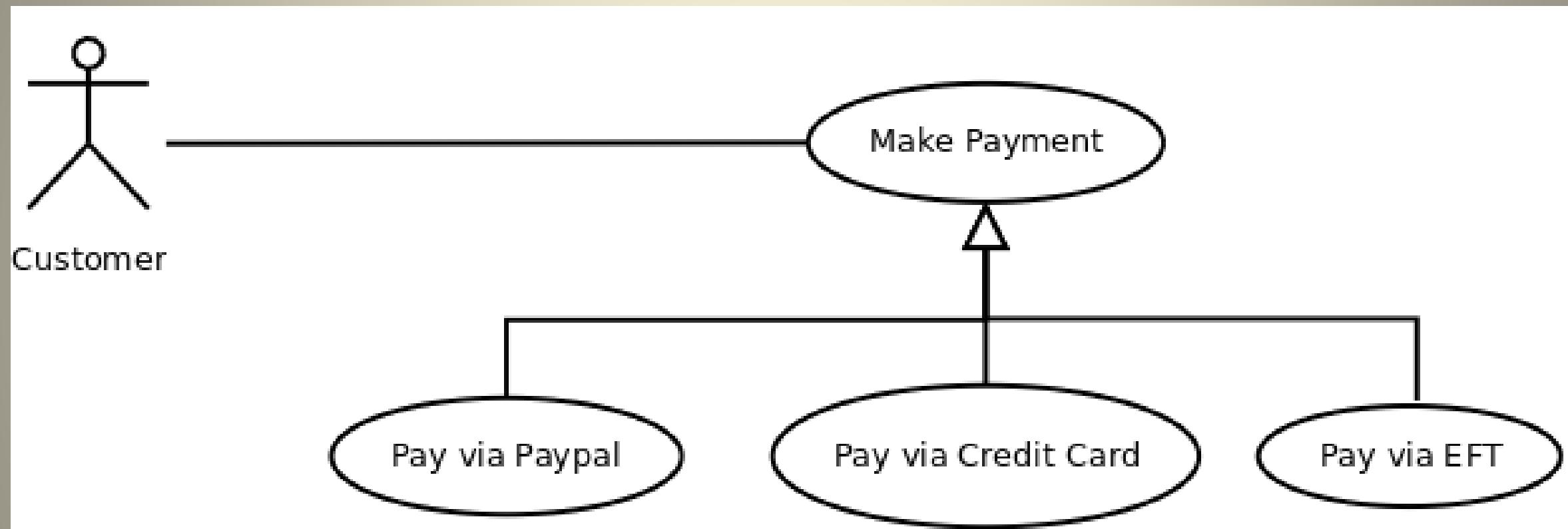
Generalization Relationship

3) Generalization

- The child use case inherits the behavior and meaning of the parent use case.
- The child may add to or override the behavior of its parent.



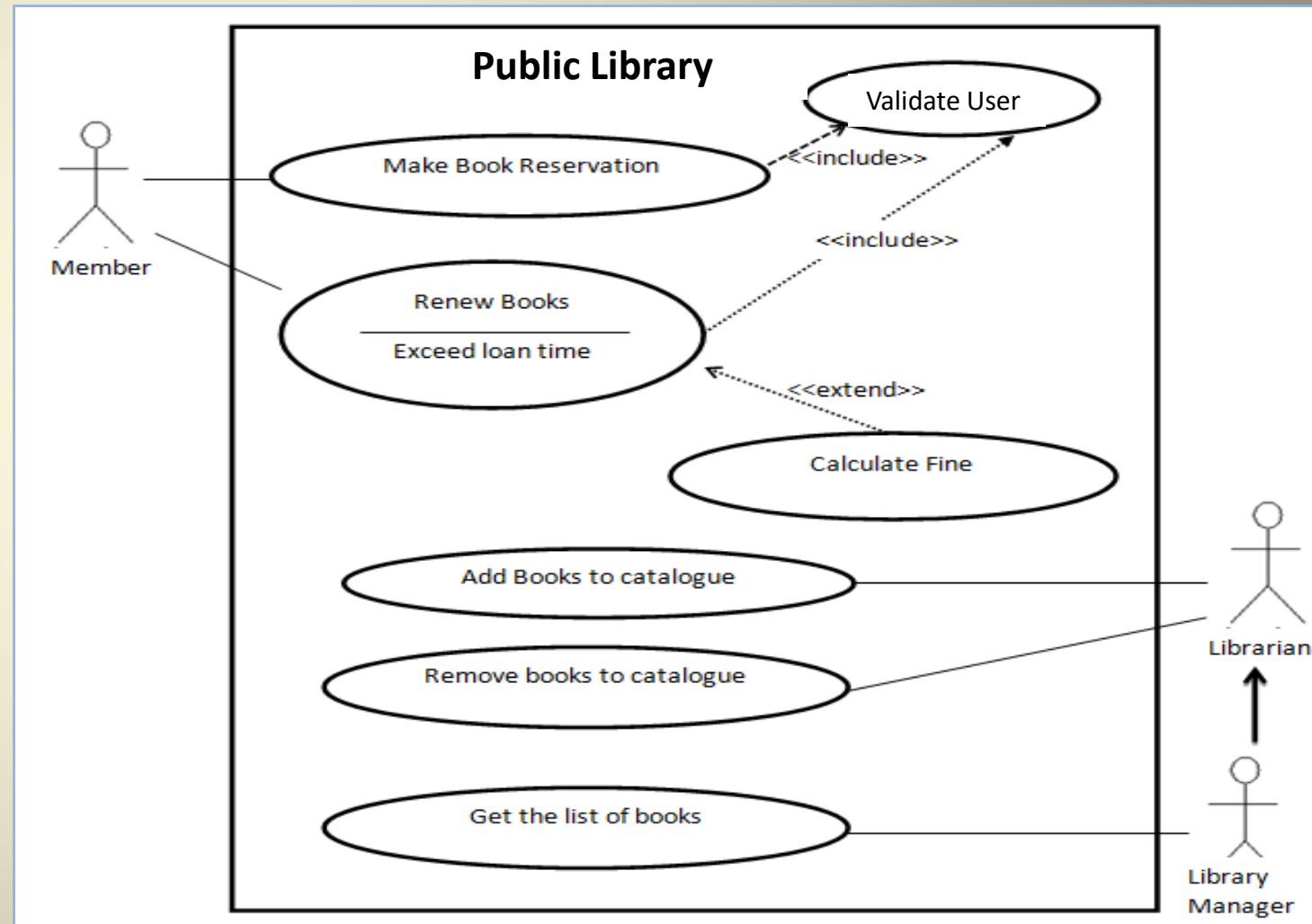
Generalization Relationship



Activity

Update the use case diagram of the Public Library for the below given criteria.

Library Manager can generate reports of the Borrowed books, Overdue books at the end of each month.



Relationship Summary

Table 6-1: *Kinds of Use Case Relationships*

<i>Relationship</i>	<i>Function</i>	<i>Notation</i>
association	The communication path between an actor and a use case that it participates in	_____
extend	The insertion of additional behavior into a base use case that does not know about it	«extend» - - - - →
include	The insertion of additional behavior into a base use case that explicitly describes the insertion	«include» - - - - →
use case generalization	A relationship between a general use case and a more specific use case that inherits and adds features to it	→

Use Case Scenarios

- A Scenario is a formal description of the flow of events that occur during the execution of a Use Case instance. It defines the specific sequence of events between the system and the external Actors.
- There is usually a **Main scenario**, which describes what happens when everything goes to plan. It is written under the assumption that everything is okay, no errors or problems occur, and it leads directly to the desired outcome of the use-case.

Use Case Scenarios

- Other scenarios describe what happens when variations to the Main scenario arise, often leading to different outcomes.
- So the flow of events should include:
 - How and when the use case starts and ends
 - When the use case interacts with the actors
 - What objects are exchanged
 - The basic flow and
 - Alternative flows (exceptional) of the behavior.

Use Case Sample Template

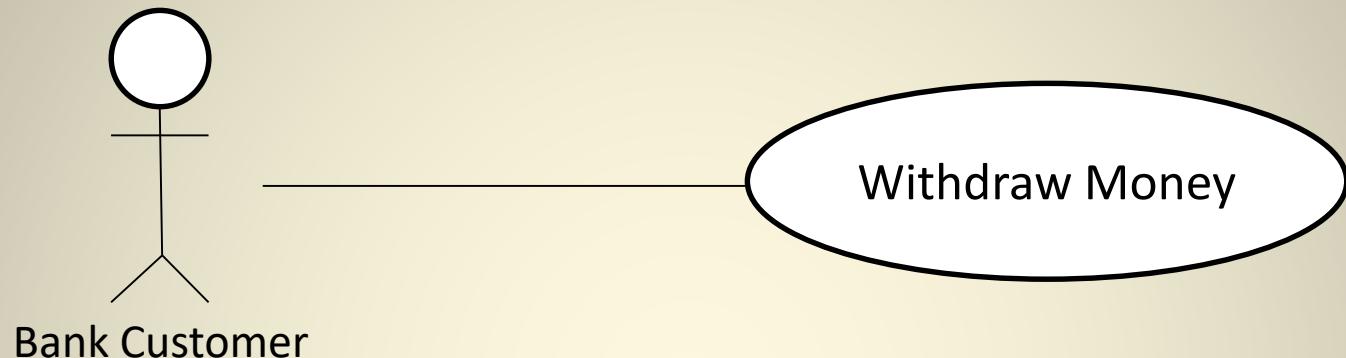
1. Use Case ID and name
2. Characteristic Information
 - » Goal in Context
 - » Scope
 - » Level
3. Pre-Conditions
4. Primary Actor
5. Main Success Scenario Steps
6. Extensions
7. Optional Information

Use Case Specification Template*

Number	<i>Unique use case number</i>	
Name	<i>Brief noun-verb phrase</i>	
Summary	<i>Brief summary of use case major actions</i>	
Priority	<i>1-5 (1 = lowest priority, 5 = highest priority)</i>	
Preconditions	<i>What needs to be true before use case “executes”</i>	
Postconditions	<i>What will be true after the use case successfully “executes”</i>	
Primary Actor(s)	<i>Primary actor name(s)</i>	
Secondary Actor(s)	<i>Secondary actor name(s)</i>	
Trigger	<i>The action that causes this use case to begin</i>	
Main Scenario	Step	Action
	Step #	<i>This is the “main success scenario” or “happy path.”</i>
	...	<i>Description of steps in successful use case “execution”</i>
	...	<i>This should be in a “system-user-system, etc.” format.</i>
Extensions	Step	Branching Action
	Step #	<i>Alternative paths that the use case may take</i>
Open Issues	<i>Issue #</i>	<i>Issues regarding the use case that need resolution</i>

*Adapted from A. Cockburn, “Basic Use Case Template”

Use Case Specification Template Example



Number	1
Name	Withdraw Money
Summary	User withdraws money from one of his/her accounts
Priority	5
Preconditions	User has logged into ATM
Postconditions	User has withdrawn money and received a receipt
Primary Actor(s)	Bank Customer

Continued ...

Trigger	User has chosen to withdraw money	
Main Scenario	Step	Action
	1	System displays account types
	2	User chooses account type
	3	System asks for amount to withdraw
	4	User enters amount
	5	System debits user's account and dispenses money
	6	User removes money
	7	System prints and dispenses receipt
	8	User removes receipt
	9	System displays closing message and dispenses user's ATM card
	11	User removes card
	10	System displays welcome message
Extensions	Step	Branching Action
	5a	System notifies user that account funds are insufficient
	5b	System gives current account balance
	5c	System exits option
Open Issues	1	Should the system ask if the user wants to see the balance?

Activity

- Write a Use Case Scenario for “**Borrowing a Book**”

You could consider the process given below as the manual system procedure.

The member identifies him or herself to the librarian and indicates which books they wish to borrow.

If it is acceptable for them to borrow these books, i.e. they are not marked “for reference only”, or the number of books on loan to the customer is less than some predetermined maximum, then the books are loaned to the customer for a specified loan period.

The members loan record is updated to reflect the loaned books. The libraries card index system is updated to show who has borrowed the books.

References

- *Writing Effective Use Cases*
 - By Dr. Alistair Cockburn
- UML 2 Bible

SOFTWARE PROCESS MODELING

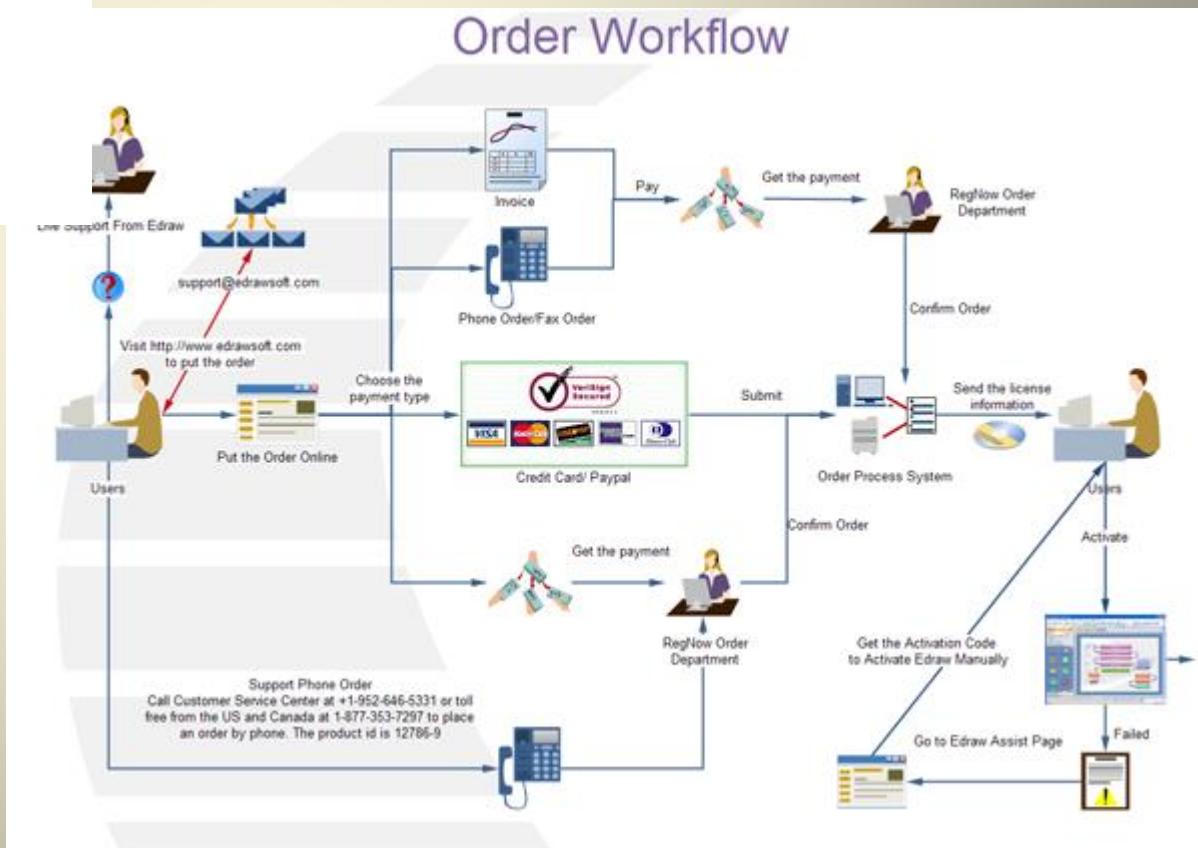
Activity Diagrams

Session Outcomes

- Introduction.
- Elements of Activity diagram.
- Partitioning an activity diagram.
- Applying activity diagrams in real world applications.

What is an activity diagram?

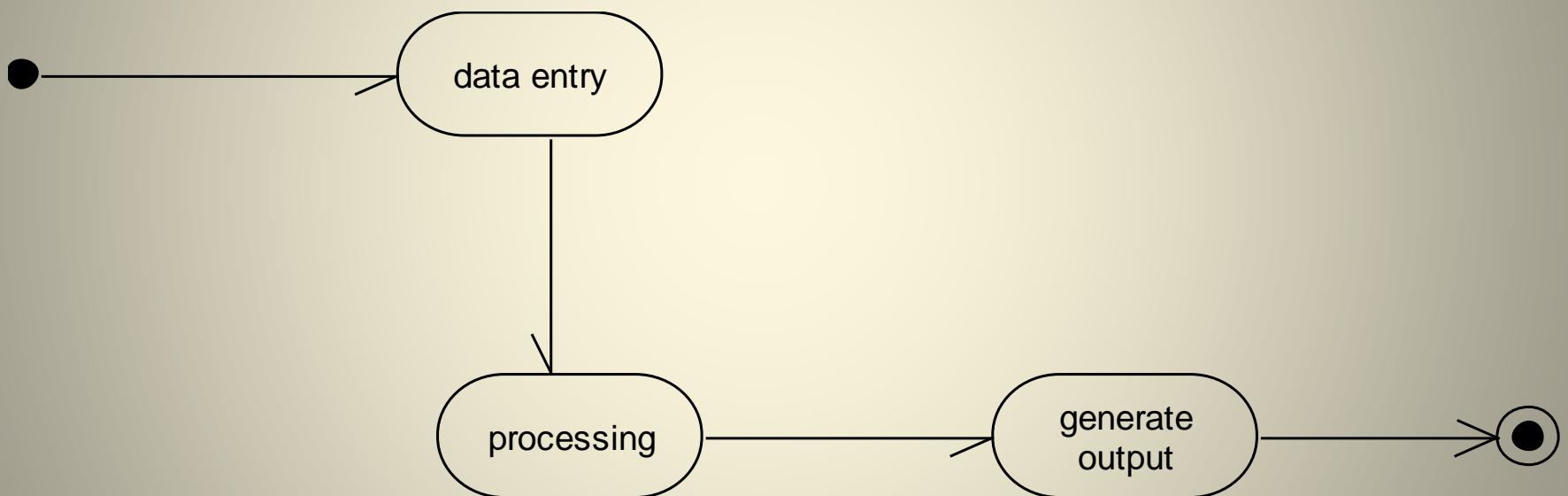
- An Activity Diagram models
 - Activities of a system.
 - Dependencies between activities.
 - Represents Workflows.



When to Use Activity Diagrams ?

- ❖ Modeling **business processes**.
- ❖ Analyzing **system functionality** to identify the use cases.
- ❖ Analyzing **individual use cases** in detail.
- ❖ Clarifying **concurrency issues**.

A simple Activity Diagram



Elements of an Activity Diagram

- ❖ Action
 - Simple Action
 - Call Action
- ❖ Transition
- ❖ Controls
 - Initial/Start
 - Final/End
 - Nodes
 - ✧ Decision/Branch
 - ✧ Merge
 - ✧ Fork
 - ✧ Join

Initial & Final Nodes

❖ Represent the beginning and end of a diagram respectively.

- Initial/Start Node – Filled Circle



- Final node is within circle



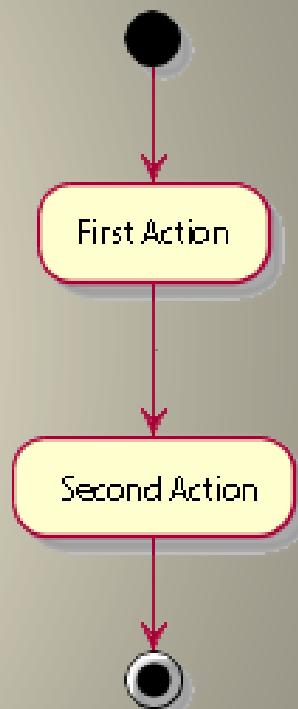
Action Node

- An action is some task which needs to be done.
- Each action can be followed by another action .
- Represented with a **rounded rectangle**.
- Text in the action box should represent an activity (**verb phrase** in present tense).
- An activity is a sequence of actions.

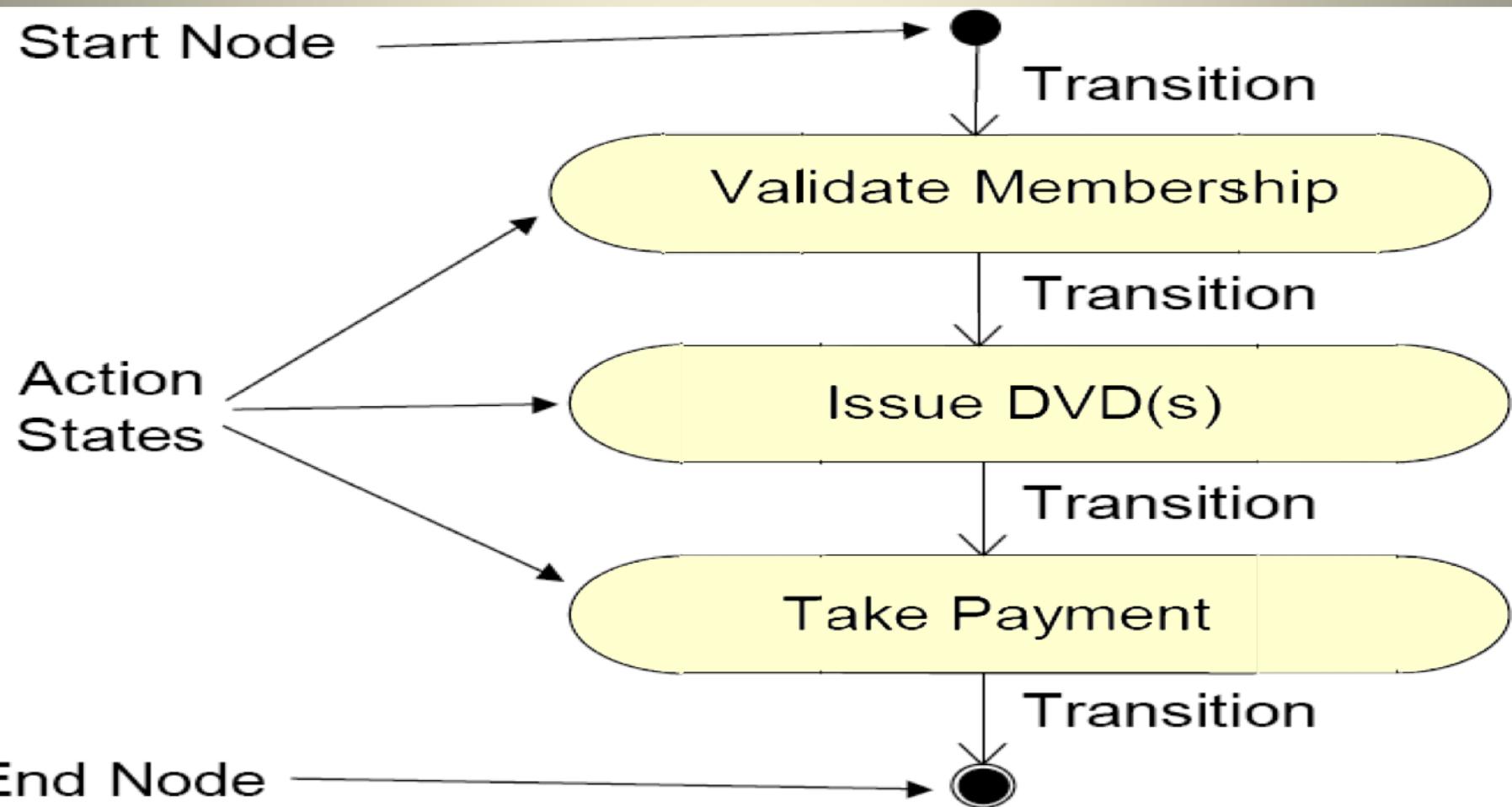
Generate Report

Transition

- Activity/Action nodes are connected by Transitions.
- Also known as a control flow/directed flow /edge
- When the execution of the node completes, execution proceeds to the node found on the output flow.



Activity Diagrams: Example



Activity - 1

Draw an activity diagram for becoming a member of the Library.

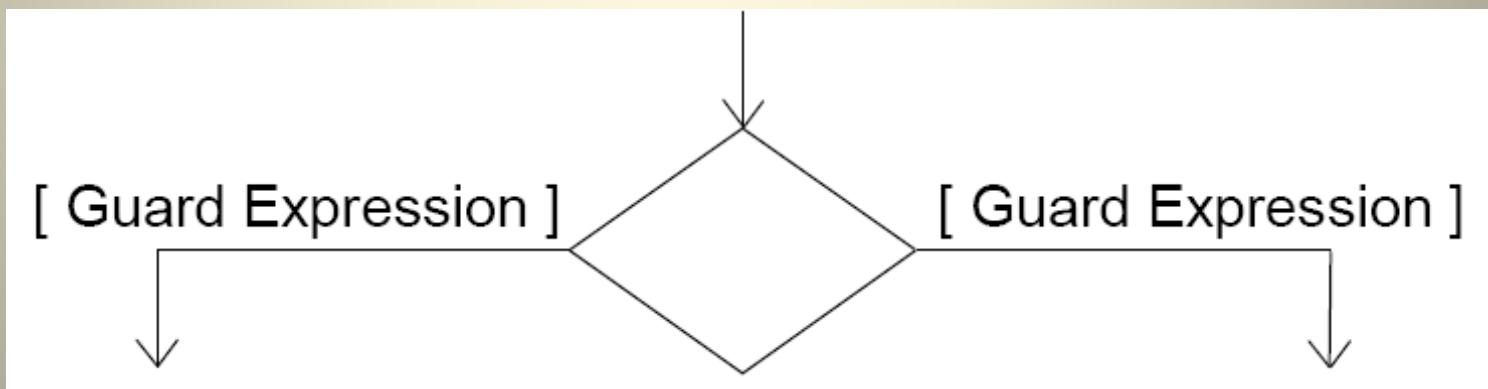
- To become a library member a student must keep a refundable security deposit of Rs.3000/=.
- Few registration forms should be filled in order to handle membership.
- Upon registration a member will be given a membership ID and a password.

Decision Node

- A Decision represents a **conditional flow of control**:
 - the alternatives are mutually exclusive.
- Similar to IF/ELSE statements in programming languages like C/C++/Java.

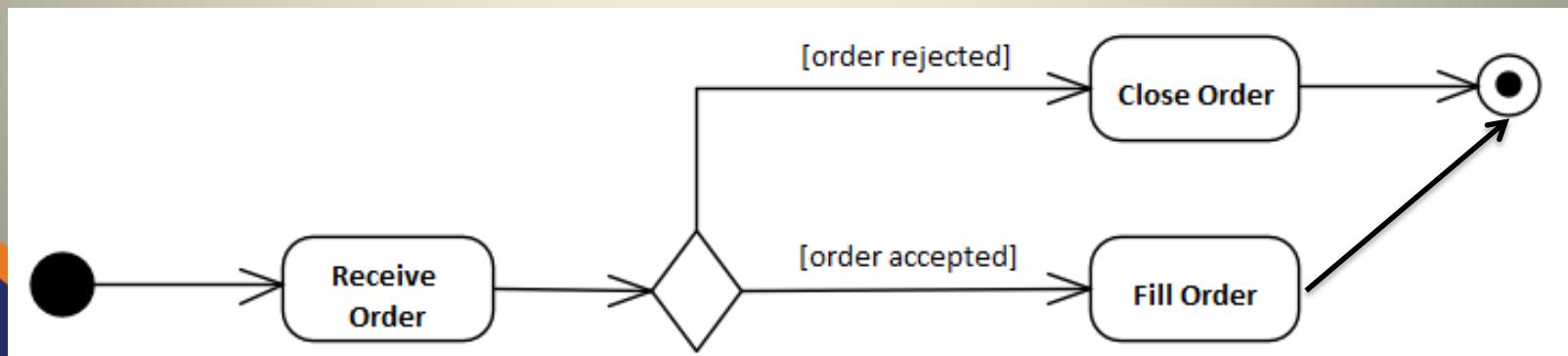
Decision Node

- A Decision/Branch is represented by:
 - a diamond at the branch point.
 - a guard expression for each possibility.

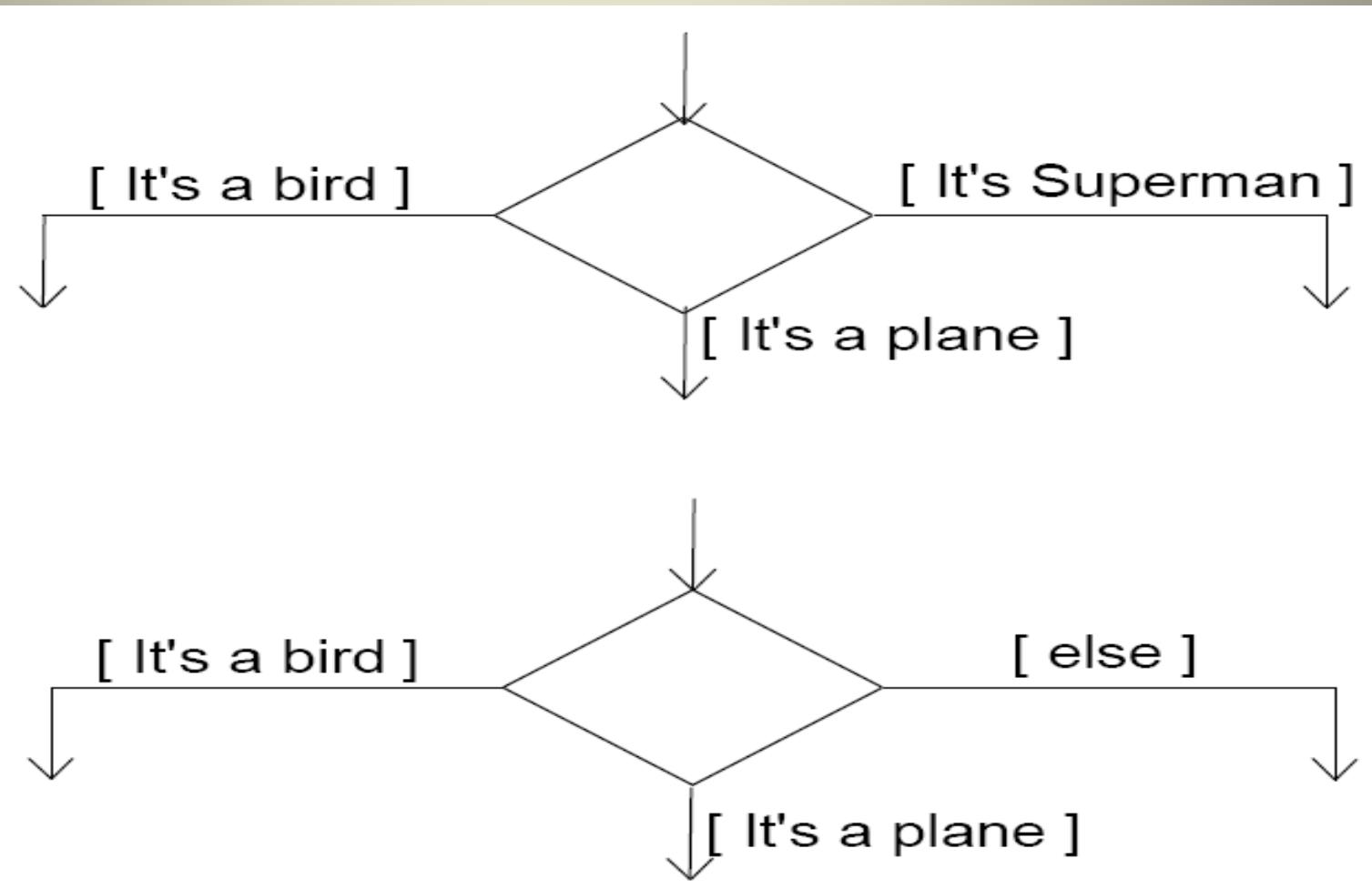


Decision Node

- Decision/Branch points.
 - Each branch must have a **guard condition**.
 - The **flow of control** flows down the **single path** where the branch condition is true.
 - There is **no limit** on the number of **branches** each branch point may have.

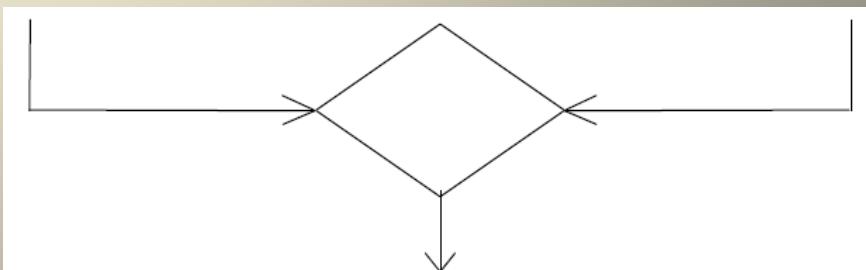


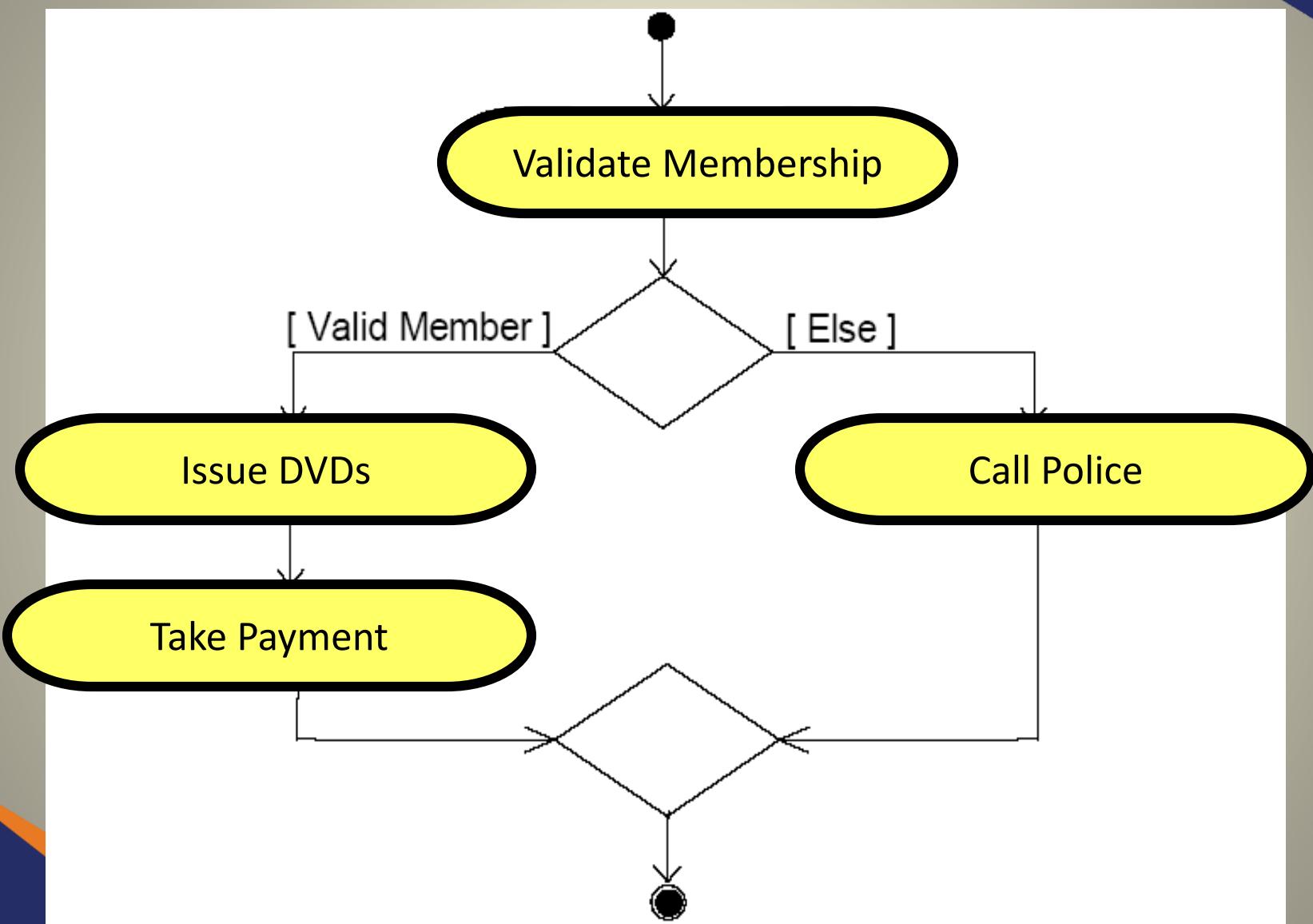
Multiple Branches: Examples



Merge Node

- A **merge point** is used to merge the flow of control from two or more branch points back together.
- The UML equivalent of **ENDIF** in pseudo code, or “**“{”**” in C/C++/Java.
- The diagram below shows a merge graphically – the **diamond is optional**:





Activity - 2

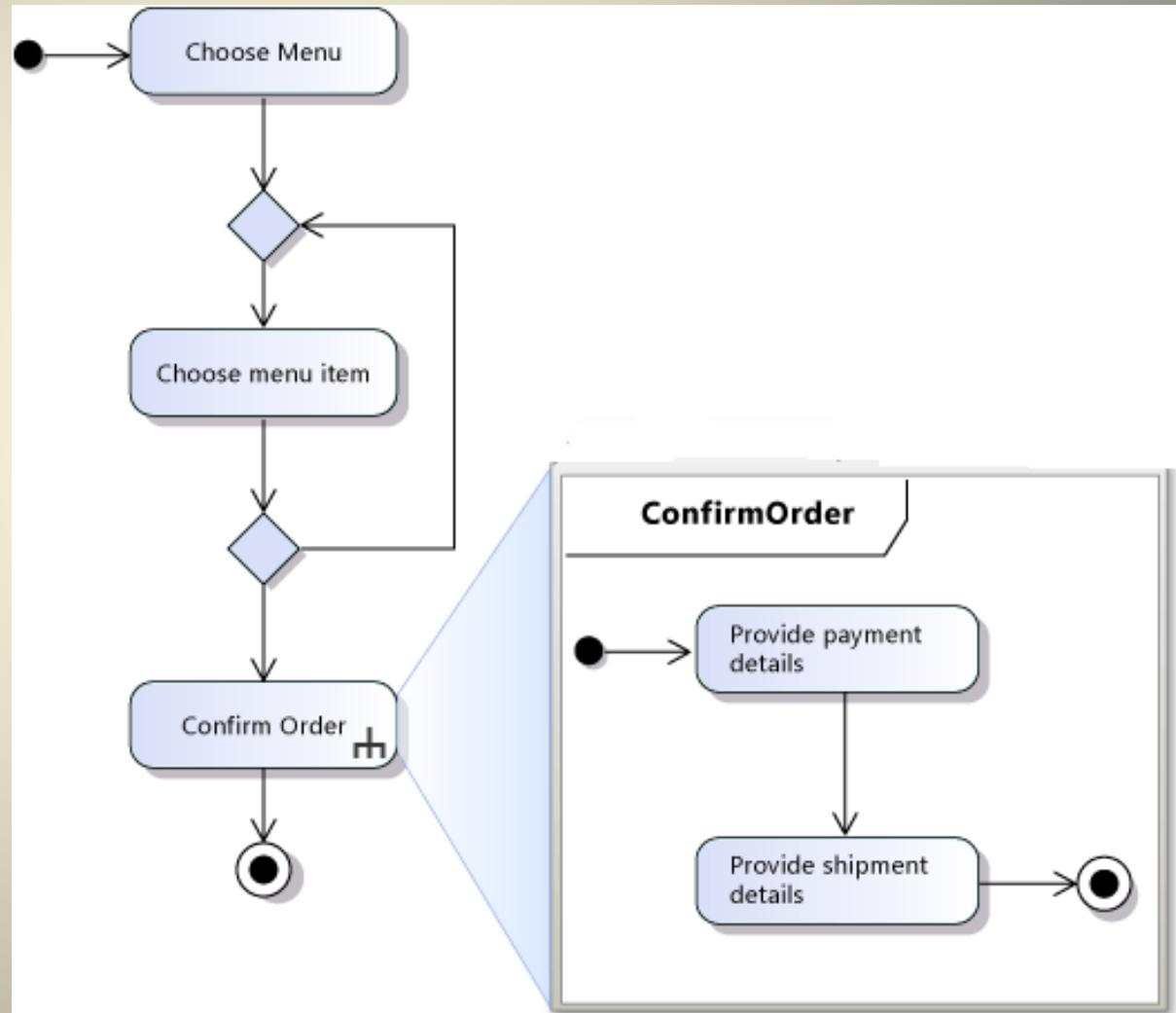
Draw an activity diagram for “**Borrow Books**” activity.

- To borrow books one must become a member.
- Members have access to core textbooks, reference books, general reading materials, CDs and DVDs.
- If a member student needs to borrow more than one book at a time, he/she can do so after depositing an additional refundable deposit of Rs.

3000/-

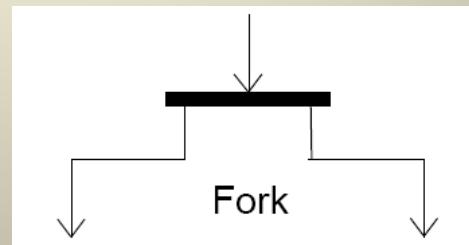
Call action /Sub Activity

- **Sub activity** is an activity that is defined in more detail on another activity diagram.
- It is indicated by a rake-style symbol within the action symbol.



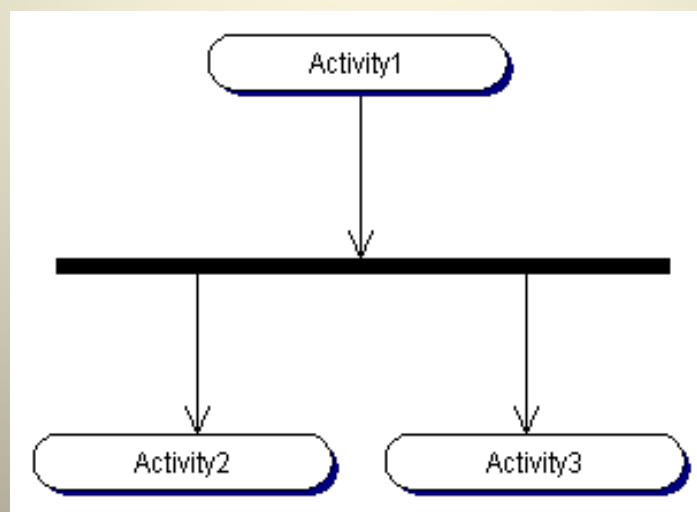
Forks & Joins

- Forks and joins are used to showing activities that can occur at the **same time (in parallel)**.
 - this does not mean that the activities **must** occur concurrently in the finished software system.
 - it means that the **order of execution of the activities can be whatever** is convenient for the implementation.



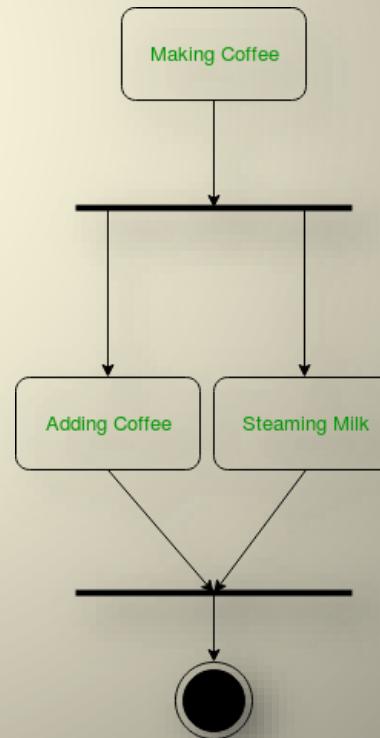
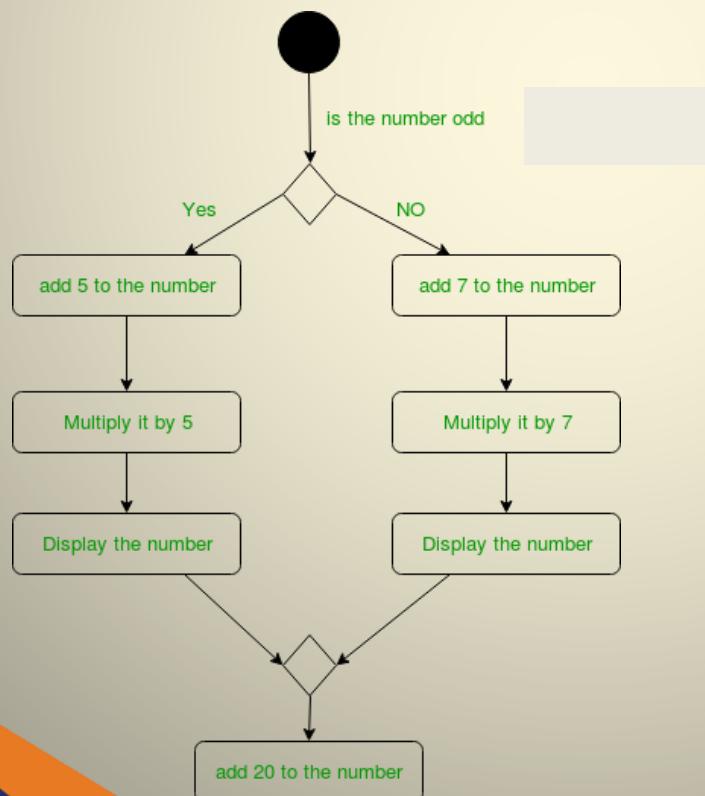
Fork

- A *fork* is when a single flow of control splits into two or more **parallel** (concurrent) flows of control.
- Represents a split in the flow of control.



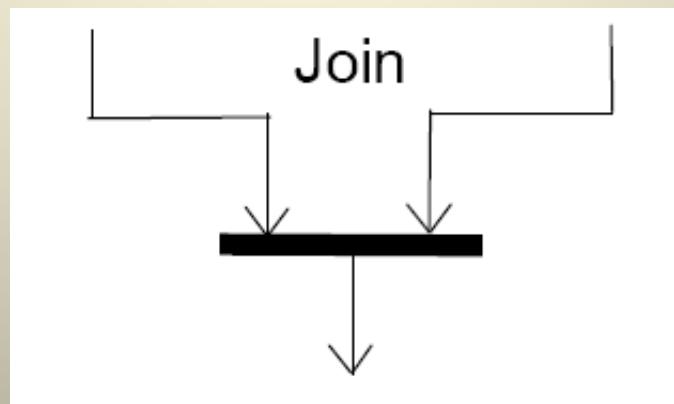
Fork

- Unlike a branch point, the **control flows down all forked paths.**
- With a branch point, the control flows down only **one path.**



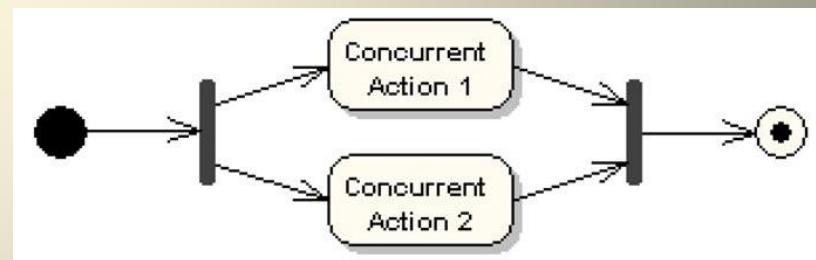
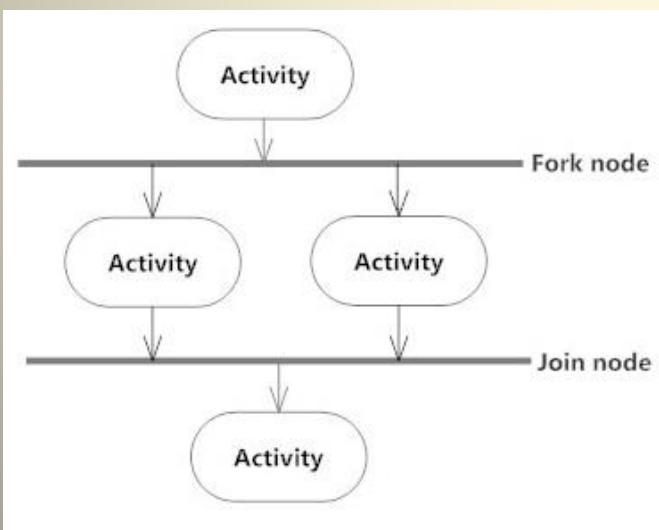
Join

- A *join* is when **two or more flows of control merge into a single flow of control.**
- Represents the merging of multiple **flows of control** back into one.
- Every fork **must** have a join associated with it.

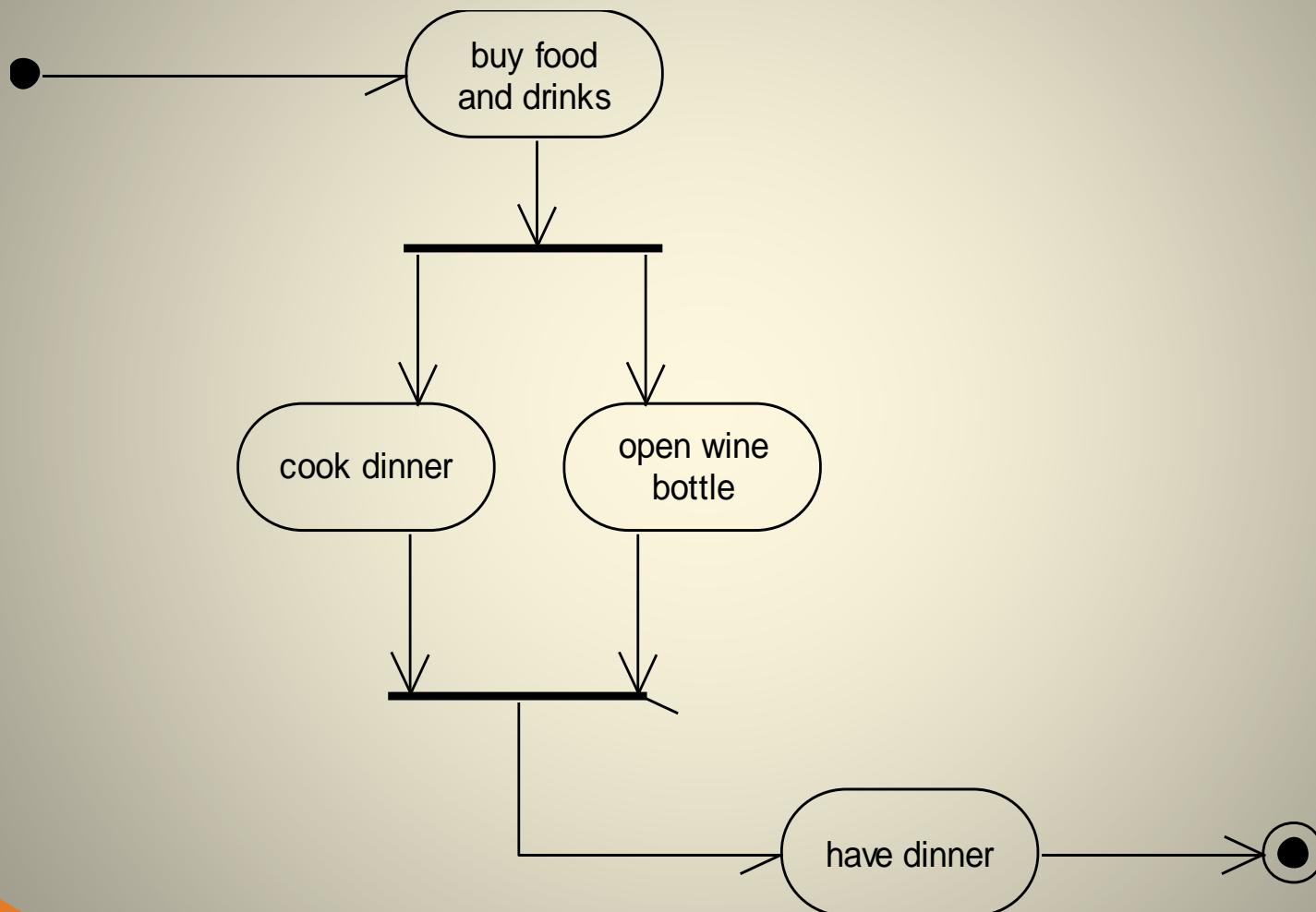


Join

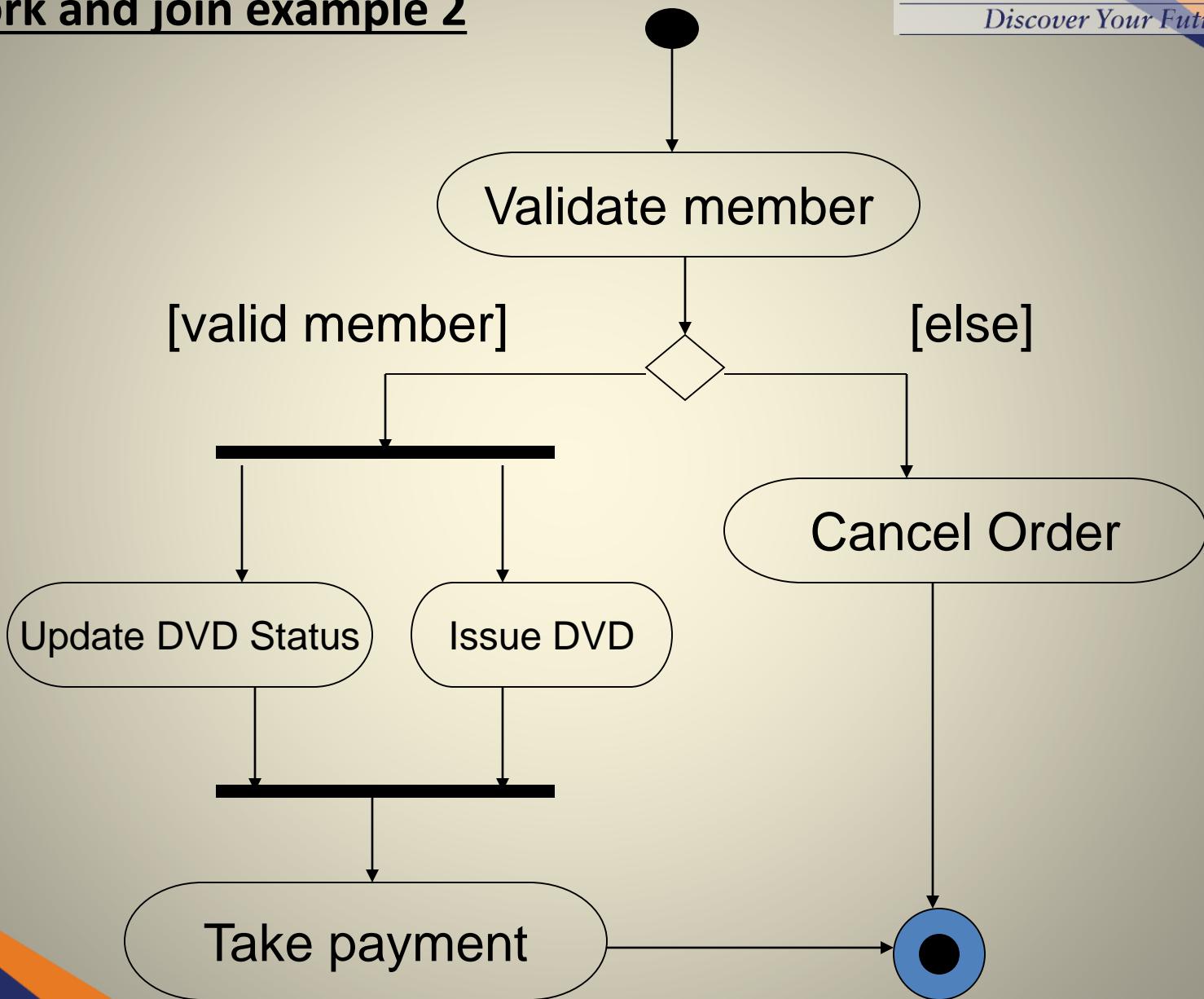
- A **flow of control** is also known as a **thread**.
- A **synchronization bar** is used to model forking and joining, and is modeled as a thick **horizontal or vertical bar**.



Fork and join example 1



Fork and join example 2

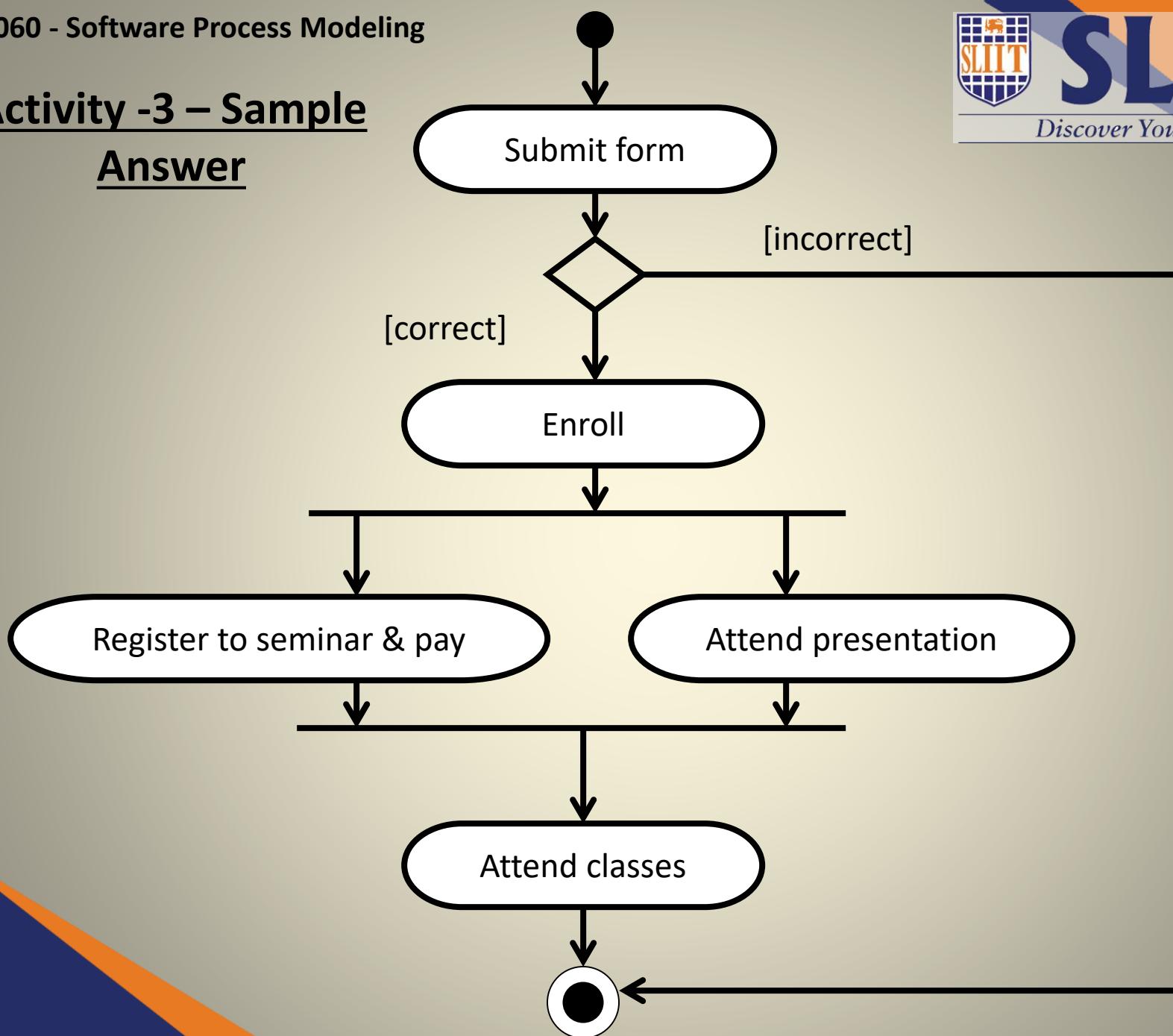


Activity - 3

“Enrolling in University”

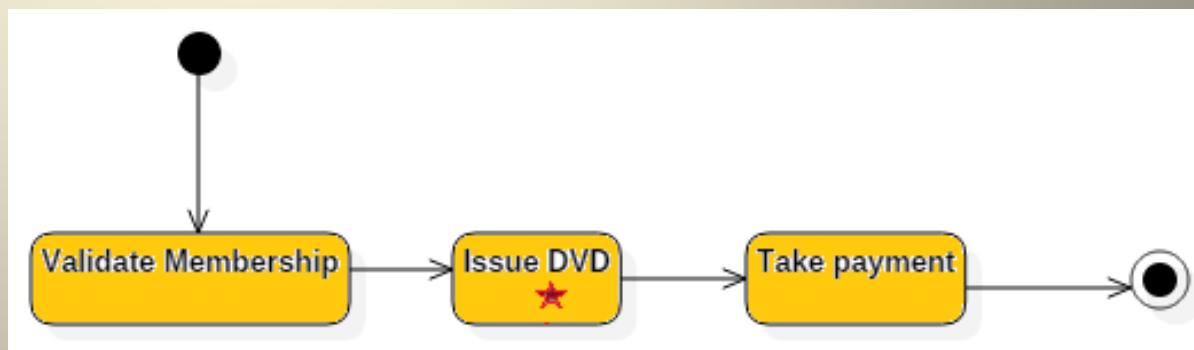
1. Candidates who wish to register for university have to fill out enrolment forms and submit promptly.
2. The forms which are being incorrectly filled will be rejected.
3. The candidates who submitted properly filled forms are being enrolled.
4. The enrolled candidates then have to register for a seminar and make payments for initial tuitions.
5. Meanwhile they have to attend university overview presentation as a start.
6. Then only students can start classes.

Activity -3 – Sample Answer



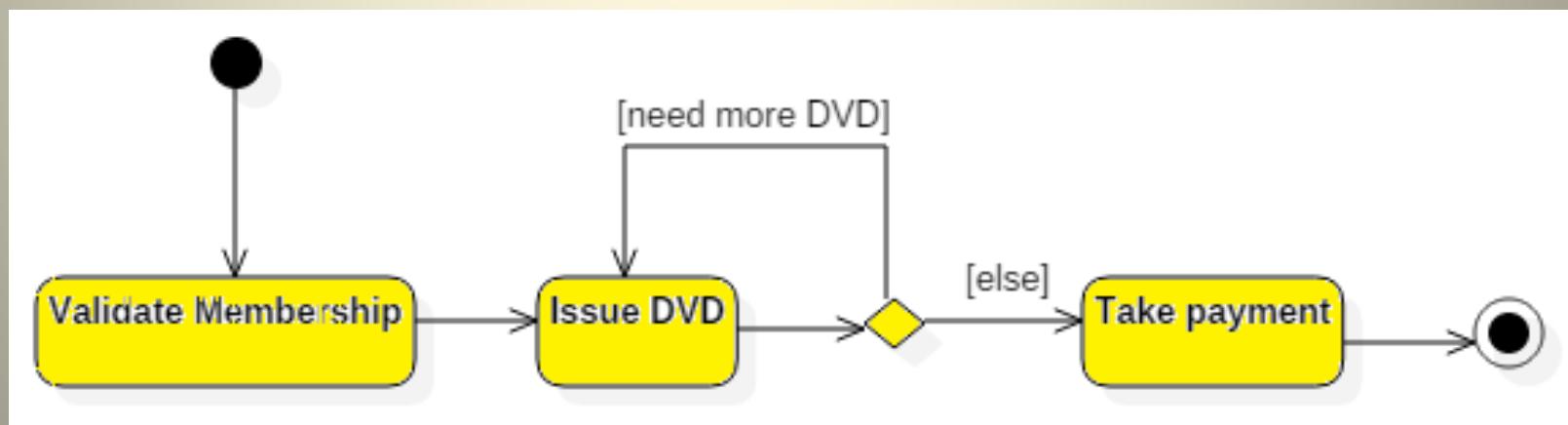
Iteration ★

- An **asterisk** inside an action state indicates that it may need to be **performed more than once**.
 - This notation is used to show iteration, without the unnecessary details of a loop construct.
- The next action state does not occur until the loop is finished.



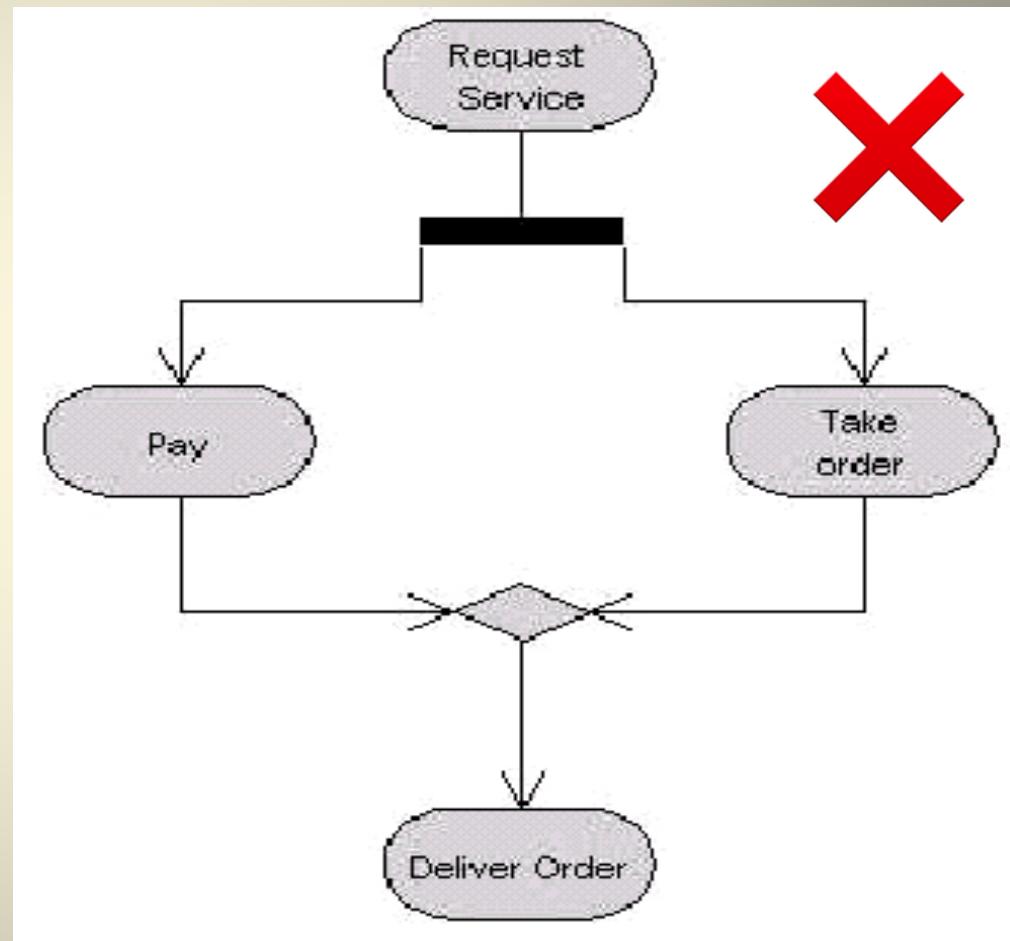
Iteration

- Use of the **asterisk** to represent an iteration will not clearly highlight the **loop termination conditions** and **number of repetitions**.
- Therefore, use of a **Decision Node** to represent an iteration would be ideal.



Common Mistakes

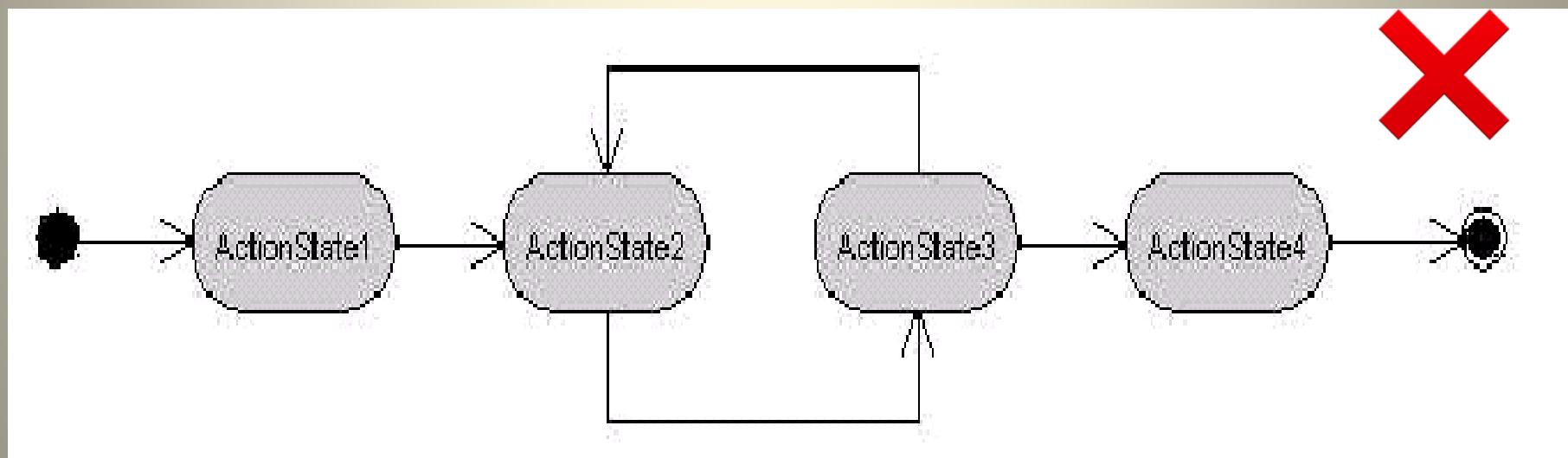
- Incorrect use of **forks, branch points, merge points and joins.**
- The fork means that the flow of control goes down both paths.
- The merge point indicates a merging of divergent paths, not flows.



Common Mistakes

Loops.

- Guard conditions should be mentioned



Activity - 4

“Develop a software system for a client”

1. Project Manager (**PM**) first gathers the requirements
2. UI Engineer (**UIE**) develops the prototype
3. PM shows the prototype to **client**
4. Till the client is satisfied, UIE modifies the prototype.
5. Client signs off the prototype
6. Once the client is satisfied, UIE develop the UI screens, Software Engineer (**SE**) develops the system.
7. Once both (system and UI development) are done, SE integrates the system.
8. PM delivers the system to client
9. Client signs off the delivery.

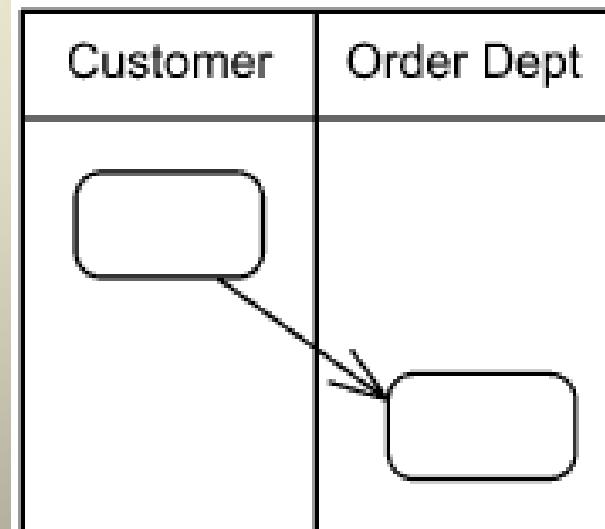
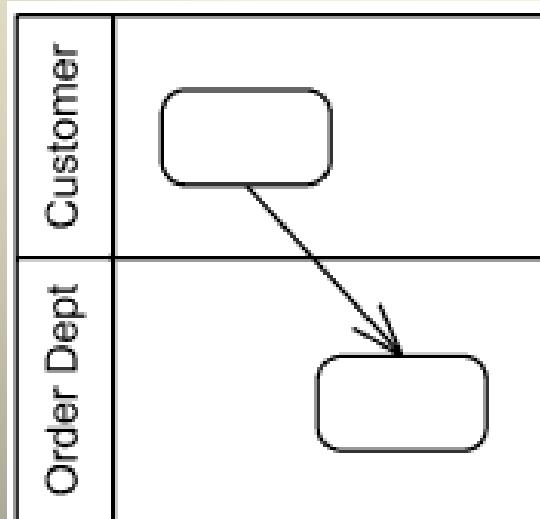
Partitioning

- An activity **partition** is an **activity group** for actions that have some common characteristic.
- Partitions often correspond to **organizational units** or **business actors** in a **business model**.

Partitions: SwimLanes

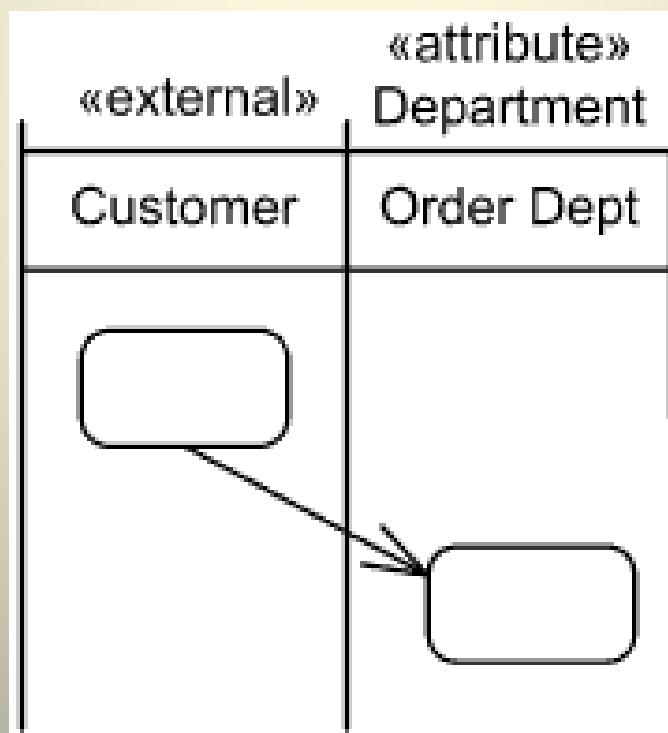
Activity **partition** may be shown using a **SwimLane** notation:

- with two, usually parallel lines, either horizontal or vertical
- a name labeling the partition in a box at one end.



Sub Partitioning in SwimLanes

- Order department is a sub class under Department class.



Activity - 5

“Develop a software system for a client”

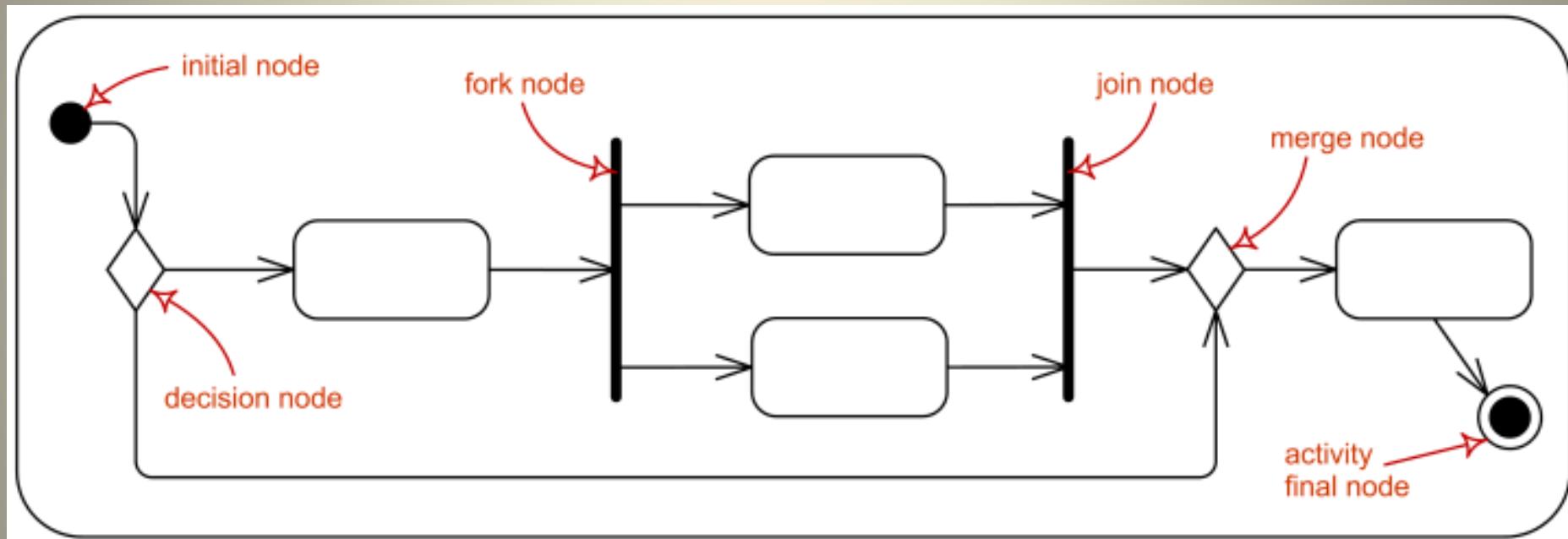
1. Project Manager (**PM**) first gathers the requirements.
2. UI Engineer (**UIE**) develops the prototype.
3. PM shows the prototype to client.
4. Till the client is satisfied, UIE modifies the prototype.
5. Once the client is satisfied, signs off the prototype
6. UIE develop the UI screens while Software Engineer (**SE**) develops the system.
7. Once both (system and UI development) are done, SE integrates the system.
8. PM delivers the system to client
9. Client signs off the delivery.

Activity - 6

“Order Processing System”

1. Once the order is received by the customer service department, several other departments too operate on various activities to complete the order.
2. The fulfillment department should fill the order (prepare goods to deliver) and deliver the order to customer.
3. Meanwhile, as soon as the order is received, customer service department works on sending an invoice to the customer.
4. The finance department will handle the payments from the customer. The shop operates on credit basis. Therefore, receiving payments is not mandatory prior to the delivery of the goods.
5. Customer service department can close the order only after completing all the above activities.

Activity Diagram Notations



How to create an Activity Diagram

1. Identify activities (steps) of a process
2. Identify who/what performs activities (process steps)
3. Draw swim lanes
4. Identify decision points (if-then)
5. Determine if a step is in loop (*For each..., or if-then loop*)
6. Determine if step is parallel with some other
7. Identify order of activities, decision points

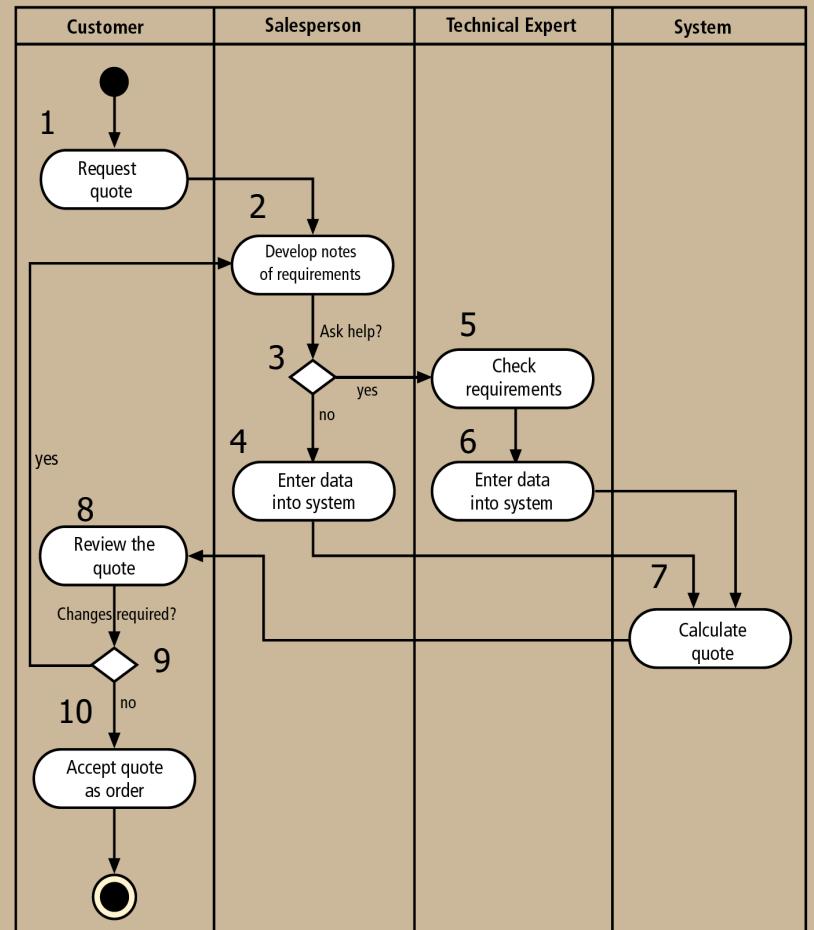
Continues...

How to create an Activity Diagram (cont.)

8. Draw the start point of the process in the swimlane of the first activity (step)
9. Draw the oval of the first activity (step)
10. Draw an arrow to the location of the second step
11. Draw subsequent steps, while inserting decision points and synchronization/loop bars where appropriate
12. Draw the end point after the last step.

You can tabulate these data (see next slide).

How to create an Activity Diagram (cont.)



Step ID	Process Activity or Decision	Who/What Performs	Parallel Activity	Loop	Preceding Step
1	Request quote	Customer	No	No	-
2	Develop requirement notes	Salesperson	No	Yes	1
3	Decision: Help?	Salesperson	-	Yes	2
4	Salesperson enters data	Salesperson	No	Yes	3
5	Check requirements	Technical Expert	No	Yes	3
6	Enter data into system	Technical Expert	No	Yes	5
7	Calculate quote	System	No	Yes	4, 6
8	Review quote	Customer	No	Yes	7
9	Decision: Changes?	Customer	No	Yes	8
10	Accept quote as order	Customer	No	No	9

References

- The Unified Modeling Language Reference Manual – James Rumbaugh, Ivar Jacobson, et al
- UML Distilled: A Brief Guide to the Standard Object Modeling Language by [Martin Fowler, Kendall Scott](#)
- Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process by [Craig Larman](#)
- [The Complete UML Training Course, Student Edition](#) by Grady Booch, et al
- UML Explained by [Kendall Scott](#)

More Examples

Use Case for Order Processing

The following scenario explains how an order is processed.

1. When the Order Processing Department (OPD) receives an order, they have to check whether the order is in proper order as given.
2. OPD checks each line item on the order to see whether they have the goods in stock. If they do, they assign the goods to the order.
3. After assigning, if the goods are not in stock or if the stock level of those assigned goods goes below the reorder level, OPD will reorder the goods.

Use Case for Order Processing

4. While the OPD is checking the order, Finance Department (FD) checks to see if the payment is OK.
5. Once the payment is OK and OPD has assigned the goods OPD will check whether any items are sent for reorder. If there are items to reorder, OPD will leave the order waiting.
6. Otherwise OPD will dispatch the order. If the payment isn't OK, OPD will cancel the order.”

Software Process Modeling

Software Design

Session Outcomes

- What is Software design?
- Design Types
- Object Oriented Design
 - Understand System and interactions
 - Design System Architecture
 - Identify main classes and objects
 - Develop Design Models
 - UML
 - SysML
 - Specify Interfaces

Story so far ...

- Feasibility study
- Requirement phase
 - Requirements elicitation and analysis
 - Requirements Specification
 - Use case diagrams
 - Activity Diagrams
 - Requirement validation
- Today's lecture : Software Design

Programmer's Approach to Software Engineering

Skip requirements engineering and design phases;
start writing code



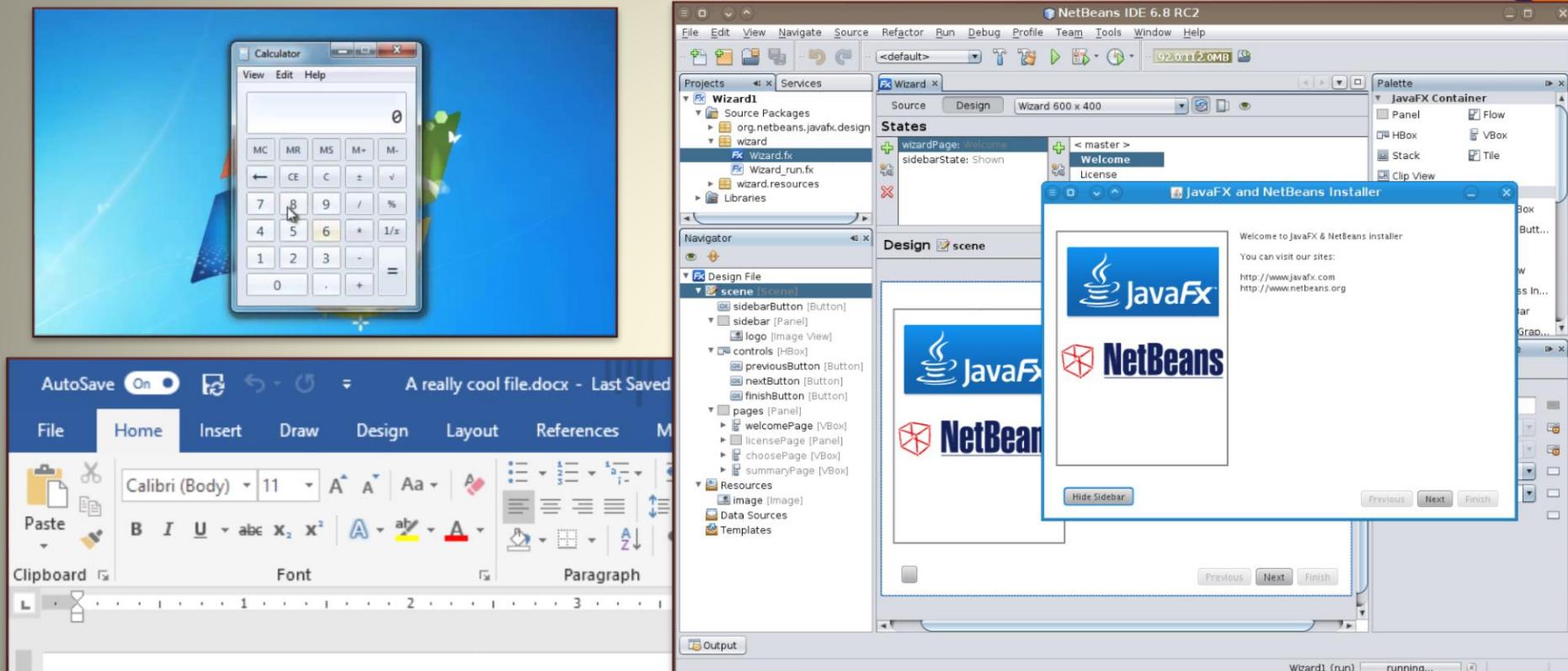
Why this programmer's approach?

- Design is a waste of time
- We need to show something to the customer really quickly.
- We are judged by the amount of LOC/month
- We expect or know that the schedule is too tight

Design of small and large systems



Design of small and large systems



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet comm quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque ha

Importance of design

- Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software.

Ref: Software Engineering A Practitioner's approach , R.S. Pressman, 7th Edition

- Design is a highly creative stage in software development where the designer plans
 - how the system or program should meet the customer's requirements
 - how to make system effective and efficient.

Ref: Software Engineering, I. Sommerville, 10th Edition

Stages of design

- Understand the problem
 - Look at the problem from different angles to discover the design requirements
- Identify one or more solutions
 - Evaluate possible solutions and choose the most appropriate
- Describe solution abstractions
 - Use graphical, formal or other descriptive notations to describe the components of the design
- Repeat process for each identified abstraction until the design is expressed in primitive terms

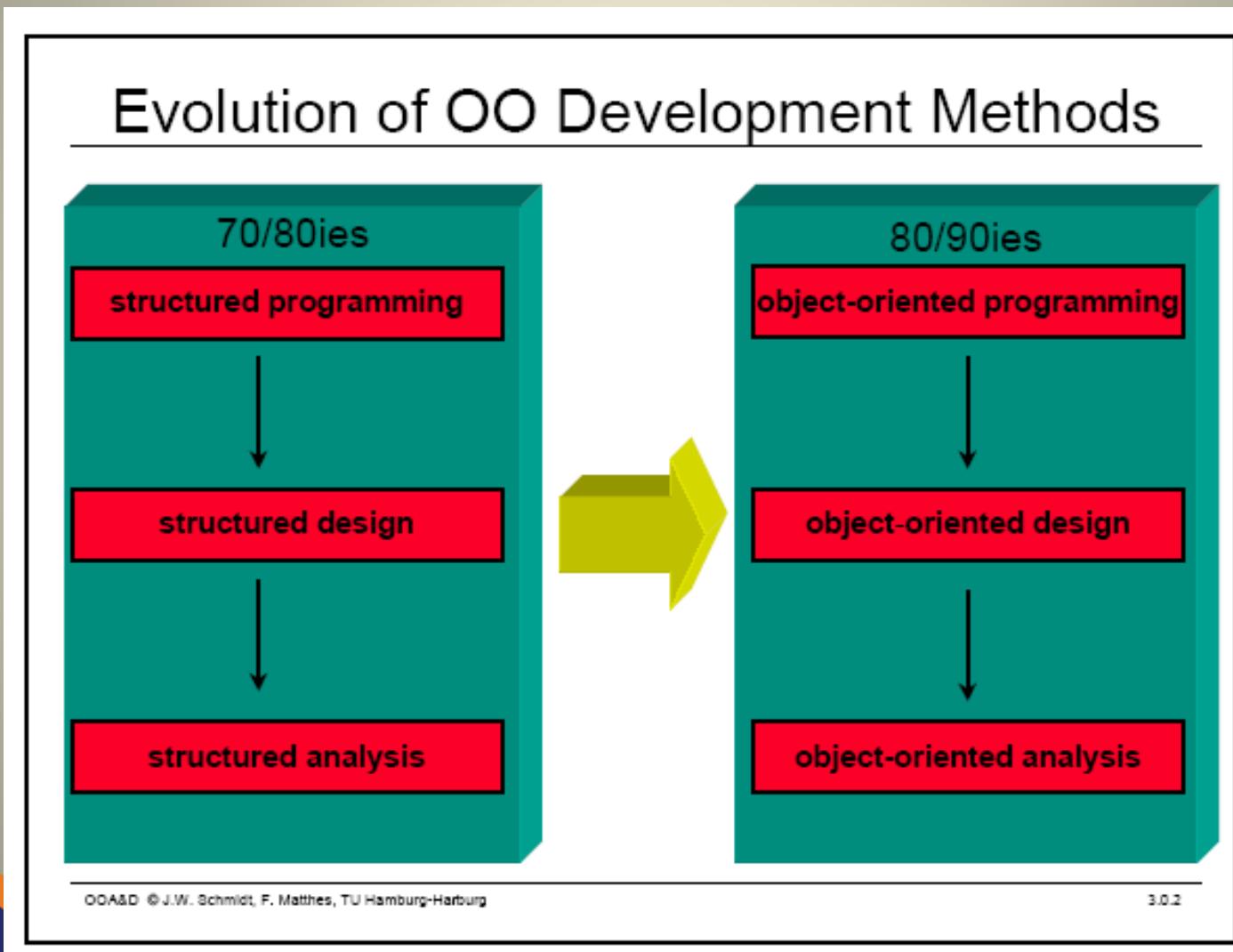
Software Design methods

- Function oriented software design
- Object oriented software design

Software Design

Object Oriented Design

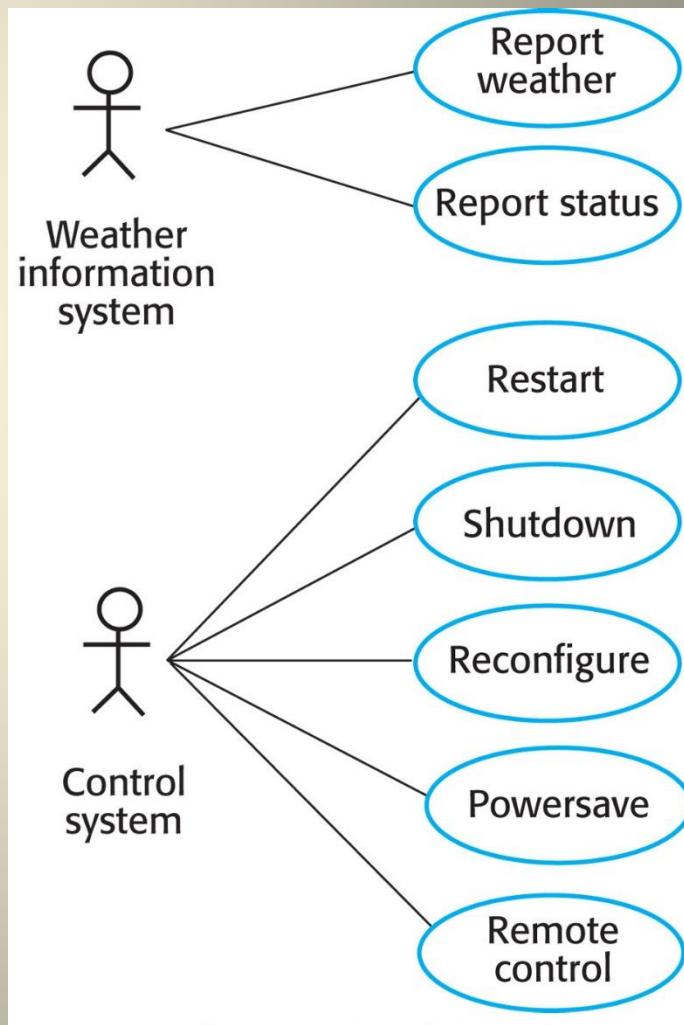
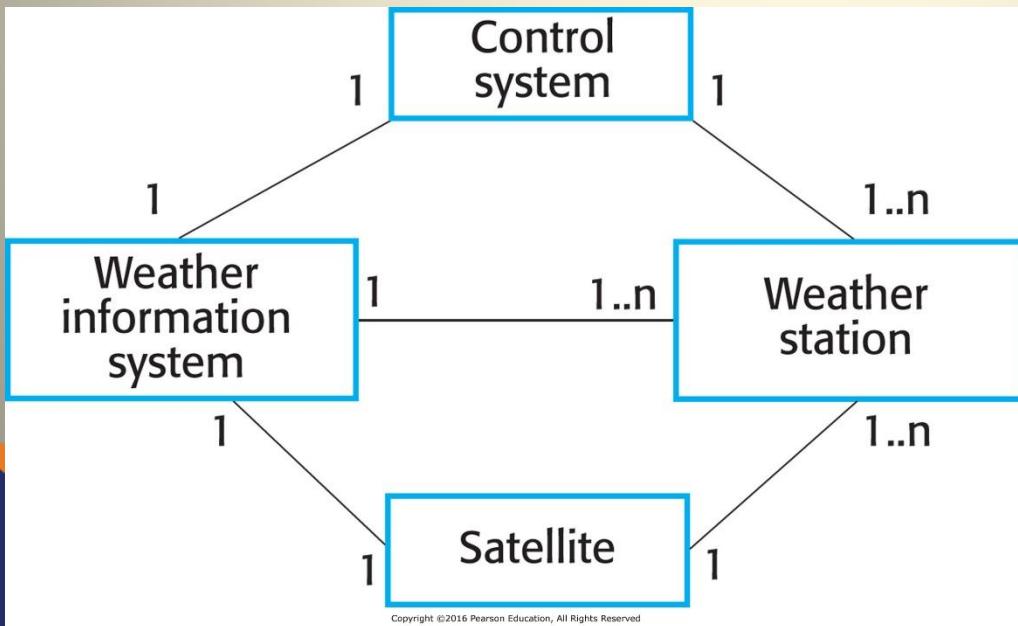
Object oriented software design



Object Oriented Design

1. Understand System and interactions

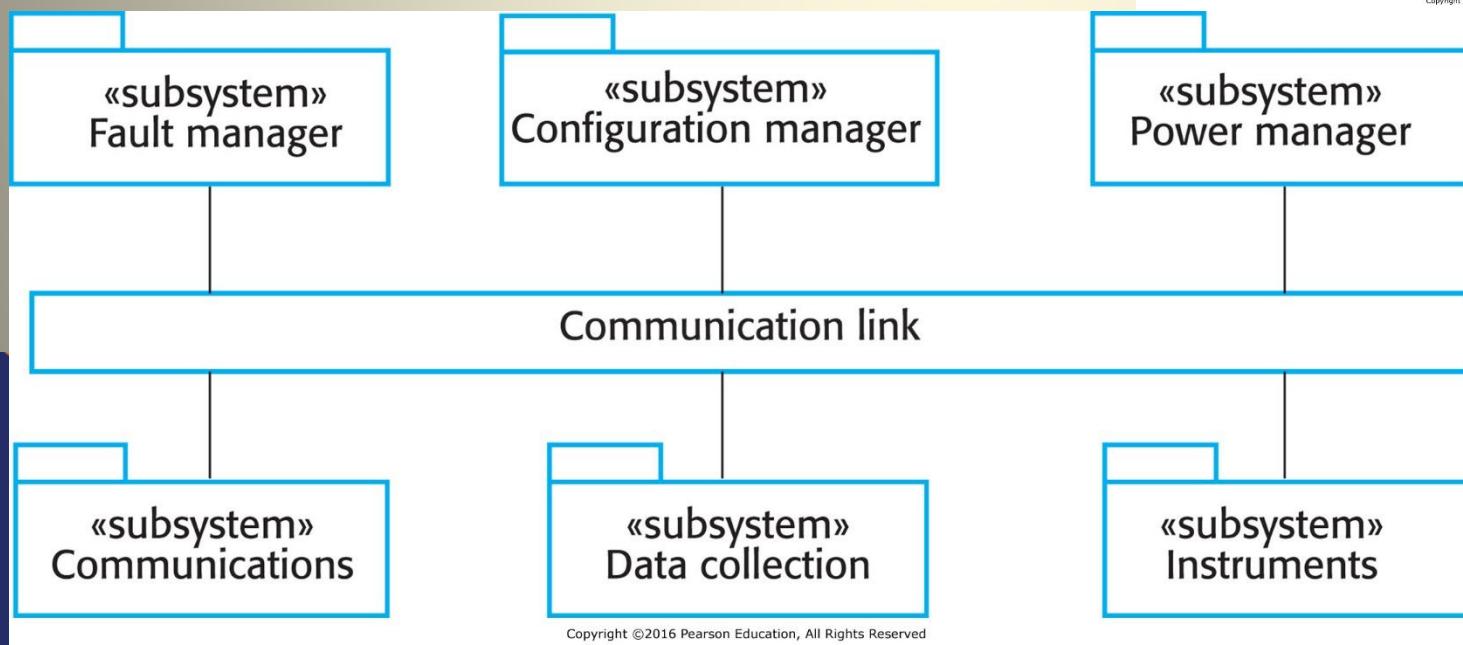
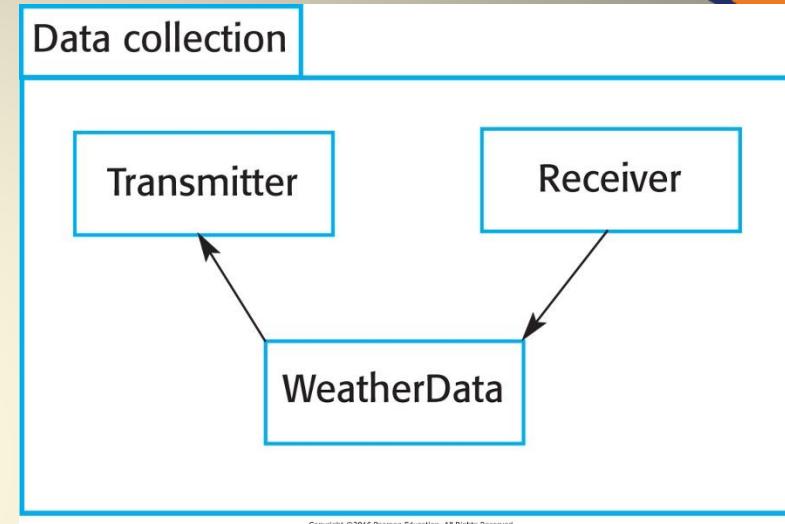
- Use case Diagrams
- Activity Diagrams
- Use case Scenarios



Object Oriented Design

2. Design System Architecture

– Subsystems and communication between the subsystems.



Object Oriented Design

3. Identify main classes and objects

WeatherStation
identifier
reportWeather ()
reportStatus ()
powerSave (instruments)
remoteControl (commands)
reconfigure (commands)
restart (instruments)
shutdown (instruments)

WeatherData
airTemperatures
groundTemperatures
windSpeeds
windDirections
pressures
rainfall
collect ()
summarize ()

Ground thermometer
gt_Ident
temperature
get ()
test ()

Anemometer
an_Ident
windSpeed
windDirection
get ()
test ()

Barometer
bar_Ident
pressure
height
get ()
test ()

Activity

- What are the CRC cards you identified for the Library system?

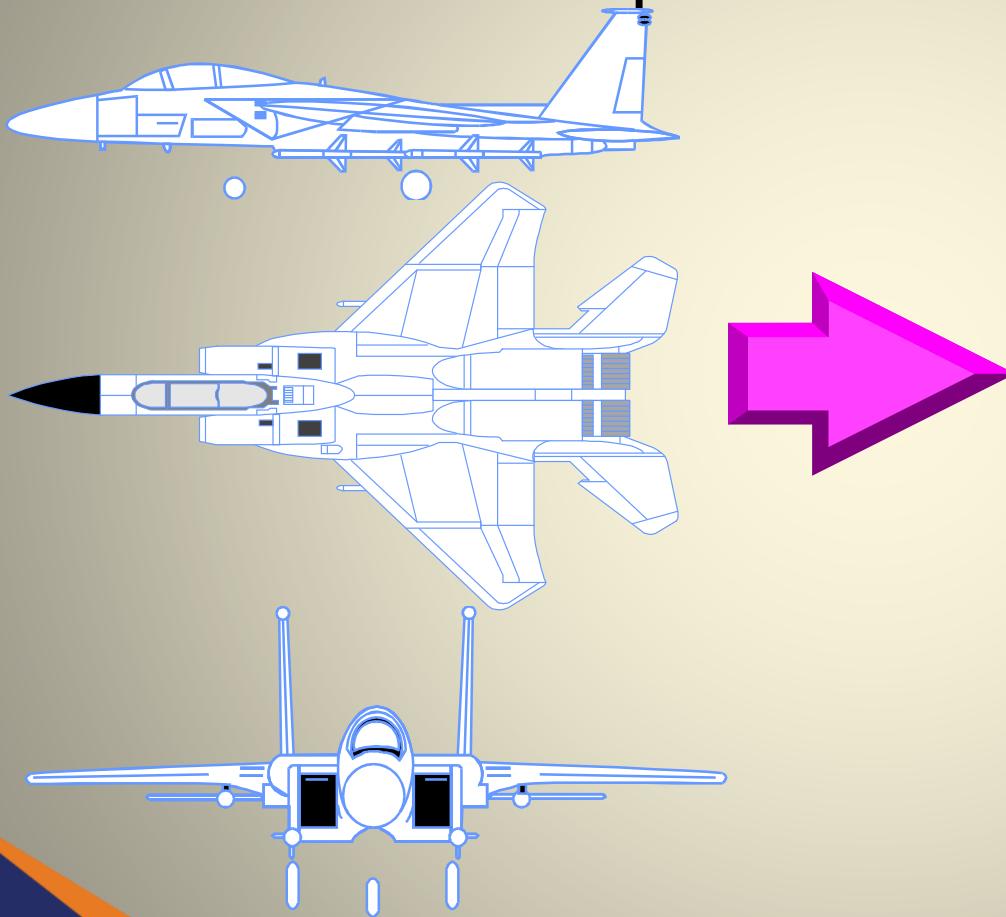
Object Oriented Design

4. Develop Design Models

- Describing a system at a high level of abstraction
- Design Model types
 - Structural models
 - Dynamic models
- Is it necessary to model software systems?

What Is a Model?

- A model is a simplification of reality.

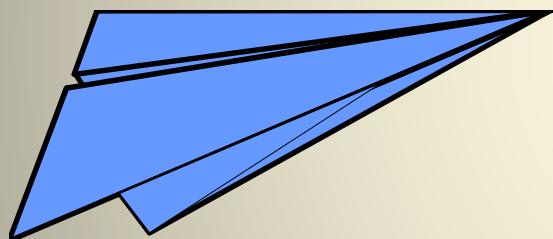


Ref: Fundamentals of Visual Modeling with UML

The Importance of Modeling

Less Important

More Important



Paper Airplane



Fighter Jet

Ref: Fundamentals of Visual Modeling with UML

Software Teams Often Do Not Model

- Many software teams build applications approaching the problem like they were building paper airplanes
 - Start coding from project requirements
 - Work longer hours and create more code
 - Lacks any planned architecture
 - Doomed to failure
- Modeling is a common thread to successful projects.

Ref: Fundamentals of Visual Modeling with UML

Why Do We Model?

- Modeling achieves **four aims**:
 - Helps us to **visualize** a system as we want it to be.
 - Permits us to **specify** the structure or behavior of a system.
 - Gives us a template that guides us in **constructing** a system.
 - **Documents** the decisions we have made.
- We build models of complex systems because we cannot comprehend such a system in its entirety.
- We build models to better understand the system we are developing.

Ref: Fundamentals of Visual Modeling with UML

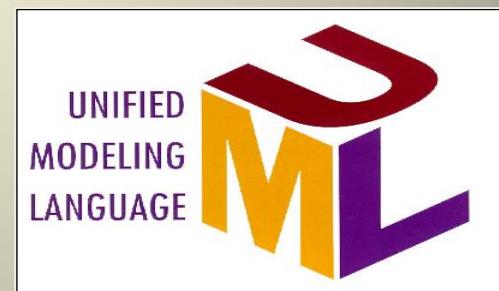
Object Oriented Design

4. Develop Design Models

- Design Model types
 - Structural models
 - Dynamic models
- Modeling Languages
 - UML
 - SysML
 - Refer <https://modeling-languages.com/#>

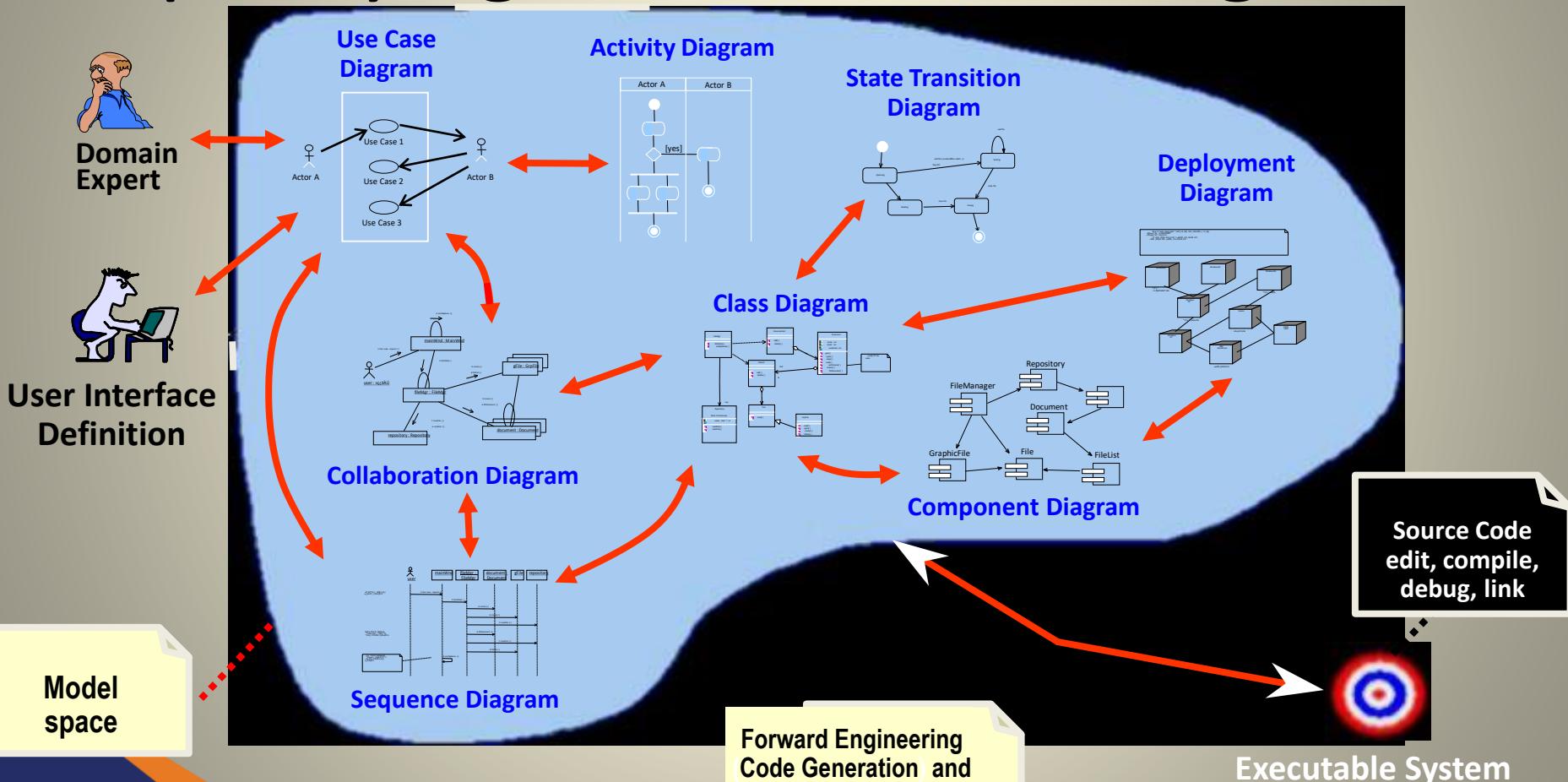
What Is the UML?

- The UML is a language for
 - Visualizing
 - Specifying
 - Constructing
 - Documenting
- Out of the above, **SPM** and **SE** modules specially focus on using UML as a language for specifying and documenting.



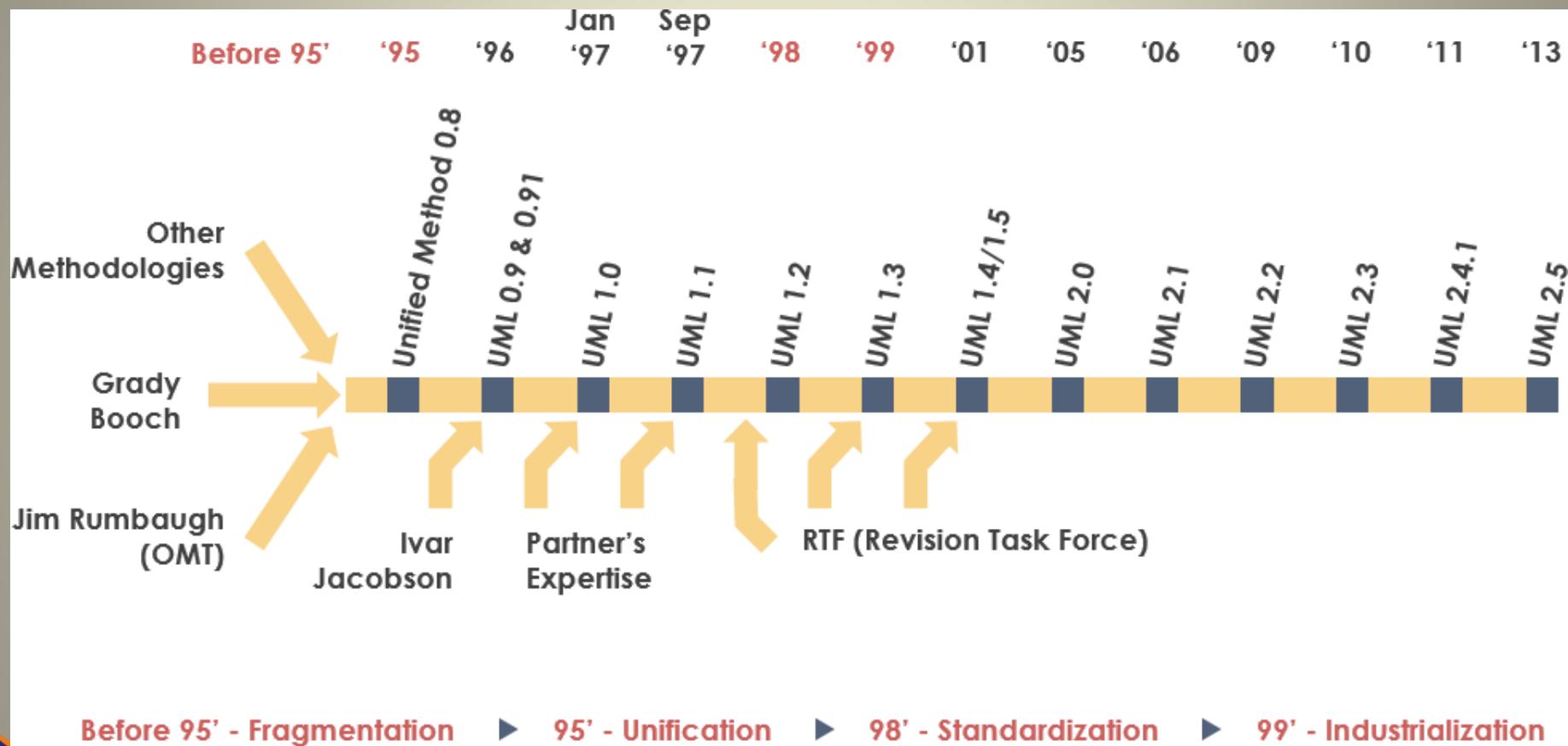
Ref: Fundamentals of Visual Modeling with UML

The UML Is a Language for Specifying and Documenting

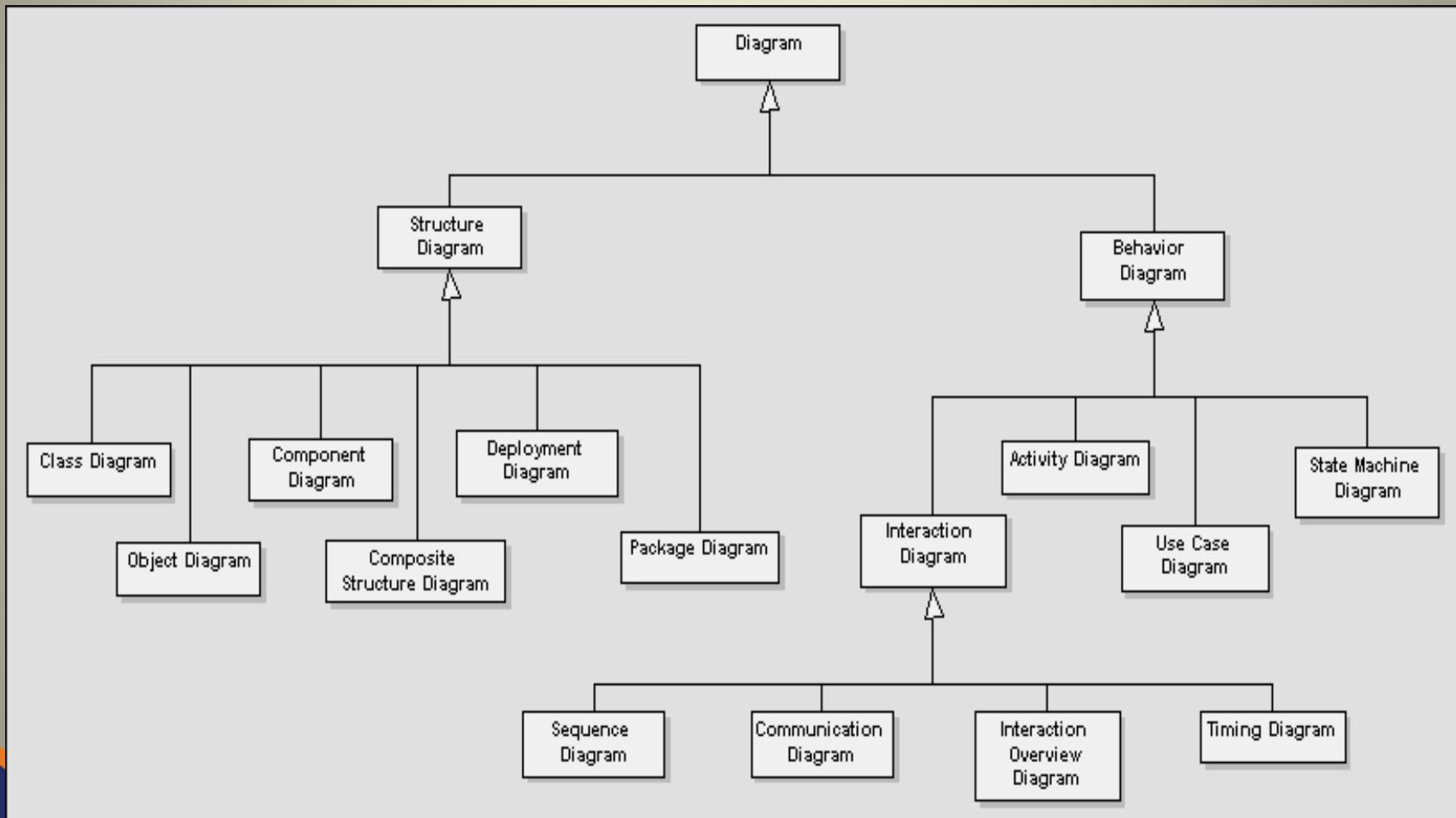


Ref: Fundamentals of Visual Modeling with UML

UML - History



UML 2 Structure



Types of UML diagrams

- There are different types of UML diagram, each with slightly different syntax rules:
 - use cases- **Covered in RE**
 - activity diagrams- **Covered in RE**
 - class diagrams. – **Cover in OOC**
 - sequence diagrams.
 - collaboration diagrams.
 - state diagrams
 - component diagrams.
 - deployment diagrams.



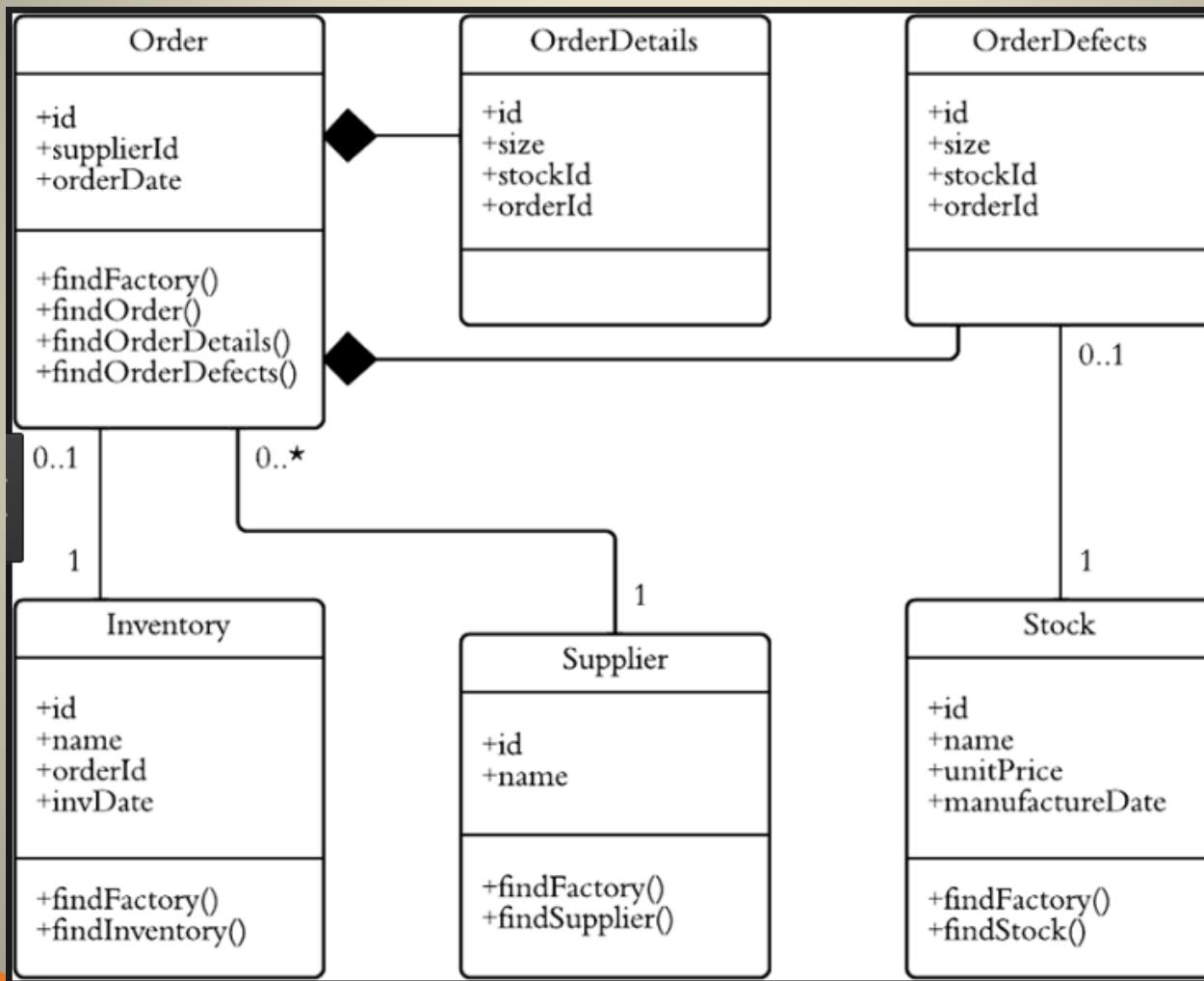
**Covers in SE in the
next semester**

Design

- When you use UML to develop a design, you will normally develop two kinds of design models:
 1. Structural models :
 - describe the static structure of the system using objects, classes and their relationships.
 - Important relationships that may be documented at this stage are generalization (inheritance), aggregation, dependency, and composition relationships. (**class diagram relationships in OOC**)

Ref: Software Engineering, I. Sommerville, 10th Edition

Structural models Example



Design

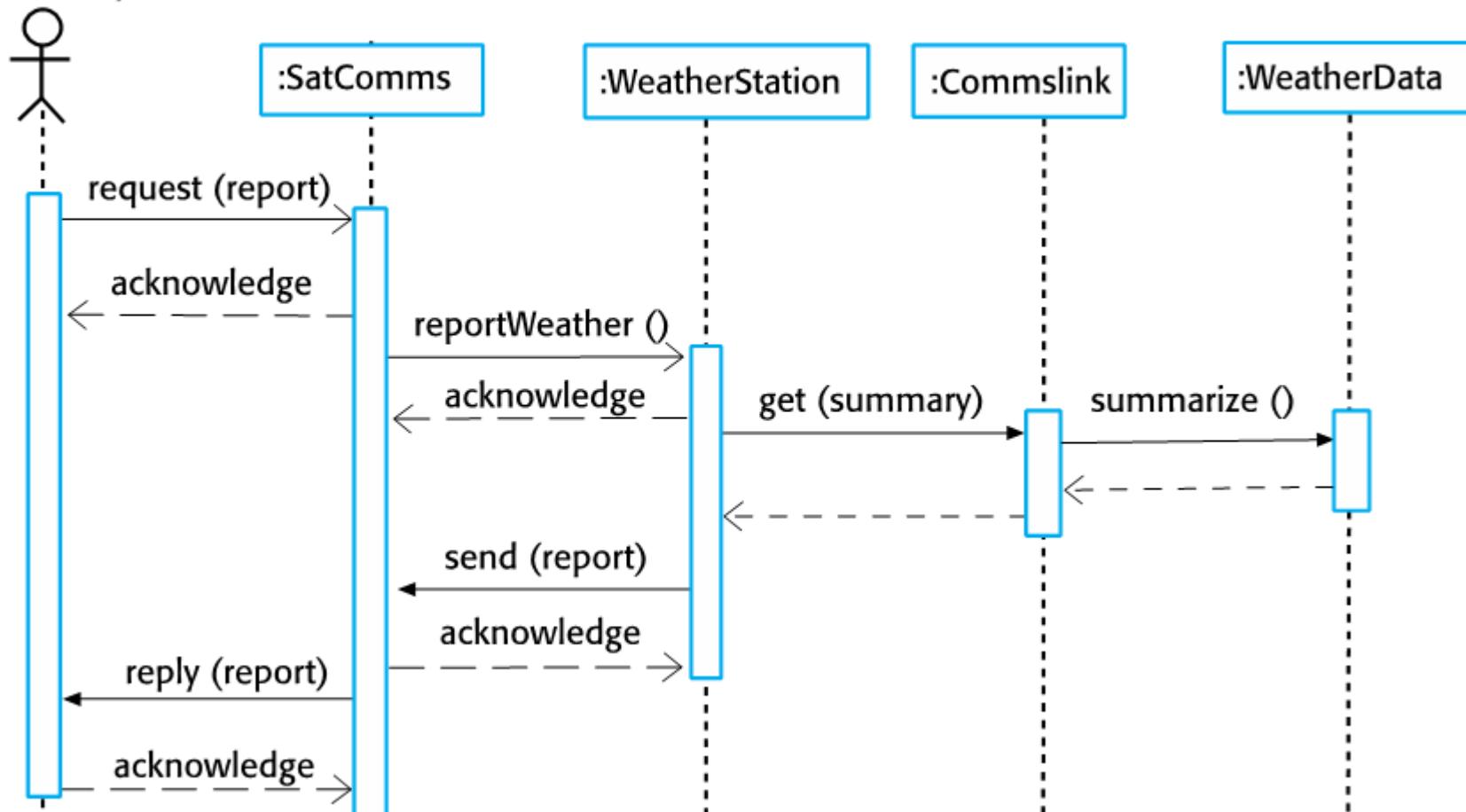
2. Dynamic models :

- Describes the dynamic structure of the system and shows the interactions between the system objects.
- Interactions that may be documented include the sequence of service requests made by objects and the state changes that are triggered by these object interactions. (You will learn them in SE next semester)

Ref: Software Engineering, I. Sommerville, 10th Edition

Dynamic models Example

information system

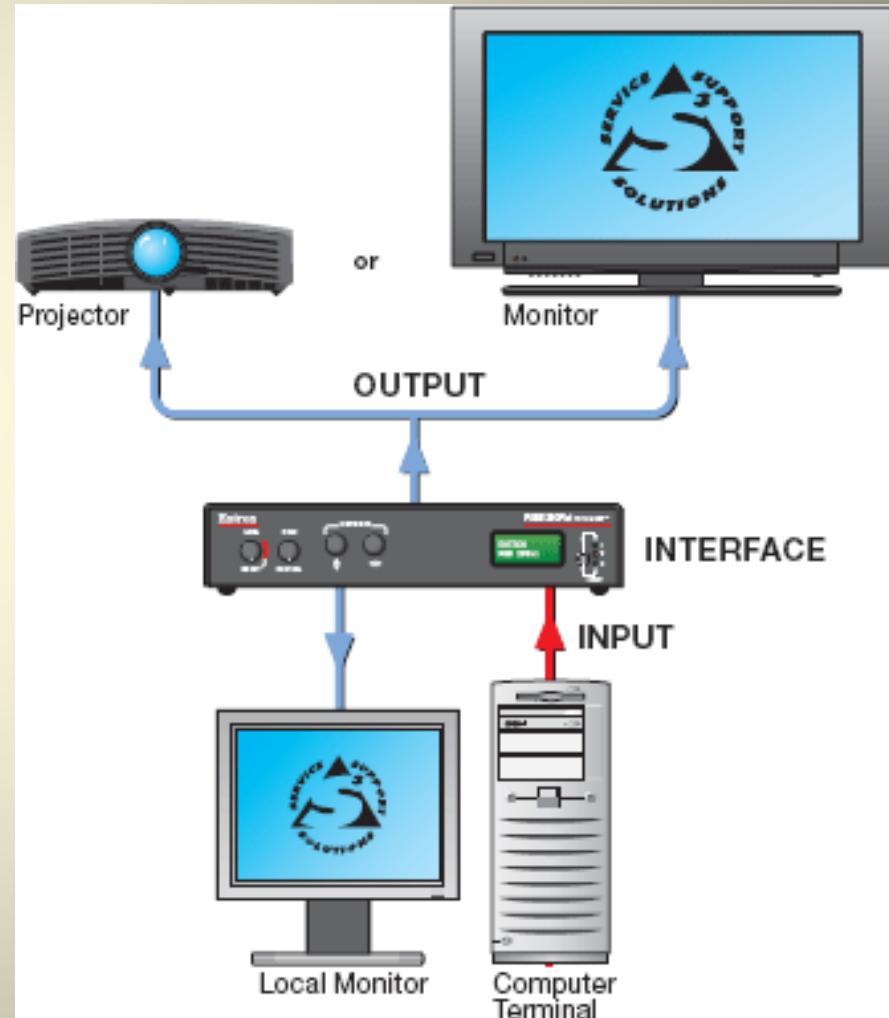


Ref: Software Engineering, I. Sommerville, 10th Edition

Object Oriented Design

5. Specify Interfaces

- Interfaces can be
 - devices
 - software
- The collection of all the inputs and outputs of a system defines its *interface*.

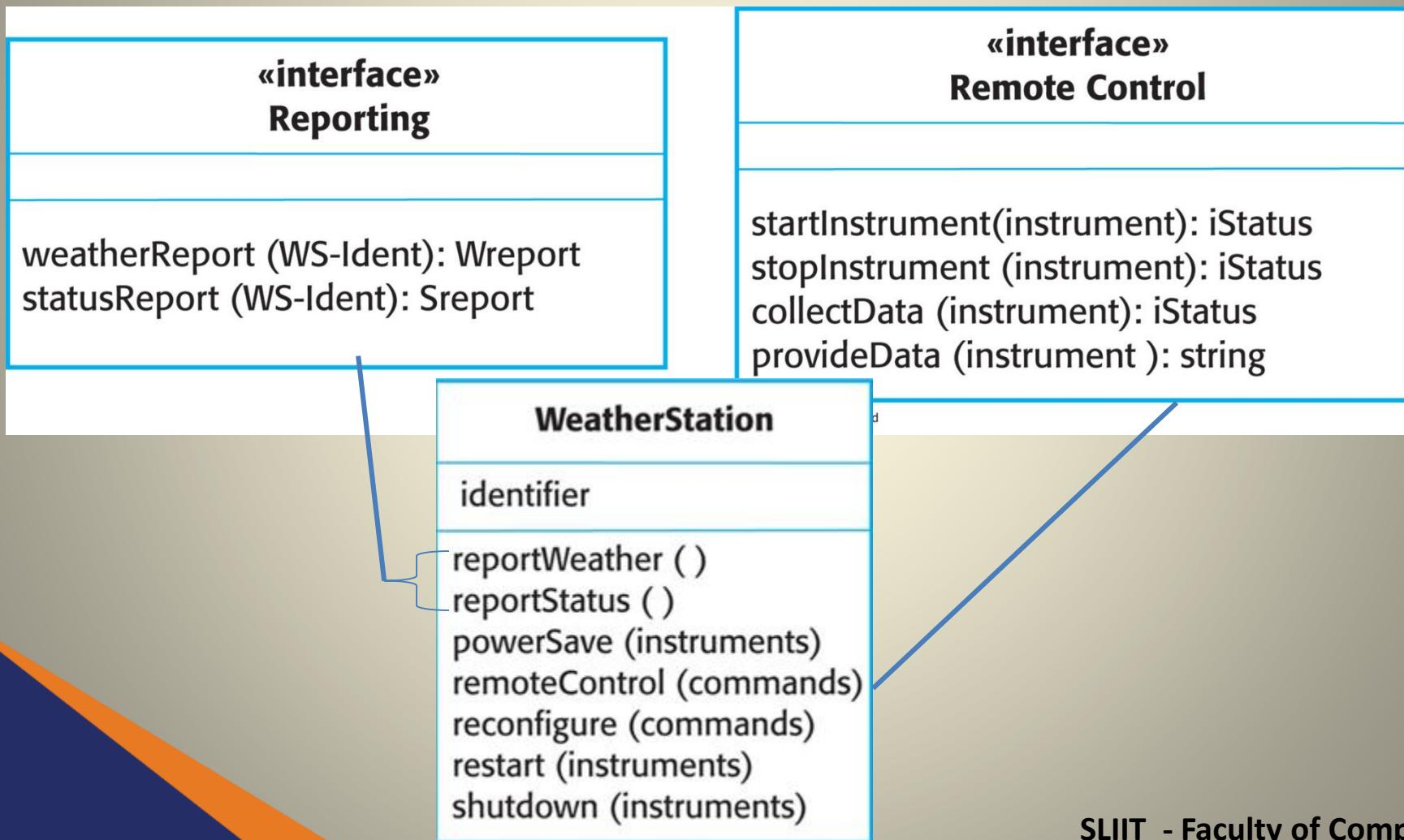


Activity

- Draw a wireframe for the SLIIT Library System

Object Oriented Design

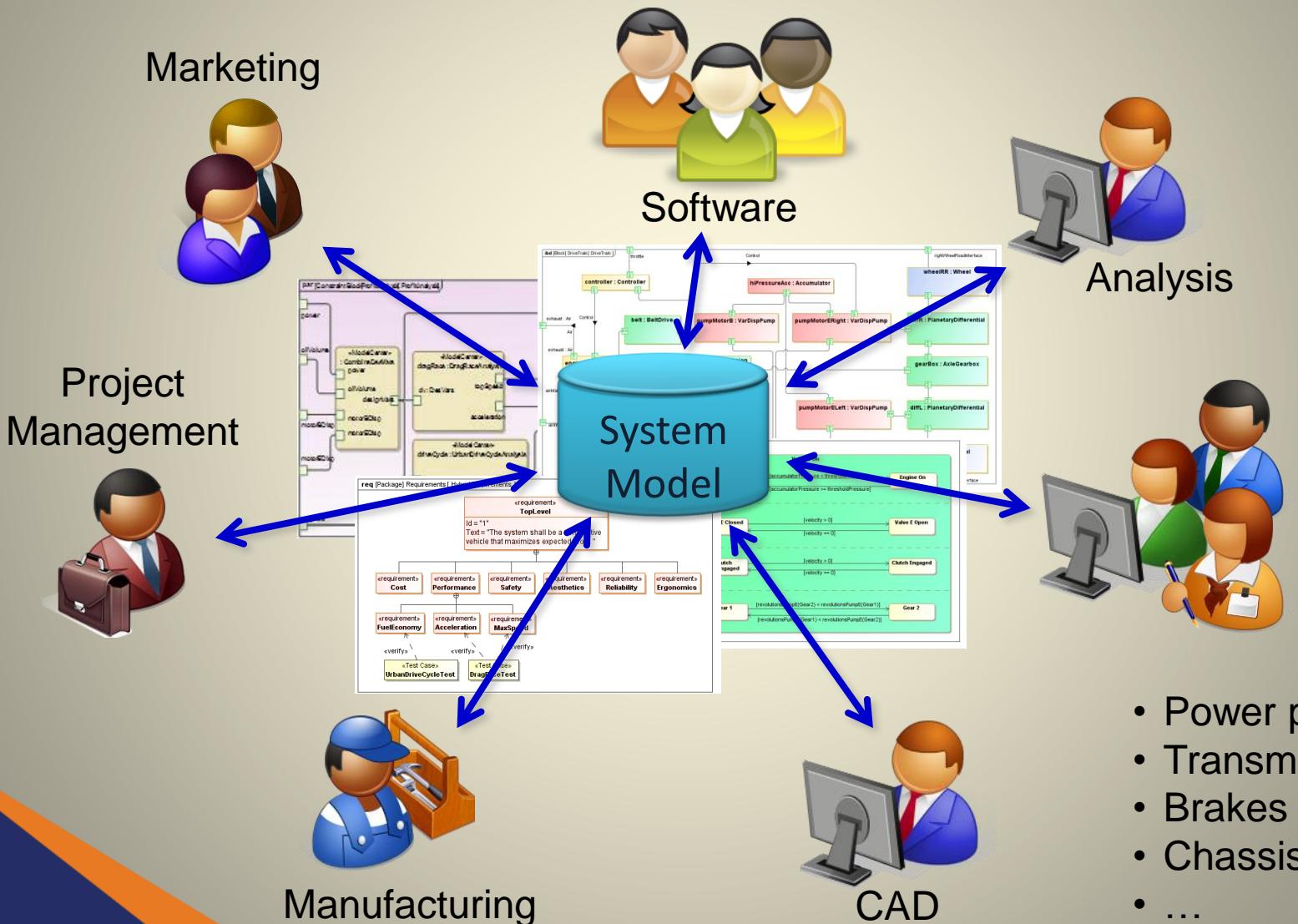
Interfaces



SysML

- What is SysML? A graphical modeling language in response to the UML.
 - It is a UML Profile that represents a subset of UML 2 with extensions.
- Supports the specification, analysis, design, verification and validation of systems that include hardware, software, data, personnel, procedures, and facilities.

SysML

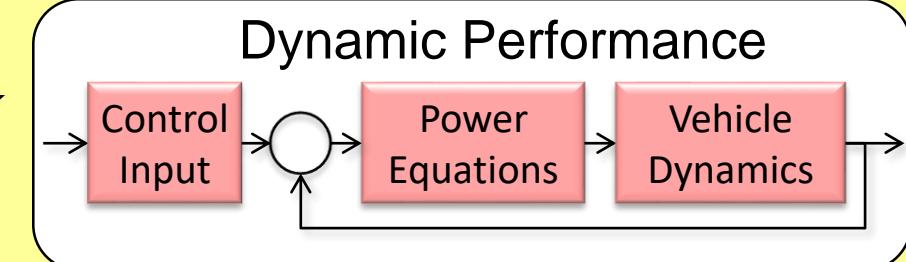


- Power plant
- Transmission
- Brakes
- Chassis
- ...

SysML

Requirements

Structure / Physical Architecture

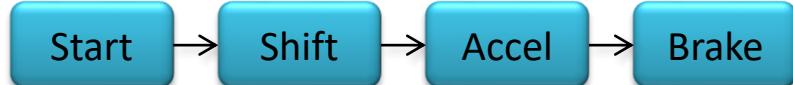


Integrated System Model

Must Address Multiple Aspects of a System



Behavior / Functional Architecture



Mass

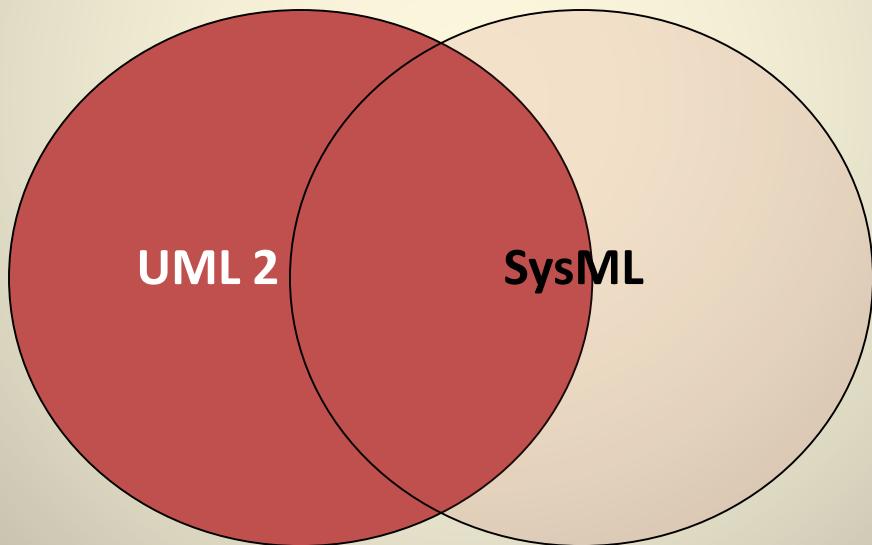
Cost

Manufacturing

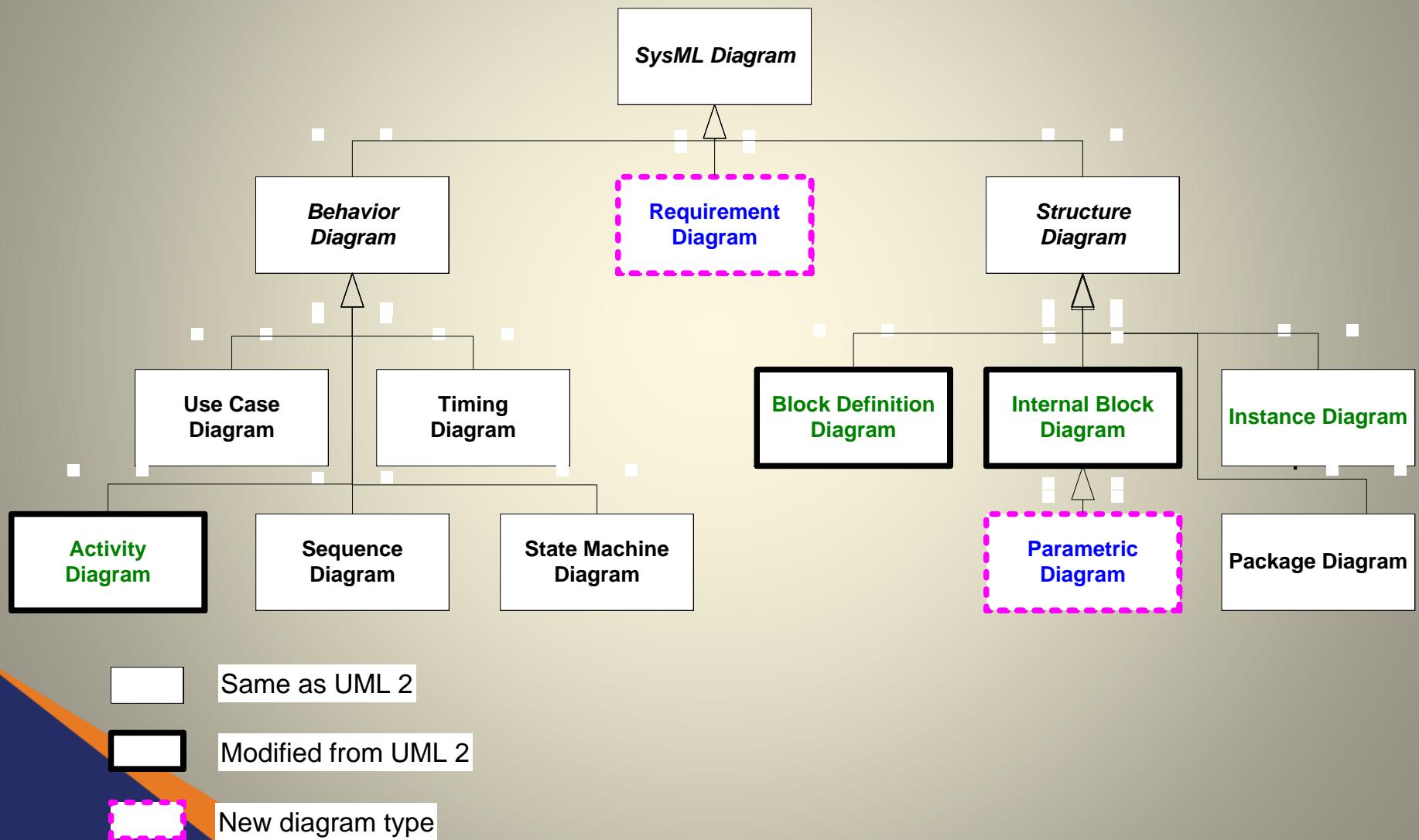
Reliability

Models are more formal, complete & semantically rich

Relationship shared by the SysML and UML Standards



- Package hierarchy in SysML



References

- Software Engineering – 10th Edition by Ian Sommerville, Chapter 7
- <https://modeling-languages.com/#>
- <http://www.omg.sysml.org/>
- <http://www.omg.org/>

Software Process Modeling

Software Implementation

Session Outcomes

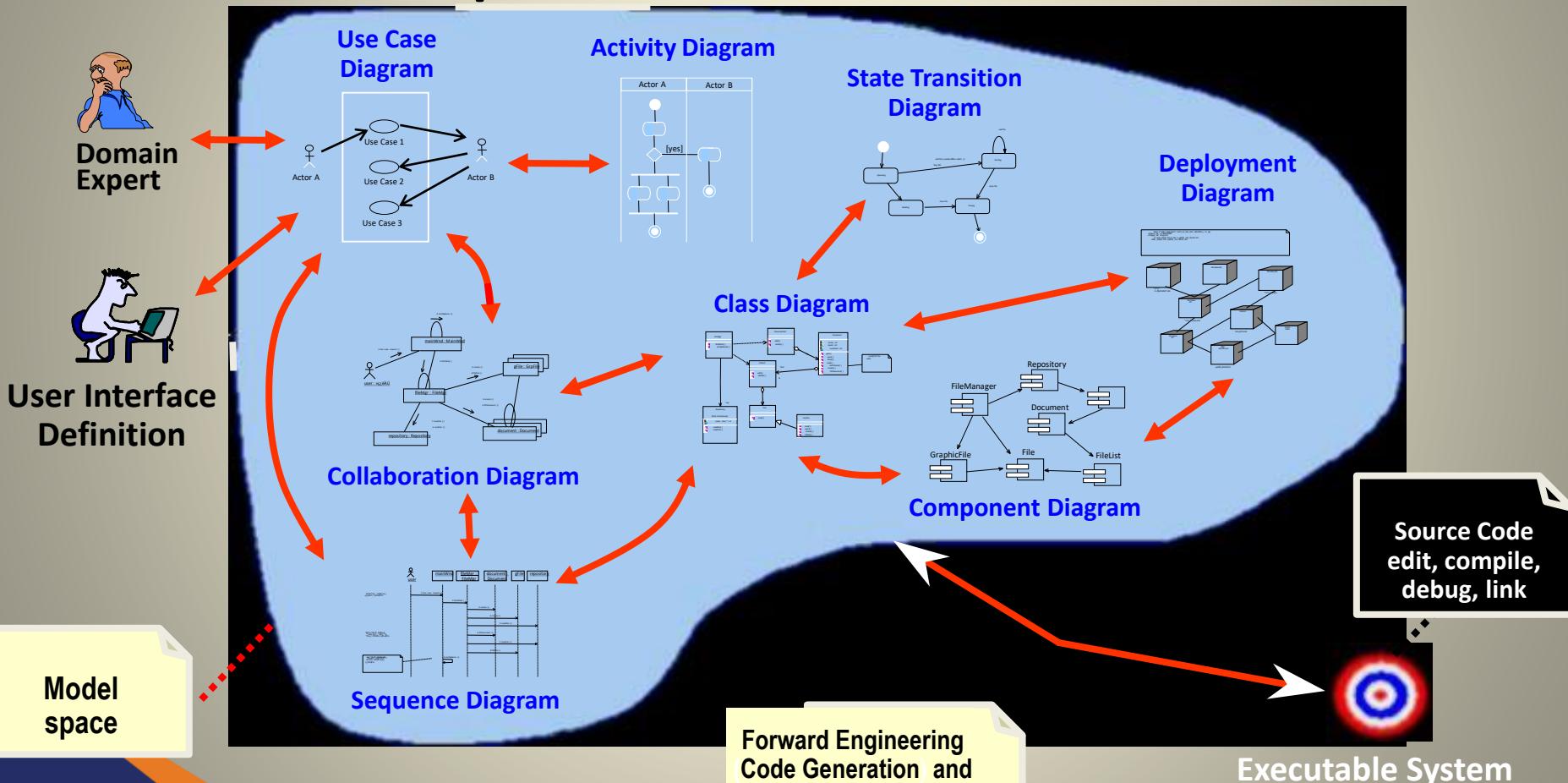
- Implementation
 - Design to Implementation
 - Round Trip Engineering
 - Implementation types
 - Coding standards

Design to Implementation

- **Implementation** is the process of realizing the design as a program.
- During the Design Phase, you learnt to build the design models which are independent of the programming language.
- Design Models aims to
 - Visualize
 - Specify
 - **Construct**
 - Document

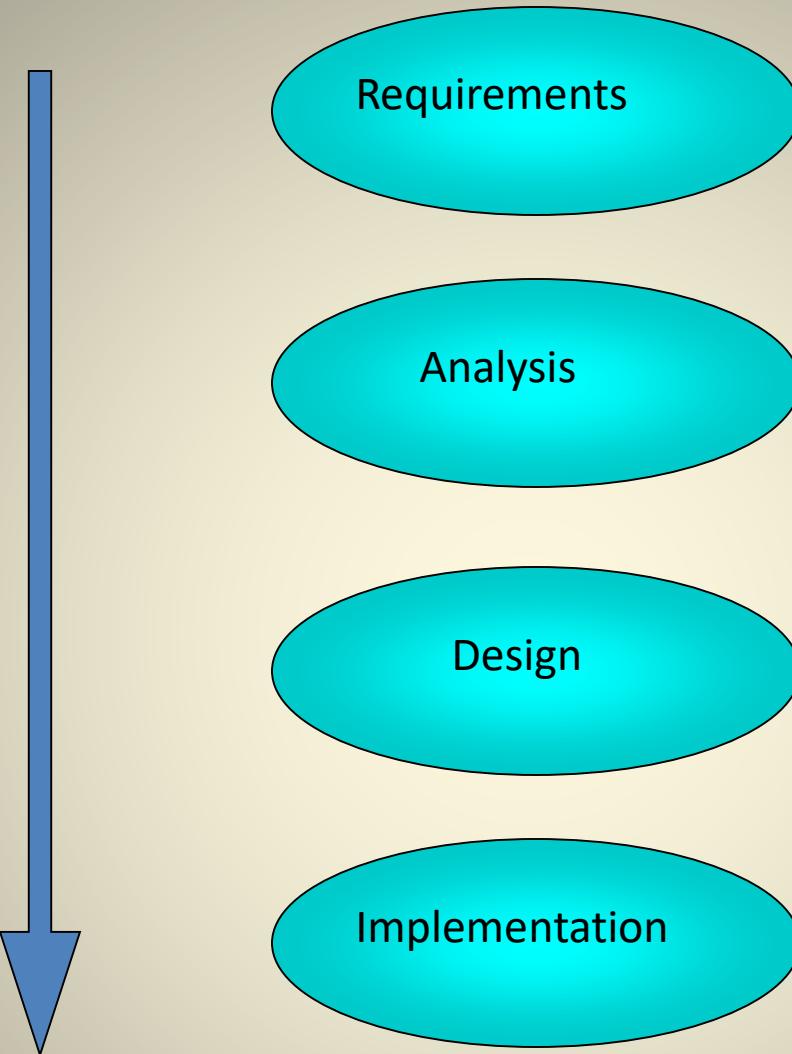
Ref: Software Engineering, I. Sommerville, 10th Edition

Design as a template for Implementation



Ref: Fundamentals of Visual Modeling with UML

Forward
engineering



Reverse
engineering

Reverse and Forward Software engineering

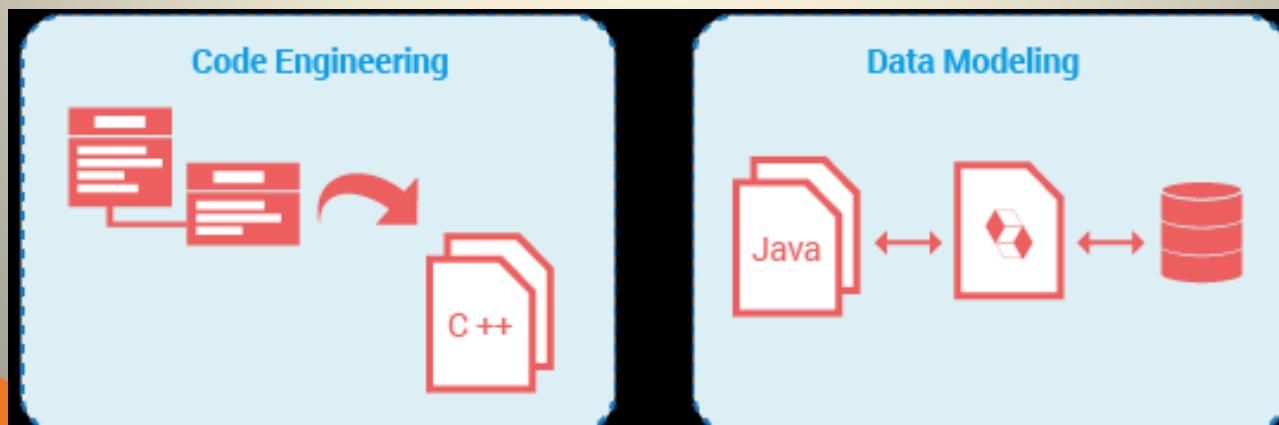


Round-trip engineering (RTE)

- Forward and Reverse engineering combined
- Examples:
 - producing source code from class diagrams and class diagrams from source code.
 - translation from ER-model to relational model and back.
- You can use RTE to build a system in a given programming language.

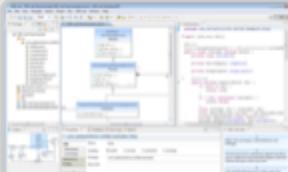
UML to Code, Code to UML

- <https://www.visual-paradigm.com/features/code-engineering-tools/>
 - **Java Round-Trip Engineering**
 - Generate Java source code from UML class model
 - **C++ Round-Trip Engineering**
 - Generate ANSI C++ source code from your UML class model



Model to Code

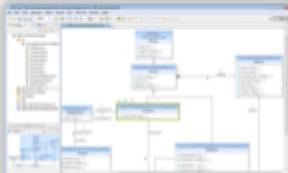
- <http://www.uml-lab.com/en/uml-lab/features/roundtrip/>



GETTING STARTED WITH UML LAB

FROM SOURCE CODE TO UML

Creating a UML model from your existing source code is really easy with UML Lab. We call this Reverse Engineering. Find out just how easy it is in this tutorial.

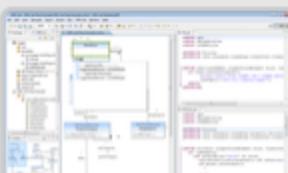


MODELING AND CODE GENERATION

Modeling is a good way to design your software. And when it comes to implementing your design, UML Lab's integrated code generator will save you a lot of time - while keeping you fully agile. This short tutorial will s

ADVANCED TUTORIALS

CREATE YOUR OWN TEMPLATES & CODE STYLES



Create your own templates and Code Styles with UML Lab. Profit from a flexible round-trip engineering that fits your individual needs. Get a better overview of your software projects and save valuable development time by customizing UML Lab.

Implementation types

1. Build

- The previous slides explained how you could build your own system
- You could even get a third party to develop the system / part of system for you.

2. Buy

- In a wide range of domains, it is now possible to buy COMMERCIAL OFF-THE-SHELF systems (COTS) that can be adapted and tailored to the users' requirements.
- For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language

COTS

- Advantages
 - Cheaper
 - Shortens design-to-production cycles
 - General Purpose (more flexible for different applications)
- Disadvantages
 - May not be suitable for all applications
 - May not meet reliability requirements of mission critical systems (flight control, weapons direction, medical equipment)

Implementations types

3. Open source development

- Project/Community open source
 - developed and managed by a distributed community of developers
 - volunteers are invited to participate in the development process.
- Commercial Open Source Software/ COSS
 - controlled by a single entity
 - The owner only accepts code contributions if the contributor transfers copyright of the code to this entity.
 - They may distribute their software for free or a fee.

Activity

- Give few examples of open source products you know and use.

Activity Answer

- Project/Community
 - The best-known open source product is, of course, the Linux operating system which is widely used as a server system and, increasingly, as a desktop environment.
 - Other important open source products are Java, the Apache web server and the mySQL database management system (mySQL has a commercial license as well).
- COSS
 - Red Hat
 - Facebook is the largest COSS code contributor

Open Source

- Advantages
 - Free to Try
 - Free Support
 - Fewer Bugs and Faster Fixes
- Disadvantages
 - Free support is not always the fastest support
 - Not a favorite for unskilled users

Build

- When building your own system
 - Confirm the detailed design
 - Understand required standards. For example,
Coding Standards
 - Implement Code
 - Plan the structure based on the SDD (Software Detailed Design)
 - Code -> apply standards -> self inspect -> compile -> unit test
 - Try to reuse as much as possible

Coding Standards

Coding Standards

- You have already used Coding Standards in IP.
 - Indentation
 - Commenting Code
 - Whitespace
 - Naming Variables

```
// Printing on one line with two printf statements.
#include <stdio.h>

// function main begins program execution
int main( void )
{
    printf( "Welcome " );
    printf( "to C!\n" );
} // end function main
```

← 1, 2, 3

```
printf( "Sum is %d\n", sum ); // print sum
scanf( "%d", &integer1 ); // read an integer
```

Code A

```
if (g < 17 && h < 22 || i < 60) {  
    return true;  
} else {  
    System.out.println ("incorrect");  
    return false;  
}
```

Code B

```
if (g < 17 && h < 22 || i < 60)  
{  
    return true;  
}  
else  
{  
    System.out.println("incorrect");  
  
    return false;  
}
```

What code is following indentation ?

Code A

```
if (g < 17 && h < 22 || i < 60) {  
    return true;  
} else {  
    System.out.println ("incorrect");  
    return false; }
```

Code B

```
if (g < 17 && h < 22 || i < 60)  
{  
    return true;  
}  
else  
{  
    System.out.println("incorrect");  
  
    return false;  
}
```

What code is following indentation?

Code B

Activity

Code A

```
for(int i=0;i<40;i++)  
    {system.out.println(i);}
```

Rewrite the above code according to the coding standards you learnt.

Activity Answer

Code A

```
for(int i=0;i<40;i++)  
    {system.out.println(i);}
```

Code Answer

```
for( int i = 0 ; i < 40 ; i++ ){  
    system.out.println(i);  
}
```

References

- Software Engineering – 10th Edition by Ian Sommerville, Chapter 7
- <https://www.visual-paradigm.com/features/code-engineering-tools/>
- <http://www.uml-lab.com/en/uml-lab/features/roundtrip/>
- Courseweb Documents
 - Coding Standard Document
 - open-source-vs-proprietary-software-pros-and-cons

Software Process Modeling

Software Testing and Maintenance

Session Outcomes

- Testing
 - V & V
 - Types of testing
 - Black Box testing
 - White box testing
 - Software Testing Strategies
 - Unit testing
 - Integration Testing
 - System Testing
 - Acceptance Testing
- Maintenance

Story So Far ...

- You have already learned how to carry out
 - Feasibility Study
 - Requirements Gathering, Analysis and Specification
 - Design
 - Implementation
- Now you have an implemented system with you to start testing.....

Importance of testing

- Defective Software
 - In 1992, Mary Bandar received an invitation to attend a kindergarten in Winona, Minnesota, along with others born in '88. Mary was born in 1888 and 104 years old at the time.
 - <http://www.apnewsarchive.com/1993/Woman-Born-in-1888-Gets-Surprise-Invitation-To-Kindergarten/id-a0d5e1ddb4be59cdc6b066d0e9517e29>
 - An F-18 crashed because of a missing exception condition: if ... then ... without the else clause that was thought could not possibly arise.
 - Computer-Related Risks -By Peter G. Neumann

What is Software Testing?

- “Software Testing is the process of executing a program or system with the intent of finding errors” [Myers, 79].
- “Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence” [Dijkstra, 1972]

Verification vs validation

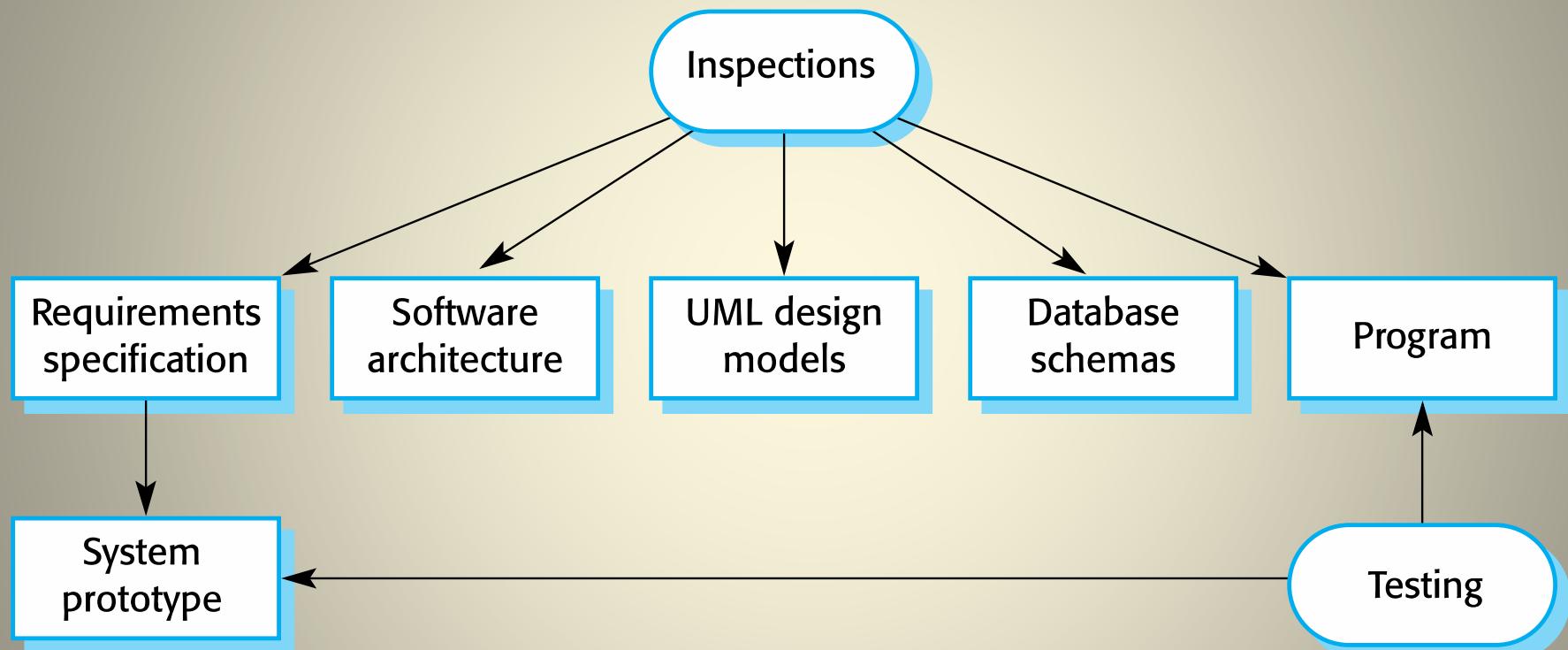
- Verification:
"Are we building the product right"
 - The software should conform to its specification – functional and non-functional requirements
- Validation:
"Are we building the right product"
 - The software should do what the user really requires which **might** be different from specification

V & V Techniques

- *Software inspections* - Concerned with analysis of the static system representation to discover problems (static verification)
 - No need to execute a software to verify it.
- *Software testing* - Concerned with exercising and observing product behaviour (dynamic verification)
 - The system is executed with test data and its operational behaviour is observed

Ref: Software Engineering, I. Sommerville, 10th Edition

Inspections and testing



Ref: Software Engineering, I. Sommerville, 10th Edition

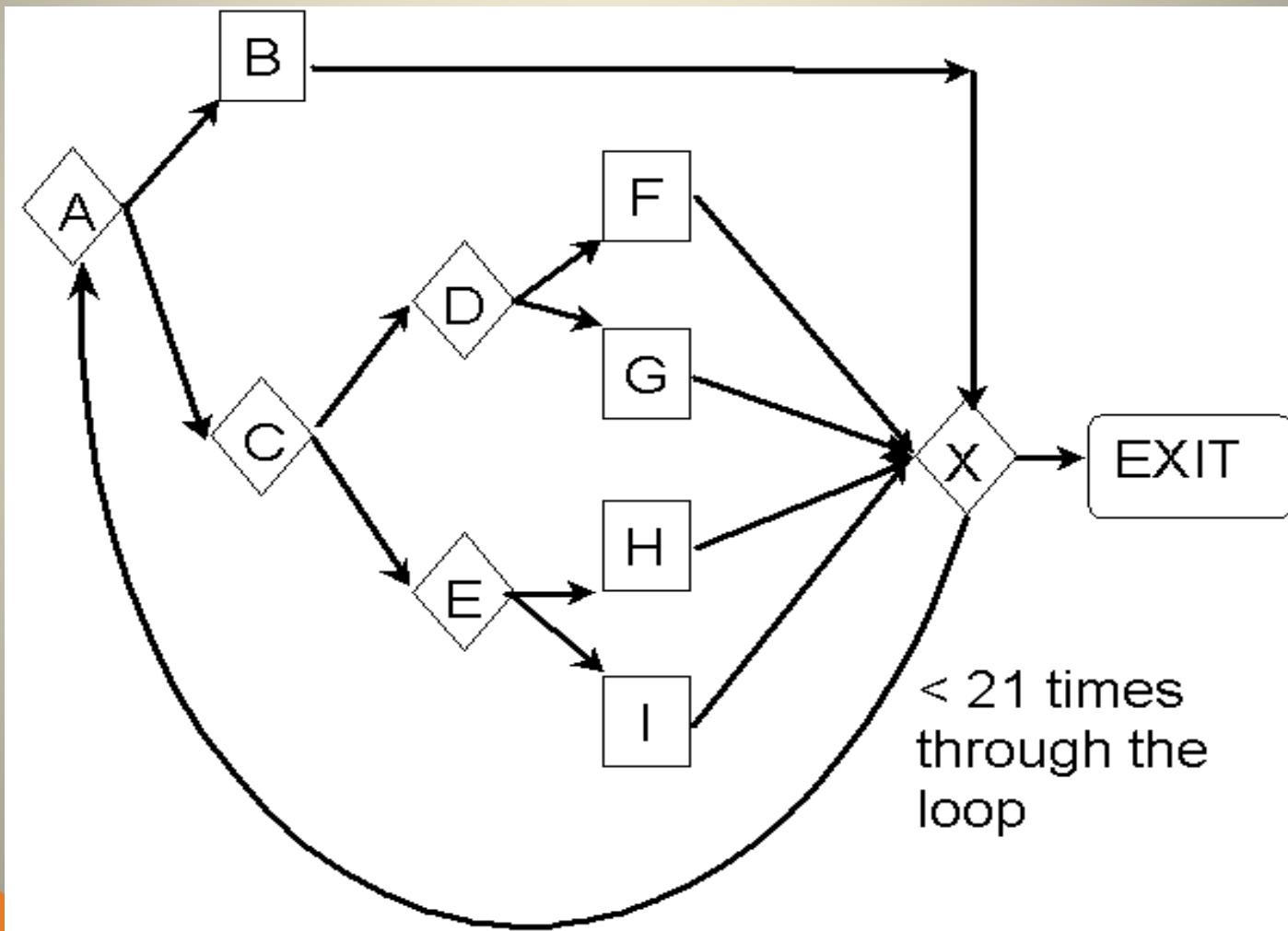
Think...

- Can we have a 100% error free s/w?
 - It is not possible to *prove* that there are no faults in the software using testing
- Why?

Why Can't Every Defect be Found?

- Too many possible paths.
- Too many possible inputs.
- Too many possible user environments.

Too Many Possible Paths

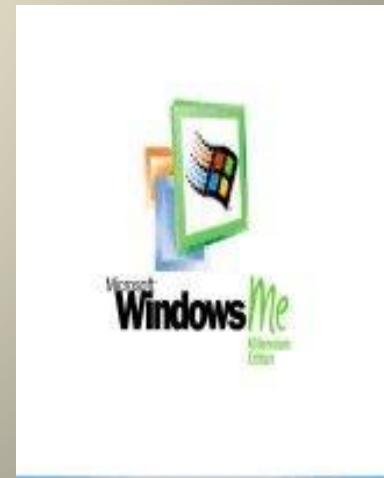
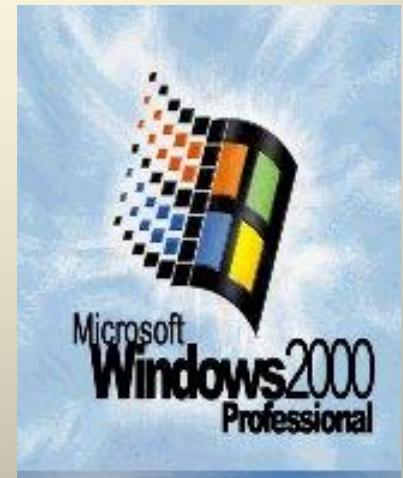
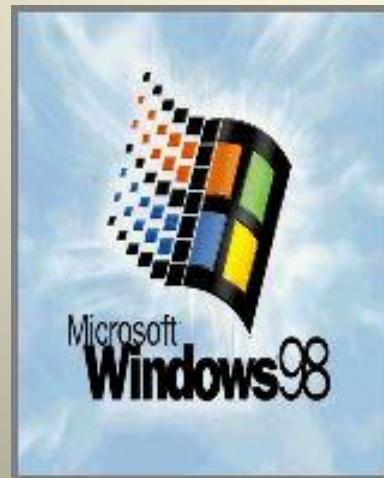
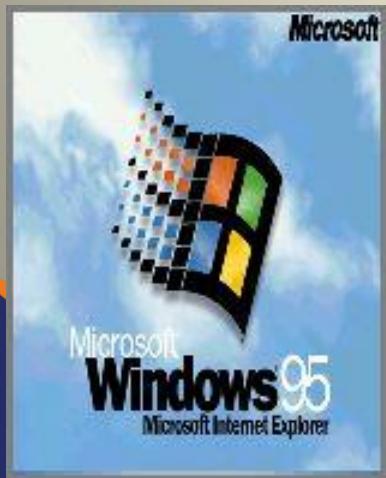


Too Many Possible Inputs

- Programs take input in a variety of ways: mouse, keyboard, and other devices.
- Must test Valid and Invalid inputs.
- Most importantly, there are an infinite amount of sequences of inputs to be tested.

Too Many Possible User Environments

- Difficult to replicate the user's combination of hardware, peripherals, OS, and applications.
- Impossible to replicate a thousand-node network to test networking software.



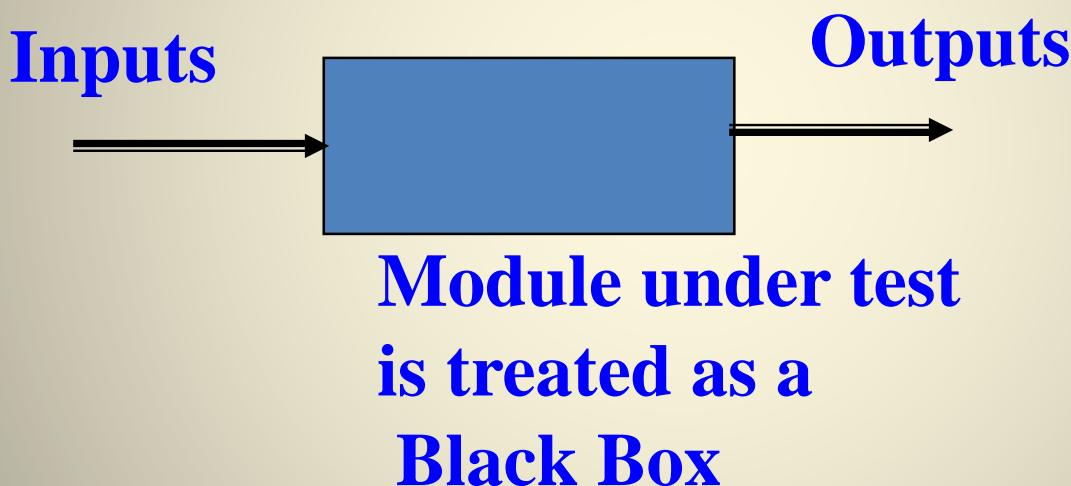
Types of Testing

- Exhaustively testing of all possible input/output combinations is too expensive
- There are 2 types of testing.
 - Black box testing (Functional testing)
 - Structural testing (White box testing)

Ref: Software Engineering, I. Sommerville, 10th Edition

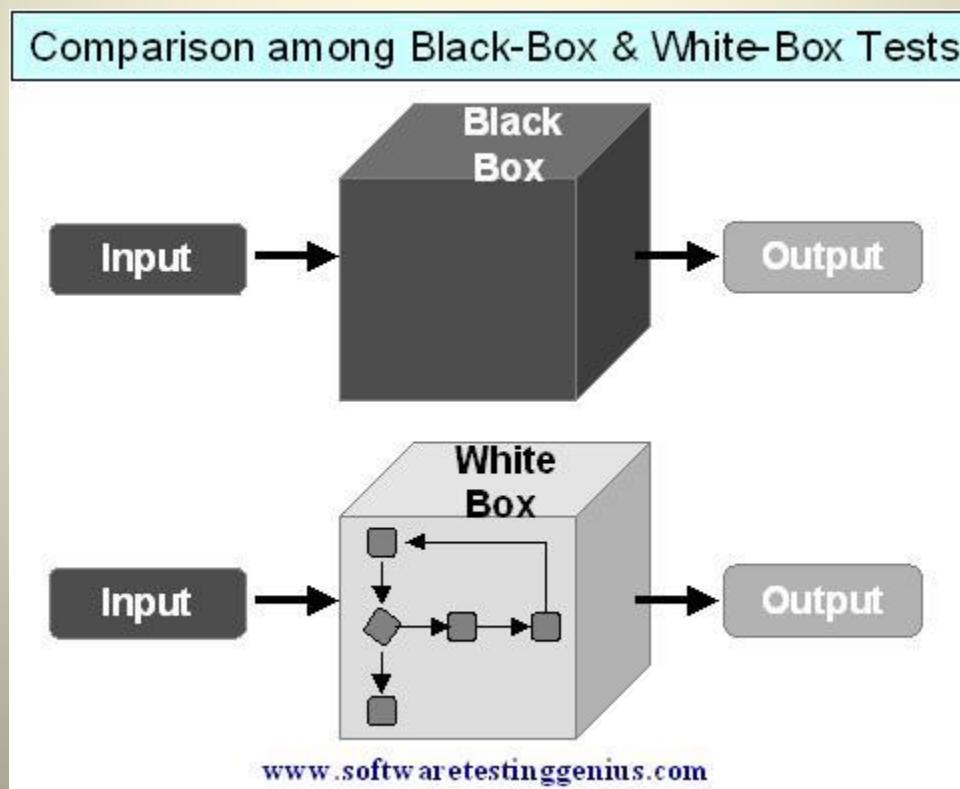
Black Box Testing

- Testing focus on the software functional requirements, and input/output.



White box testing

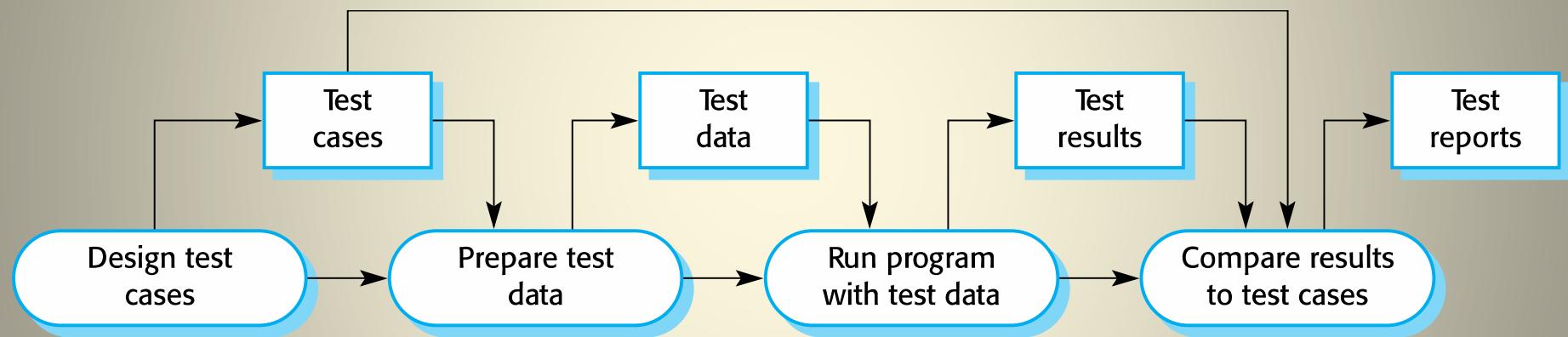
- Testing is based on the structure of the program.
 - In white box testing internal structure of the program is taken into account
 - The test data is derived from the structure of the software



Activity 1

- What are the differences between Black Box and White box Testing ?

A model of the software testing process



Ref: Software Engineering, I. Sommerville, 10th Edition

Black box testing strategies

- **Equivalence Class Testing:**
 - It is used to minimize the number of possible test cases to a best level while maintaining reasonable test coverage.
- **Boundary Value Testing:**
 - Boundary value testing is focused on the values at boundaries.
 - This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases.

Equivalence partitioning

Example

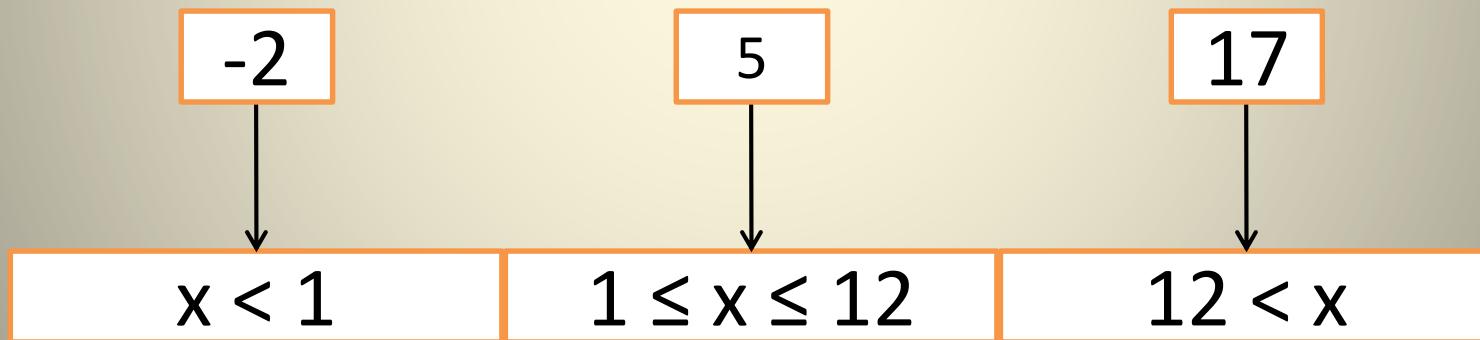
- Example of a function which takes a parameter “month”.
- The valid range for the month is 1 to 12, representing January to December. This valid range is called a partition.
- In this example there are two further partitions of invalid ranges.

 $x < 1$ $1 \leq x \leq 12$ $12 < x$

Equivalence partitioning

Example

- Test cases are chosen so that each partition would be tested.



Equivalence partitions

- In equivalence-partitioning technique we need to test only one condition from each partition.
 - This is because we are assuming that all the conditions in one partition will be treated in the same way by the software.
- Equivalence partitioning is an effective approach to testing because it helps to reduce the number of tests.

Equivalence partitioning

Example

- In company XYZ, to become a staff member one has to be 18 years old. The retirement age is 57.
 - Find the test values for the employment age.

Activity

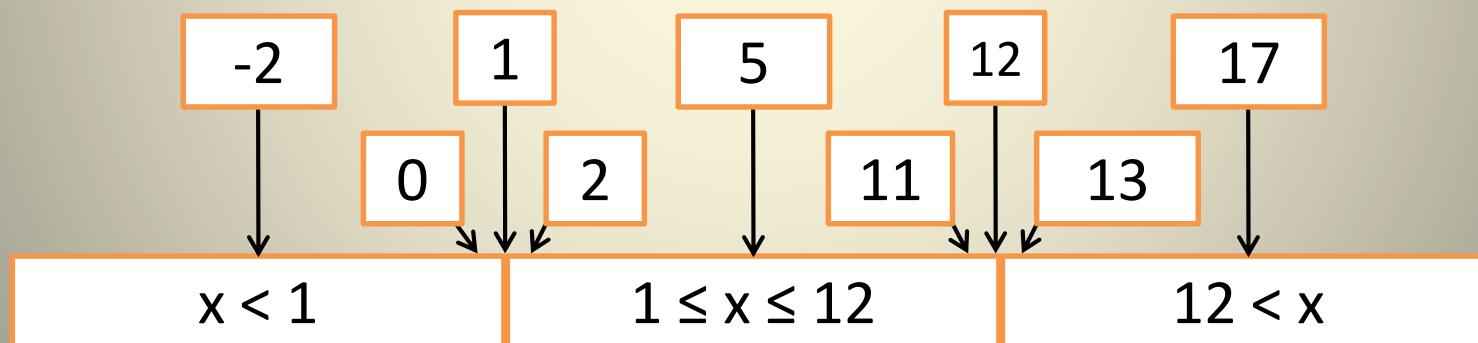
- In a Library, students can borrow books. There is a limit given for students on the number of books they can borrow at one time. A student can borrow 2-5 books at one time.
- With equivalence partitioning identify the test values to check the borrowing **limit** of a student at **one time**.

Boundary value analysis

- Equivalence partitioning is not a stand alone method to determine test cases. It is usually supplemented by ***boundary value analysis***.
- ***Boundary value analysis*** focuses on values on the edge of an equivalence partition or at the smallest value on either side of an edge.

Equivalence partitioning with boundary value analysis

- Example of a function which takes a parameter “month”.
- Test cases are supplemented with ***boundary values***.



Boundary Value Analysis Example

- Write test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis

Boundary Value Analysis - Example

- Test Cases
 - 1: Consider test data exactly as the **input boundaries** of input domain i.e. values 1 and 1000.
 - 2: Consider test data with values just **below the extreme edges** of input domains i.e. values 0 and 999.
 - 3: Consider test data with values just **above the extreme edges** of input domain i.e. values 2 and 1001.

Activity

An online Movie ticket booking system lets users book movie tickets for currently playing movies and newly releasing movies. When a user is going to make a booking, he/she must select type of movie (English/Tamil/Hindi) and enter Type of ticket (Full/Half). Also must enter No of Tickets wanted (1 to 10) and if taking any children (age 3 to 12) must fill as “Yes” if not “No”. The maximum no of tickets allowed for a booking is 10. The system interface shows multiple text boxes for the user to enter the needed values.

Write 8 Sample test data to check the format and correctness of user-entered data to the above system using equivalence partitioning and Boundary Value Analysis. Use the Format given below.

Software Testing Levels

- Unit testing : Individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
- Integration testing : Several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
- System testing: Some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.
- Acceptance Testing: Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

Unit testing

- Unit testing is the process of testing individual components in isolation.
- Unit testing can be automated, but may still be performed manually.
- Units may be:
 - Individual functions or methods within an object
 - Object classes with several attributes and methods

Ref: Software Engineering, I. Sommerville, 10th Edition

Assert

- Last semester in IP module you used “assert” expression.
 - It can be used as a means of checking the programs expected behavior.
 - It can be used for unit testing.

assert(<boolean expression>)

Introduction

- This simple function requires a boolean (conditional) expression.
- An assertion is a fact that we strongly believe to be true.
 - e.g. assert (5 > 2) or assert (20 == 20)
- When your program is executed the assertion is checked and if the condition is true, it simply moves to the next statement in the code.
- The assert () function requires the <assert.h> header file.
- You can disable assertions from production code by using the following directive before #include <assert.h>

#define NDEBUG

Quiz

- What will be the outcome of the following assertions (pass/fail).
- Assume that char grade = 'B'; int radius= 254; int perimeter = 40
 - 1. assert(grade == 'A') ;
 - 2. assert(grade == 'A' || grade == 'B' || grade = 'C') ;
 - 3. assert(radius == 34) ;
 - 4. assert(radius == 254) ;
 - 5. assert(perimeter < 50) ;

Activity

- Write a unit testing function for the following program using “assert”.

```
interface Adder {  
    int add(int a, int b);  
}  
  
class AdderImpl implements Adder {  
    int add(int a, int b) {  
        return a + b;  
    }  
}
```

Unit Testing

Method Used for unit testing: White Box Testing method is used for executing the unit test.

When should Unit testing be done?

Unit testing should be done before Integration testing.

By whom should unit testing be done?

Unit testing should be done by the developers.

Advantages of Unit testing:

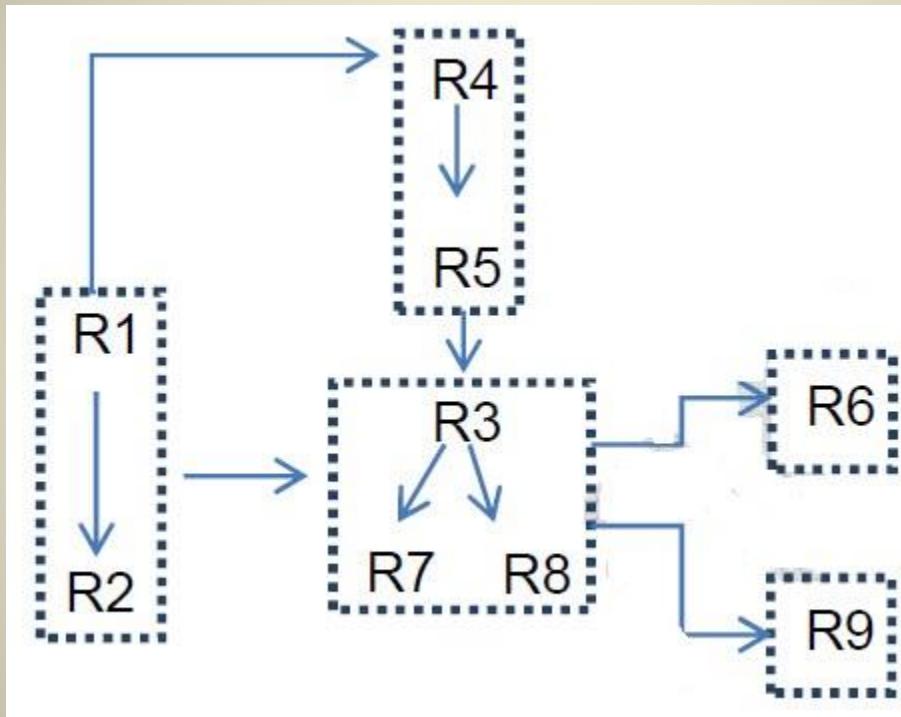
1. Issues are found at early stage. Since unit testing are carried out by developers where they test their individual code before the integration. Hence the issues can be found very early and can be resolved then and there without impacting the other piece of codes.
2. Unit testing helps in maintaining and changing the code.
3. Since the bugs are found early in unit testing it also helps in reducing the cost of bug fixes.

Integration Testing

- Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.
- It occurs **after** unit testing and **before** system testing.
- Integration Testing
 - Big Bang Testing
 - Top Down Testing
 - Bottom Up Testing

Activity

- Requirements logical dependencies ($A \rightarrow B$ means that B is dependent on A.)



How would you structure the test execution schedule according to the above criteria?

Big Bang Testing

- In Big Bang Integration testing, individual modules of the programs are not integrated until **every thing** is ready. It is called 'Run it and see' approach.
- In this approach, the program is integrated **without any formal integration testing**, and then run to ensure that all the components are working properly.



Activity

- **Big Bang Approach** : In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole.
- Is it a good method? Why ?

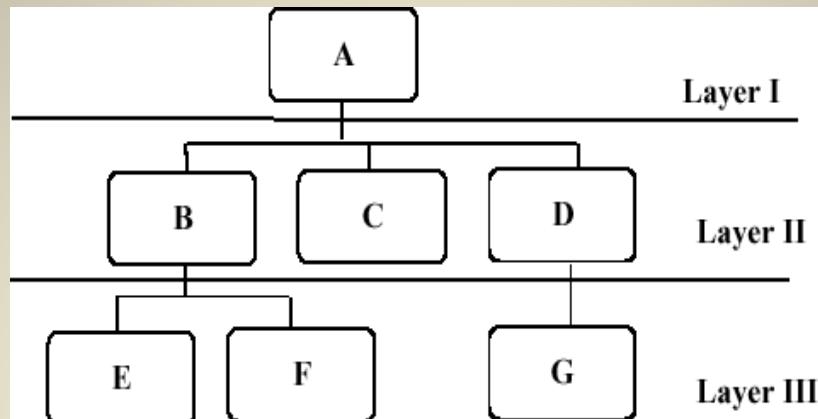
Activity Answer

- **Disadvantage:** The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.
- The chances of having critical failures are more because of integrating all the components together at same time.

Top-down Testing

- Top down integration testing is an incremental integration testing technique which begins by testing the top level module and progressively adding in lower level module one by one.
- It provides early working module of the program and so design defects can be found and corrected early.

Top-down Testing



Test A

B,C,D can be Stubs

Layer I

Test A, B, C, D

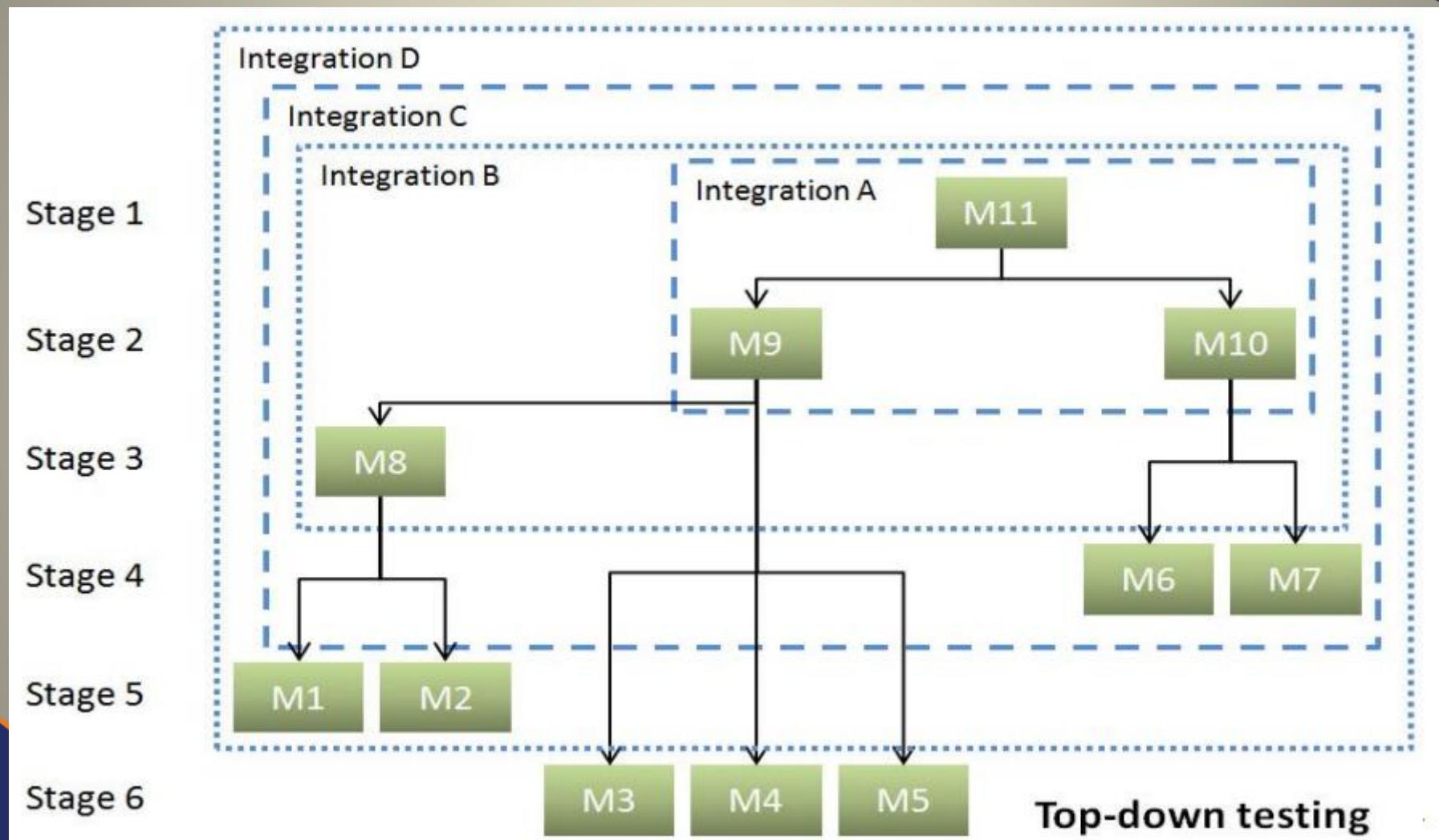
E, F, G can be Stubs

Layer I + II

**Test
A, B, C, D,
E, F, G**

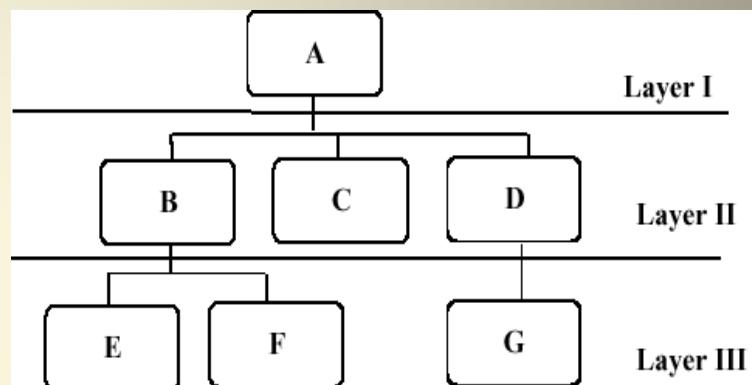
All Layers

Top-down Testing

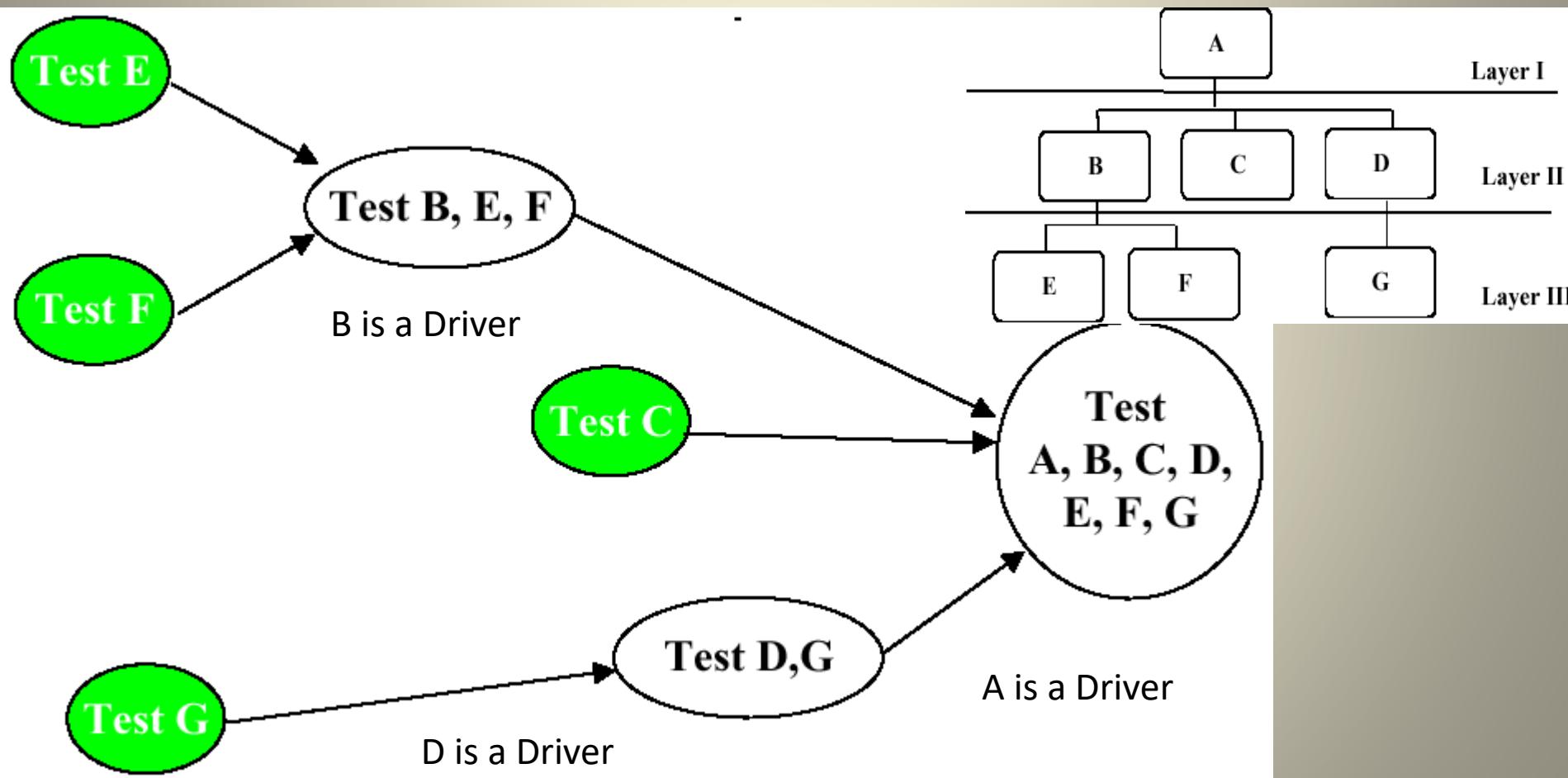


Bottom-up Integration Testing

- In bottom up integration testing, module at the lowest level are developed first and other modules which go towards the 'main' program are integrated and tested one at a time.
- In this approach, lower level modules are tested extensively thus make sure that highest used module is tested properly



Bottom-up Testing Graphical Representation



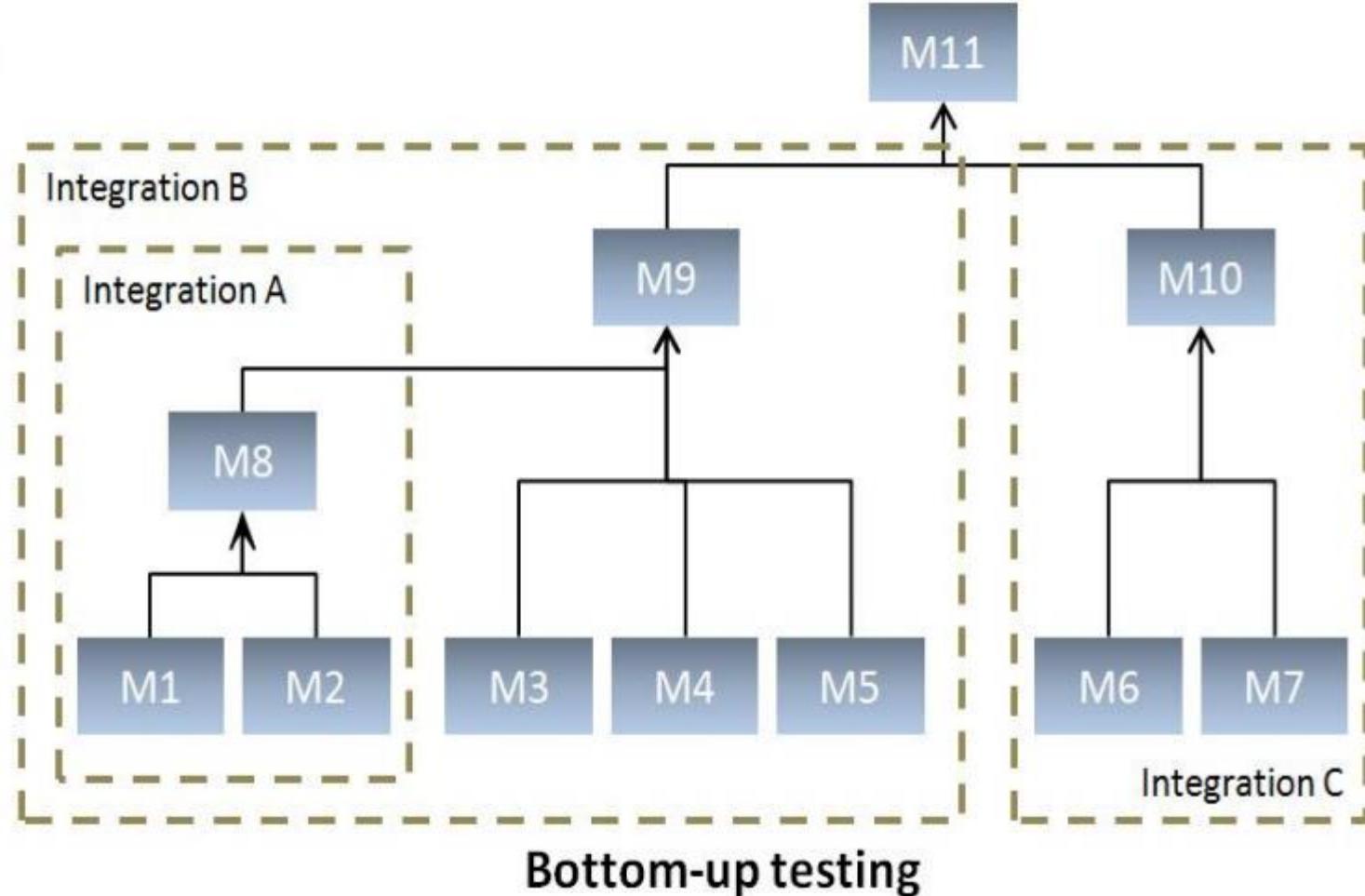
Bottom-up Testing

Stage 4

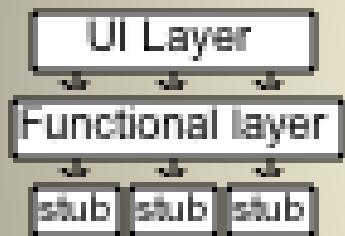
Stage 3

Stage 2

Stage 1

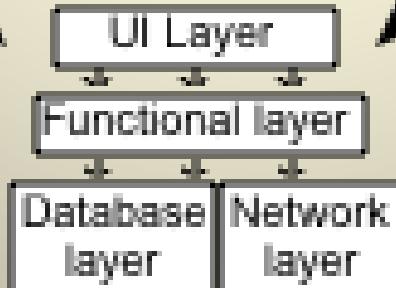
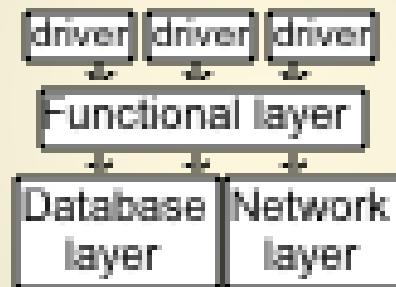


Top-down testing



Fully integrated system

Bottom-up testing



Sandwich testing



System testing

- System testing is the testing of a complete and fully integrated software product.
- System testing is actually a series of different tests whose sole purpose is to exercise the full computer based system.

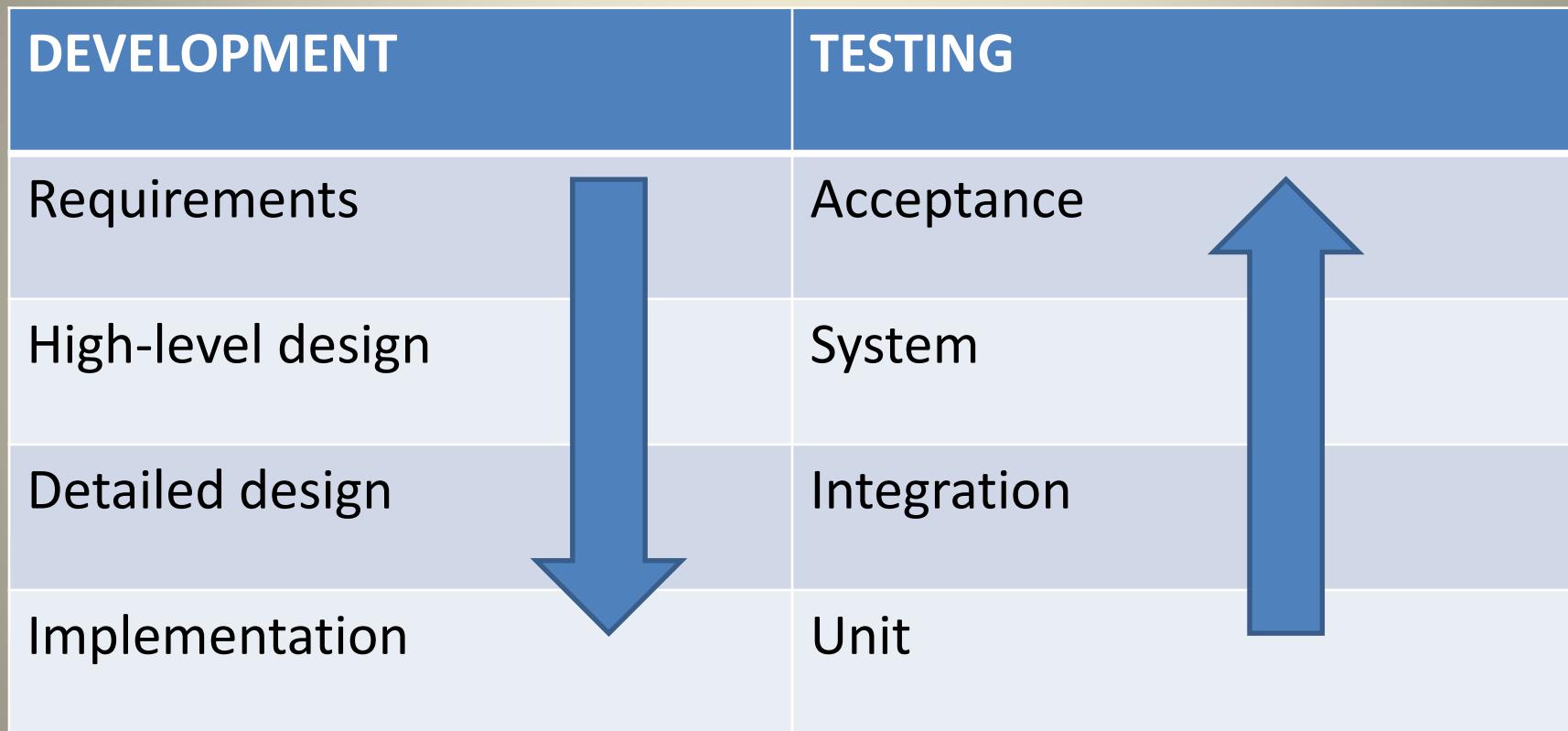
Ref: Software Engineering, I. Sommerville, 10th Edition

Acceptance testing

- Alpha testing
 - Users of the software work with the development team to test the software at the **developer's site**.
- Beta testing
 - A release of the software is made available to users to allow them to experiment and to raise problems that they discover.
 - tested by users in an **uncontrolled environment**

Ref: Software Engineering, I. Sommerville, 10th Edition

Life cycle model for testing



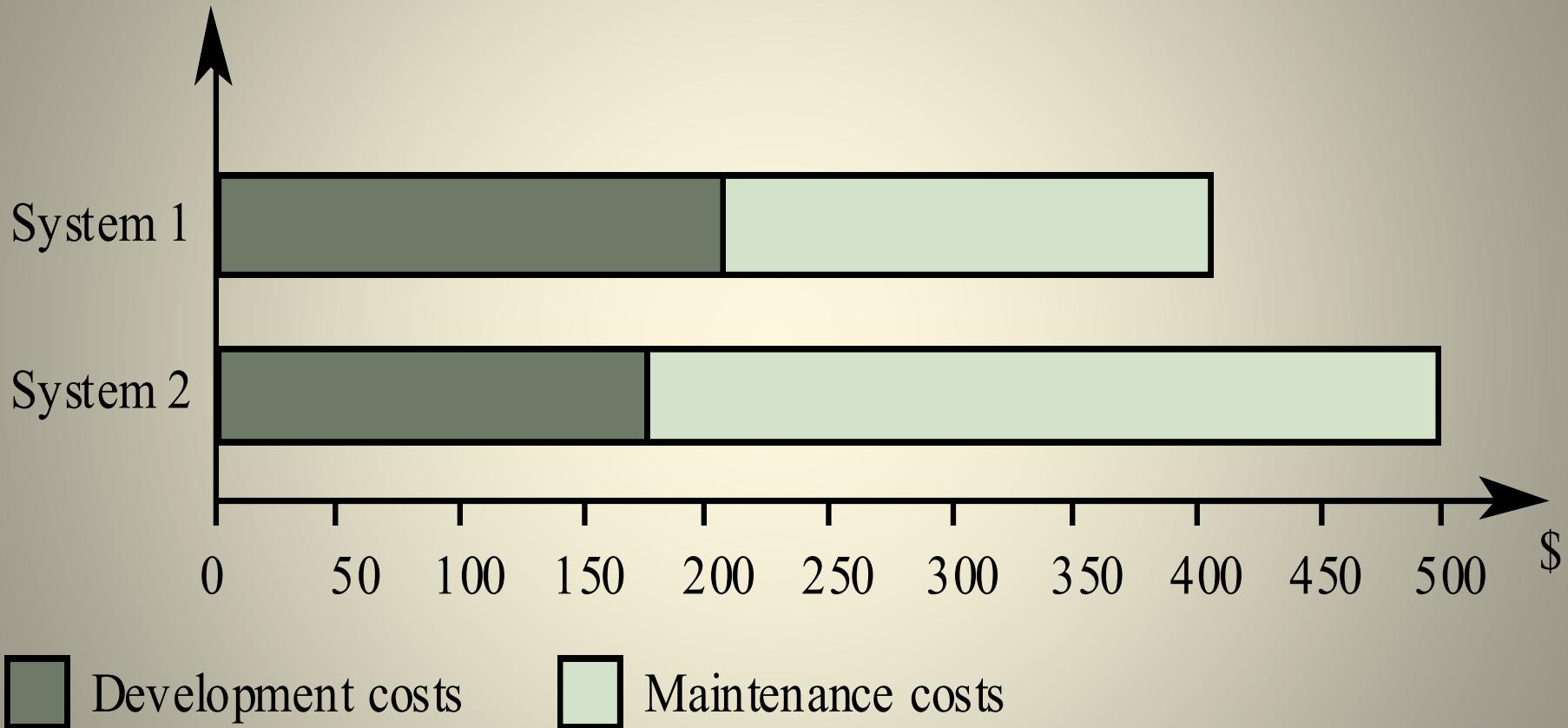
Maintenance

Software Maintenance

“Managing the processes of system change”

- Modifying a program after it has been put into use.
- Software Maintenance results in a service being delivered to the customer.
- The change may be simple changes to correct coding errors, more extensive changes to correct design errors or significant enhancement to correct specification error or accommodate new requirements

Development/Maintenance Costs



Types of Maintenance

- **Perfective maintenance**
 - Changing a system to make it meet its requirements more effective. Adding Functionality.
- **Adaptive maintenance**
 - Maintenance due to changes in environment. New Operating Systems, new hardware
- **Corrective maintenance**
 - Correct newly found bugs

References

- Software Engineering – 10th Edition by Ian Sommerville, Chapter 8

Modern Software Development Methodologies

AGILE

Session Outcomes

- Traditional Development Methodologies
- What is AGILE?
- Agile Development Methodologies
 - SCRUM
 - SCRUM Roles
 - SCRUM Processes
 - SCRUM Artefacts
 - SCRUM Tools

Traditional Models

- What are the traditional development models you know of?
 - Classical Waterfall model
 - Iterative waterfall model
 - Prototype model
 - Spiral model

Traditional Models

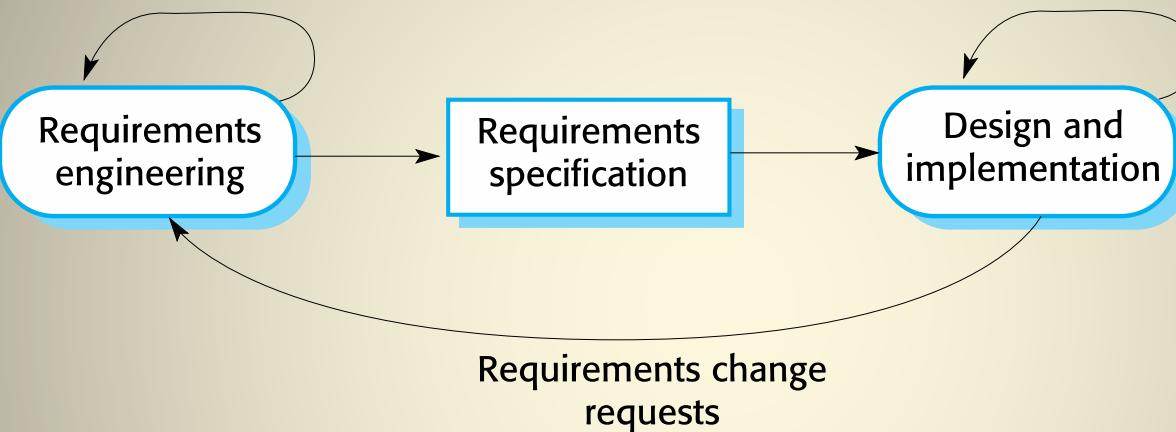
- What are the known issues in traditional development methods?
 - High Cost
 - Changes are not acceptable
 - Can detect errors only in the latter part of the SDLC
 - Less or no iterations
 - Lack of transparency

Plan-based Vs AGILE

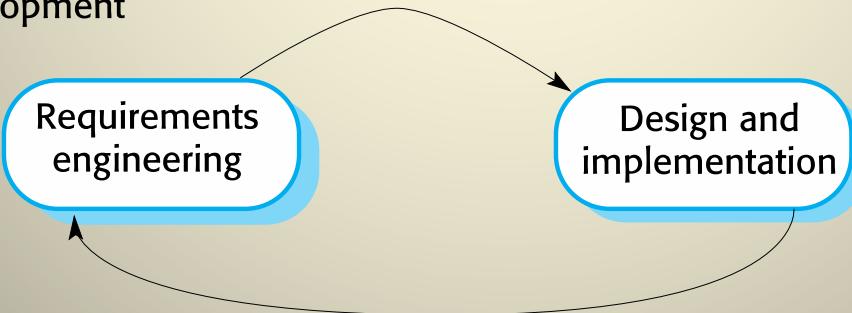
- Some types of software need a complete analysis of the system and proper planning beforehand. **Plan-based development**
 - Safety critical control systems
- But what about a system used in a fast moving business environment? Can we use plan-based development?
 - **AGILE Development !**

AGILE Vs Plan-based development

Plan-based development



Agile development



Ref: Software Engineering, I. Sommerville, 10th Edition

AGILE

What is AGILE?- Definition

- Agile Software Development is an umbrella term for a set of methods and practices based on the values and principles expressed in the **Agile Manifesto**.
- Solutions evolve through collaboration between self-organizing, cross-functional teams utilizing the appropriate practices for their context.

Ref: What is Agile Software Development?", Agile Alliance, 2017

In Summary

- AGILE provides a foundation for the teams to build software
 - With the **highest value**
 - With **high quality**
 - With in the **shortest time**

Activity

- Do you have an Agile or Fixed mindset?
 - Form your project group.
 - Choose a Scrum Master for your group.
 - Scrum Master with the help of the group categorize the following items into An agile or growth mindset/A fixed mindset
 - an agile or growth mindset is an attitude that equates failure and problems with opportunities for learning
 - a fixed mindset believes that basic skills, intelligence, and qualities are inherent and fixed.

Activity Items

- Learn by doing
- Feedback is about current capabilities
- Right the first time
- Fail fast
- Think and learn
- Feedback is personal
- Embrace challenges
- Desire to look smart
- No need to change or improve
- Avoid Failure
- Can be wrong the first time
- Trust team members

AGILE manifesto

- The Agile Manifesto, also called the Manifesto for Agile Software Development, is a formal proclamation of **four key values** and **12 principles** to guide an iterative and people-centric approach to software development.

Ref: What is Agile Manifesto? - Definition from [WhatIs.com](#)

4 Key values

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

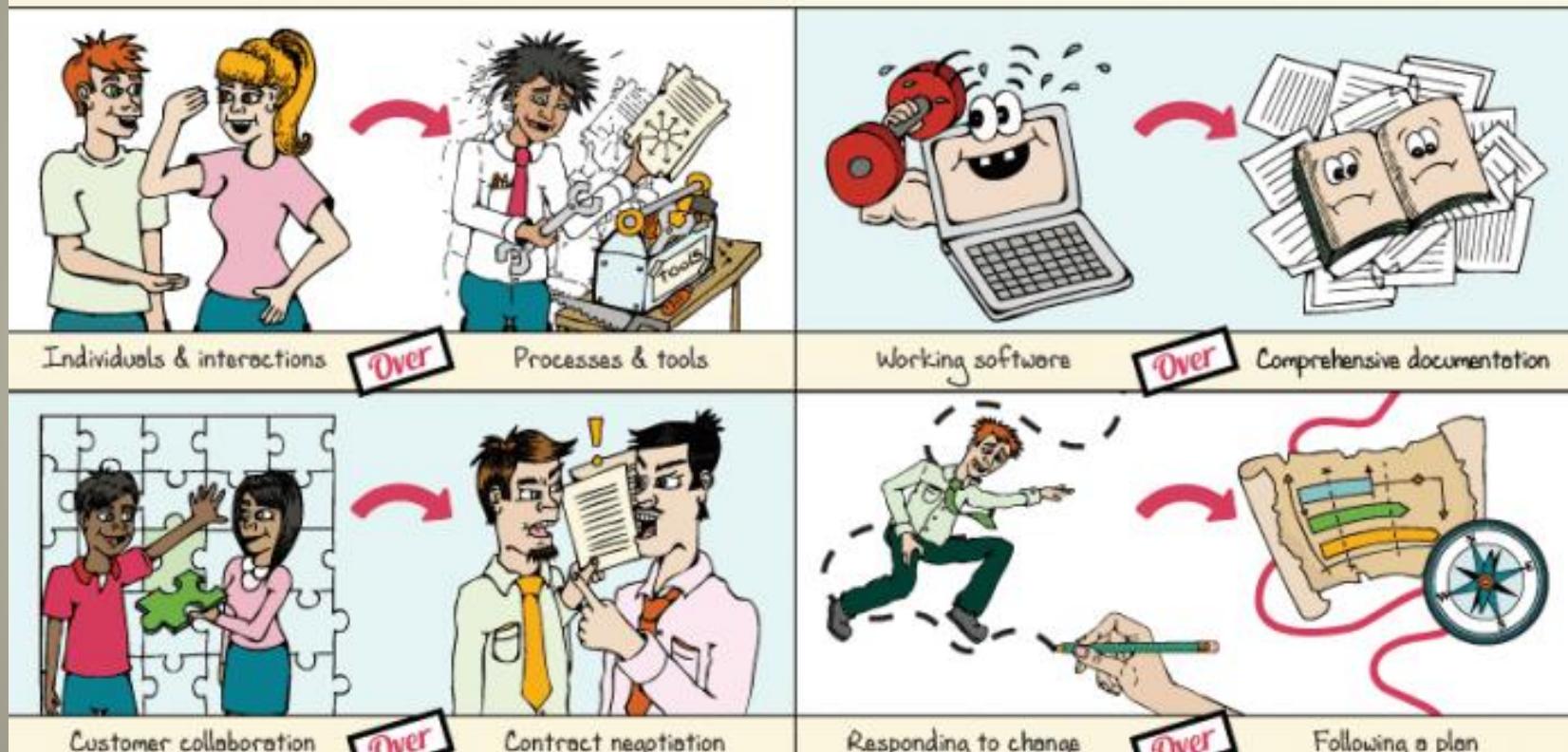
- 1. Individuals and interactions** over processes and tools
- 2. Working software** over comprehensive documentation
- 3. Customer collaboration** over contract negotiation
- 4. Responding to change** over following a plan

That is, while there is value in the items on the right , we value the items on the left more (bold ones). ”

Ref: “Manifesto for Agile Software Development”, Agile Alliance, 2017 / <http://www.agilemanifesto.org>

Agile Manifesto

"We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:



Activity

- Compare and contrast a Library Management System if developed with traditional way and with AGILE.

Different Trends of AGILE

- SCRUM
- eXtreme Programming (XP)
- Test Driven Development (TDD)
- Pair Programming
- Behaviour Driven Development
- Lean Software Development
- Kanban

Agile Alliance (www.agilealliance.org)

– A non-profit organization promotes agile development

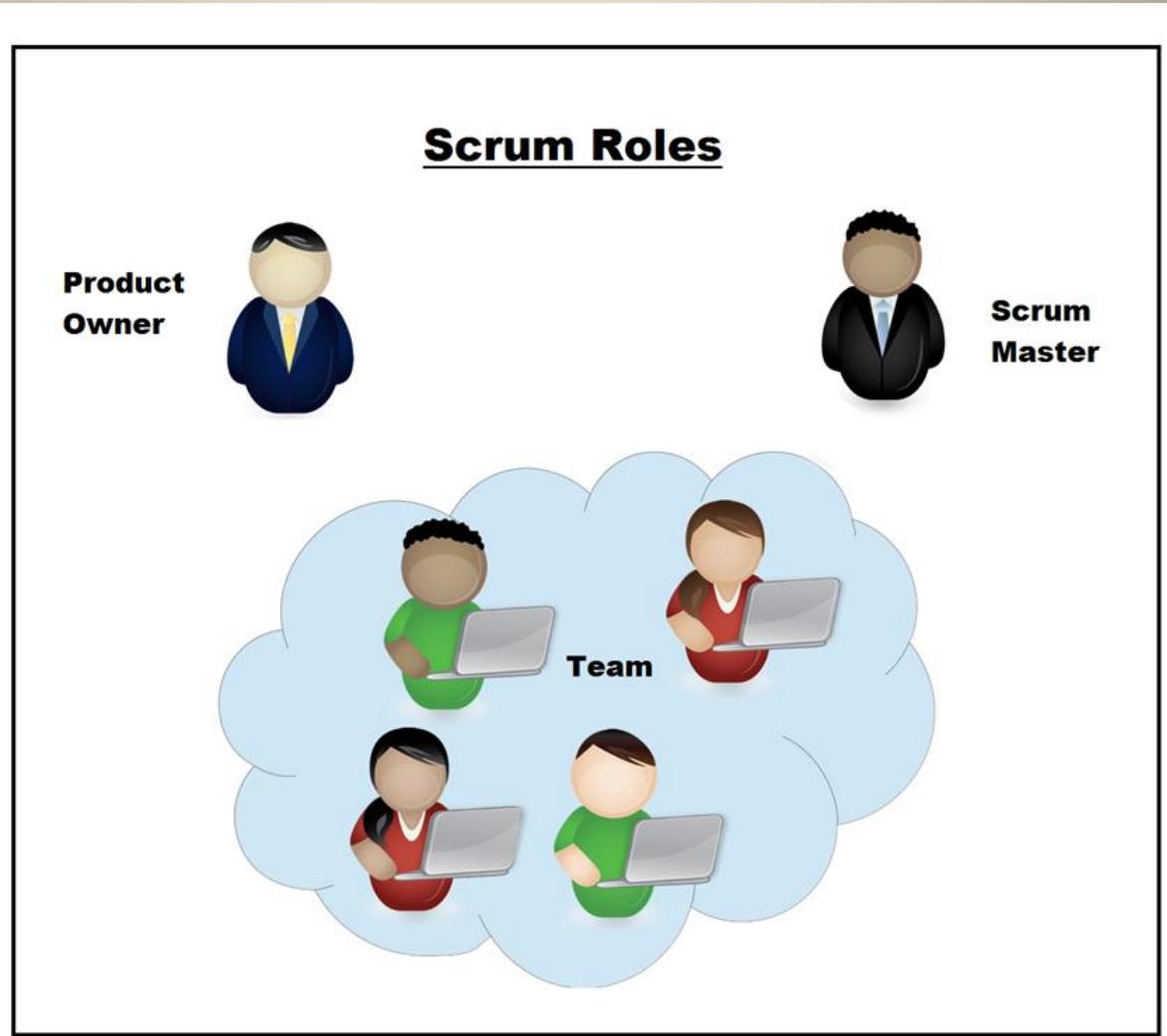
SCRUM

What is SCRUM?

- Scrum is a management and control process that cuts through complexity to focus on building software that meets business needs.
- Components
 - SCRUM Roles
 - SCRUM Activities
 - SCRUM Artifacts

Ref: "What is Scrum?", Scrum.org, 2017

SCRUM Roles



SCRUM Roles

- **Product Owner**
 - Client's representative
 - Define the features of the product
 - Decide on release date and content
 - Accept or reject work results
- **SCRUM Master**
 - Represents management to the project
 - Removes the impediments
 - Shield the team from external interferences

SCRUM Roles

- **Dev Team**
 - Cross-functional
 - QA, Programmers, UI Designers, etc.
 - Work collaboratively and share responsibilities.
 - Typically 5-10 people
- **Users/Stakeholders**
 - Those who are going to use the product or have a vested interest in how it turns out.

Activity

1. How should work be allocated to the team in an Agile project?
 - a) The Team Leader(Scrum Master) should allocate specific tasks to individuals
 - b) Tasks should be randomly allocated to team members, using Planning Poker
 - c) Team members should self select tasks appropriate to their skills
 - d) The most complex tasks should be allocated by the Team Leader (Scrum Master)

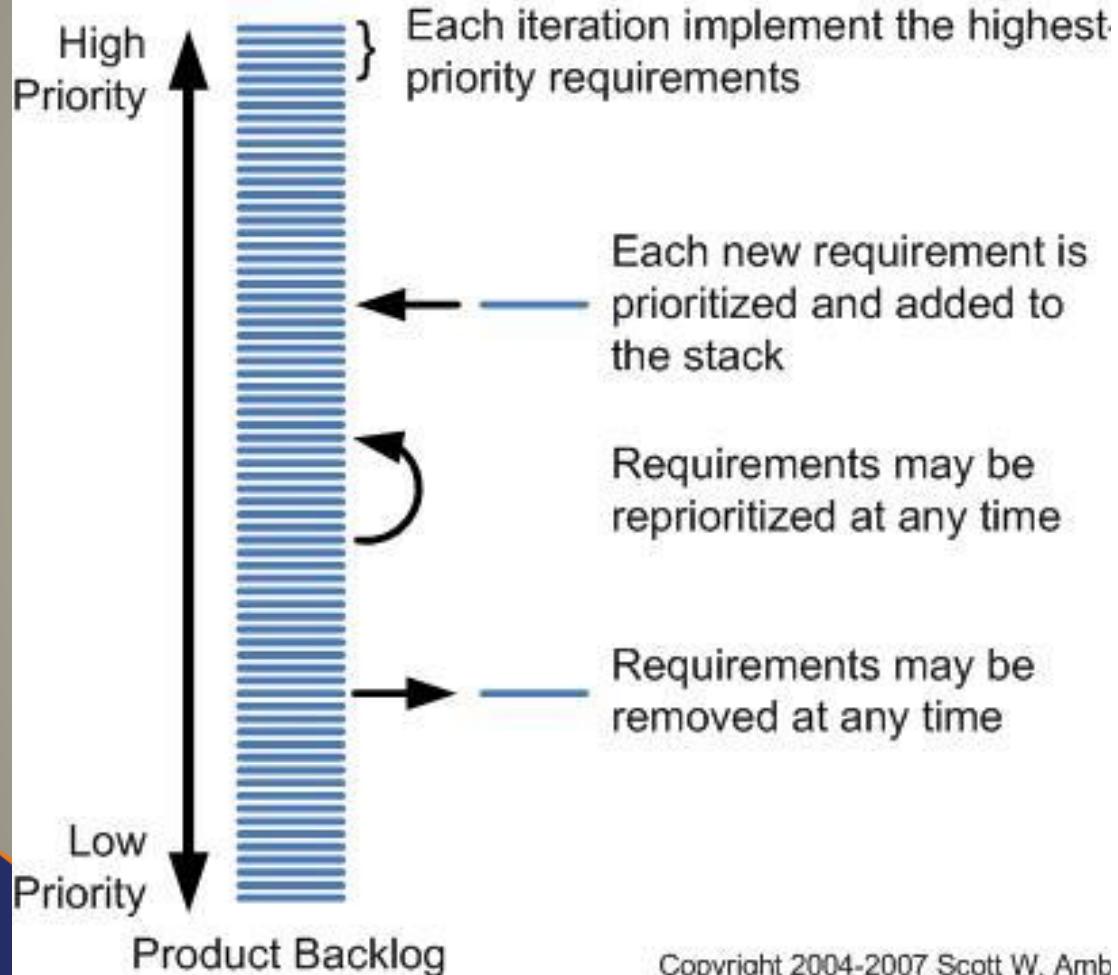
Activity

1. An Agile team ...
 - a) Is self - organizing, with each member having the same technical skills.
 - b) Collaborates and supports its team members
 - c) Ensures that weak members of the team are allocated the simpler tasks
 - d) Ensures blame is allocated fairly

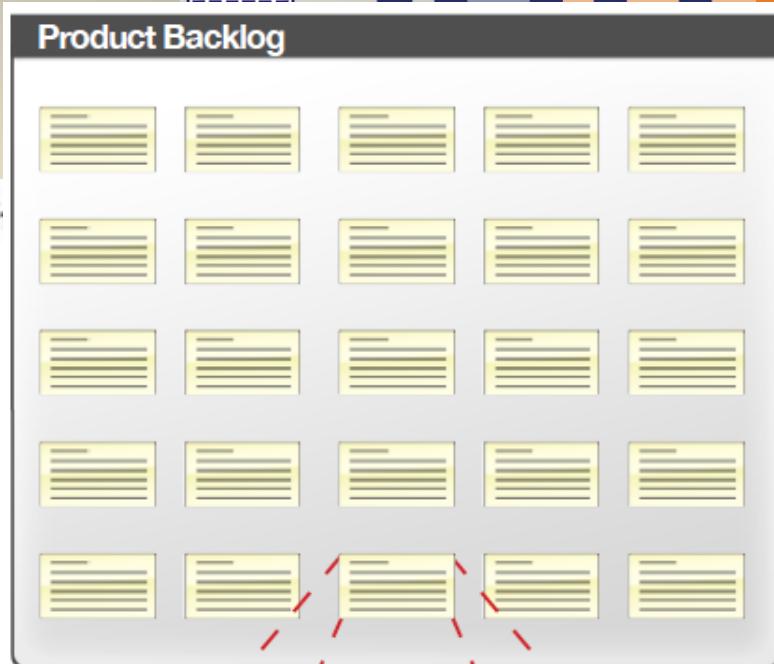
SCRUM Artifacts

- Product Backlog
- Sprint Backlog
- Burn down Charts

Product Backlog



Copyright 2004-2007 Scott W. Ambler

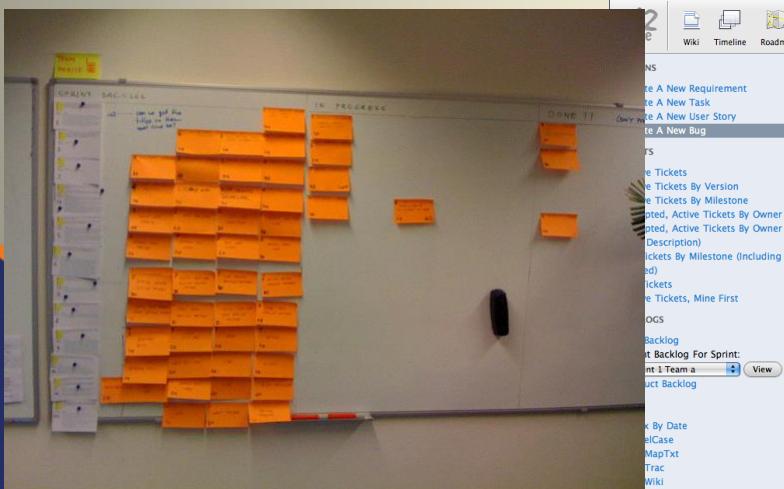


User Story

As (role), I want (feature), so that (benefit).

Creating the product backlog

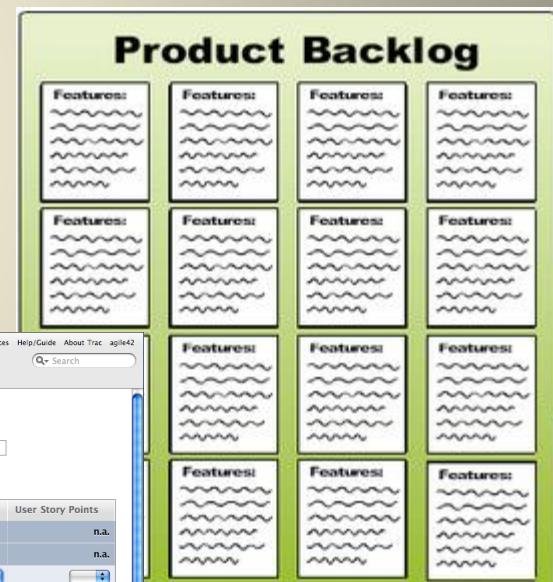
- There are many ways to store the product backlog:
 - As a collection of index cards or post-its on the wall
 - On a flip-chart
 - In a requirements management tool
 - In Excel



Product Backlog (displaying some of 13 items)

Legenda Requirement Story Task Accepted ✓ Done ✎ Closed Bug

ID	Summary	Business Value Points	Rofit	Importance	User Story Points
#8	Req 4	3000	n.a.		n.a.
#2	Req 2	2000	n.a.		n.a.
#19	5th User Story for Req 2		Mandatory		
#39	Req7	1200	n.a.		n.a.
#40	Story for Requirement 7		Linear		
#37	req 6	1200	n.a.		n.a.
#38	This Story belongs to req 6				
#1	Req 1	500	38.46		13.0
#3	3rd User story for Req 1		Linear		
#12	2nd User Story for Req 1		Exciter		
#10	Req 5		n.a.		n.a.
#5	Req 3		n.a.		n.a.
#6	Req3 user story		Mandatory		



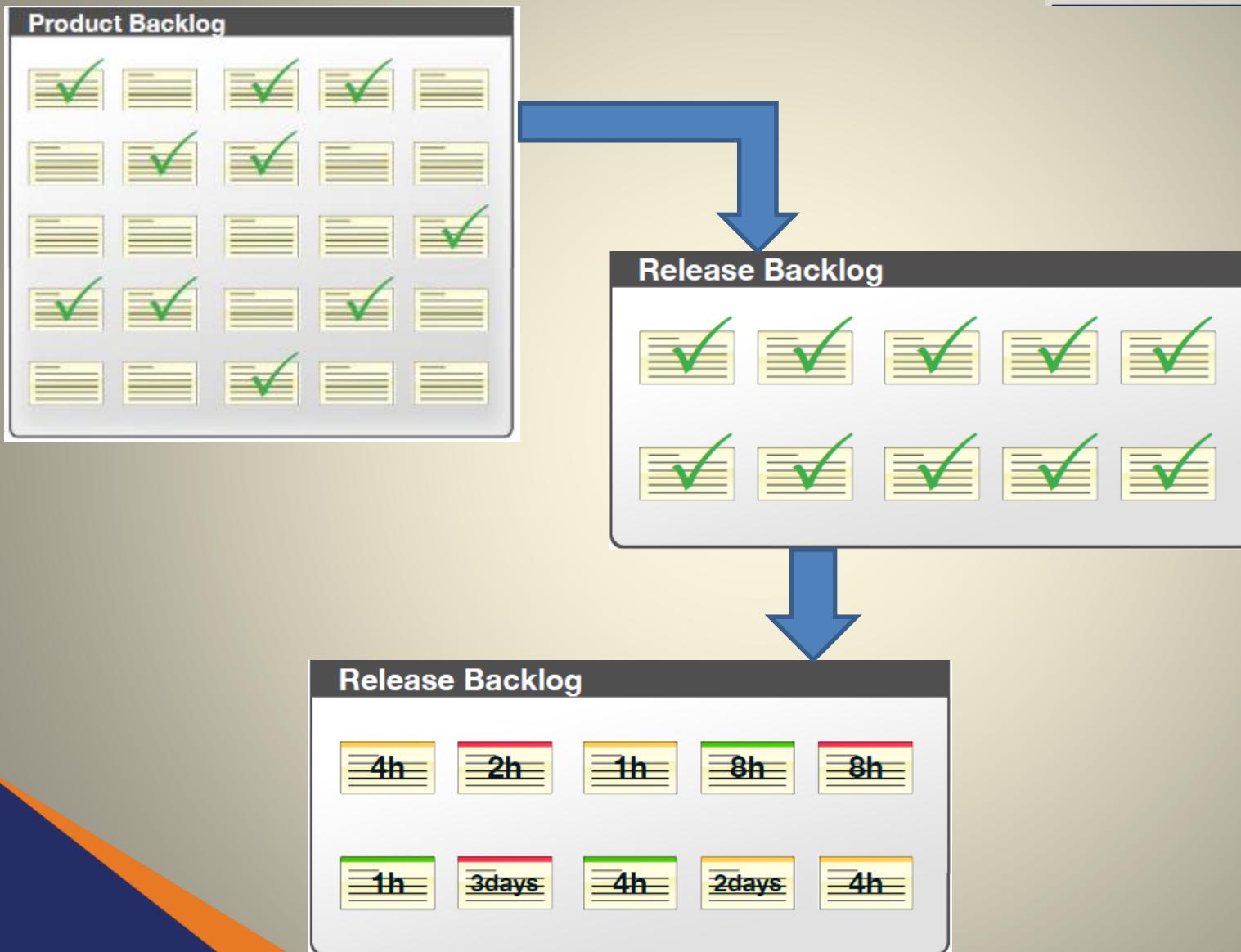
Activity

1. Who is responsible for prioritizing the product backlog?
 - a) Product Owner
 - b) SCRUM Master
 - c) Lead Developer
 - d) End Customer

Sprint and Sprint Backlog

- An iteration in a Scrum project is known as a Sprint.
- Before starting a Sprint the Team should come up with a Sprint backlog.
- The sprint backlog (release backlog) is a list of user stories identified by the Scrum team to be completed during the sprint.
- This is a subset of Product backlog user stories defined only for a particular sprint.

Sprint Backlog



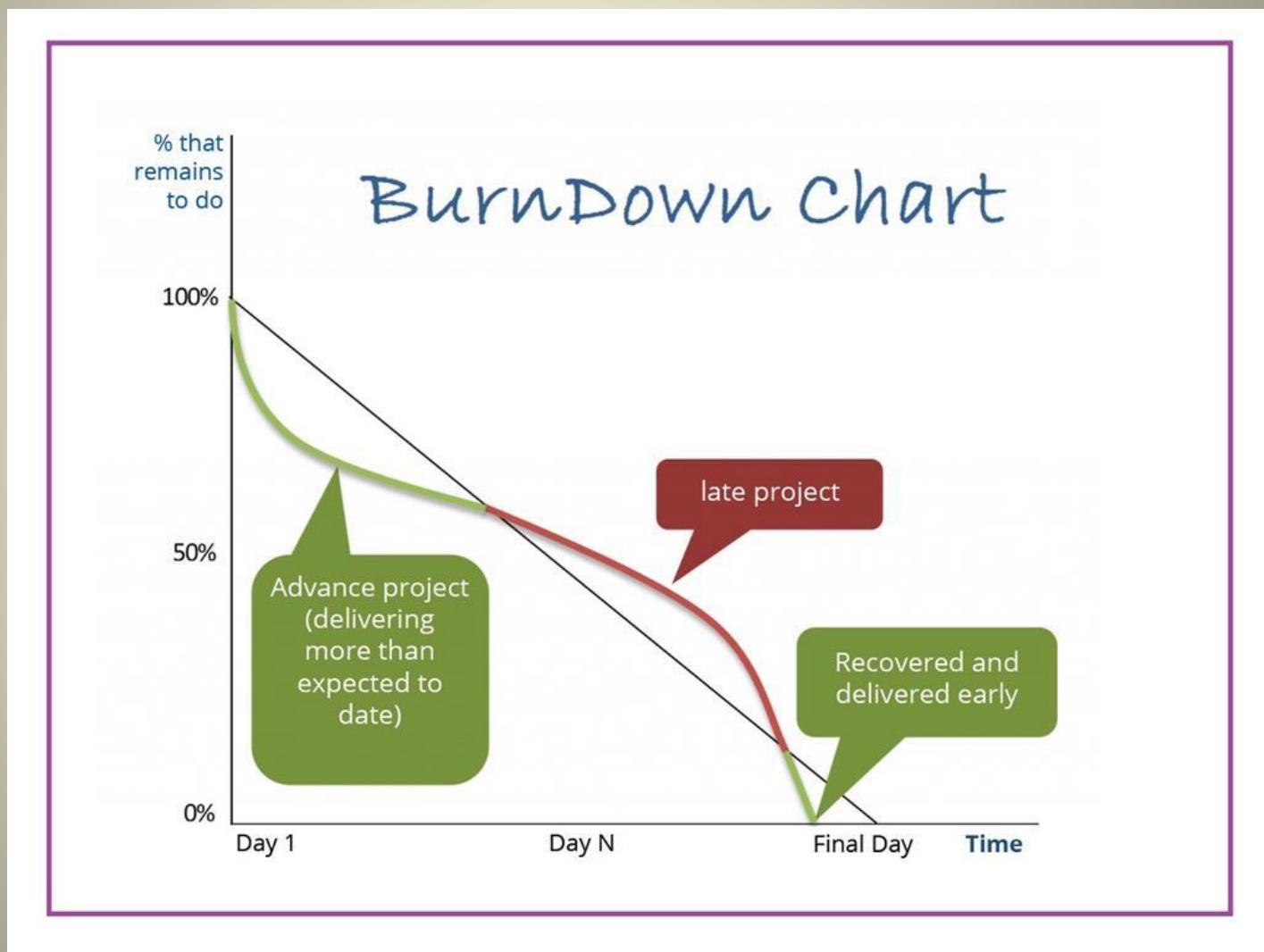
Activity

1. Who is responsible for all estimates of the Product Backlog items?
 - a) The Product Owner
 - b) The Development Team
 - c) The Scrum Master
 - d) The Scrum Team

Sprint Burn Down Chart

- A burn down chart is a graphical representation of work left to do vs time.
- The outstanding work (or backlog) is often on the **vertical axis**, with time along the **horizontal**.
- That is, it is a run chart of outstanding work. It is useful for predicting when all of the work will be completed.

Sprint Burn Down Chart

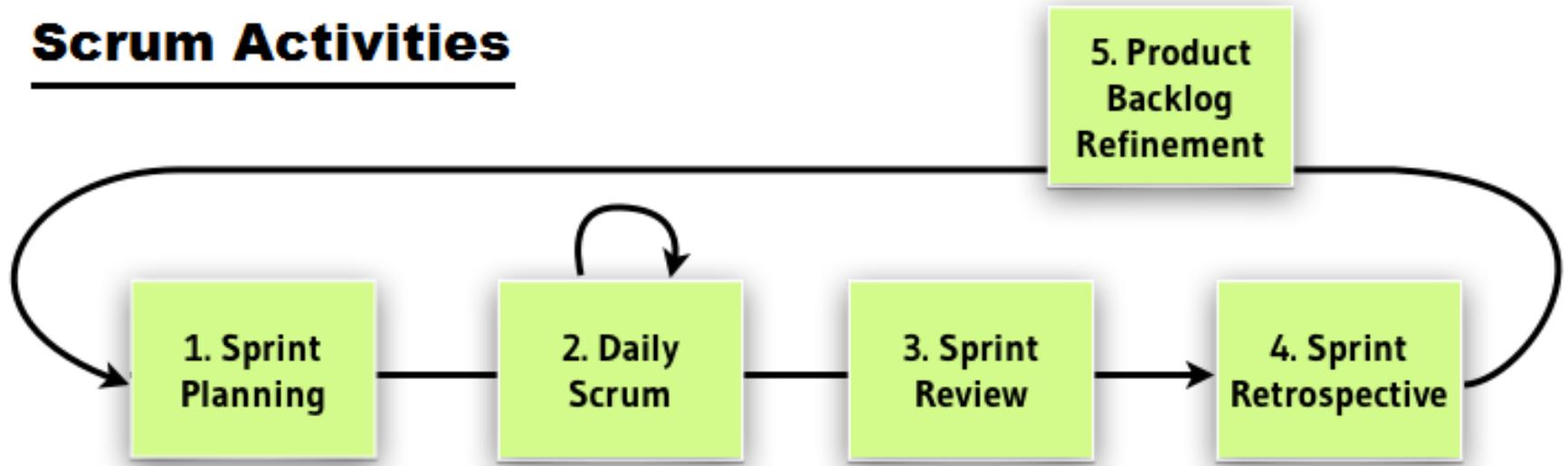


Activity

1. What is the Scrum Master responsible for?
 - a) Ensuring the Scrum Team adheres to Scrum theory, practices, and rules
 - b) Ensuring the Product Backlog is visible, transparent, and clear to all
 - c) Ensuring the Development Team understands items in the Product Backlog to the level needed
 - d) Ensuring Scrum is understood and enacted

Scrum Activities

Scrum Activities



Sprint Planning

- Each Sprint may be considered a project with no more than a one-month horizon.
- Like projects, Sprints are used to accomplish something. Each Sprint has a definition of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product.

Sprint Planning Meeting

- The work to be performed in the Sprint is planned at the Sprint Planning, it's a collaborative work of the entire Scrum Team.
- Time-boxed to a maximum of eight hours for a one-month Sprint.
- Sprint Planning answers the following:
 - What can be delivered in the Increment resulting from the upcoming Sprint?
 - How will the work needed to deliver the Increment be achieved?

Daily Scrum



Daily SCRUM

- Short (15 min) frequent meetings, facilitated by the Scrum Master.
- One activity – Scrum Master asks each attendee 3 questions.
 1. What have you completed (relative to the Backlog) since the last Scrum meeting?
 2. What got in your way of completing this work?
 3. What will you do between now and the next Scrum meeting?

Sprint Review



Sprint Review

- Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed.
- ANYTHING can be changed, work can be added, eliminated, reprioritized.
- 4 hour time boxed meeting for a 1 month sprint.

Sprint Retrospective

Sprint Retrospective Meeting



The illustration depicts a group of six cartoon characters representing different professional roles: a businessman, a chef, a painter, a pilot, a teacher, and a musician. They are standing in a circle, facing each other, with large green and orange speech bubbles floating around them, symbolizing the exchange of ideas and feedback during a retrospective session.

WHAT WENT WELL	WHAT COULD BE IMPROVED
Three green speech bubbles	Four orange/red speech bubbles

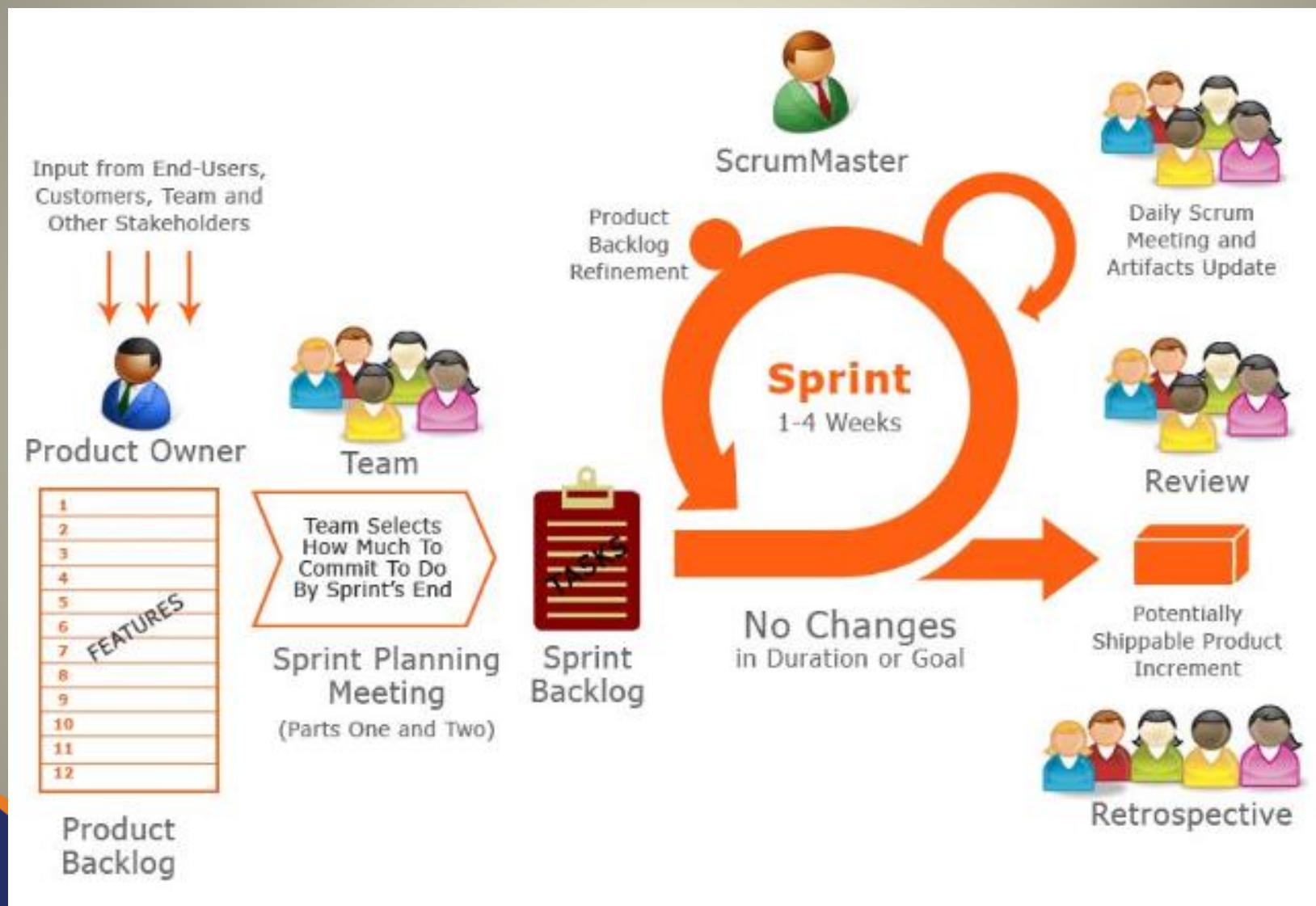
Sprint Retrospective

- The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning.
 - Three-hour time-boxed meeting for one-month Sprint.



Activity

1. When is the work planned for the first days of a Sprint decomposed?
 - a) At the end of Sprint Review
 - b) At the end of Sprint Planning
 - c) At the first Daily Scrum
 - d) At the beginning of Sprint Retrospective



SCRUM Tools

- There are many tools available to manage the SCRUM process development.
 - Targetprocess
 - Trello

Activity

- Form your project group.
- Choose a Scrum Master for your group.
- Choose 1-designer, 2-developers and 1 QA Engineer.
- Consider the lecturer as the Product Owner
- Prepare user stories for your case study (at least 10)
- Prioritize and arrange them in product backlog.

Activity Contd...

- Select user stories from the product backlog into three releases.
- Select a release and prepare the sprint backlog.

References

- "What is Agile Software Development?", *Agile Alliance*, 2017. [Online]. Available: <https://www.agilealliance.org/agile101/>
- "What is Agile Manifesto? - Definition from WhatIs.com", *SearchCIO*, 2017. [Online]. Available: <http://searchcio.techtarget.com/definition/Agile-Manifesto>
- "Manifesto for Agile Software Development", *Agilemanifesto.org*, 2017. [Online]. Available: <http://agilemanifesto.org/iso/en/manifesto.html>
- "What is Scrum?", *Scrum.org*, 2017. [Online]. Available: <https://www.scrum.org/resources/what-is-scrum/>
- Lean Software Development: An Agile Toolkit, Mary and Tom Poppendieck, 2003, Addison Wesley

Final Exam

Software Process Modelling

Assessment Criteria

Mid Term Examination LO1-LO4	30%
Assignment I LO3-LO5	10%
Assignment II LO4-LO5	10%
Final Examination LO1-LO9	50%

Final Exam

- 2 Hour Paper
- 10 minutes reading time
 - No writing during the 10 minutes
- Writing booklets and rough paper will be given
- If anything important is written, attach the rough papers
- Write all the assumptions you make

Final Exam Structure

- Question 1 – Use Case Diagram
 - Case study based
 - Marking guide – example
 - UseCases - $0.5 * 20 = 10$ marks
 - Actors - $0.5 * 4 = 2$ Marks
 - Actor Generalisation - $1 * 3 = 3$ Marks
 - Include
 - 7 to 9 = 5 Marks
 - <6 - 0.5 each
 - Extend
 - $1 * 5 = 5$ Marks
 - Generalisation
 - $0.5 * 8 = 4$ Marks
 - Overall - = 1 Mark

Final Exam Structure

- Question 2 – Activity Diagram/ Use Case Scenario
 - Case study based
 - Marking Scheme – example
 - Swim Lanes = 3 Marks
 - Actions - $0.5 * 16 = 8$ Marks
 - Decision - $1 * 4 = 4$ Marks
 - Fork - $1 * 2 = 2$ Marks
 - Start = 1 Mark
 - End - $0.5 * 2 = 1$ Mark
 - Overall = 1 Mark

Final Exam Structure

- Question 3 – covering all the sections
 - Fill in the blanks + structured + essay
 - SDLC Models
 - Phases of waterfall model
 - Agile and Scrum
 - May be related to tutorials and labs

