

# Software and Software Engineering

Lecture # 1, 2, 3  
21,22, 23 Jan

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)

# Introduction to Software Engineering SE-110



# Today's Outline

- Administrative Stuff
- Overview of 110
- Introduction to Software Engineering
- Why study Software Engineering?

# About me

- Graduated from FJWU
- Completed Masters from NUST
- Research interests:
  - Software engineering
  - Database systems
  - Algorithm analysis

# Office hours and meeting rules

- Office Room 6 (CS building)
- Office hours: 3:00 to 4:00 pm
  - (Monday, Wednesday, Thursday)

# About course

- Study and application of the principles and techniques of computer science, engineering, and mathematical analysis to the design, development, testing, and evaluation of the software and the systems that enable computers to perform their many applications.
- This is perhaps the most important course in your CS curriculum in CAREER PATH FORECAST.

# Career Forecast

- Software engineers are in demand in not only at software development companies but also in all other organizations that are involved in the development of significant information systems – including
  - governments,
  - telecommunications companies,
  - the chemical industry,
  - the bio-medical industry,
  - financial institutions,
  - pharmaceuticals,
  - healthcare sector corporations,
  - engineering and manufacturing firms,
  - e-commerce,
- In fact in the words of noted software engineer, David Parnas, "career opportunities for software engineers are essentially unlimited."

# Knowledge assumed

- Programming language concepts

# Skills assumed

- We assume you have the skills to code in programming language therefore you
  - can design, implement, test, debug, read, and understand the programs.

# Tentative Mark Distribution and Grading Policy

- ❖ Assignments: 10%
- ❖ Quizzes: 10 %
- ❖ Mid Exam 1: 15 %
- ❖ Mid Exam 2: 15 %
- ❖ Class Participation: 3 – 5 %
- ❖ Final: 40%
- ❖ Presentations / Projects: 10%
- ❖ *Absolute Grading Policy*

# Project

- An advance project
  - Provides interesting problem, realistic pressures, unclear/changing
- Small 3-4 member teams
- Emphasis on good SE practice/methodology
  - Homework's and deliverables tied to the project
  - Grading: practices more important than the end product

# Course Ethics

## Projects/Assignments

- Deadlines are always final
- No credit for late submissions
- One student per assignment at maximum
- Project Proposal Submission : 3rd Week
- Project Presentation : Last week

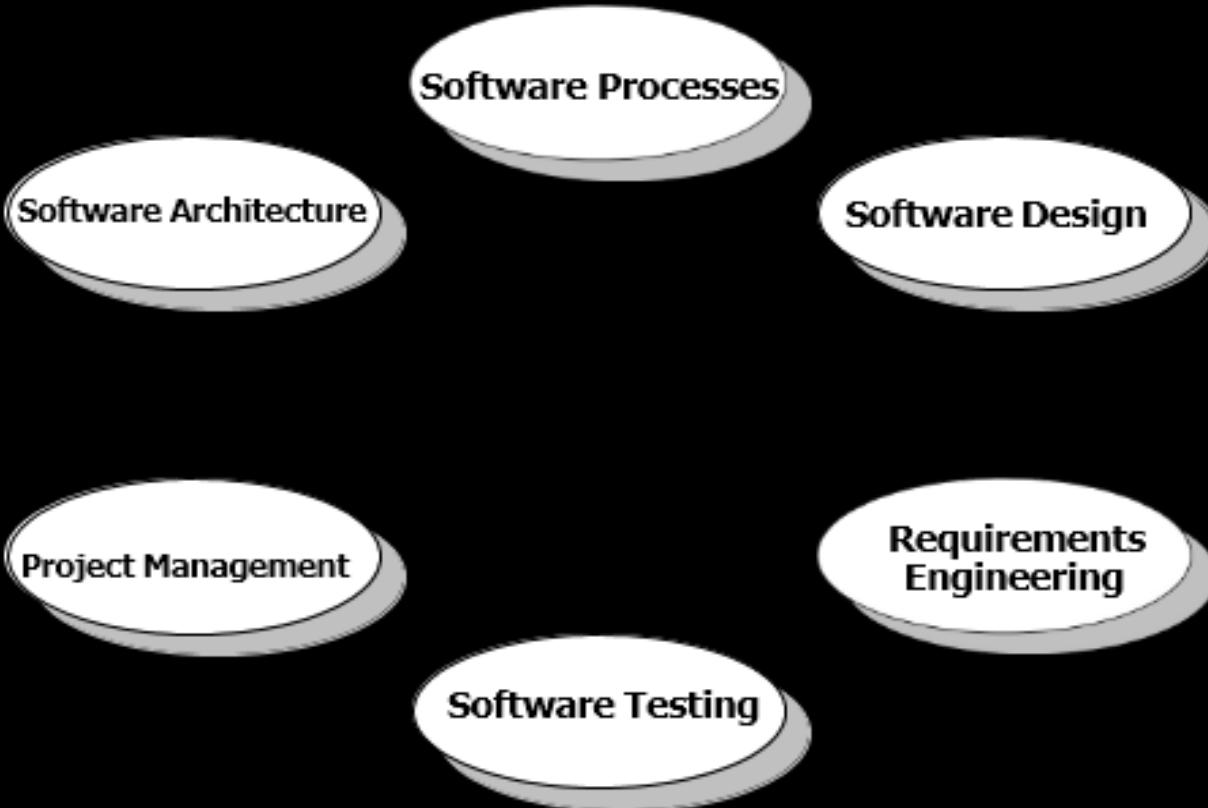
## Quizzes

- Announced
- Unannounced

## Honesty

- All parties involved in any kind of cheating in any exam will get zero in that exam.

# Major Topics of the Course



# Course Outline

- Overview of Software Engineering SE
- Process framework and models
- Requirement Engineering
- Requirement Elicitation
- Requirement Modeling
- Software Architecture
- Risk Management
- Software Quality
- Testing
- Project Management
- Work Breakdown structure

# Course Material

- You will have **Presentations** of each topic and **reference books** in PDF format will be available at slate.
- Text Books
  - Ian Sommerville, Software Engineering 10<sup>th</sup> Edition
  - Pressman, R S Software Engineering: A Practitioners Approach (7th Edition, European Adaptation), McGraw Hill, 1994
- Reference Books
  - (Undergraduate Topics in Computer Science) Pankaj Jalote - A Concise Introduction to Software Engineering-Springer (2008)

# Course Goals

- Have a sound understanding of the fundamental concepts of the software engineering paradigm
- Understand and apply different practices used in software industry for software development
- Recognize the risks of software failure
- Verify through review practice accuracy, ambiguity and completeness of a software
- Develop a software engineering mindset

# Why Study Software Engineering ?



# What is Software?

Software is:

- (1) instructions (computer programs) that when executed provide desired features, function, and performance;
- (2) data structures that enable the programs to adequately manipulate information and
- (3) documentation that describes the operation and use of the programs.

# What is Software?

- *Software is developed or engineered, it is not manufactured in the classical sense.*
- *Software doesn't "wear out."*
- *Although the industry is moving toward component-based construction, most software continues to be custom-built.*

# Wear vs. Deterioration

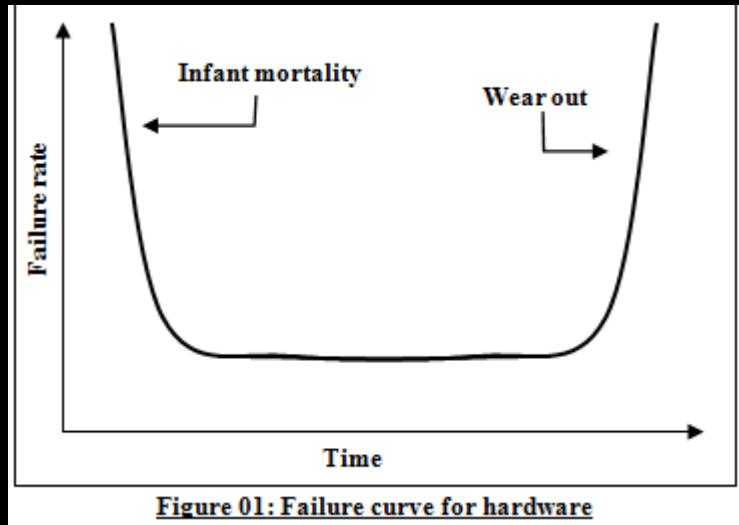


Figure 01: Failure curve for hardware

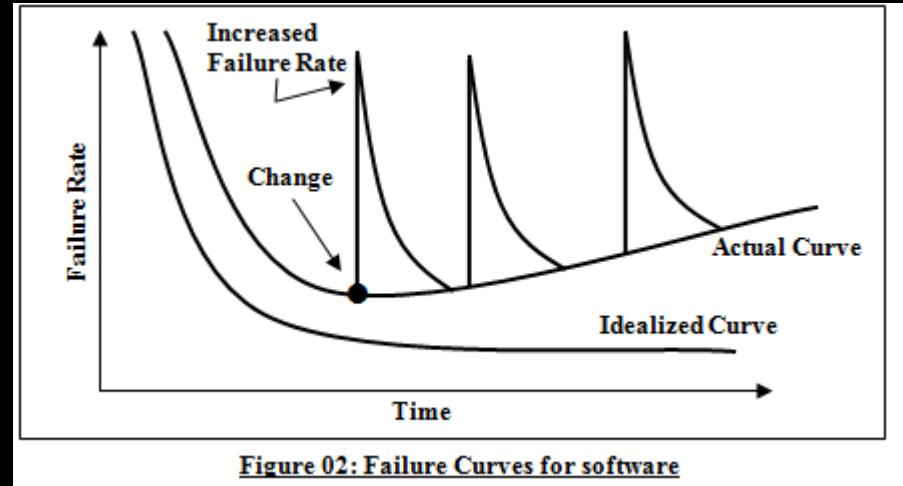


Figure 02: Failure Curves for software

# Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

# Software Application Domains

## 1. System software:

collection of programs written to service other programs. such as compilers, editors, file management utilities

## 2. Application software:

stand-alone programs for specific needs.

## 3. Engineering/scientific software:

Characterized by “number crunching” algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc

# Software Applications

## 4. Embedded software:

resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)

## 5. Product-line software:

focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)

## 6. WebApps (Web applications)

network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.

## 7. AI software

uses non-numerical algorithm to solve complex problem. E.g Robotics, expert system, pattern recognition game playing

# Software—New Categories

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Netsourcing—the Web as a computing engine
- Open source—"free" source code open to the computing community (a blessing, but also a potential curse!)
- Also ...
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

# Why study Software Engineering ?

- We will try to understand . .
  - Typical ***Engineering approach***
- How software development ***is different*** . .
  - from ***other engineering*** disciplines ?

# Engineering Approach

Before discussing this we need to understand difference between **science** and **Engineering**



# Difference b/w Science & Engg

- **What is Science ?**

- Scientist invented a **wheel**,
- **Engineer** will use that wheel for bi-cycle, motor-car, for anything.
  
- Scientist invented an **electric motor**,
- **Engineer** will use that motor to construct things like pumps, electric-lifts, toys, etc.

- **What is Engineering ?**

- Process of **productive use** of scientific knowledge is called engineering.

# Difference b/w CS & SW-Engg

- **Computer science** is knowledge contains theories and fundamentals.
- Somebody has invented followings
  - Database
  - Design methodology
  - Algorithms
  - Testing methods
  - Methods of analysis and Verification.
- *Software Engineering* is a approach of applying engineering principles in order to develop software.
- SW-Engg applies *various phases* to implement all of the components needed to satisfy the user requirements.
- SW-Engg follows engineering which is to *plan*, with *suitable skills* and craft for moving into Mass-production.

# Engineering Approach

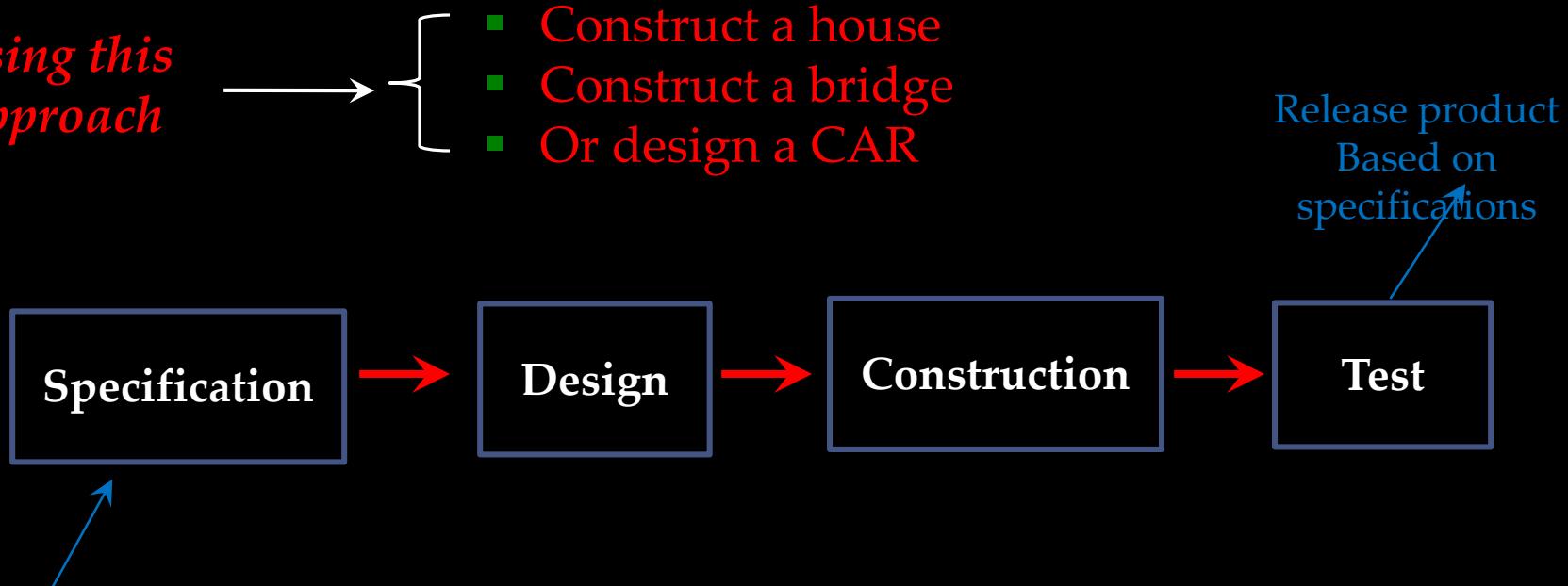
- Day to day life we see successful projects in other fields of engineering
  - Such as in civil, or mechanical engineering.
  - Building bridges & roads are also large projects and they require large efforts in construction.
- It is our common observation that usually large software projects are NOT very successful.
  - In the middle of software projects we realize that **more time** is required or
  - **more resources** are required to complete the project, than we initially anticipated.

# Engineering Approach

- Following are **Large** engineering projects **in other discipline** and are normally successful
  - Building bridges & roads
  - Power Plants
  - 60 story luxury apartments
  - Aircraft missiles
- Why such kinds of project are generally successful?
- What are the **practices being followed** in these kinds of project that make them successful?
- Is there any **common attributes** or these are different than developing a software?

# Engineering Approach

*Using this approach*



What needs  
to develop

This *engineering approach* helps engineers developing what they intend to and complete projects successfully

# Example – Construct a house

Following are logical steps to develop a house

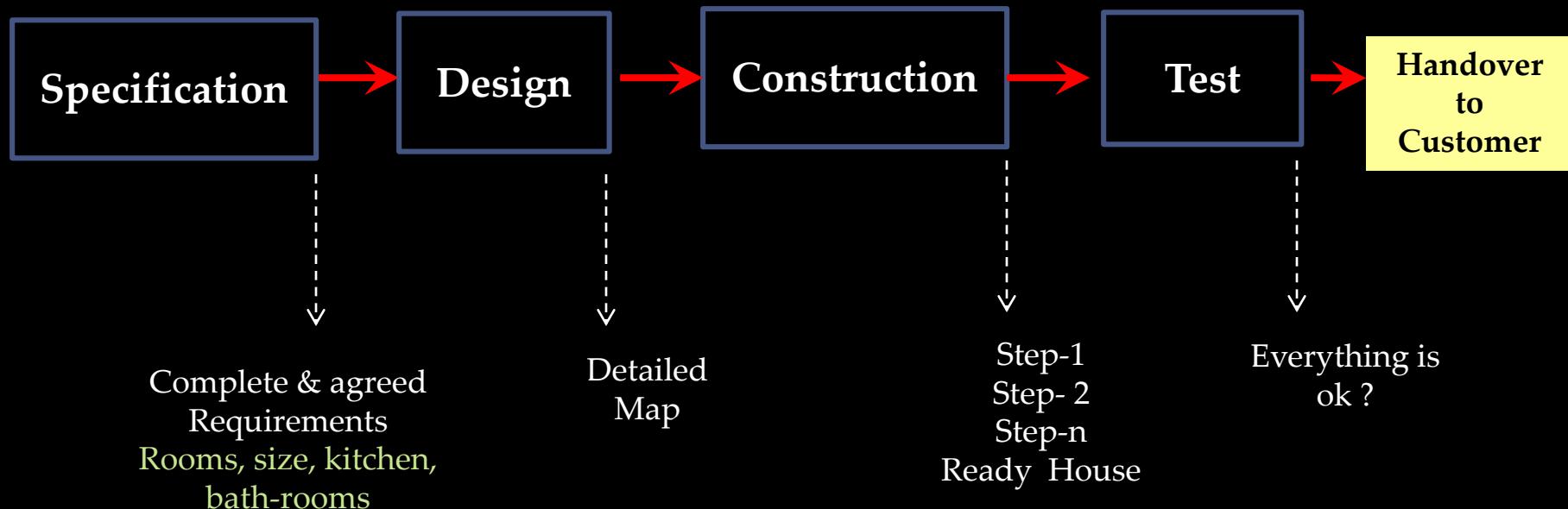
- Stable and *detail requirements* from customer.
- **Record** requirement so that we **do not miss** any requirement.
- After recording it must be *validated* by user again.
- Any **Change** means additional cost and time.
- Once the *requirements are agreed* (baseline) next step we develop **a map** (design)
  - Size of rooms
  - Doors and windows placement
  - Size of kitchen, bathrooms etc
  - Design should have electrical cabling layout
  - plumbing layouts etc

# Example – Construct a house

- Once the map is *approved* by the customer you **start** construction of house.
- **What if** you developed the structure of the house and customer request you to **change** dimensions of rooms?
  - Can you accommodate that change at this stage?
- How do you manage agreed (scope) map?

# Example – Construct a house

How Engineering approach helped **construction** of **House** ?



# Software Engg is different



# Software Engineering is different

- Requirements are complex and **multi-dimensional**.
- Difficult to asses *progress* of the project.
- Delivering an **intangible** product.
- Software **get obsolete** quickly.
- Can be developed using **different approach**
  - Approach = process, templates, techniques etc
  - Plan-driven (Predictive) & change-driven (adaptive)

Software Engineering is different

# Requirements are complex

Components of any automation Solution

- Features and functions
- Process/Practices (need to be automated)
- People (their experience, knowledge, and ability to adopt change)
- Usability (Desktop, Internet, Smart-phone)
- Environment (Temp, Dust-free etc)

Requirements are complex

## Guess my requirements

I need a  
equipment

- ❑ That is equipped with internal **combustion engine** (such as Car, truck)
- ❑ That is available in yellow & black colour
- ❑ It has three wheels
- ❑ Has 3 speed forward and 3 speed reverse
- ❑ Levels uneven surface nicely and quickly



# Difficult to asses progress

- You can asses progress in civil engineering
  - **50%** work is completed (in 60 story building)
  - You may verify that by visiting the project site and see 30 stories are built or not
- When Project Manager of software project says **90%** work complete.
  - How can you verify that statement ?
  - Is it really 90% or much less . . .

Software Engineering is different

## Delivering an intangible thing

- In civil engineering you are delivering house or a road or a bridge.
  - Can understand the time taken and cost
  - Buy a Car you look for rooms, their size, design etc
- In software project what do you deliver ?
- A CD only cannot asses **how much efforts** it had taken and **what is its cost**.

Can differentiate - 2 kilo potato or 20 Kilos  
Cannot differentiate - Small SW or large SW

# Software get obsolete quickly

- Software is developed to **automate business process**, as business grows its process changes.
- Need some changes approximately after very 8-9 months in developed software.
- This is **not that case** when you purchased a **brand-new car** or new house.

# Can be developed using different approach

- Software house use different approaches to develop same functionality
  - Plan-Driven (Sequential or Predictive )
  - Change-Driven (Agile or Adaptive)
- Building a 60 story cannot be build with **steel**, **cement** and **concrete**. Approach is not very different in other engineering discipline.

# Software Engineering is different

- It is difficult to **estimates efforts** and plan these activities for software development.
- There are **so many failures** in software development projects.
- It is **NOT difficult** to estimate efforts in construction of 50-story building.

# Software Engineering is different

- We must have, separate **specialized team** for . . .
  - Requirements elicitation
  - Dedicated team for software design
  - Team of developers
  - separate team for testing etc.
- All these team needs **supervisions** and **monitoring** for the work they do.
- Typical **engineering approach** is necessary for the successful software project.

# Software Engineering –

## Definition

Software Engineering as defined by IEEE

Application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”

# Why study Software Engineering ?



# Why study Software Engineering ?

- What are **challenges** in software development ?
  - that will be addressed by using Engineering approach
- What is the need/required ?
  - Unable to handle large software projects - Why
- Why are we studying software Engineering?



# Why study Software Engineering ?

- Historically 2-3 people used to develop **small** and **simple** software that is ok. No problems in small & medium size projects
  - Example: your FYP is small and simple project
- When a **large** software with **complex** requirements are developed through a large team (70-80 people) we had serious problems (called *Software crisis*).
  - Example: Developing a core banking software is large project, will require large team and few years.

# Software Crisis

- In early **60s** software techniques that were used to develop **small** software were **not applicable** for **large** software applications.
  - In most of the cases that software which was tried to be build using those old tools and techniques were either **not complete**.
  - Most of the times it was delivered **too late**.
  - Most of the projects were **over-budgeted**.
  - A few projects caused loss of life
  - Some projects caused property damage.

# Cost of Failure

- **Allstate Insurance** – ([www.allstate.com](http://www.allstate.com)) in 1982
  - \$8M were allocated to automate business
  - EDS providing software
  - Initial 5-year project continued for 10-years, until 1993
  - Cost approached \$100M
- **Bank of America** – ([www.bankofamerica.com](http://www.bankofamerica.com))
  - Spent \$23M on an initial **5 year** accounting & reporting system
  - Spent \$600M trying to make it work
  - Project was cancelled
  - Lost customer accounts - \$Billions



# Cost of Failure

- ***Blue Cross and Blue Shield of Wisconsin – 1983***
  - EDS was hired to build **\$200M** computer system
  - Delivered on time in 18 months
  - System didn't work – issued **\$60M** in overpayments and duplicate checks
  - Blue Cross lost 35,000 policy holders by 1987

# NATO Conference

- In the fall of 1968 and again the in fall of 1969, NATO hosted a conference devoted to the subject of software engineering.
- The motivation for these conferences was that the computer industry at large was having a great deal of trouble in producing large and complex software systems i.e. the software crisis.
- Participants were solicited from computer manufacturers, computer users, software houses, and academia. Over the years, the conference reports have gained a certain amount of classical aura.

# NATO Conference

- In late 1968 the Conference recommended the term Software Engineering. The phrase ‘software engineering’ was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.

# Software Engineering

- Writing software has evolved into a profession concerned with
  - How best to maximize the quality of software and of how to create it.
  - How best to create high quality software is a separate and controversial problem covering software design principles
  - How best to deliver software on time and as quickly as possible, work-place "culture", hiring practices, and so forth.

**All this falls under the broad rubric of software engineering**



That is all

**SDLC**

**Lecture # 4, 5, 6  
28,29, 30 Jan**

# **Introduction to Software Engineering**

**SE-110**

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)



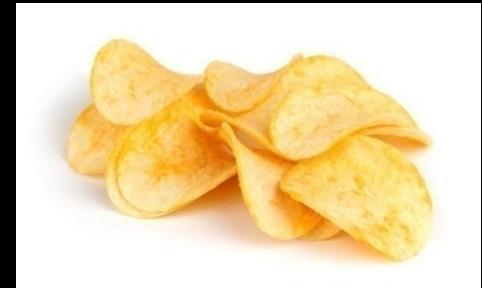
# Today's Outline

- Process and Product
- Software Engineering Framework
- What is system?
- SDLC
- SDLC Phases

# Process

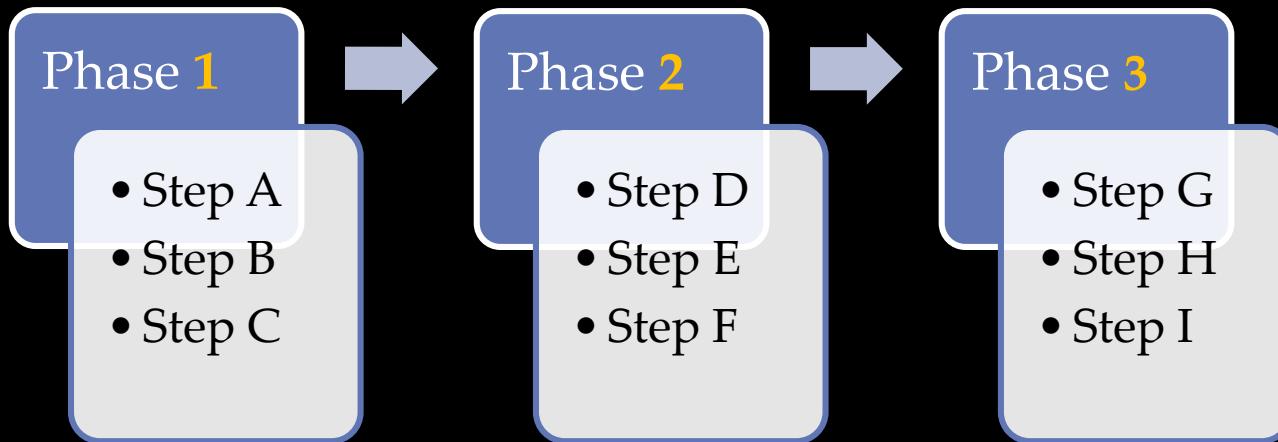
**Process:** A particular method, generally involving a number of steps.

Process for making *potato chips*



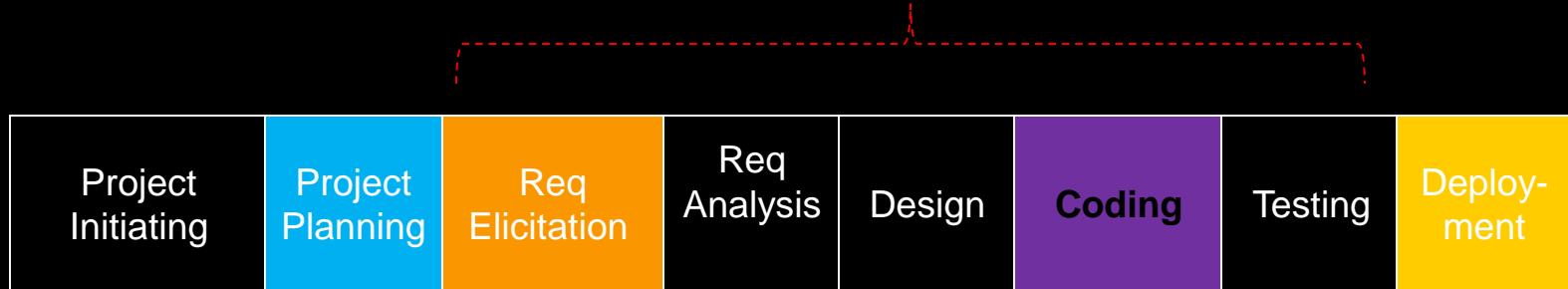
# Process

- Process is generally a set of **phases**.
- Each phase performs a well **defined task** and generally produces an output.



# Software Process

- **Software Process**: A set of steps, along with ordering constraints on execution, to produce software with desired outcome.
  - Many types of **activities** performed by **different teams**
  - Software process is comprising of many component processes



# Process

What we should do, to develop a *good quality* software ?

What are the attributes of a *good process* ?  
Visible, Repeatable, Measureable

Can you give examples of *Technical and Managerial* problems in software development process ?

Project fails due to **Managerial** problems

# Product

- **Products** are outcomes of executing a **process** for a project.
- Does Process quality and Product quality has any relation ?
- Software development life cycle (SDLC), is a structure imposed on the development of a software product.
- Software Engineering says if you follow the process the output is predictable and repeatable no matter who does that.

# Software Engineering Framework

# Software Engineering Framework

- What is **framework** and why we need framework?
  - Framework means; set of rules to be followed.
- What are **those rules**? Those rules have been **adopted** by organizations that produce good results.
- Experts convert those rules into a framework to be used by every organization with respect to their needs.
- **Example:** Framework for Requirements Development

# Software Engineering Framework

What

Definition

How

Development

Change

Maintenance

Umbrella / parallel activities



- Quality Assurance
- Configuration Mg'mt
- Project Monitoring
- Measurement

# Definition Phase

- *Definition* phase focuses on *what* (is required).
- During definition, SW-development-Team and user attempts to identify the following questions:
  - What is **need** (or problem)?
  - What *features* are required?
  - What interfaces are to be established?
  - Any budget or technical **constraints** ?
  - What is *success criteria* ?.

# Development Phase

- *Development* phase focuses on the **how**.
- During development, the SW-development-Team attempts to define how:
  - How database would be designed
  - How software *architectures* would be designed
  - How the design will translate into programming language
  - How *testing* will be performed

# Development Phase

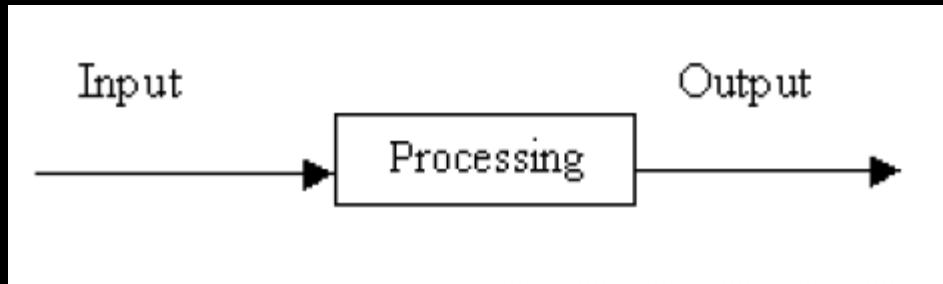
- Methods applied during development phase, *will vary* (depending on the SDLC) but the three steps will occur in some form:
- *Design*: Design translate the requirements into some graphical or tabular representations.
- *Coding*: Design is then translated into programming language.
- *Testing*: The executable code must be tested to uncover errors.

# Maintenance Phase

- Maintenance phase focuses on changes that associated with
  - Error Correction (Corrective)
  - Platform Adaptations required (Adaptive)
  - Enhancement due to change (Perfective)
  - The work carried out order to avoid any breakdown or malfunction (Preventive)

# System

- A collection of components that work together to realize some objective forms a system.
- Basically there are three major components in every system namely input, processing and output.



# System

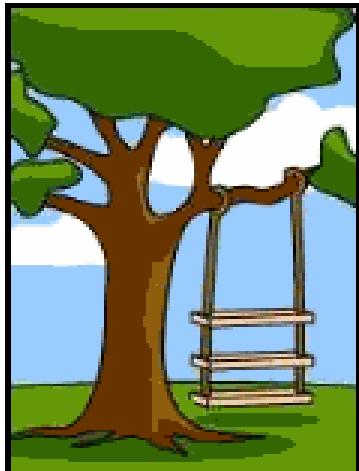
A System can be a Application program or it can be an Information System.

- Computer App: is an application program (app for short) is a computer program designed to perform a **group of coordinated functions**, tasks, or activities for the benefit of the user.
- Information System: is software that helps you organize and analyze data. This makes it possible to answer questions and solve problems relevant to the mission of an organization.

# Software Crisis

Defining the  
*problem* is the  
PROBLEM

# Why Object-Oriented?



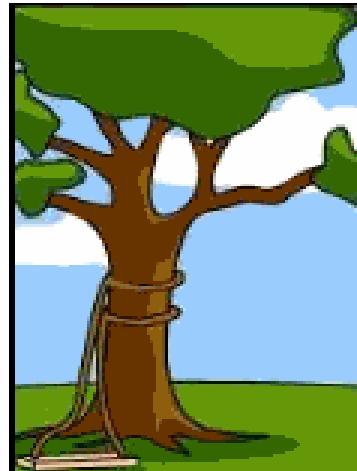
How the customer explained it



How the Project Leader understood it



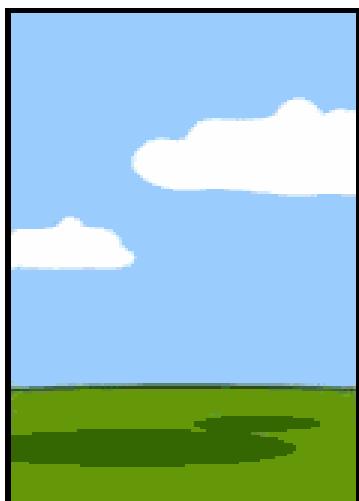
How the Analyst designed it



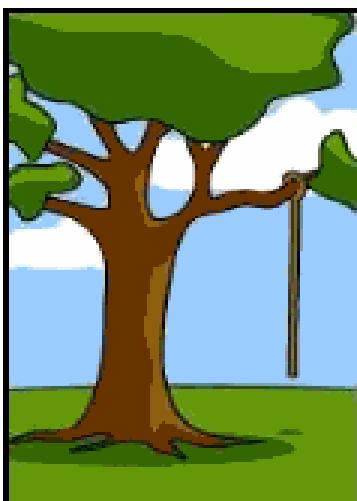
How the Programmer wrote it



How the Business Consultant described it



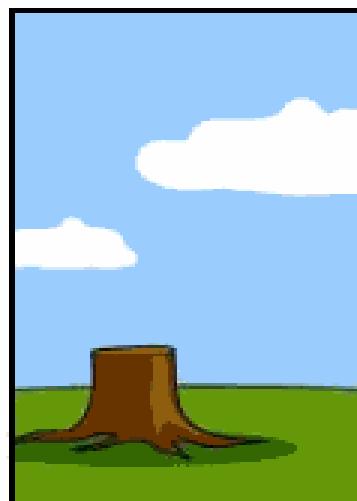
How the project was documented



What operations installed



How the customer was billed



How it was supported

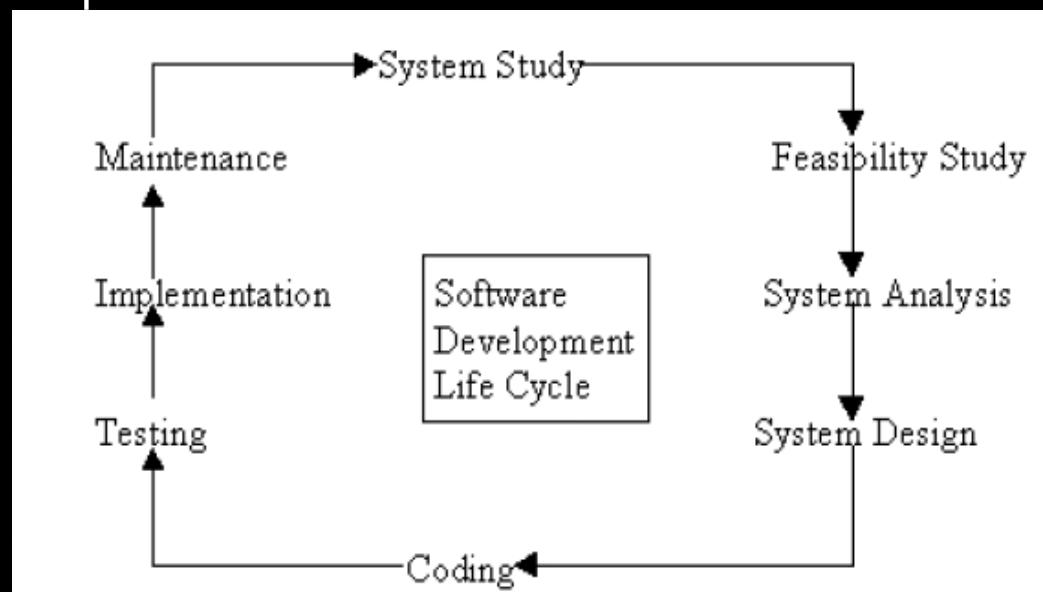


What the customer really needed

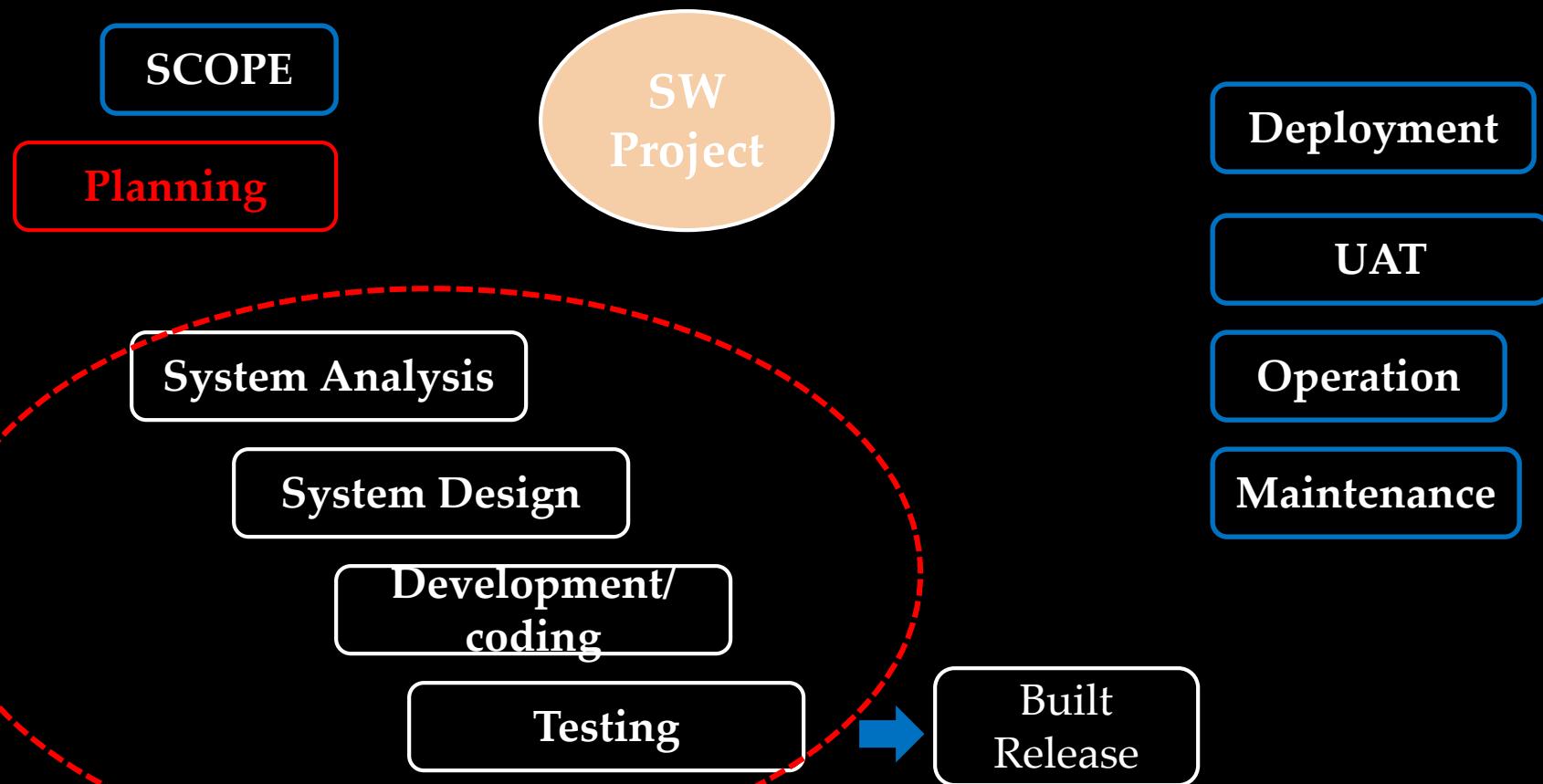


# SDLC

- Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software.
- It consists of a detailed plan describing how to develop and maintain software.
- SDLC consists of many activities/ phases.
- Following are the major phases of SDLC.
  - System study
  - Feasibility study
  - System analysis
  - System design
  - Coding
  - Testing
  - Built release
  - Maintenance

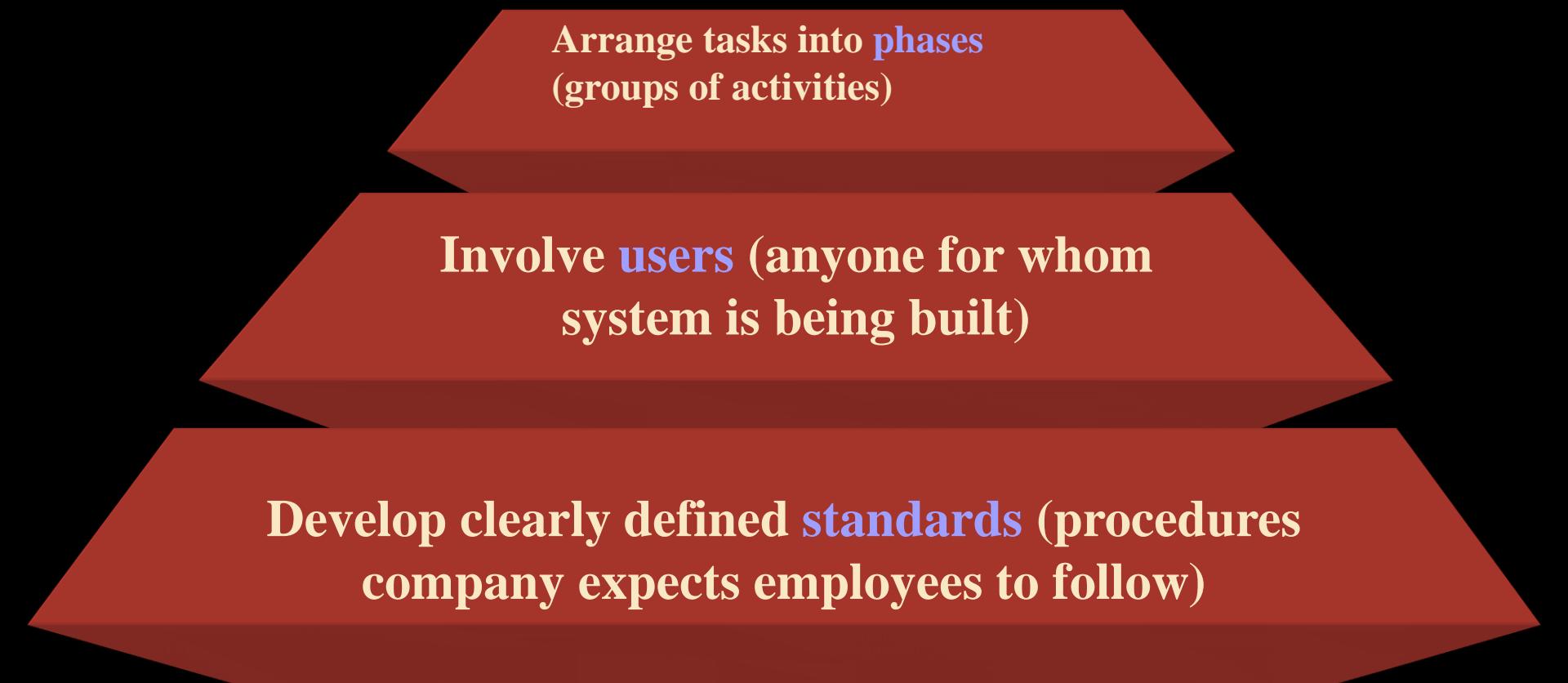


# Software Development Life Cycle



# The System Development Life Cycle

What are guidelines for system development?



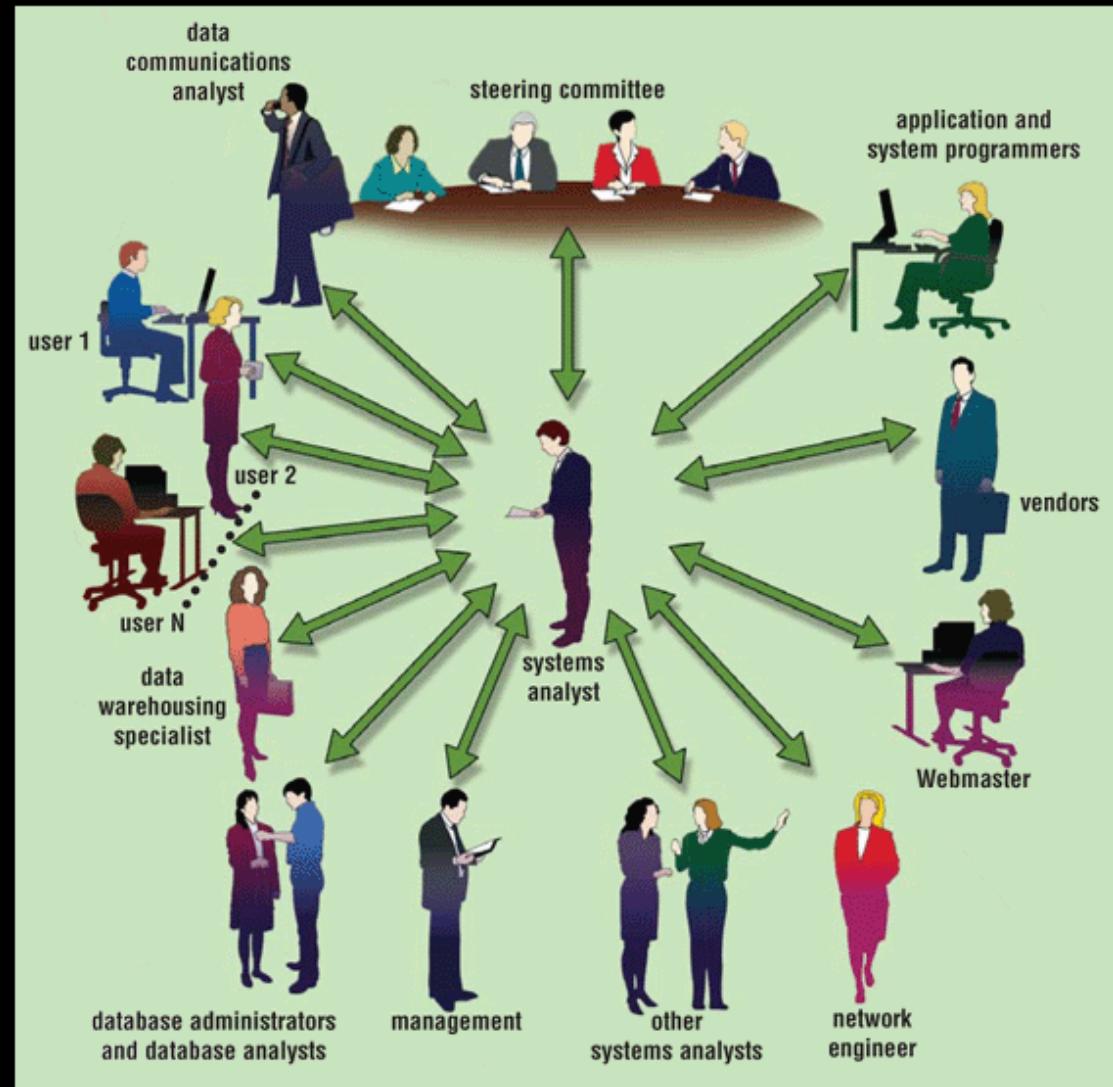
Arrange tasks into **phases**  
(groups of activities)

Involve **users** (anyone for whom  
system is being built)

Develop clearly defined **standards** (procedures  
company expects employees to follow)

# The System Development Life Cycle

Who participates in the system development life cycle?



# The System Development Life Cycle

What is the **project team**?

**Formed to work on project from beginning to end**

**Consists of users, systems analyst, and other IT professionals**

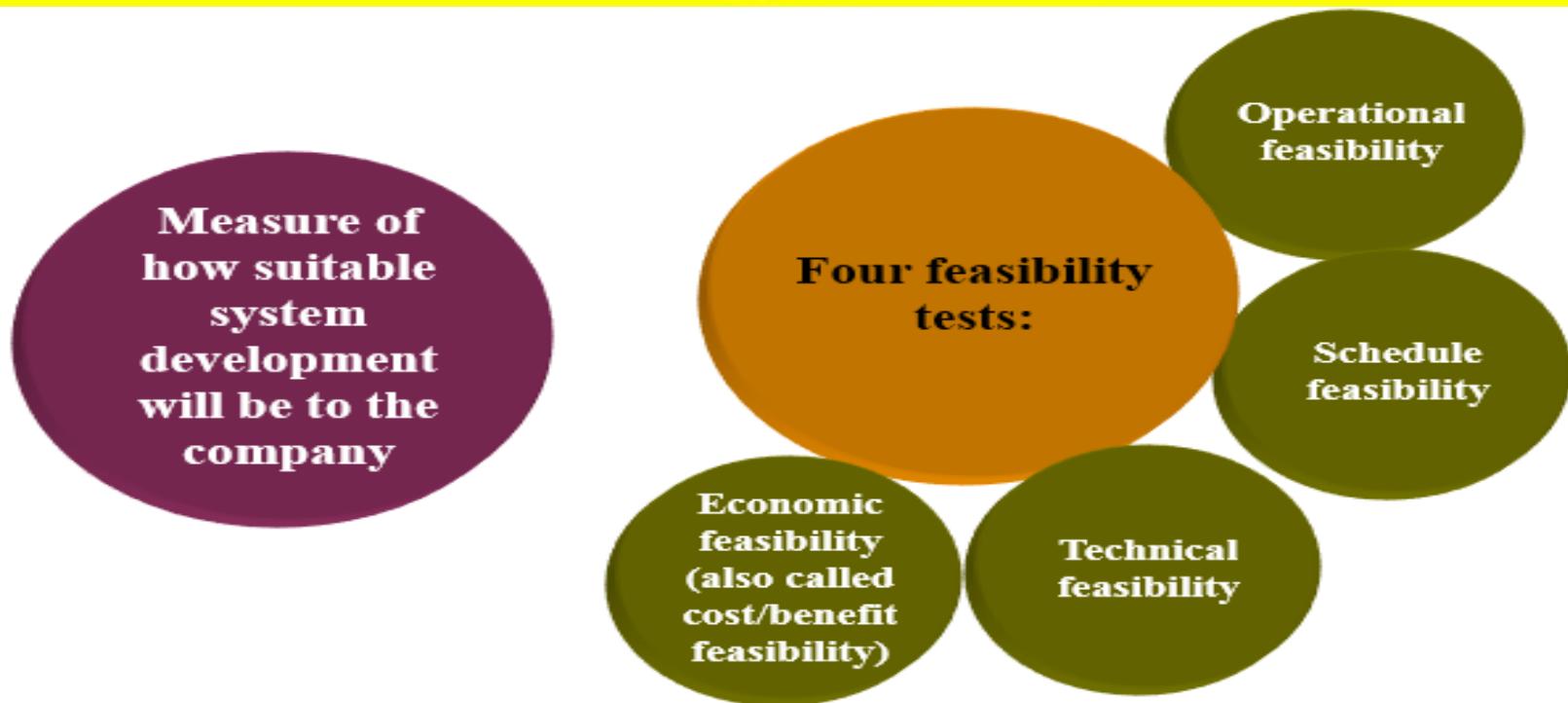
Project leader—one member of the team who manages and controls project budget and schedule

# Stage 1: System Study

- Gives clear picture of what actually the physical system is.
- System study phases(I & II):
  - I: initial survey of the system(**scope identification**)
  - II: depth study of the system (**requirement identification**, limitation & issues of the current system). It also includes the back ground analysis and inference or findings of the system.
- **Output:** system proposal or recommendations to overcome the limitations / issues of the current system.

# Feasibility Study

*An analysis of a proposed project to determine whether it is feasible and should go ahead.*



# Stage 2: Feasibility Study

- A feasibility study precedes the decision to begin a project. It is an assessment of the practicality of a proposed system.
  - **Operational feasibility:** refers to the measure of solving problems with the help of a new proposed system. It helps in taking advantage of the opportunities and fulfills the requirements as identified during the development of the project. It takes care that the management and the users support the project.
  - **Economical feasibility:** A project is considered economically feasible when the benefits that will accrue to the broad community are greater than the cost of undertaking the project.
- A feasibility study leads to a decision: **go or no-go**.
- **Output:** FSR

# Five common factors (TELOS)

1. **Technology and system feasibility**
2. **Economic feasibility**
3. **Legal feasibility**
4. **Operational feasibility**
5. **Schedule feasibility**

# 1. Technical Feasibility

- This assessment is based on an outline design of system requirements, to determine whether the company has the technical expertise to handle completion of the project.
  - Is the project possible with current technology?
  - What technical risk is there?
  - Availability of the technology:
    - Is it available locally?
    - Can it be obtained?
    - Will it be compatible with other systems?

# 2. Economic Feasibility

- Economic analysis is the most frequently used method for evaluating the effectiveness of a new system. More commonly known as cost/benefit analysis, the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. If benefits outweigh costs, then the.
  - What are the development and operational costs?
  - Are the benefits worth the costs?

# 3. Legal Feasibility

- Determines whether the proposed system conflicts with legal requirements, e.g. a data processing system must comply with the local Data Protection Acts.

# 4. Operational Feasibility

- Is a measure of how well a proposed system solves the problems, and takes advantages of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.
  - If the system is developed, will it be used?
  - Human and social issues...
    - Potential labour objections?
    - Manager resistance?
    - Organizational conflicts and policies?
    - Social acceptability?

# 5. Schedule Feasibility

- A project will fail if it takes too long to be completed before it is useful. Typically this means estimating how long the system will take to develop, and if it can be completed in a given time period using some methods like payback period. Schedule feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the project deadlines reasonable? Some projects are initiated with specific deadlines. You need to determine whether the deadlines are mandatory or desirable.
  - Is it possible to build a solution in time to be useful?
    - What are the consequences of delay?
    - Any constraints on the schedule?
    - Can these constraints be met?

# Other feasibility factors

- **Market and real estate feasibility**
- **Resource feasibility**
- **Cultural feasibility**

# Stage 3: System Analysis

- Forms the **basis of agreement** between user and developer. System analysis establish the system's services, constraints and goals by consultation with users.
- It is the study of **specifications, operations and relationships** with in the system and outside the system.
- Specifies what not how. (Hard task)
- Define the **boundary** of the new system keeping in view the problems and the new requirement.
- **Output:** is the Software Requirements Specification (SRS) document.

# Stage 4: System Design

- A major step in moving from *problem* to *solution*.
- Based on system analysis, the new system must be designed.
- *Two main tasks*
  - General design: (preliminary design) components and connectors that should be there in the system
  - Detailed design: (*Detailed Design*) logic of modules

*Output: SDS(system Design Specification)*

# Tools and Techniques for Designing

1. Flow Chart

2. Data Flow Diagram

3. Data Dictionary

4. Structured English

5. Decision Tables

# Flow Chart

- Draw a flowchart to find the sum of 5 numbers
- Draw a flowchart to log in to face book account

# Decision Tables

- 1) List all actions that can be associated with a specific procedure (or module)
- 2) List all conditions (or decisions made) during execution of the procedure

List of Conditions	Combination Of Conditions
List of Actions	The Corresponding Set of Actions

Conditions	Rules					
	1	2	3	4	5	6
Regular customer	T	T				
Silver customer			T	T		
Gold customer					T	T
Special discount	F	T	F	T	F	T
Actions						
No discount		✓				
Apply 8 percent discount			✓	✓		
Apply 15 percent discount					✓	✓
Apply additional x percent discount		✓		✓		✓

# Decision Tables

## Activity Task:

For the SafeHome problem, assume that the system is connected to the network. Write a decision table based on the following facts;-

The homeowner is supposed to get an E-Mail if and only if noise level goes beyond a level. If the temperature goes beyond a level not only homeowner will be getting an E-Mail, but also alarm has to be switched on along with a telephone call to a local police station. Same thing goes by for the fact when pressure goes beyond certain level.

# Stage 5: Coding

- Converts design into **code** in specific language
- **Goal:** Implement the design with simple and easy to understand code
- programs must be **modular** in nature. This helps in fast development, maintenance and future changes, if required.
- Coding phase affects both **testing** and **maintenance**.
  - Well written code reduces testing and maintenance effort.
- **Output:** is source-code.

# Stage 6: Testing

- **Defects** are introduced in each phase
  - Must be found and removed to achieve high quality
- Software testing is a process of analyzing software for the purpose of finding bugs.
- Using test data, following test runs are carried out
  - **Unit test:** performed by the respective developers on the individual units of source code to ensure that the individual parts are correct in terms of requirements and functionality.
  - **System test:** done after unit test. System testing tests the system as a whole. Actual output of the system is matched with the expected outputs. Errors are identified and fixed.
  - **User acceptance testing (UAT)** – determines if the system satisfies the business requirements
- **Outputs:** are
  - Test plans/results
  - Final tested (reliable) code

# Stage 7: Built Release

- After UAT, deployment phase begins.
- Final phase of SDLC, puts the product into production
- All programs of system are loaded onto the user's computer.
- Then training of user starts including
  - how to execute the package
  - how to enter the data
  - how to process data

# Built Release Strategies

- **Parallel run:** computerized & manual systems are executed in parallel.
- Advantages of Parallel run:
  - Manual results comparison with the computerized one.
  - Failure of the computerized system at the early stage, does not affect the working of the organization.
- **Pilot run:** New system is installed in parts. Some part of the new system is installed first and executed successfully for considerable time period.
- **Advantages:**
  - When results are found satisfactory then only other parts are implemented.
  - This strategy builds the confidence and the errors are traced easily.

# Stage 8: Maintenance

- Maintenance phase focuses on changes that associated with
  - Error Correction
  - Platform Adaptations required
  - Enhancement due to change
  - Re-engineering
- Maintenance is required to:
  - eliminate errors in the system during its working life
  - tune the system to any variations in its working environment.
- System Review: is necessary from time to time for:
  - knowing the full capabilities of the system
  - knowing the required changes or the additional requirements
  - studying the performance
- Major change during the review:
  - If a major change to a system is needed, a new project may have to be set up to carry out the change.
  - New project will then proceed through all above life cycle phases.

# System Environments

- Development
- Test
- Staging
- Pre-Production
- Production
- Mirror
- Roles involved:
- D, PM, TM, BA.



That is all

# **Software Process & Process Models**

**Lecture # 7,8  
04, 06 Feb**

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)

## **Software Engineering CS-303**



# Today's Outline

- DFD
- Context Level DFD

# Data Flow Diagrams

- DFDs are graphic representation of the **flow of data** or information into and out of a system
  - ***what does the system do to the data?***
- A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system.
- The DFD also provides information about the outputs and inputs of each entity and the process itself.
- A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart

# Why DFD?

- DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer.
- DFD has often been used due to the following reasons:
  - Logical information flow of the system
  - Determination of physical system construction requirements
  - Simplicity of notation

# DFD Symbols

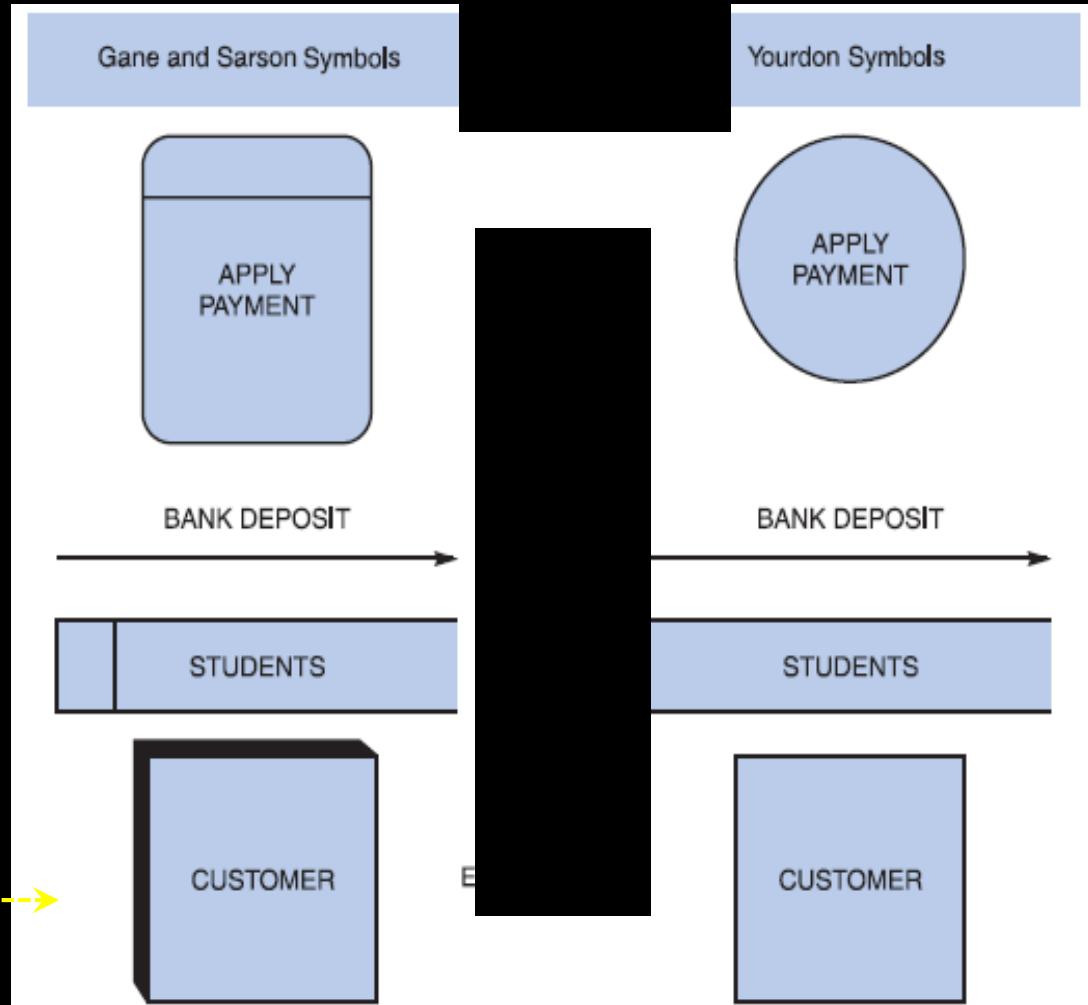
DFDs use **four basic symbols** that represent

Processes

Data flows

Data stores

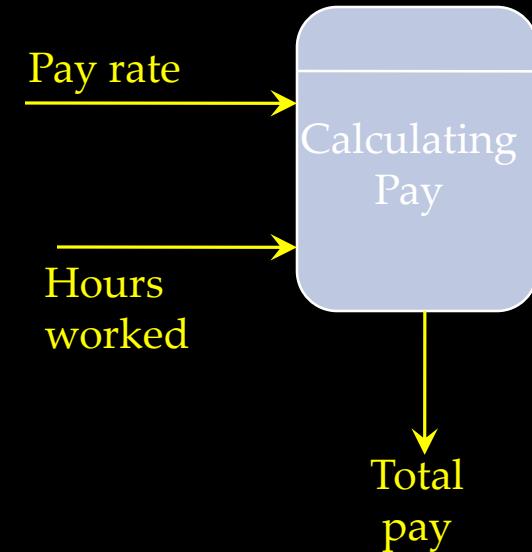
External Entities



# Data Flow Diagrams

## Process Symbol

- A process receives ***input data*** and produces ***output***.
- For instance, the process for ***calculating pay*** uses two inputs (**pay rate** and **hours worked**) to produce one output (**total pay**).
  - Processes can be very simple or quite complex.
  - Note: Processing details are not shown in a DFD.



# Process Symbol

- A process symbol can be referred to as a **black box**, because the inputs, outputs, and general functions of the process are known, but the ***underlying details*** and ***logic*** of the process are **hidden**.
- By showing processes as black boxes, an analyst can create DFDs that show how the ***system functions***, but ***avoid unnecessary detail***.
- When the analyst wishes to show additional levels of detail, he/she can zoom in on a process symbol and create a more in-depth DFD (level-1 and level-2) that shows the process's internal workings.

# Data Flow Diagrams

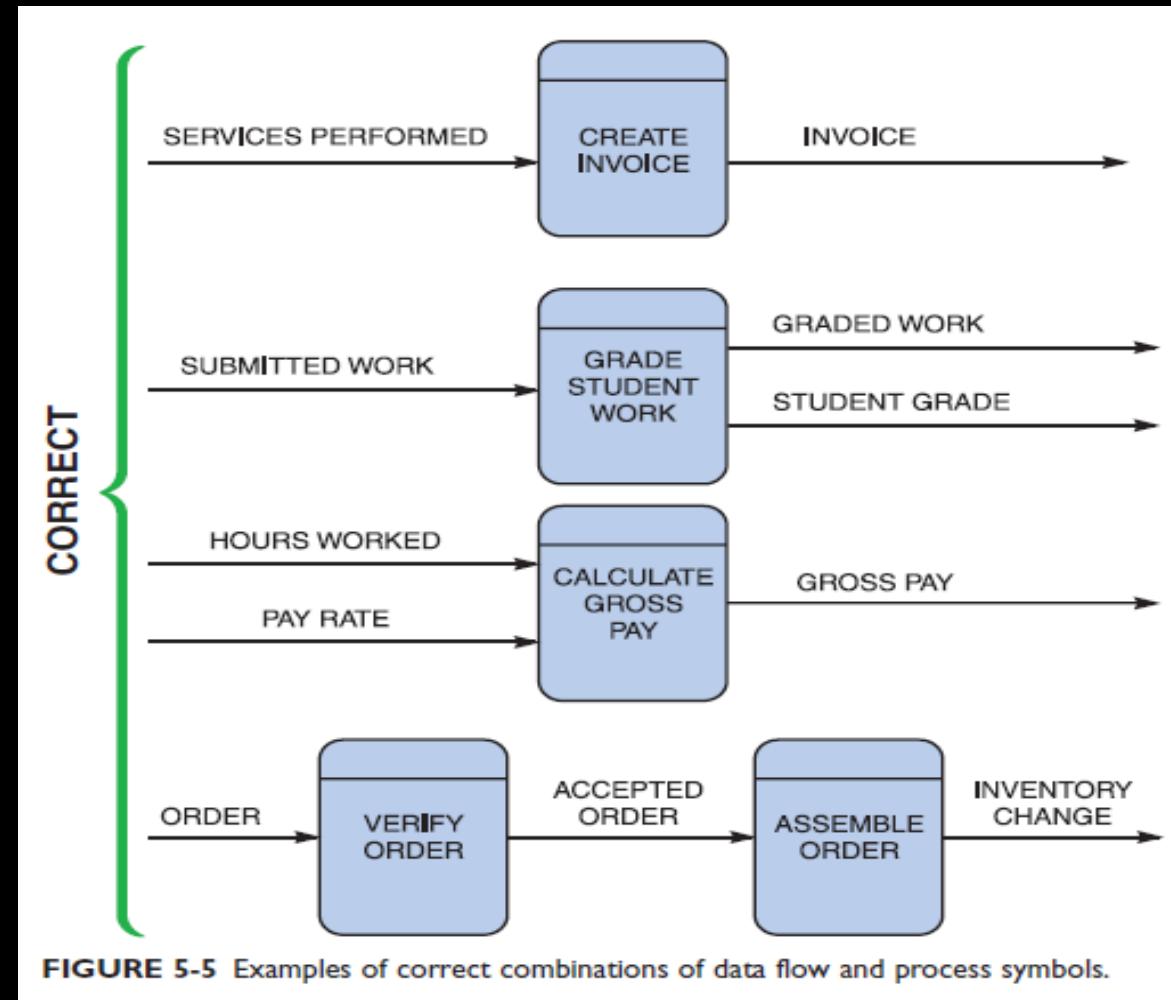
## Data Flow Symbol



- A data flow is a path for data to move from one part of the information system to another.
- A data flow in a DFD represents one or more **data items**.
- In previous example we saw , the process for **calculating pay** uses two inputs (pay rate and hours worked) to produce one output (total pay).

# Data Flow Symbol

- The symbol for a data flow **is a line** with a arrowhead.
- Data flow name** appears above, below, or alongside the line.
- A **data flow name** consists of a singular noun and an adjective, if needed.



# Data Flow Diagrams

## Data Store Symbol



- A **data store** is used in a DFD to represent data that the system stores because one or more processes need to use the data at a later time.
- For instance, **teachers** need to store student scores on tests and assignments during the semester so they can assign final grades at the end of the term.
- Similarly, a **company** stores employee salary and deduction data during the year in order to print W-2 forms with total earnings and deductions at the end of the year.
- A DFD **does not show** the detailed contents of a data store.

# Data Store Symbol

Examples of **data store** names are

DAILY PAYMENTS

ACCOUNTS RECEIVABLE

PATIENTS

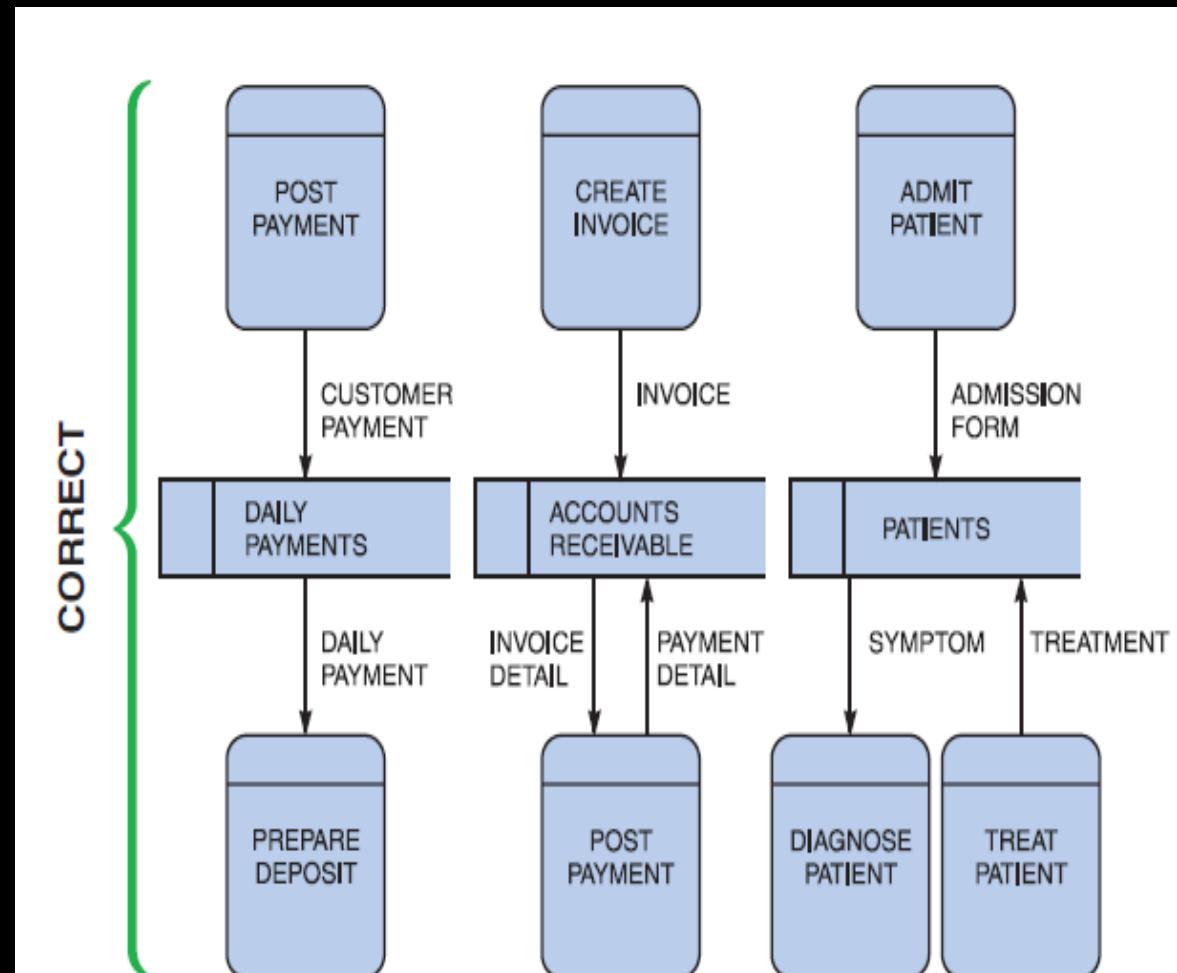


FIGURE 5-7 Examples of correct uses of data store symbols in a data flow diagram.

# Data Flow Diagrams

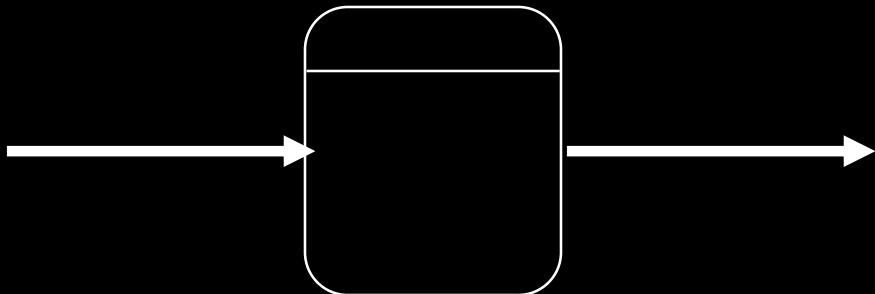
## External Entity Symbol

- The symbol for an **entity** is a **rectangle**. The name of the entity appears inside the symbol.
- DFD shows only **external entities** that **provide data** to the system or **receive output** from the system.
- DFD shows the **boundaries** of the system and how the system interfaces with the outside world.
- DFD entities also are called **terminators**, because they are **data origins** or **final destinations**.

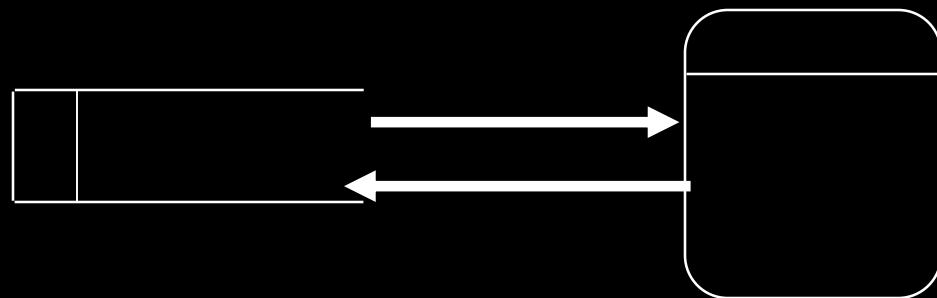
# External Entity Symbol

- Systems analysts call an entity that supplies data to the system **a source**, and an entity that receives data from the system **a sink**.
  - An entity name is the singular form of a
    - department, outside organization,
    - other information system, or person.
  - An external entity can be a source or a sink or both, but each entity must be connected to a process **by a data flow**.
- Intro. To Software Engineering

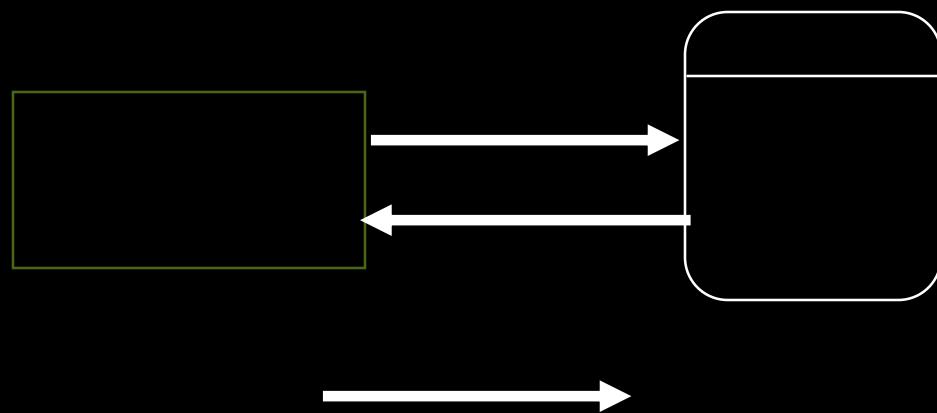
# Rules for Drawing DFD's



A **minimum** of one data flow in and one data flow out of a process



A datastore **must** be connected to a process (either in, out, or both)



An external entity **must** be connected to a process (either in, out, or both)

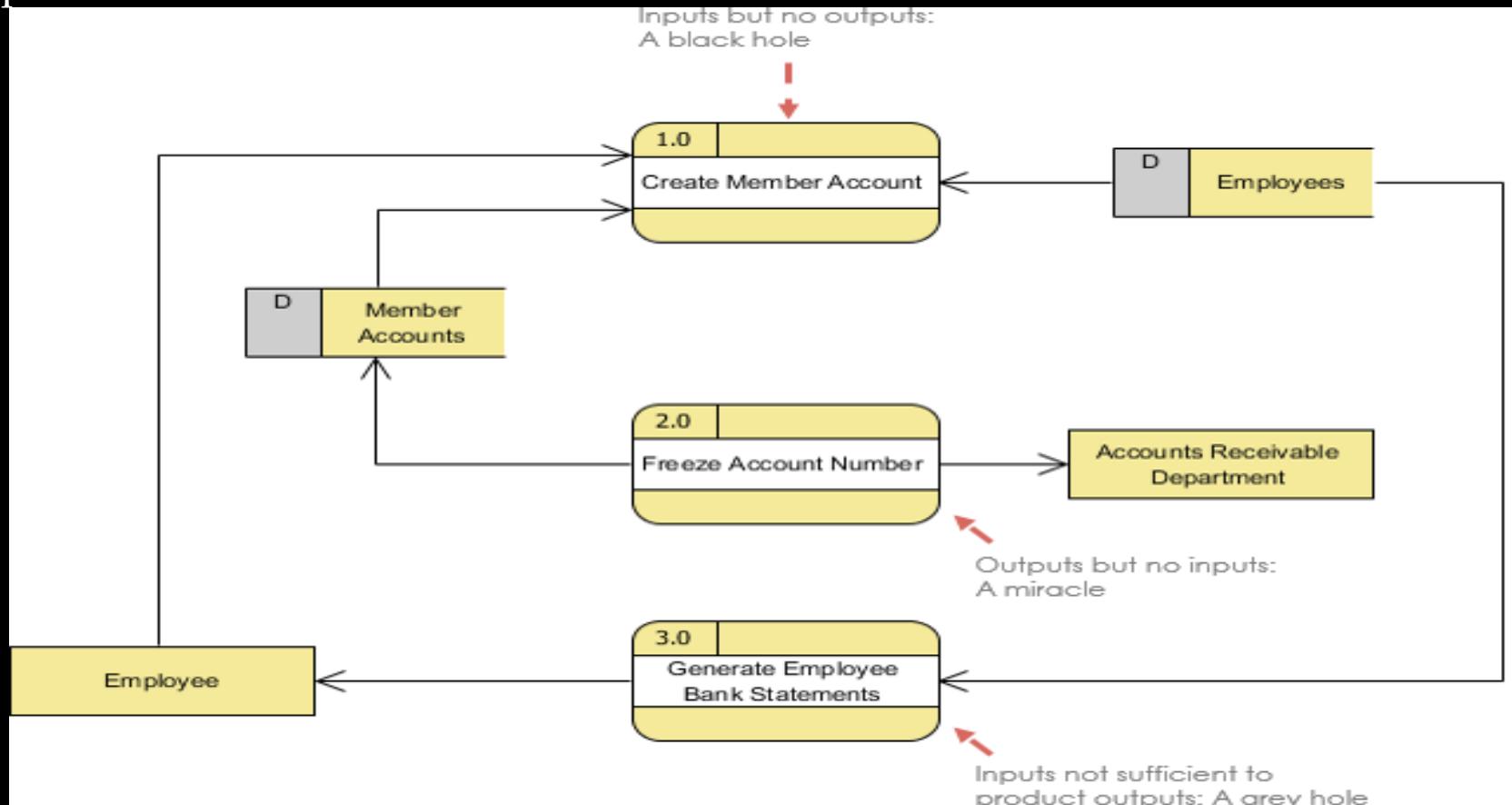
A single data flow **must** only flow one way

# DFD: Common Mistakes

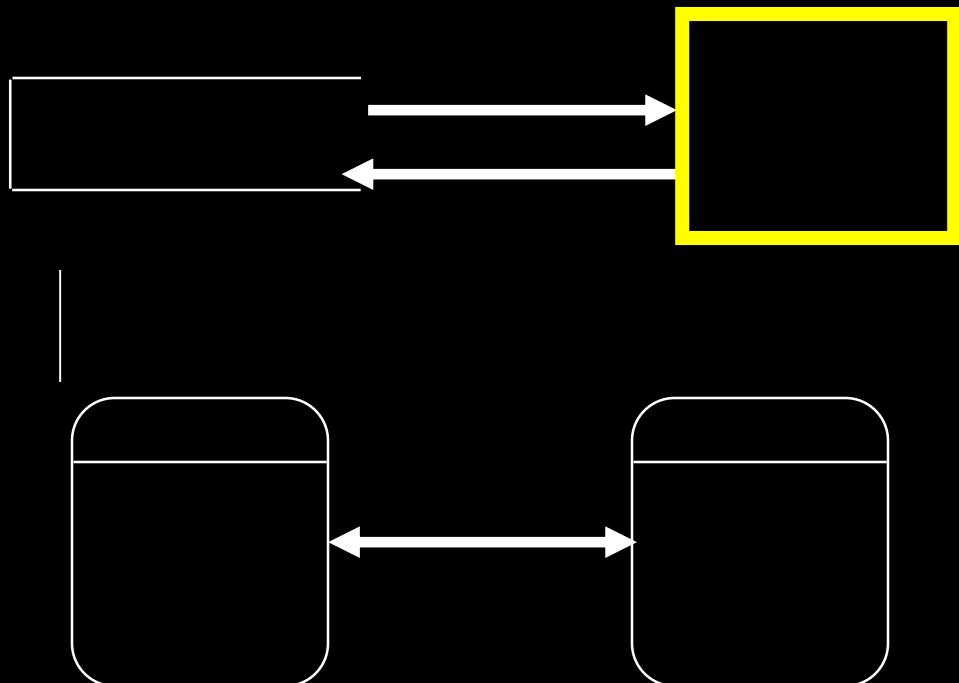
Black holes - A processing step may have input flows but no output flows.

Miracles - A processing step may have output flows but no input flows.

Grey holes - A processing step may have outputs that are greater than the sum of its inputs.



# DFD: Common Mistakes



- Data store is hooked to external entity. This means external entity can read and write to your data file without auditing!!
- The data flow goes in two directions at once. Two or more arrows should be used to show the flow to and from each process.

# DFD Levels: Context Diagram

- First we can start to draw a general overview. This general overview is called a “Context Diagram”.
- A context diagram is a data flow diagram that only shows the top level, otherwise known as Level 0. At this level, there is only one visible process node that represents the functions of a complete system in regards to how it interacts with external entities. Some of the benefits of a Context Diagram are:
  - Shows the overview of the boundaries of a system
  - No technical knowledge is required to understand with the simple notation
  - Simple to draw,
- A Context Diagram shows three things:
  - all external entities
  - a single process labeled “0”that represents the entire system
  - the major information flows between the external entities and the system.

# Data Flow Diagram Tips and Cautions

1. Process labels should be verb phrases; data stores are represented by nouns
2. A data store must be associated with at least a process
3. An external entity must be associated with at least a process
4. Don't let it get too complex; normally 5 - 7 average people can manage processes
5. Datastores should not be connected to an external entity, otherwise, it would mean that you're giving an external entity direct access to your data files
6. Data flows should not exist between 2 external entities without going through a process

# Building a DFD

- It would be impossible to understand all of the data flows, and to identify all of the ‘external entities’ relating to our information system in one pass, so we tend to draw DFD’s incrementally.
- We tend to start at the context level, break processes down to Level 0, and then start to consider where data enters and exits our information system, where it is stored, and how a process converts it from one form to another. We are interested here in the movement of *data* and its conversion.

# *DFD Example: Bus Garage Repairs*

- Buses come to a garage for repairs.
- A mechanic and helper perform the repair, record the reason for the repair and record the total cost of all parts used on a Shop Repair Order.
- Information on labor, parts and repair outcome is used for billing by the Accounting Department, parts monitoring by the inventory management computer system and a performance review by the supervisor.

# *DFD Example: Bus Garage Repairs*

*(cont'd)*

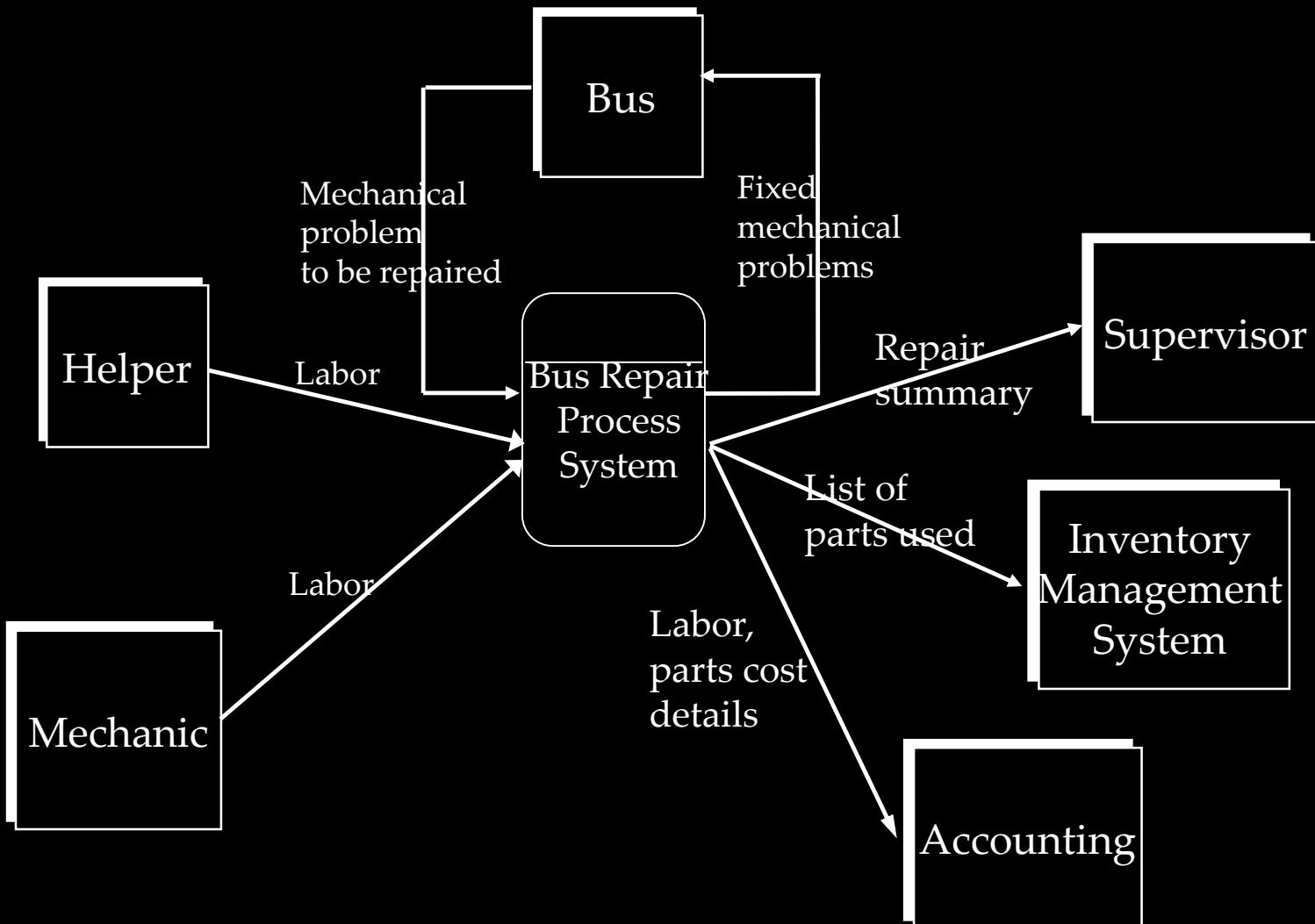
- *External Entities:* Bus, Mechanic, Helper, Supervisor, Inventory Management System, Accounting Department, etc.
- Key process ("the system"): performing repairs and storing information related to repairs
- *Processes:*
  - Record Bus ID and reason for repair
  - Determine parts needed
  - Perform repair
  - Calculate parts extended and total cost
  - Record labor hours, cost
- Intro. To Software Engineering

# *DFD Example: Bus Garage Repairs*

## *(cont'd)*

- ***Data stores:***
  - **Personnel file**
  - **Repairs file**
  - **Parts list**
- ***Data flows:***
  - **Repair order**
  - **Bus record**
  - **Parts record**
  - **Employee timecard**
  - **Invoices**

# *Bus Garage Context Diagram*



# Building a DFD

- A DFD is NOT a flowchart. It simply shows how and where data itself progresses through our system.

# Today's Outline

- Software process models
  - Water Fall Model
  - Evolutionary Development
  - Component base Software Engineering / Reuse-oriented development
- Process iteration
  - Incremental Model
  - Spiral Model
- Process activities

# The software process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

# Plan-driven and Agile Processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

# Software Process Model

- **Software Process** : is coherent sets of activities for specifying, designing, implementing and testing software systems.
- A **software process model** is an abstract representation of a software process.
- *It is a description of the sequence of activities carried out in an SE project, and the relative order of these activities. It presents a description of a process from some particular perspective.*

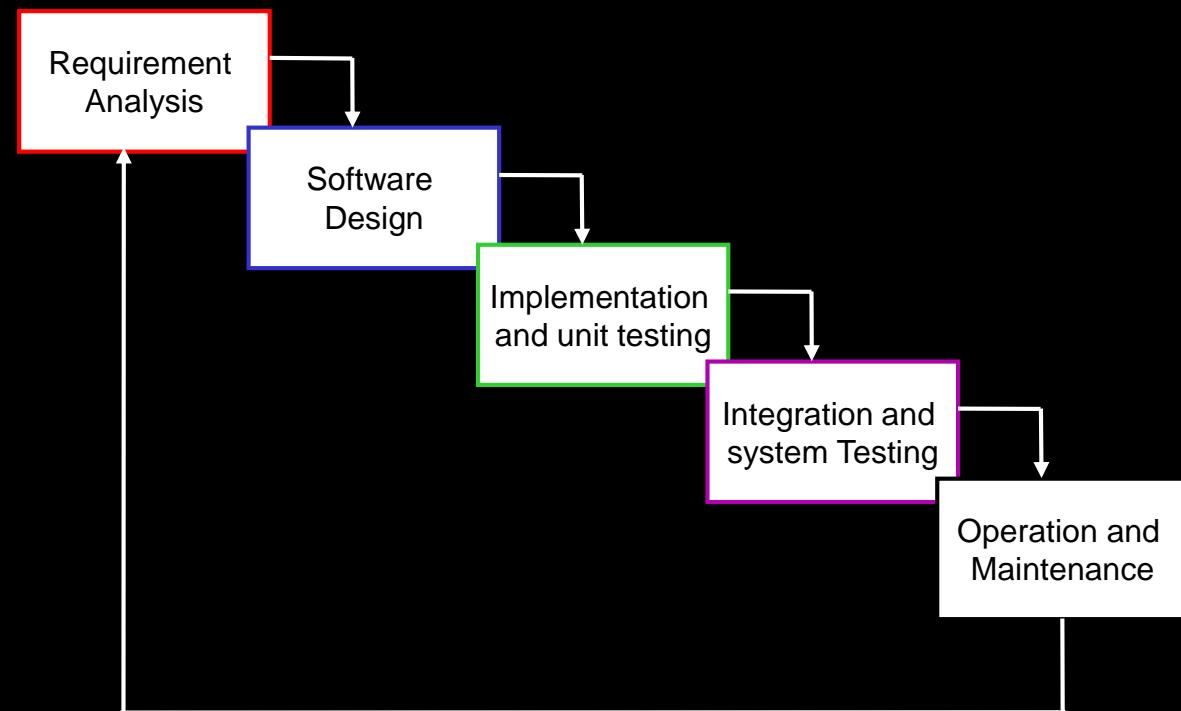
# Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development. It is the oldest paradigm for SE. When requirements are well defined and reasonably stable, it leads to a linear fashion.
- Incremental development
  - Specification, development and validation are interleaved.
- Integration and configuration
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

# Waterfall

Each box represents a **set of tasks** that results in the production of each or more work products.

Each new phase begins when the **work products** of the previous phase as completed, frozen and signed off.



## Waterfall (when to use)

- Well suited for projects where requirements can be understood easily and technology decisions are easy
- Has been used widely
- For standard/familiar type of projects it still may be the most optimum.

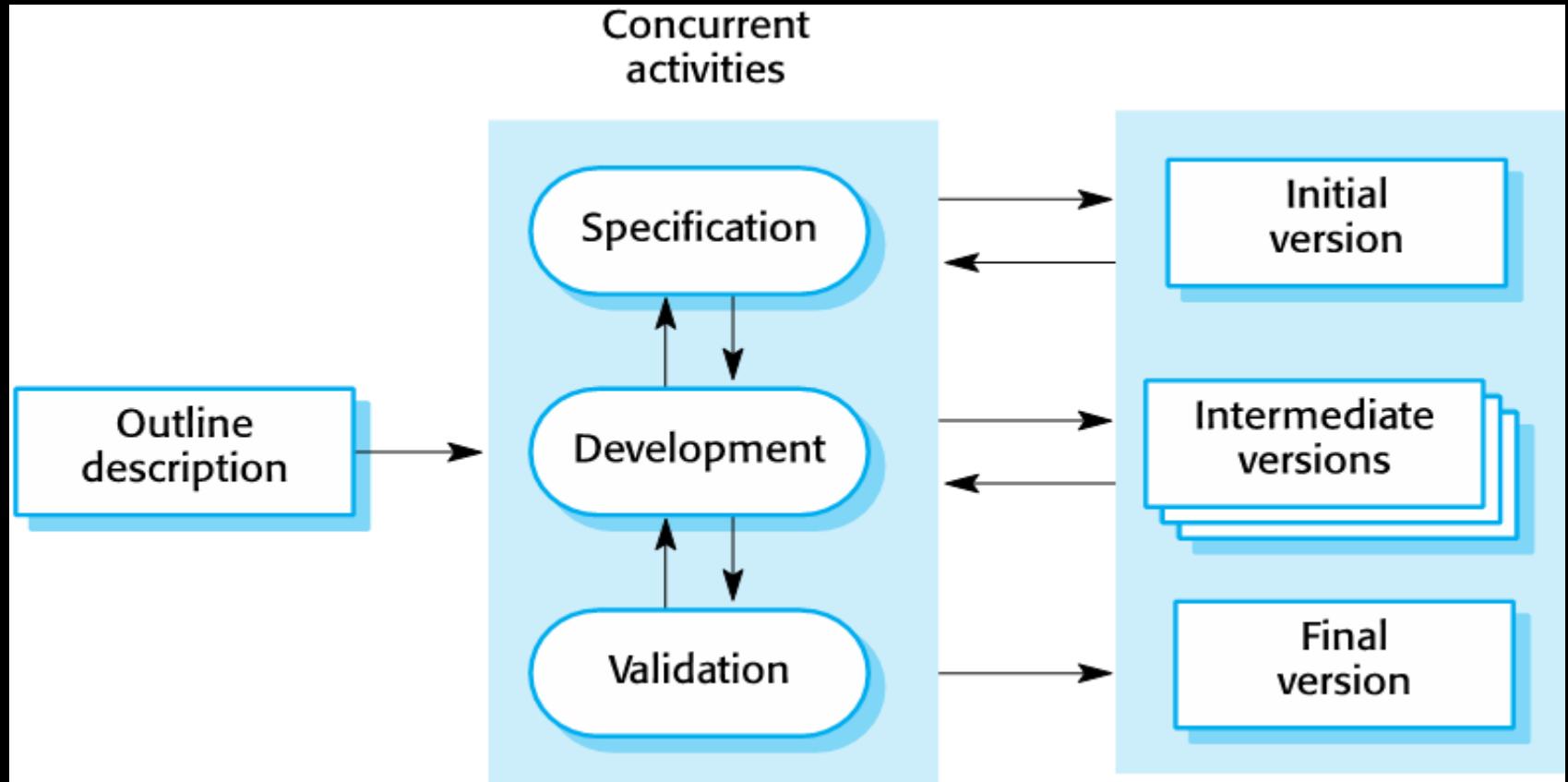
# Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

# Disadvantages of Waterfall Model

- Unable to work where high level of uncertainty is involved
- Requirements need to be stable hence making model rigid
- Unrealistic to state all requirements at the beginning
- Does not handle concurrent events – development teams are delayed waiting for others
- Difficult and expensive to change decisions
- The user is involved only in the beginning phase of requirement gathering and than during acceptance phase

# Iterative Development



# Iterative Development Benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

# Iterative development problems

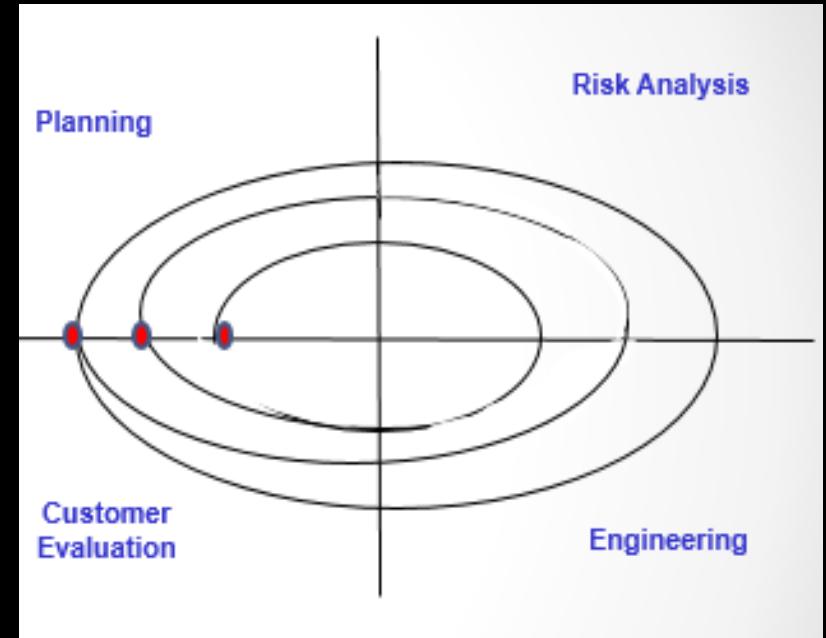
- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

# Evolutionary Models: Spiral

- Spiral is primary **Risk Driven** approach. Spiral model consist of different **cycle**. **In each cycle we try to address some risk elements.**
- **Planning:** Determines objectives, alternatives and constraints.
- **Risk Analysis:** Analysis of alternatives as well as an identification and/or resolution of risks.
- **Engineering:** Development of the next level of product
- **Customer evaluation:** Assessment of the results of engineering

# Spiral

- With each iteration around the spiral progressively more complete versions of the software are built.
- Spiral model enables the developer, and the customer, to understand and **react to risk** at each evolutionary level.
- Each loop around the spiral implies that project costs and schedules may be modified.



- This creates problems in fixed-price project.



That is all

# **Software Process & Process Models**

**Lecture # 9, 10  
11, 12 Feb**

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)

# **Software Engineering**

**CS-303**



# Today's Outline

- Integration and Configuration
- Software prototyping
- Incremental development process model
- Software process Activities

# DFD Example level 0 and level 1

- <https://www.visual-paradigm.com/tutorials/data-flow-diagram-example-food-ordering-system.jsp>

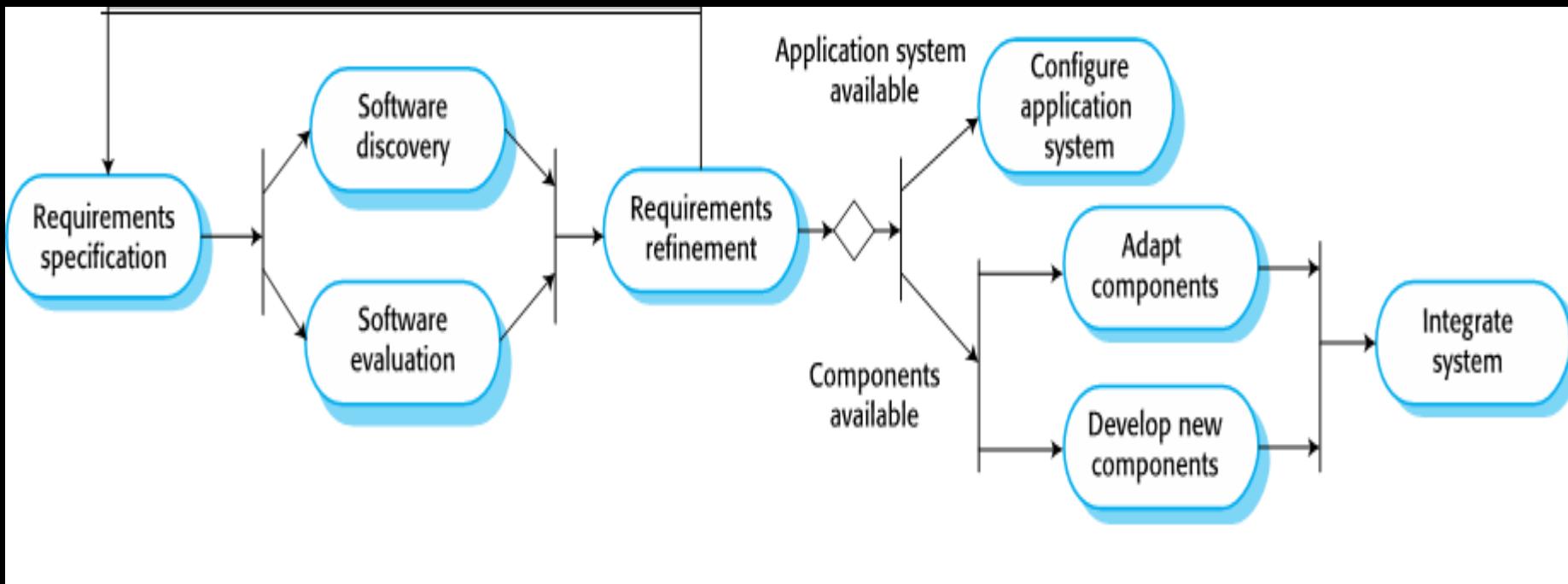
# Integration and Configuration

- Based on software reuse where systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf- systems).
- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
- Reuse is now the standard approach for building many types of business system
  - Reuse covered in more depth in Chapter 15.

# Types of Reusable Software

- Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Web services that are developed according to service standards and which are available for remote invocation.

# Reuse-oriented Software Engineering



# Key Process Stages

- Requirements specification
- Software discovery and evaluation
- Requirements refinement
- Application system configuration
- Component adaptation and integration

# Advantages and Disadvantages

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- But requirements compromises are inevitable so system may not meet real needs of users
- Loss of control over evolution of reused system elements

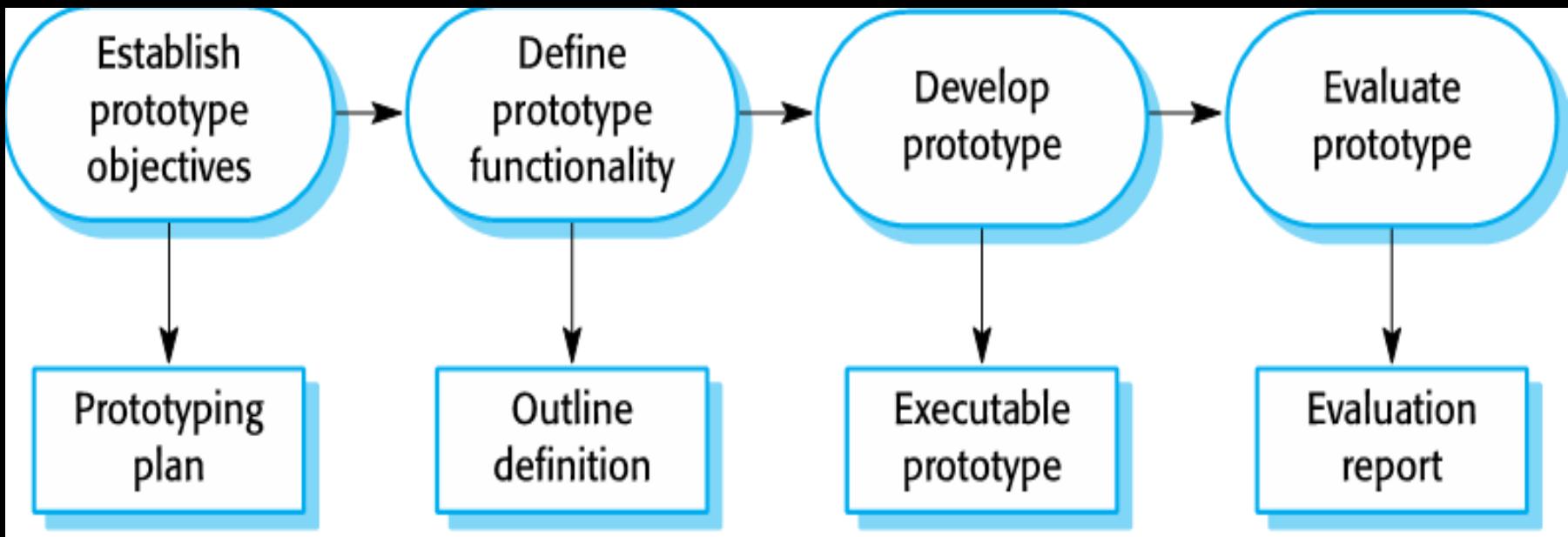
# Software Prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation;
  - In design processes to explore options and develop a UI design;
  - In the testing process to run back-to-back tests.

# Benefits of Prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

# The Process of Prototype Development



# Prototype Development

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - Prototype should focus on areas of the product that are not well-understood;
  - Error checking and recovery may not be included in the prototype;
  - Focus on functional rather than non-functional requirements such as reliability and security

# Throw-away Prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organisational quality standards.

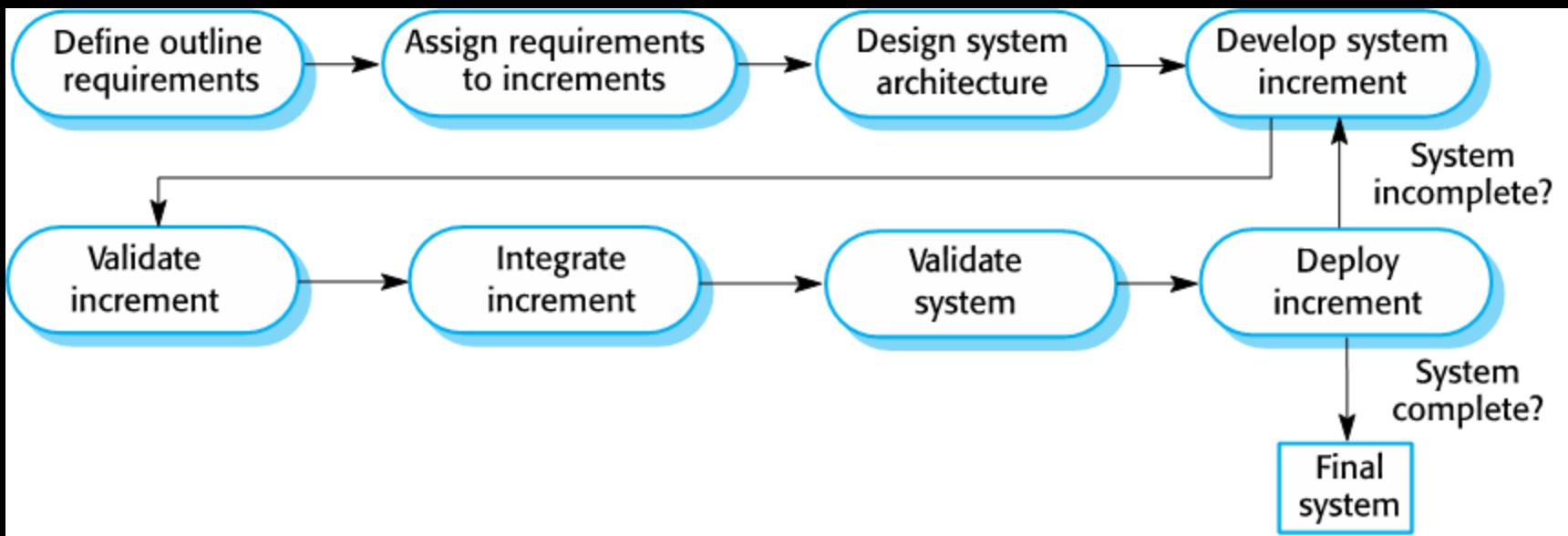
# Incremental Delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

# Incremental Development and Delivery

- Incremental development
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  - Normal approach used in agile methods;
  - Evaluation done by user/customer proxy.
- Incremental delivery
  - Deploy an increment for use by end-users;
  - More realistic evaluation about practical use of software;
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

# Incremental Delivery



# Incremental Delivery Advantages

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

# Process Activities

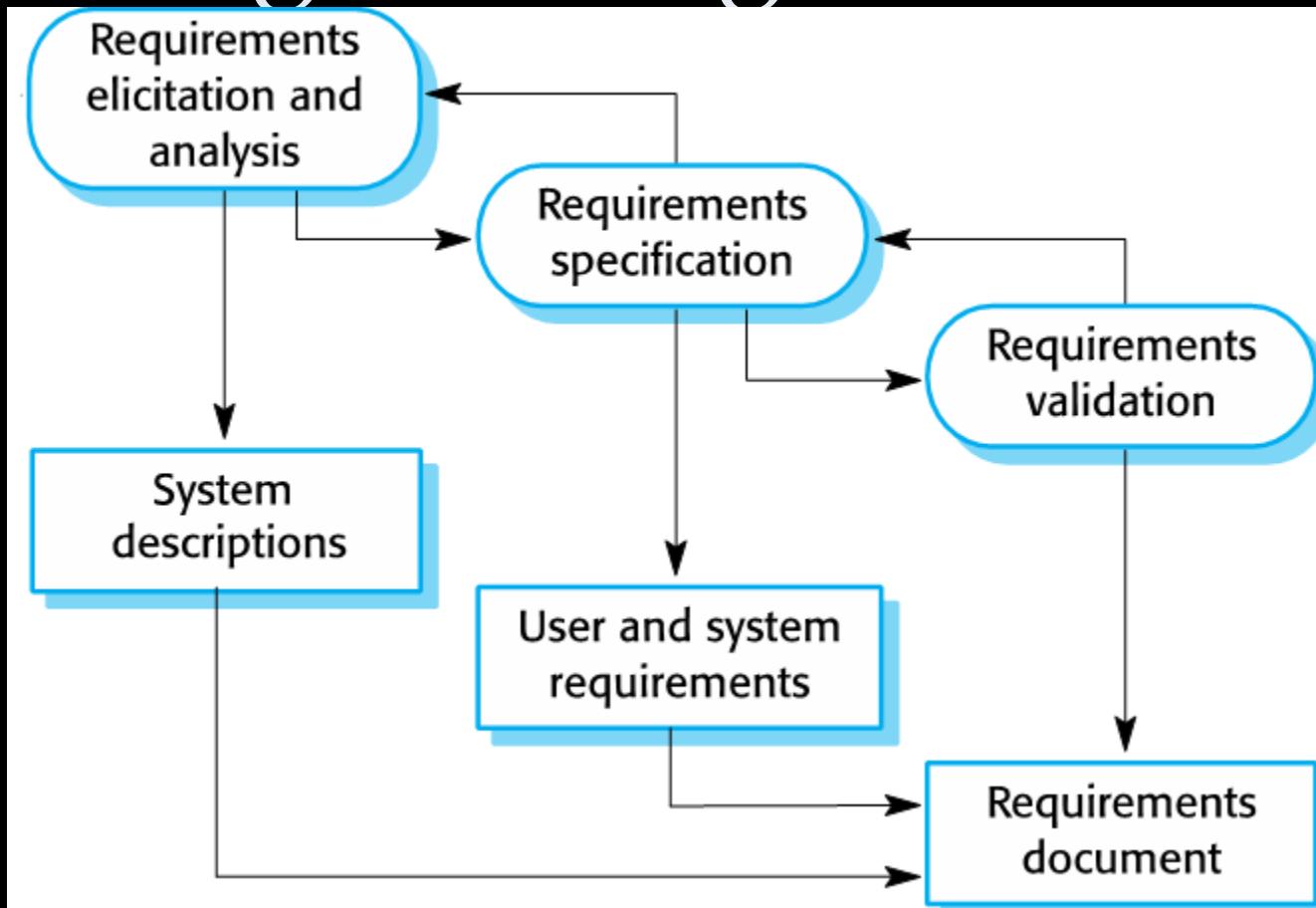
# Process Activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
- For example, in the waterfall model, they are organized in sequence, whereas in iterative development they are interleaved.

# Software Specification

- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
  - Requirements specification
    - Defining the requirements in detail
  - Requirements validation
    - Checking the validity of the requirements

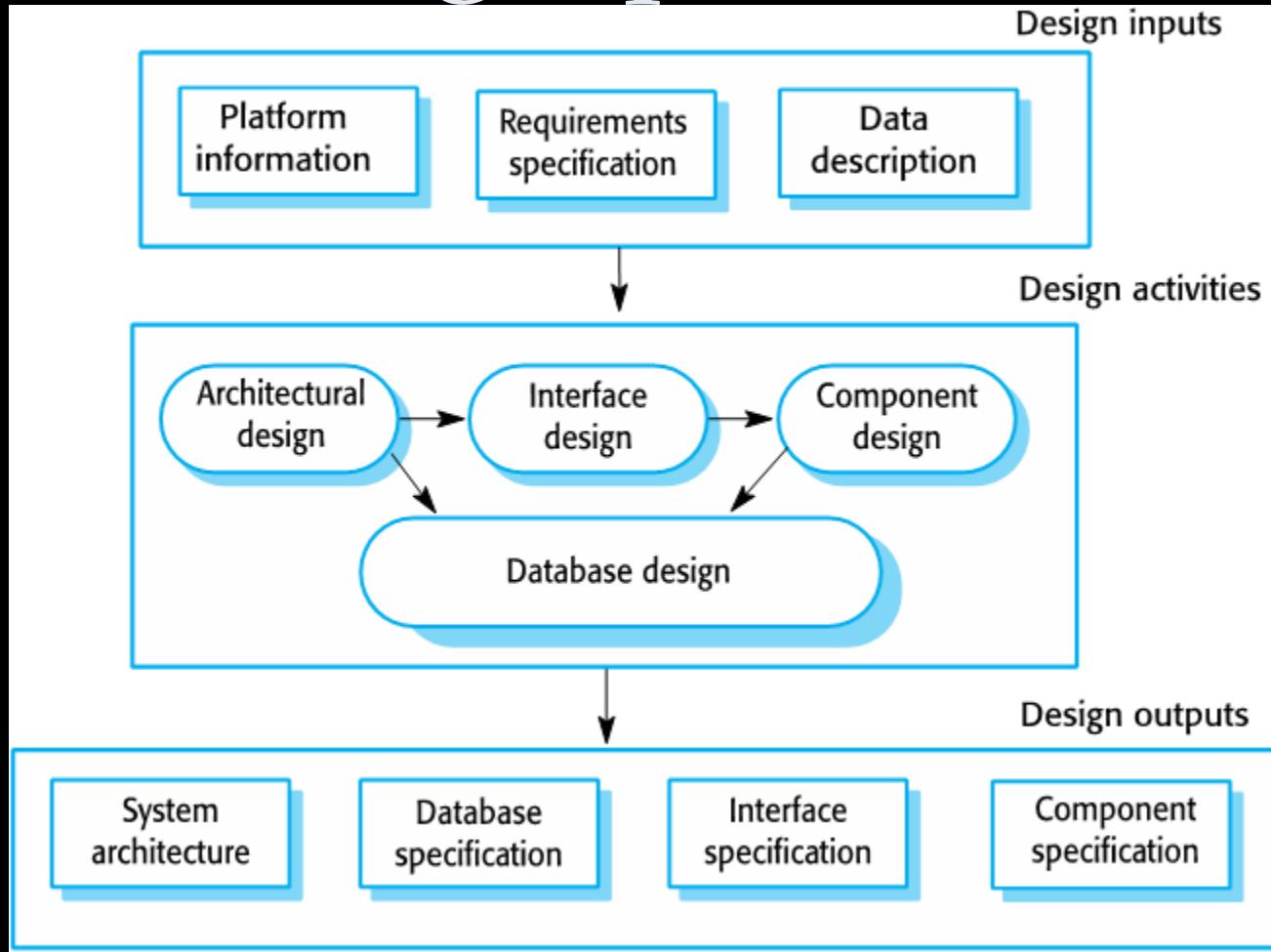
# The Requirements Engineering Process



# Software Design and Implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

# A General Model of the Design process



# Design Activities

- *Architectural design*, where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
- *Database design*, where you design the system data structures and how these are to be represented in a database.
- *Interface design*, where you define the interfaces between system components.
- *Component selection and design*, where you search for reusable components. If unavailable, you design how it will operate.

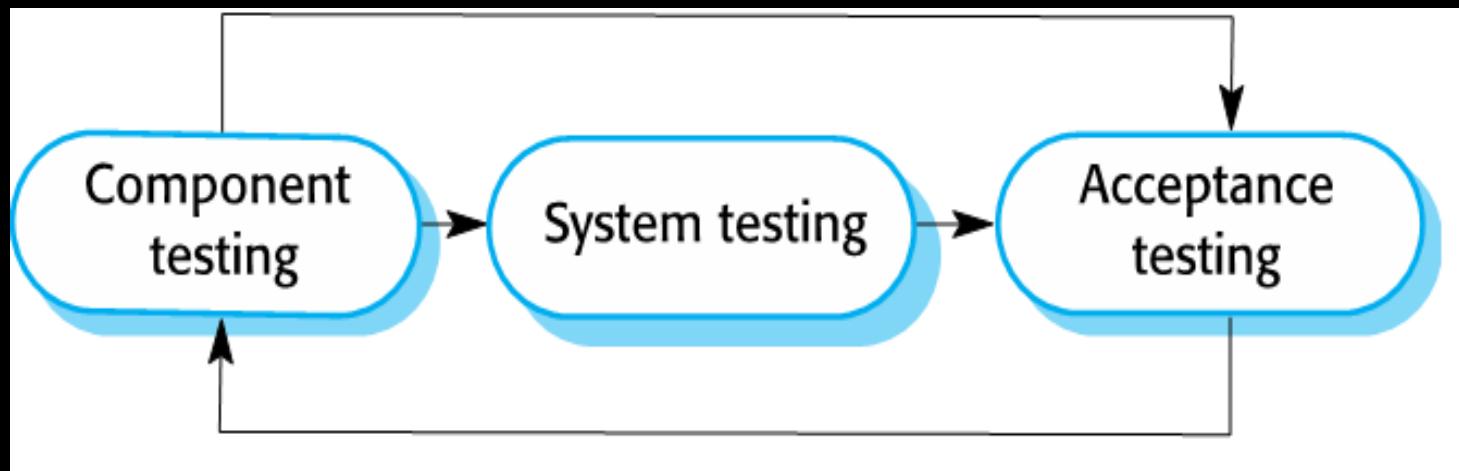
# System Implementation

- The software is implemented either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- Programming is an individual activity with no standard process.
- Debugging is the activity of finding program faults and correcting these faults.

# Software Validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

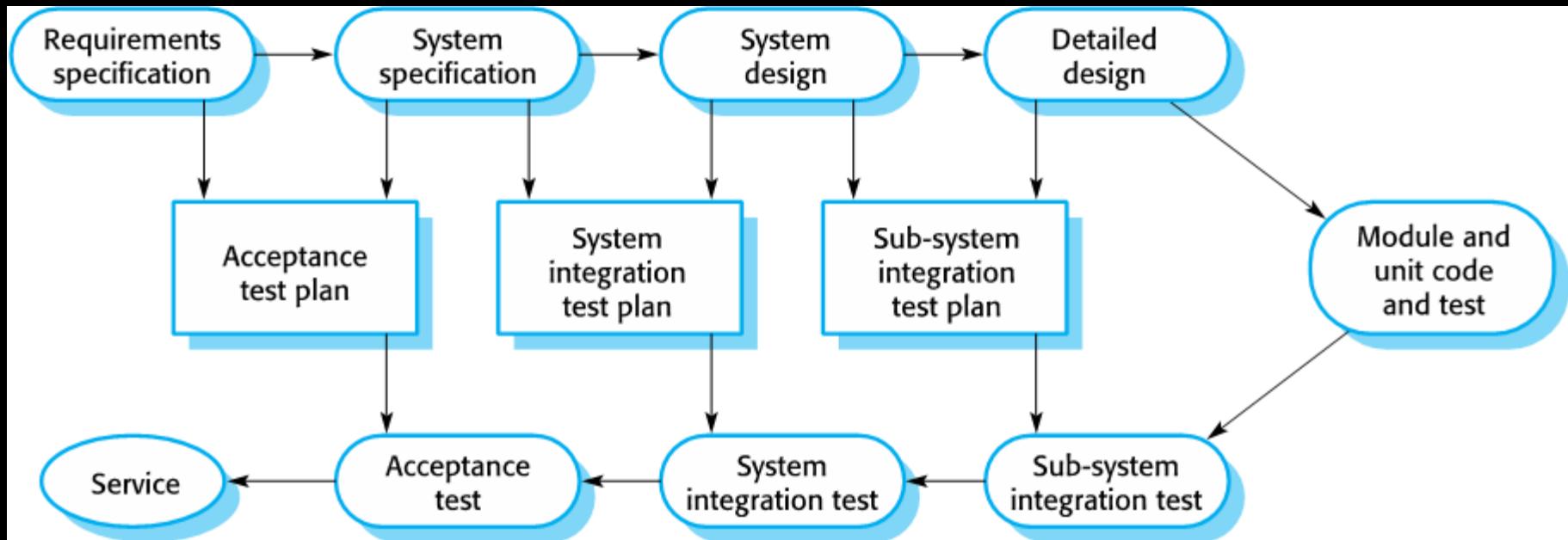
# Stages of Testing



# Testing Stages

- Component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- Customer testing
  - Testing with customer data to check that the system meets the customer's needs.

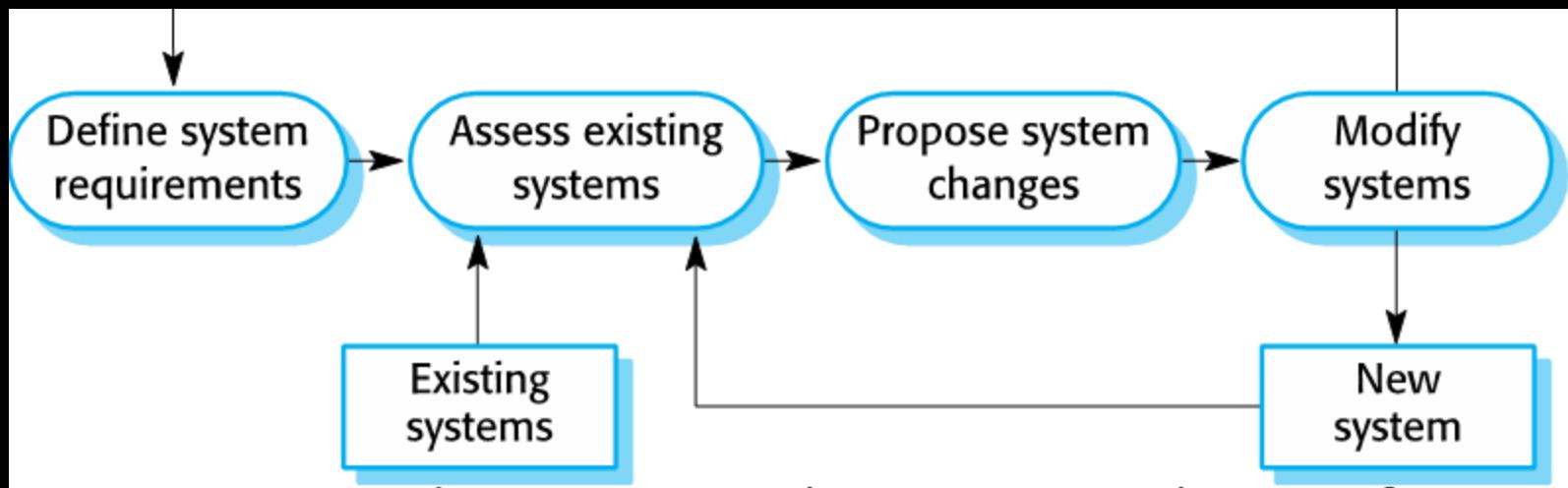
# Testing phases in a plan-driven software process (V-model)



# Software Evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

# System Evolution





That is all

# Agile Software Development

Lecture # 12, 13  
13, Feb

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)

# Software Engineering

CS-303



# Topics covered

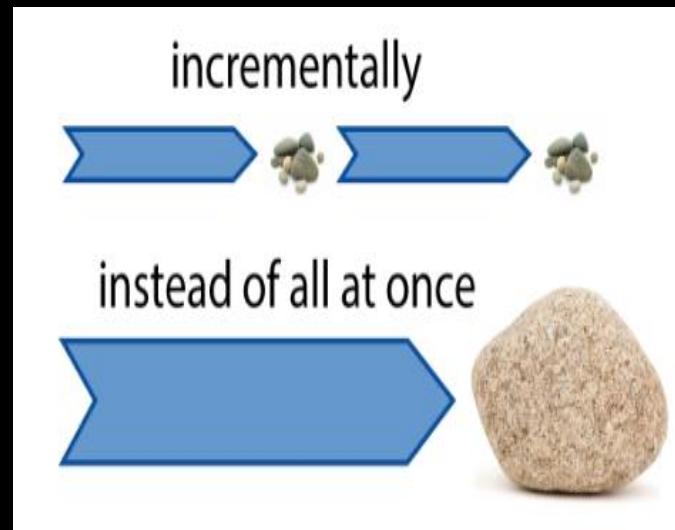
- Agile methods
- Agile development techniques
- Quiz

# Rapid Software Development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems

# About Agile

- ❑ Agile is a Software Development Methodology
- ❑ It means 'ability to move quickly and easily' and responding swiftly to change
- ❑ It's a time boxed, iterative approach
- ❑ Develop and deliver software incrementally from the start of the project, instead of trying to deliver it all at once near the end.



# Example - Incremental



In the diagram above when we work **incrementally** we are adding piece by piece but expect that each piece is fully finished. Thus keep on adding the pieces until it's complete. As in the image above a person has thought of the application. Then he started building it and in the first iteration the first module of the application or product is totally ready and can be demoed to the customers. Likewise in the second iteration the other module is ready and integrated with the first module. Similarly, in the third iteration the whole product is ready and integrated. Hence, the product got ready step by step.

# Example-- Iterative



In the diagram above when we work **iteratively** we create rough product or product piece in one iteration, then review it and improve it in next iteration and so on until it's finished. As shown in the image above, in the first iteration the whole painting is sketched roughly, then in the second iteration colors are filled and in the third iteration finishing is done. Hence, in iterative model the whole product is developed step by step.

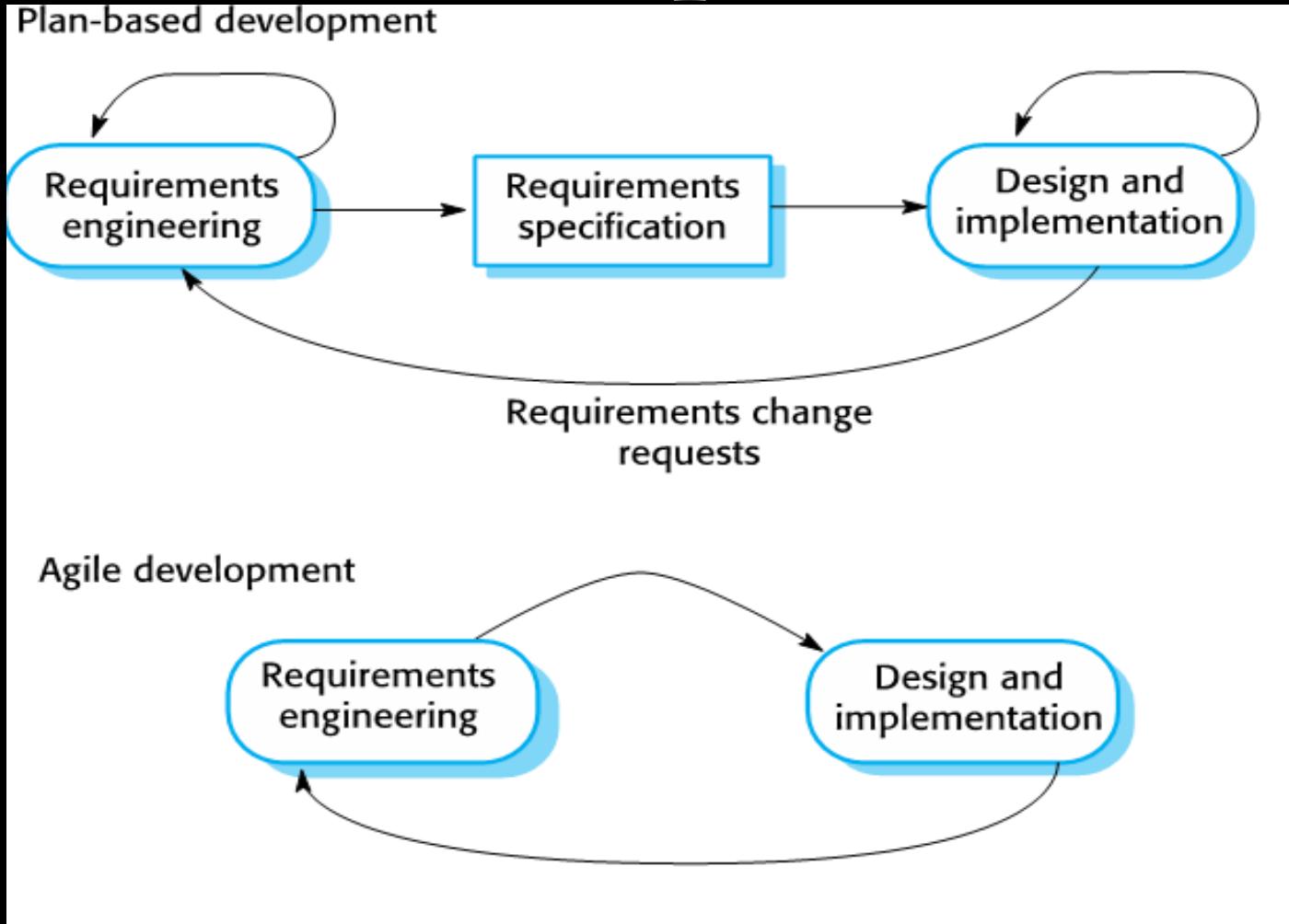
# Agile Development Characteristics

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

# Plan-driven and Agile Development

- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

# Plan-driven and Agile Development



# Agile Methods

# Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

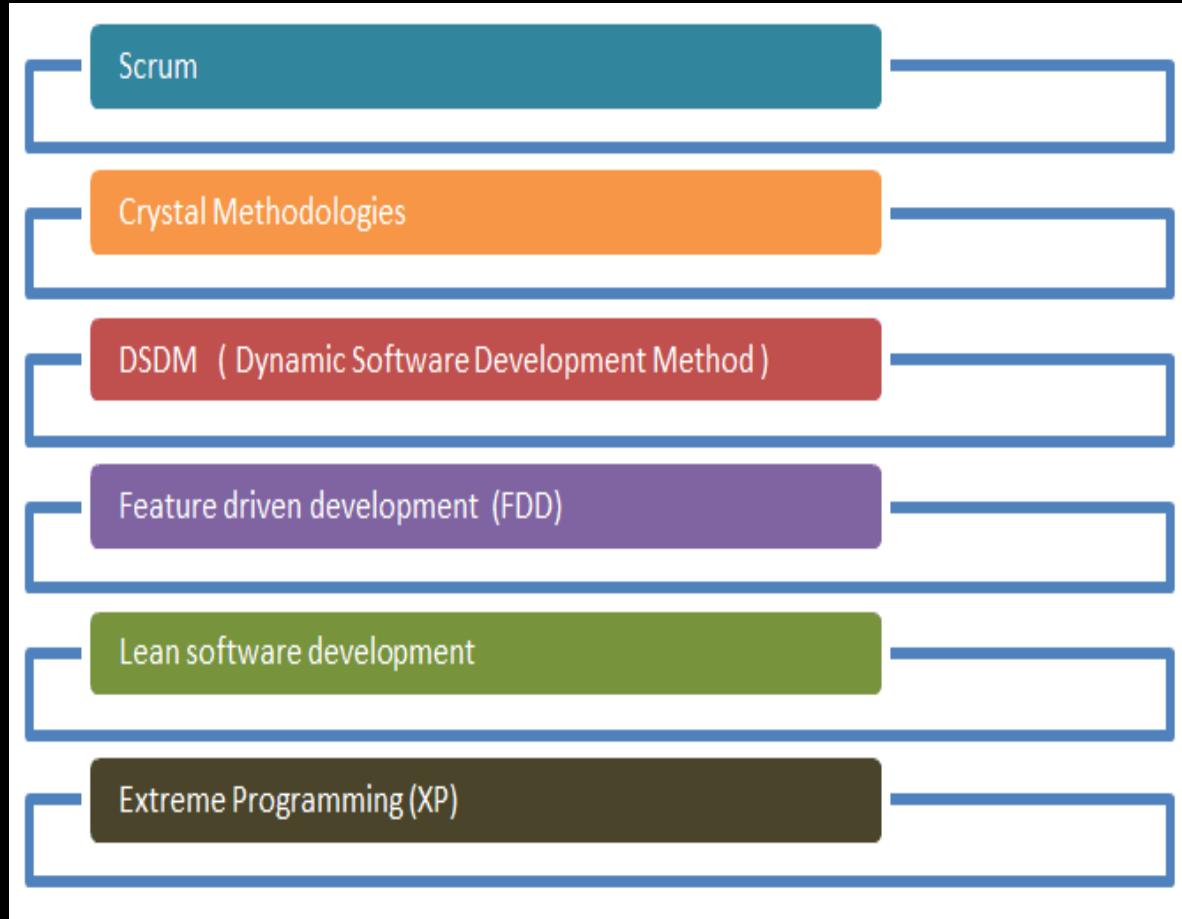
# Agile Manifesto

- We are *uncovering better ways of developing software by doing it and helping others do it.* Through this work we have come to value:
  - Individuals and interactions over processes and tools*
  - Working software over comprehensive documentation*
  - Customer collaboration over contract negotiation*
  - 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.*

# Agile Core Values

- **Individuals and interactions** over processes and tools---in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software** over comprehensive documentation---working software will be more useful and welcome than just presenting documents to clients in meetings.
- **Customer collaboration** over contract negotiation---requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.
- **Responding to change** over following a plan--- agile development is focused on quick responses to change and continuous development.

# Agile Methods



# The Principles of Agile Methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

# Agile Method Applicability

- Product development where a software company is developing a small or medium-sized product for sale.
  - Virtually all software products and apps are now developed using an agile approach
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software.

# Agile Development Techniques

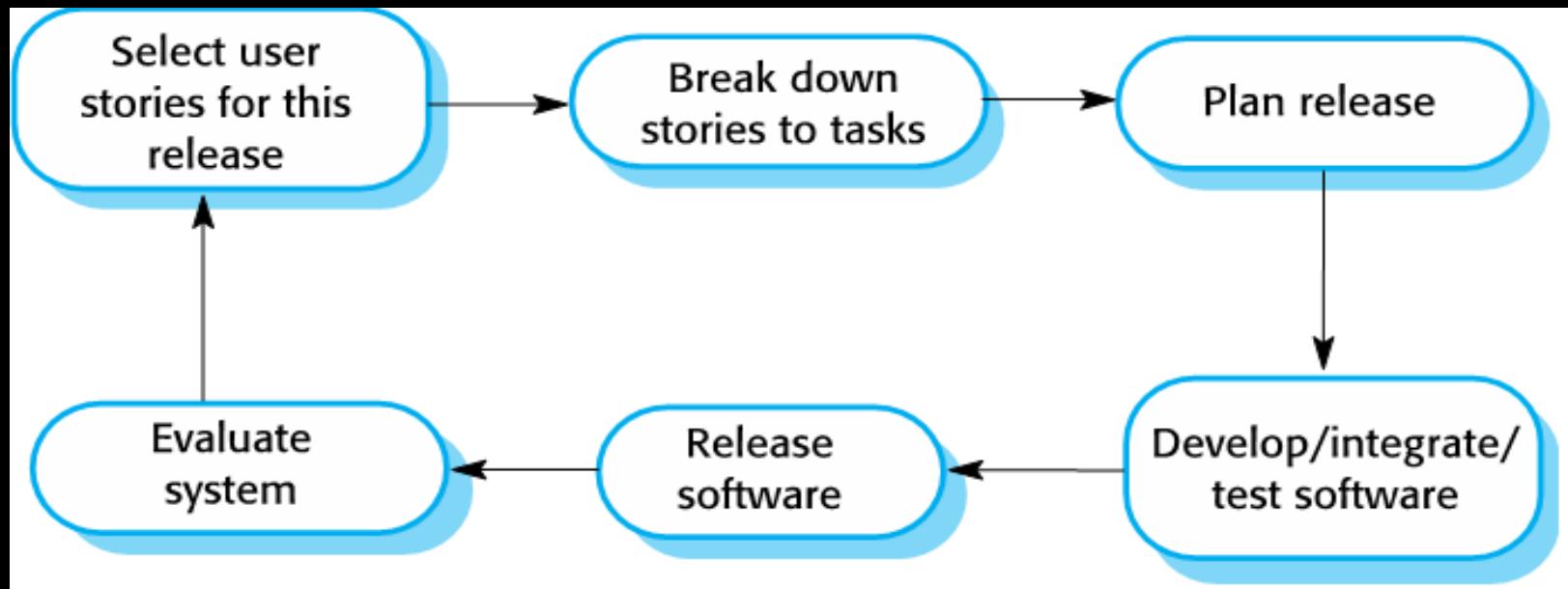
# Extreme Programming(XP)

- The origin of extreme programming (XP) started in 1990s when Kent Black tried to find a better way of doing software development when he was handling a project. One significant difference in its approach is that it focuses on **adaptability rather than on predictability**. The reason behind this approach is that software development is a very **fluid process** where requirements cannot be fully predicted from the beginning but will always change as projects move on. Hence software development needs a methodology that is capable to adapt to changing requirements at any point during the project life.

# Extreme Programming

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

# The Extreme Programming Release Cycle



# Extreme Programming Practices (a)

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

# Extreme Programming Practices (b)

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

# XP and Agile Principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

# Influential XP Practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
- Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.
- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming



That is all

# Agile Software Development

Lecture # 14, 15  
18, 19 Feb

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)

# Software Engineering

CS-303



# Topics Covered

- Agile methods
- Agile development techniques
- Agile project management

# Influential XP Practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
- Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.
- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming

# User Stories for Requirements

- Sitting down with the customers
- make a list of features they would like to see in their software.
- We call these things **user stories** and they become the **To Do list** for project. These tasks are the basis of schedule and cost estimates
- Like any other lists, there always seems to be more to do than time allows.
- Ask the customer to prioritize their list
- Get the most important stuff done first, and save the least important for last.



# A 'prescribing medication' Story

## Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

# Examples of Task Cards for Prescribing Medication

**Task 1: Change dose of prescribed drug**

**Task 2: Formulary selection**

**Task 3: Dose checking**

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.

Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

# Refactoring

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

# Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

# Examples of Refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

# Test-first Development

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

# Test-driven Development

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

# Customer Involvement

- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

# Test-case Description for Dose Checking

## **Test 4: Dose checking**

### **Input:**

1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

### **Tests:**

1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose \* frequency is too high and too low.
4. Test for inputs where single dose \* frequency is in the permitted range.

### **Output:**

OK or error message indicating that the dose is outside the safe range.

# Test Automation

- Test automation means that tests are written as executable components before the task is implemented
  - These testing components should be stand-alone, should simulate the **submission of input** to be tested and should **check** that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- As testing is automated, there is always a set of tests that can be quickly and easily executed
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

# Pair Programming

- In pair programming, programmers sit together at the same computer to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.

# Pair Programming Advantages

- Pair programming involves programmers working in pairs, developing code together.
- The **sharing of knowledge** that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- This helps develop **common ownership** of code and spreads knowledge across the team.
- It serves as an **informal review** process as each line of code is looked at by more than 1 person.
- It encourages **refactoring** as the whole team can benefit from improving the system code.

# Agile Project Management

# Agile Project Management

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is **plan-driven**. Managers draw up a plan for the project showing **what** should be delivered, **when** it should be delivered and **who** will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the practices used in agile methods.

# Scrum

- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.
  - The initial phase is an **outline planning phase** where you establish the general objectives for the project and design the software architecture.
  - This is followed by a **series of sprint cycles**, where each cycle develops an increment of the system.
  - The **project closure phase** wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

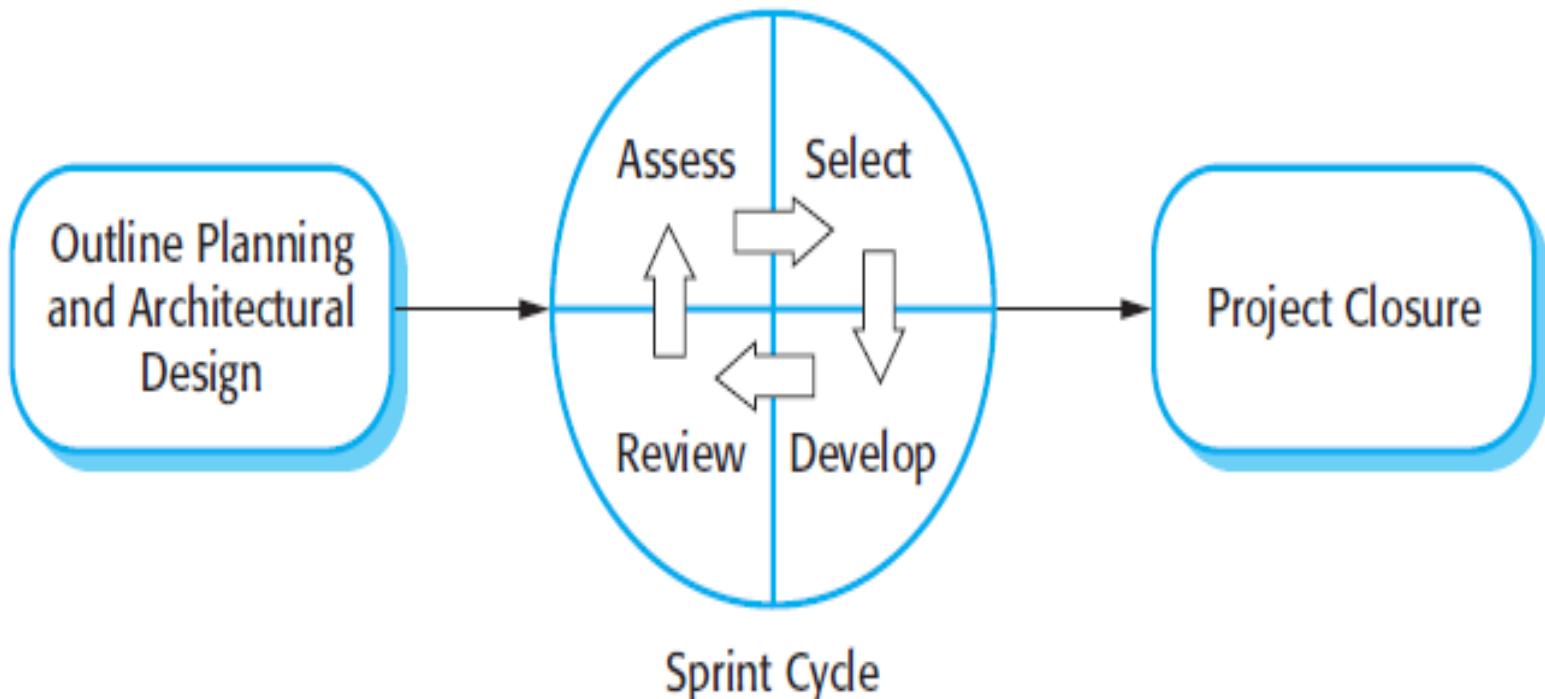
# Scrum Terminology (a)

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

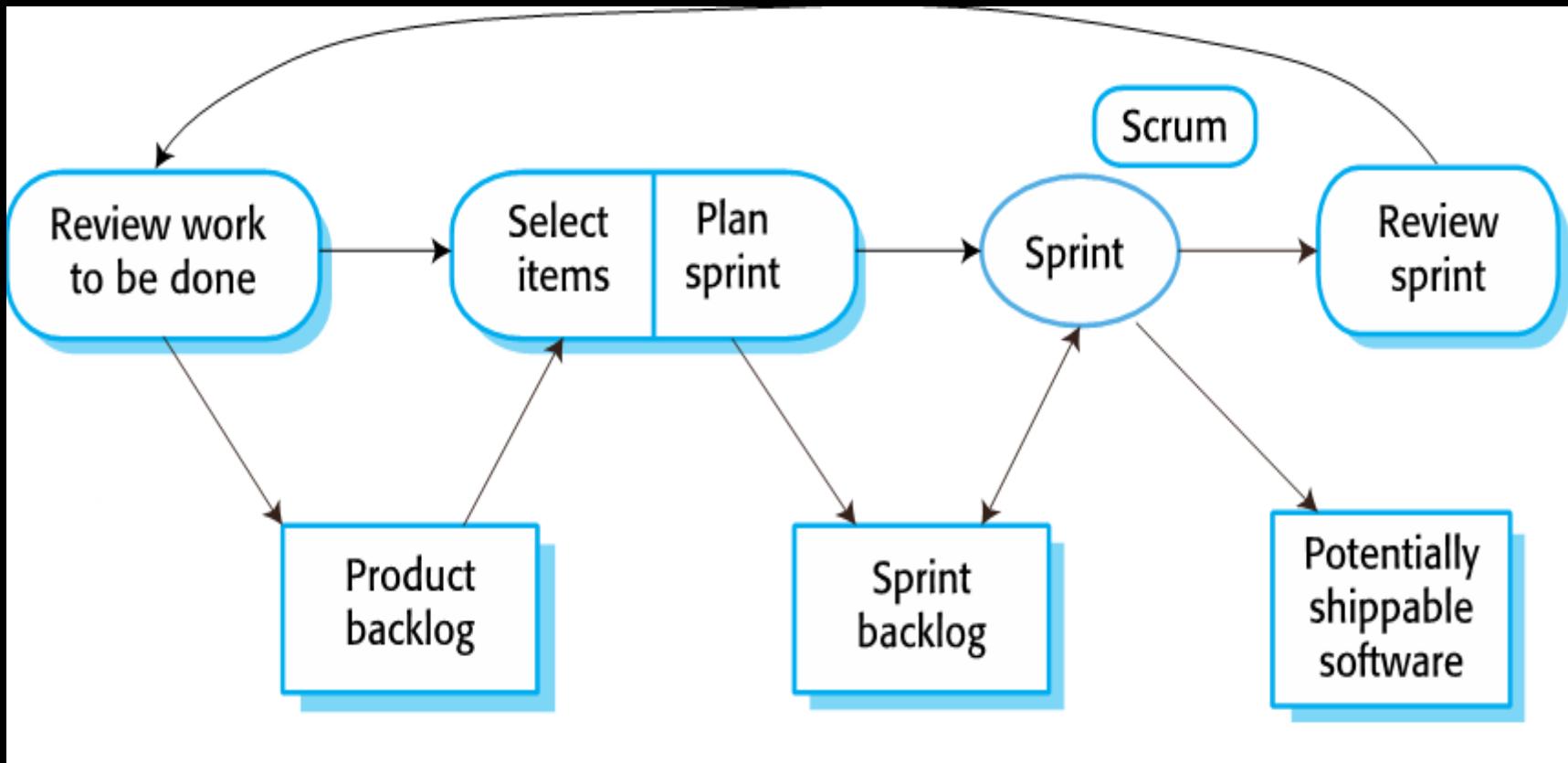
# Scrum Terminology (b)

Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

# The Scrum Process

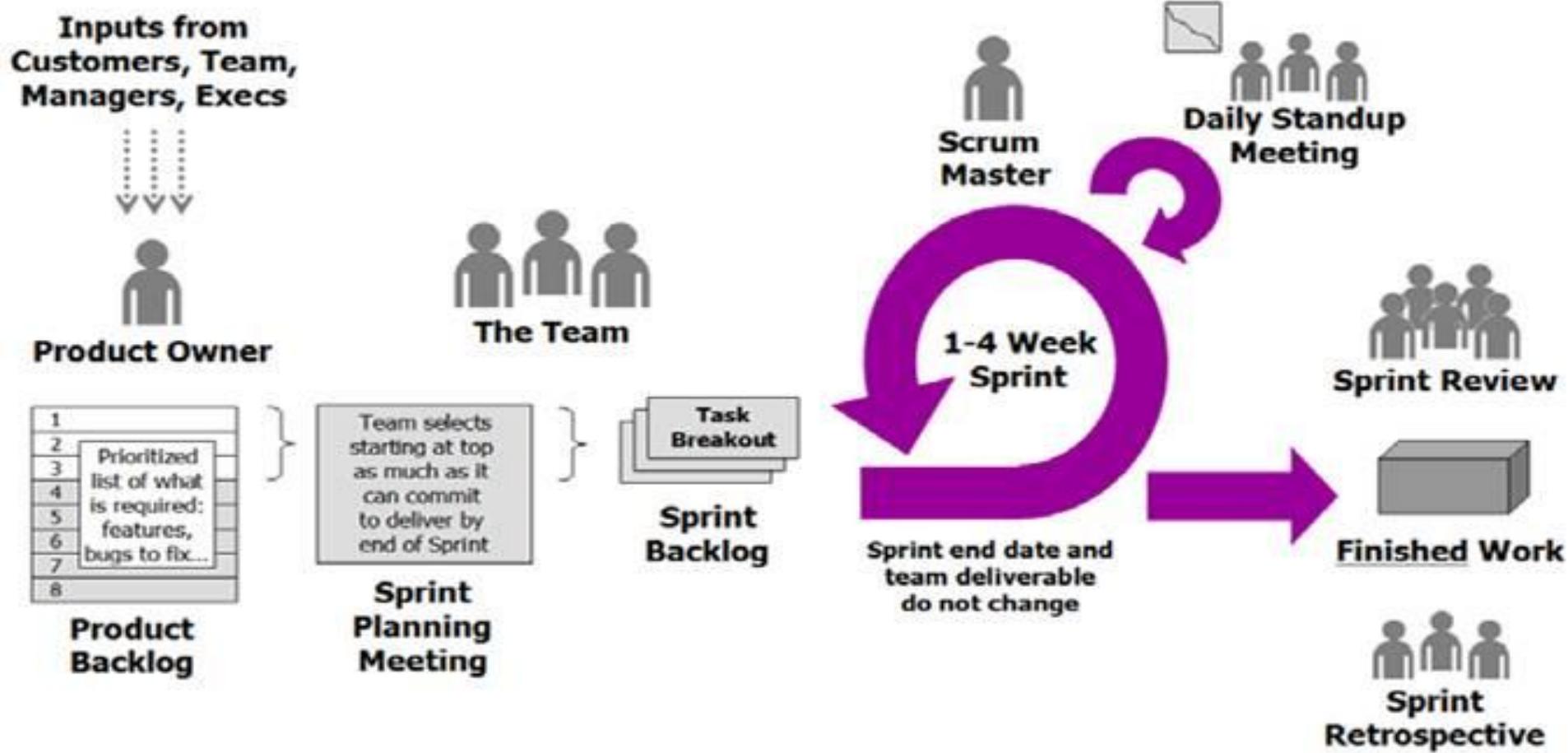


# Scrum Sprint Cycle



# The Scrum Sprint Cycle

- 1-What have you accomplished since yesterday?
- 2-Are your Sprint Backlog estimates accurate?
- 3-What are you working on today?
- 4-Is there anything blocking you?



# The Scrum Sprint Cycle

- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.

# The Sprint Cycle

- Once these are agreed, the team organize themselves to develop the software.
- During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

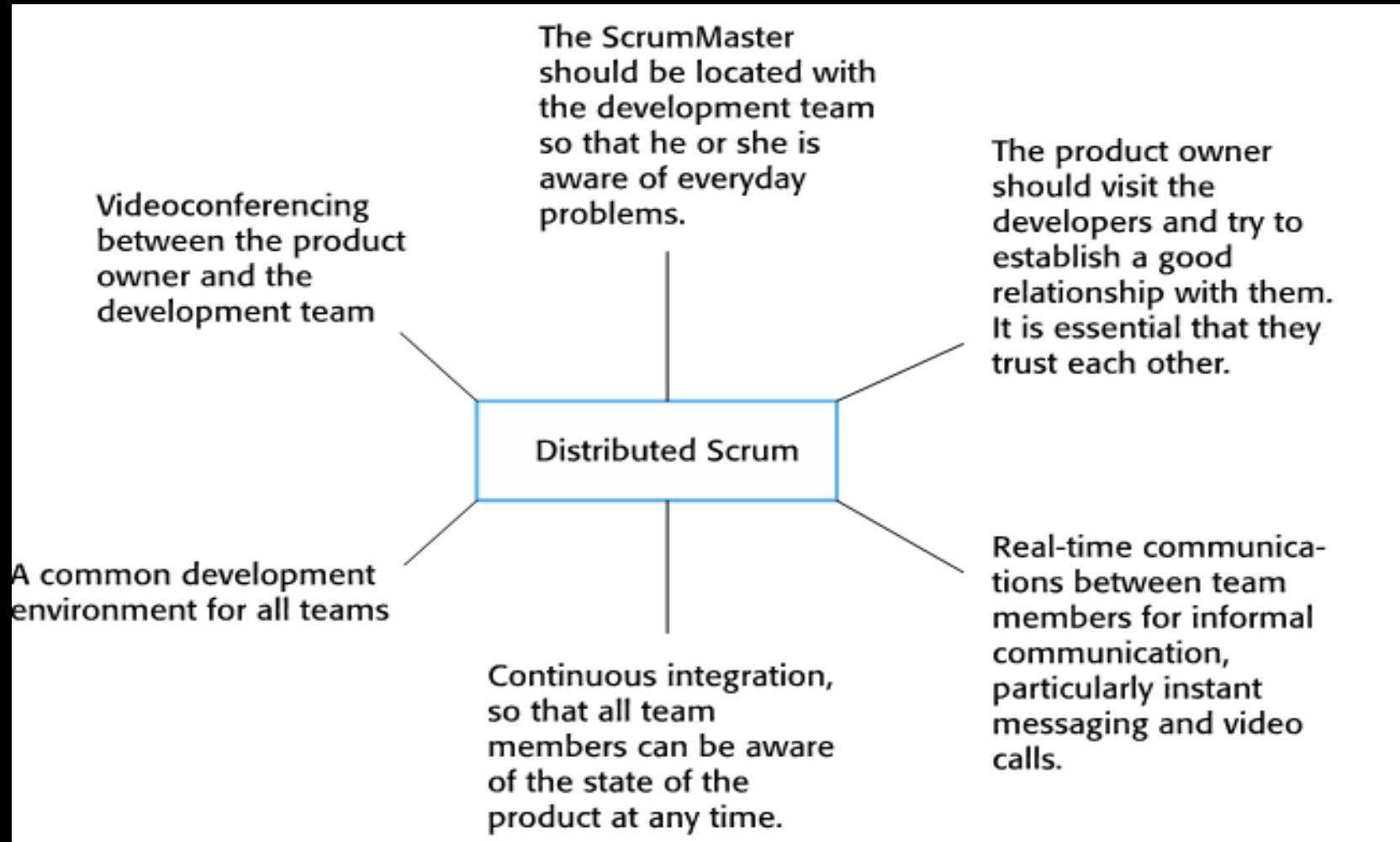
# Teamwork in Scrum

- The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

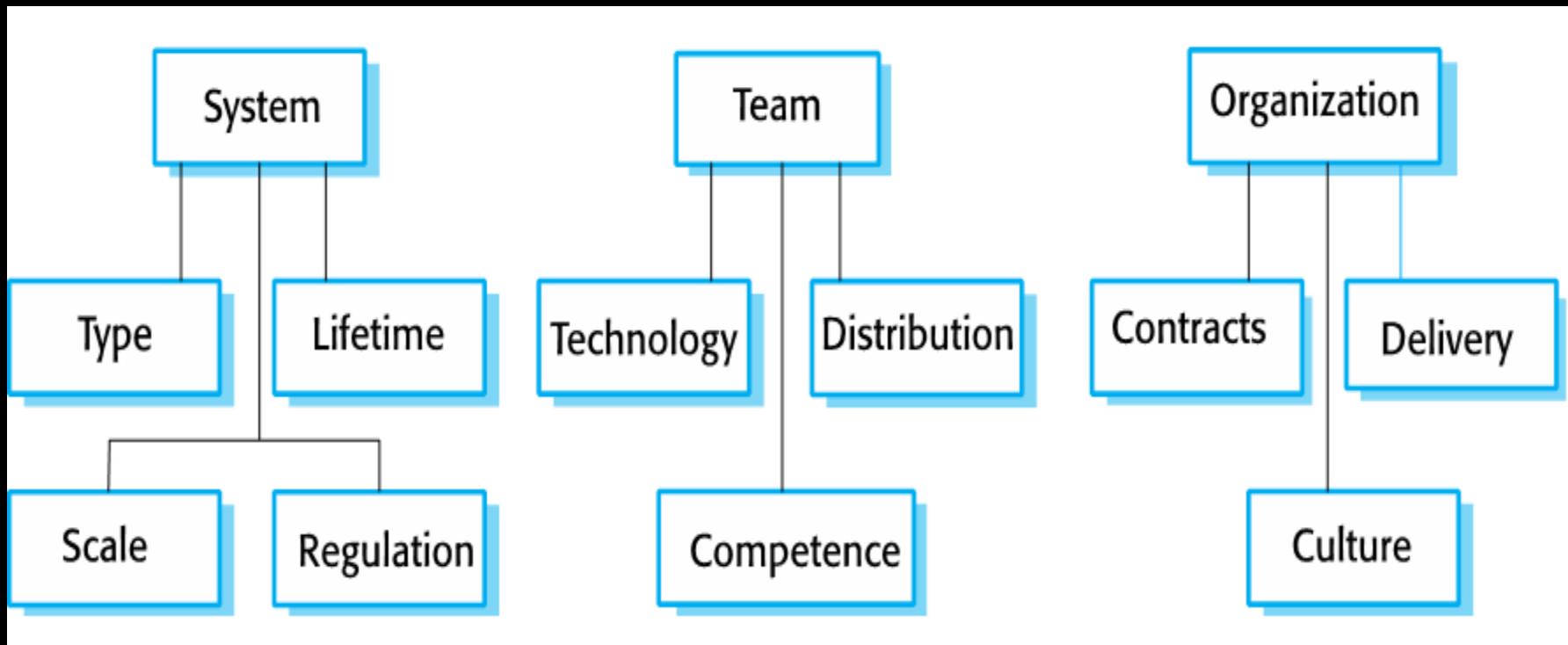
# Scrum Benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

# Distributed Scrum



# Agile and Plan-based Factors





That is all

# Project Management

Lecture # 15, 16  
19, 20 Feb

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)

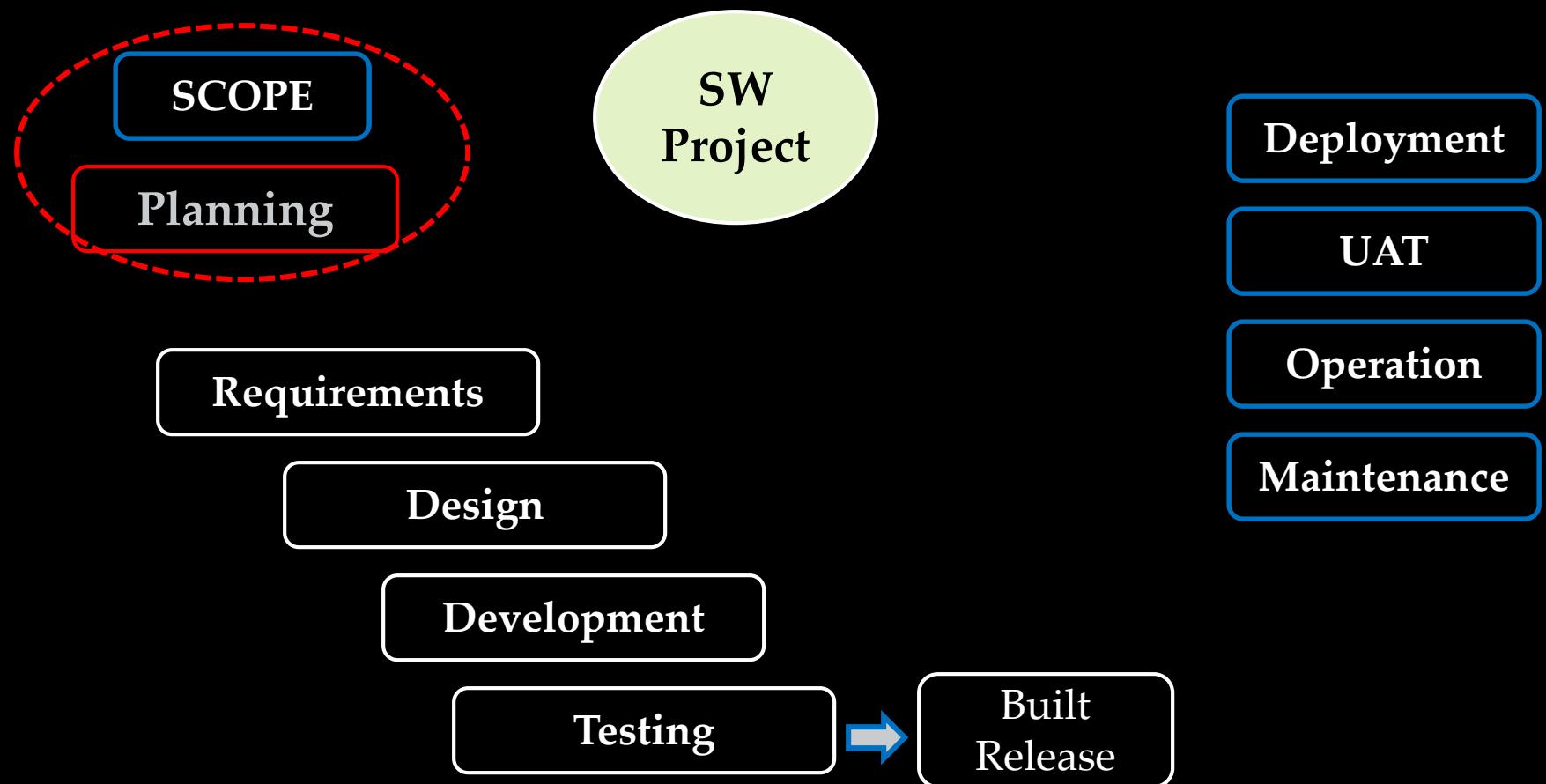
# Intro to Software Engineering SE-110



# Today's Outline

- What is project?
- What is Management?
- Project Management
- Software Project Management(SPM)
- Project Management activities
  - Project planning
  - Project scheduling
  - Risk management

# SDLC



# What is not Project?

- It seems that everything we do is a project.
- A project is different from day-to-day business.
- Ongoing work effort is generally a repetitive process because it follows an organization's existing procedures.
- Any routine work is NOT a project.
  - Coming to university on regular basis is not a project.
  - Monthly/weekly grocery shopping is not a project.
  - Production of cars at automobile factory is not a project.
  - Conducting classes at University is not a project.

# Project Concepts

**It is a project when it meets the following criteria;**

1. Any non-repetitive activity.
2. A temporary endeavor undertaken to create a unique product or service so require resources.
3. Any activity with a start and a finish. (Temporary)
4. A unique set of co-ordinated activities, undertaken by an individual or team to meet specific objectives within defined schedule, cost and performance parameters. *Project should have a primary customer/sponsor.*

# What is Management?

- It is a set of activities undertaken by one or more persons for the purpose of planning & controlling the activities of others in order to achieve an objective or complete the project.



# What is Project Management?

- Project management is “the application of knowledge, skills, tools and techniques to project activities to meet the project requirements”.
- PM is the one who is responsible for establishing a communication in between the project team and the user.
- Project Manager and his/her team should collectively possess the necessary and requisite inter-personal and technical skills to facilitate control over the various activities within the project.

# Project Manager Role

- **A Good Project Manager**

- **Takes ownership of the whole project**
- **Is proactive not reactive**
- **Adequately plans the project**
- **Is Authoritative (NOT Authoritarian)**
- **Is Decisive**
- **Is a Good Communicator**
- **Manages by data and facts not unformed optimism**
- **Leads by example**
- **Has sound Judgement**
- **Is a Motivator**
- **Is Diplomatic**



# Software Project Issues

- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.
- Almost 70% of software organization have no defined engineering methods
- Software ends up
  - Late
  - Over budget
  - Fails to meet requirements
  - In a 1998 survey, 26% of software projects failed outright, 46% experienced cost and schedule overruns
- The major reason of these software projects are not technical problems, but management problems”

# Project Management....



**Work Smart Not Hard !!!**

# Software Management Distinctions

- The product is intangible.
- The product is uniquely flexible.
- Software engineering is not recognized as an engineering discipline with the same status as mechanical, electrical engineering, etc. The software development process is not standardised.
- Many software projects are 'one-off' projects.

# 4 P'S OF PROJECT MANAGEMENT

People



Product



Process



Project

# The Management Spectrum

- Effective software project management focuses on these items (in this order)
  - The people
    - Deals with the cultivation of motivated, highly skilled people
    - Consists of the stakeholders, the team leaders, and the software team
  - The product
    - Product objectives and scope should be established before a project can be planned
  - The process
    - The software process provides the framework from which a comprehensive plan for software development can be established
  - The project
    - Planning and controlling a software project is done for one primary reason

# Project Staffing

- May not be possible to appoint the ideal people to work on a project
  - Project budget may not allow for the use of highly-paid staff;
  - Staff with the appropriate experience may not be available;
  - An organisation may wish to develop employee skills on a software project.
- Managers have to work within these constraints especially when there are shortages of trained staff.

# What is planning?



Planning is the process of stating objectives and then determining the most effective activities or accomplishments necessary to reach the objectives



# Who makes the plans?



Everybody must plan

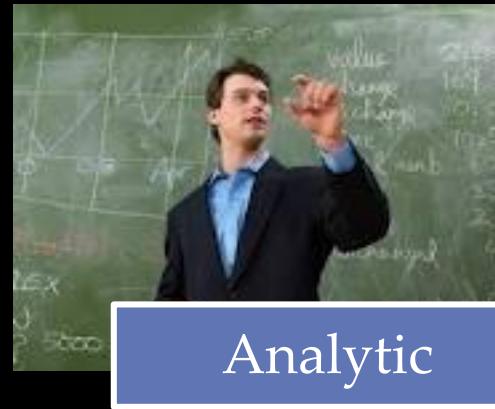


Project manager initiate the planning process



Project manager coordinates planning activities into the overall project master plan

# Characteristics of a project planner



# Planning Process



Devising and maintaining a workable scheme to accomplish the business need that the project undertaken was to address

# Project Planning

- Adequate planning leads to the correct completion of work



# Planning

- Inadequate planning leads to frustration towards the end of the project & poor project performance



Project Start



Project End

# Project Planning

- Probably the most time-consuming project management activity.
- Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
- Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

# Project Planning Process

Establish the project constraints

Make initial assessments of the project parameters

Define project milestones and deliverables

while project has not been completed or cancelled loop

    Draw up project schedule

    Initiate activities according to schedule

    Wait ( for a while )

    Review project progress

    Revise estimates of project parameters

    Update the project schedule

    Re-negotiate project constraints and deliverables

    if ( problems arise ) then

        Initiate technical review and possible revision

    end if

end loop

# Types of Project Plan

Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project. See Chapter 27.
Validation plan	Describes the approach, resources and schedule used for system validation. See Chapter 22.
Configuration management plan	Describes the configuration management procedures and structures to be used. See Chapter 29.
Maintenance plan	Predicts the maintenance requirements of the system, maintenance costs and effort required. See Chapter 21.
Staff development plan.	Describes how the skills and experience of the project team members will be developed. See Chapter 25.

# The Software Development Plan /Project Plan

- This is usually what is meant by a project plan.
- Specifies the order of work to be carried out, resources, responsibilities, and so on.
- The project plan sets out:
  - The work breakdown;
  - The resources available to the project;
  - A schedule for the work.

# Project Plan Structure

- Introduction.
- Project organisation.
- Risk analysis.
- Hardware and software resource requirements.
- Work breakdown.
- Project schedule.
- Monitoring and reporting mechanisms.

# Work Breakdown Structure (WBS)

- The Work Breakdown Structure is the foundation for effective project planning, costing and management.
  - It is the most important aspect in setting-up a Project
  - A Work Breakdown Structure (WBS) is a hierarchical (from general to specific) tree structure of deliverables and tasks that need to be performed to complete a project.”
- It is the foundation on which everything else builds



# 5 reasons to create a Work Breakdown Structure

- It helps with correct project organization.
- Assists in describing the project scope to stakeholders.
- Helps to distribute responsibilities.
- Shows the project's milestones and all the points to control.
- Allows estimating costs, risks and time correctly.

# WBS

- There are many ways of breaking down the activities in a project, but the most usual is into:
  - work packages;
  - tasks;
  - deliverables;
  - milestones.

# WBS

- A work package is a large, logically distinct section of work:
  - typically at least 12 months duration;
  - may include multiple concurrent activities;
  - independent of other activities;
  - but may depend on, or feed into other activities;
  - typically allocated to a single team.
- A task is typically a much smaller piece of work: A part of a work package.
- typically 3–6 person months effort;
- may be dependent on other concurrent activities;
- typically allocated to a single person.

# WBS

- A deliverable is an output of the project that can meaningfully be assessed.
- Examples:
  - a report (e.g., requirements spec);
  - code (e.g., alpha tested product).
- Deliverables are indicators (but only indicators) of progress.
- A milestone is a point at which progress on the project may be assessed.
- Typically a major turning point in the project.
- EXAMPLES:
  - delivery of requirements spec;
  - delivery of alpha tested code.

# WBS

- Usually. . .
- work packages are numbered WP1, WP2, . . . ;
- tasks are numbered T1.1, T1.2, etc,
  - the first number is the number of the workpackage;
  - the second is a sequence number.
- deliverables are numbered D1.1, D1.2, etc
- milestones are numbered M1, M2 etc.

# Steps to build a WBS

- **Begin with the Charter, focusing on Objectives and Deliverables**
- **Break the main product(s) down into sub-products**
- **Set the structure to match how you'll manage the project**
- **Lowest level not too detailed, not too large**
- **Is there a need for Integration?**
- **Identify support activities**
- **Check for completeness - is all the effort included?**
- **Develop a coding structure if needed**
- **Assign work package managers**

# Displaying the WBS

## Example of outlined WBS.

Project Name	Task 1	Subtask 1.1	Work Package 1.1.1
		Subtask 1.2	Workpackage 1.2.1
			Workpackage 1.2.2
	Task 2	Subtask 2.1	Workpackage 2.1.1
			Workpackage 2.1.2

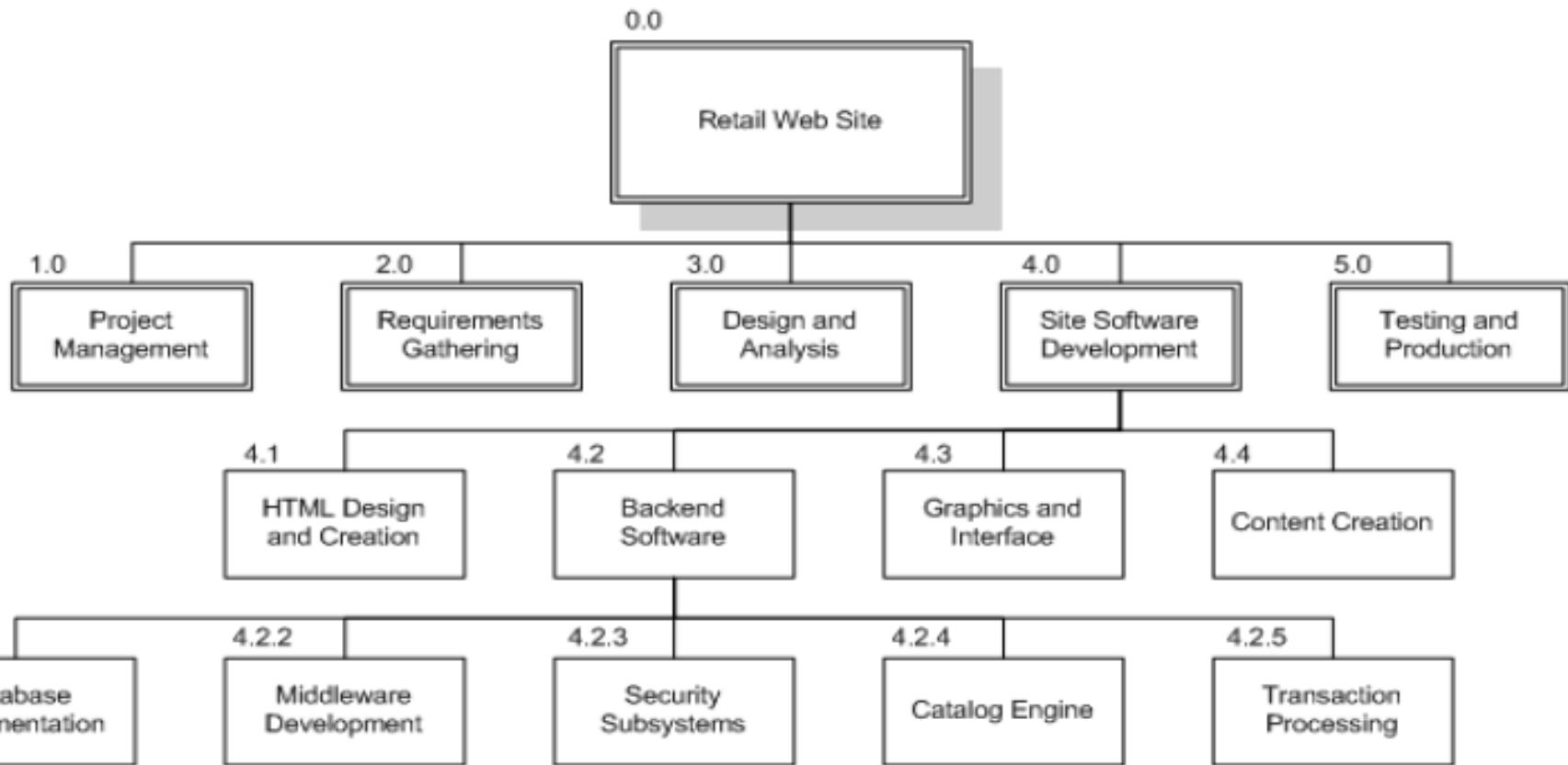
# Displaying the WBS

# Example of outlined WBS.

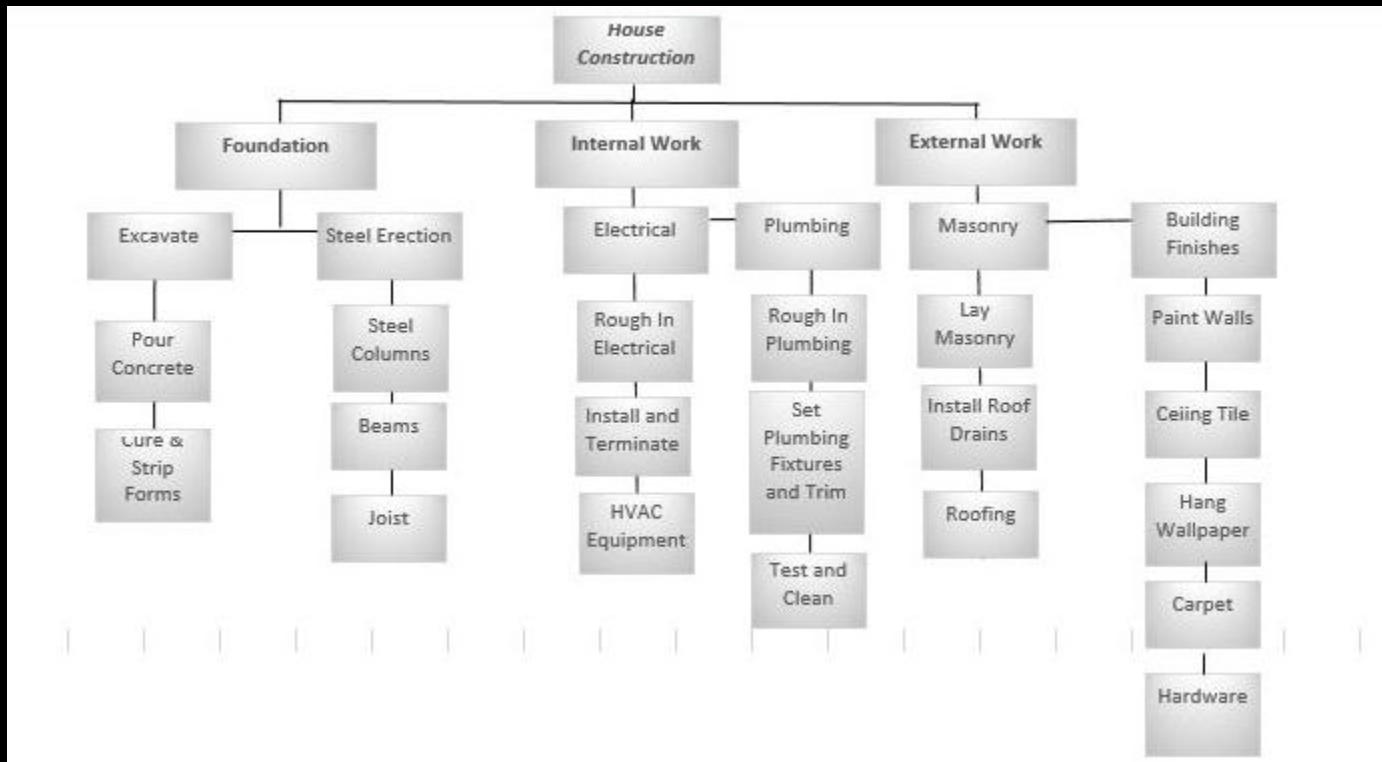
- 0.0 Retail Web Site
- 1.0 Project Management
- 2.0 Requirements Gathering
- 3.0 Analysis & Design
- 4.0 Site Software Development
  - 4.1 HTML Design and Creation
  - 4.2 Backend Software
    - 4.2.1 Database Implementation
    - 4.2.2 Middleware Development
    - 4.2.3 Security Subsystems
    - 4.2.4 Catalog Engine
    - 4.2.5 Transaction Processing
  - 4.3 Graphics and Interface
  - 4.4 Content Creation
- 5.0 Testing and Production

# Displaying the WBS

## Example of Chart WBS.



# WBS– House Construction



# Example WBS

- **Redecorate Room**
  - **Prepare materials**
    - Buy paint
    - Buy a ladder
    - Buy brushes/rollers
    - Buy wallpaper remover
  - **Prepare room**
    - Remove old wallpaper
    - Remove detachable decorations
    - Cover floor with old newspapers
    - Cover electrical outlets/switches with tape
    - Cover furniture with sheets
  - **Paint the room**
  - **Clean up the room**
    - Dispose or store left over paint
    - Clean brushes/rollers
    - Dispose of old newspapers
    - Remove covers



That is all