# What Is Engineering?

- **Engineering** is the discipline, art and profession of acquiring and applying scientific, mathematical, economic, social, and practical knowledge to design and build structures, machines, devices, systems, materials and processes that safely realize solutions to the needs of society.
- The American Engineers' Council for Professional Development has defined "engineering" as:

  The creative application of *scientific principles* to *design* or *develop* structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all in the context of an intended function, economics of operation and safety to life and property

# What Is Software?

- **Computer software**, or just **software**, is the collection of computer programs and related data that provide the instructions telling a computer what to do.
- We can also say software refers to one or more computer programs and data held in the storage of the computer for some purposes. Program software performs the function of the program it implements, either by directly providing instructions to the computer hardware or by serving as input to another piece of software.

  NOTE: The term was coined to contrast to the old term hardware (meaning physical devices). In contrast to hardware, software is **intangible**, meaning it "cannot be touched".
- Software is also sometimes used in a more narrow sense, meaning application software only. Sometimes the term includes data that has not traditionally been associated with computers, such as film, tapes, and records

# What Is Software Engineering? (1/3)

- **Software engineering** (**SE**) is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build.
- It is a "systematic approach to the analysis, design, assessment, implementation, test, maintenance and re-engineering of a software by applying engineering to the software".

  NOTE: The term *software engineering* first appeared in the 1968 NATO Software Engineering Conference, and was meant to provoke thought regarding the perceived "software crisis" at the time.

# What Is Software Engineering? (2/3)

- Since the field is still relatively young compared to its sister fields of engineering, there is still much debate around what *software engineering* actually is, and if it conforms to the classical definition of engineering.
- The IEEE Computer Society's *Software Engineering Body of Knowledge* defines "software engineering" as

  “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software”.
- It is the application of Engineering to software because it integrates significant mathematics, computer science and practices whose origins are in Engineering.

# What Is Software Engineering? (3/3)

- *Software development*, a much used and more generic term, *does not necessarily subsume* the engineering paradigm.
- Objective of Software Engineering (SE):
  - To "systematically" develop "high quality" software in "timely and cost effective" manner
- Basic knowledge for SE comes from various engineering disciplines
- SE methodologies, techniques and guidelines have been evolved by systematically organizing:
  - Past innovations in software development
  - Past experience in software development

# Why Software Engineering? (1/3)

- Computers have been in use for commercial purposes for last 60+ years.
- Early software used exploratory programming styles in
  - Assembly language programming
    - Small programs
    - Written by one programmer
    - Programs lacked sophistication
    - Every programmer had own individualistic style
  - Higher-level language programming
    - Helped programmers to write larger programs
    - Considerably reduced software development effort

# Why Software Engineering? (2/3)

- With time, the computers are becoming:
  - Faster
  - More capable
  - Easy to use
  - Cheaper
- Sophistication in computers has resulted in demand for large and complex software.
- Exploratory style of software development are
  - Adequate only for small tasks
  - Inadequate to cope with the changing scenario

# Why Software Engineering? (3/3)

- Also, the software needs to be:
  - Developed within the specified time
  - In a cost effective manner
  - Amenable for maintenance by someone other than the author
- All this requires a systematic approach to developing software i.e. software has to be engineered.
- Systematic and engineering like approach to software development is inevitable for developing large and complex software

  NOTE: It involves use of techniques like systems analysis, estimation, prototyping, designing, testing, etc.

# Top Five Causes of Project Failure

1. Lack of attention to human and organizational factors.
2. Poor project management.
3. Poor articulation of user requirements.
4. Inadequate attention to business needs and goals.
5. Failure to involve users appropriately.

# Software Development Practices - Changes (1/2)

- More focus is on requirements specification
- Design has become a distinct phase between the requirements and the coding
- Coding has become small part of the project (from being the dominant part of project)
- Focus has shifted from error detection (and correction) to error prevention
- Reviews are carried out after each phase

# Software Development Practices - Changes (2/2)

- Testing has become more systematic e.g. test cases are developed right after the requirements are identified
- Improved documentation has resulted in:
  - Better visibility thru consistent use of standards
  - Smoother fault diagnosis and maintenance
- Use of Computer Aided Software Engineering (CASE) tools has increased

# Software Development Methodologies (1/2)

- Software Development Methodologies specify:
  - How to do the job?
    - e.g. analysis, design, coding, testing
  - What outputs to produced?
  - How to check for entry/exit criteria?
  - How to measure progress?
- Typical phases associated in software project life cycle:
  - Feasibility (pre-development)
  - Requirements
  - Design

# Software Development Methodologies (2/2)

- Typical phases associated in software project life cycle (Continued):
  - Coding and Unit testing
  - Testing
  - Maintenance (post-development)
- Software development organizations expect their employees to master their development methodologies (before being assigned to a job)

# Software Project Phases

- Pre-development Phase
  - Feasibility Study
  - Technical Proposals
- Development Phase
  - Requirements Study
  - Designing
  - Implementation - Coding and unit testing
  - Integration and System testing
- Post-Development Phase
  - Maintenance

# Why Software Project Phases?(1/2)

- It breaks the overall problem of developing required software, into performing (successfully) a set of phases, each handling a different concern of software development.
- It ensures that the cost of development is lower than what it would have been if the whole problem was tackled together.
- A phased process allows proper checking for quality and progress at some defined points during development (end of phases).

# Why Software Project Phases?(2/2)

- Thus having the development process in phases aids in
  - Managing the complexity
  - Project tracking
  - Ensuring quality.
- The development process in phases increases the probability of developing the software in a timely and cost effective manner.

# Feasibility Study (1/2)

- Objective of feasibility phase:
  - Determine whether developing the software is FINANCIALLY and TECHNICALLY feasible
- Involves collection of data pertaining to:
  - Inputs to the system
  - Processing to be carried out
  - Outputs from the system
  - Constraints to be adhered to

# Feasibility Study (2/2)

- Analyzing the data to arrive at:
  - Abstract definition of the problem
  - Formulation of different solution strategies
  - Examination of alternative solutions strategies i.e. benefits, resources input, cost, time, development related issues
  - Performing cost/benefit analysis to determine best solution under current circumstances

NOTE: Feasibility study may lead to a decision of not pursuing the project further

# Proposals (1/2)

- Depending on the findings of the feasibility study, a proposal is prepared and submitted for carrying out the project.
- The proposal be based on
  - An idea of the broad project requirements
  - An idea of the capability to deliver the project
  - An estimate of  the time and effort
  - An estimate of the cost and pricing of the project.

# Proposals (2/2)

NOTE: It is a good idea to upfront state all the assumptions made in making the proposal. Also dependencies and expectations of and from both the parties should be clearly stated.

- Based on the acceptability of the proposal, contracts are usually signed, which usually marks the beginning of the development phase of a project.

# Requirements Study (1/8)

- Objective:
  - To establish user requirements
- Main activities:
  - Requirements analysis to understand the exact requirements of the customer
  - Requirements specification to properly document the requirements
- Outputs at the end of this phase:
  - System Requirements Specifications (SRS) document

# Requirements Study (2/8)

- Outputs at the end of this phase (continued):
  - System Requirements Specifications (SRS) document
  - User manual
  - System Test Plan
  - Requirements traceability matrix
- Responsibility for this phase:
  - Requirements analysis team

# Requirements Study (3/8)

REQUIREMENTS ANALYSIS

- Purpose:
  - To clearly understand the requirements
- Involves collecting all relevant data (about the product) from the users
- Data collection techniques include:
  - Interviews
  - Discussions
  - Questionnaires

# Requirements Study (4/8)

- Caution:
  - Data collected from different users may contain ambiguities and contradiction

  NOTE: Each user typically has only a partial and incomplete view of the system
- Ambiguities and contradictions need to be identified and resolved (by further discussions) i.e. remove inconsistencies and overcome incompleteness

# Requirements Study (5/8)

REQUIREMENTS SPECIFICATIONS

- Involves systematic documentation of the user requirements into a SRS document
- SRS document concentrates only on the “external behavior” of the system i.e. only on “WHAT the system is suppose to do?” and not on “HOW it is suppose to do it?”

# Requirements Study (6/8)

- SRS document addresses:
  - Functional requirements
  - Performance requirements
  - Formats of the inputs and outputs
  - Design Constraints (due to political, economic, environmental and security reasons)

NOTE : It is advisable that besides the required functionality all the factors that may affect the proper functioning of the system is specified in the requirement document.

# Requirements Study (7/8)

- SRS establishes the basis for agreement between the client and the developing organization on what the software product will do.
- SRS document serves as a contract between the customer and the developer
- SRS provides a reference for validation of the final product

# Requirements Study (8/8)

- SRS document must be:
  - Written in a language that user can understand
  - Thoroughly understood by both the customer as well as the developer
  - Must be reviewed
  - Kept current

# Designing (1/4)

- Objective:
  - To transform requirements specifications into a structure that is -> suitable for implementation in some programming language i.e. to propose a software architecture that meets the software requirements
- Design approaches:
  - (Traditional) structured design approach
  - Object Oriented approach
- Design documents need to be reviewed (before proceeding to the implementation phase)
- Responsible person: System Designer

# Designing (2/4)

- The design of the system has a major impact on the later phases like testing and maintenance and is perhaps the most critical factor affecting the quality of the software.
- The output of this phase is the design document, similar to the blueprint or plan for the solution.
- The design activity is often divided into two sub-phases viz.
  - System design (top-level design)
  - Detailed design (low-level design)

# Designing (3/4)

STRUCTURED DESIGN

- Distinct activities:
  - Carry out structured analysis on the requirements specification
  - Evolve structured design (also called as software architecture)
- Components of software design:
  - Architectural (high level) design depicts:
    - System modules
    - Invocation relationships between modules
  - Detailed (low level) design depicts:
    - Detailed description of each module e.g. data structures, algorithms

# Designing (4/4)

## OBJECT ORIENTED DESIGN

- Involves building an object structure:
  - Identifying the objects in software domain
  - Identifying the relationships between these objects
- Object structure is then further refined to obtain detailed design
- Advantages of object oriented approach:
  - Less development effort
  - Less development time
  - Better maintainability

NOTE: Several well known methodologies are available for undertaking high level design as well as low level design

# Coding & Unit Testing(1/2)

- Purpose of implementation phase:
  - Implement the software design into code (using some programming language)
  - A well-written code reduces the testing and maintenance effort.
  - Focus should be on developing programs that are easy to read and understand.
- Activities involved (for each module):
  - Coding

    Purpose of coding:
    - To translate program specification into source code

# Coding & Unit Testing(2/2)

- Activities involved (for each module) (continued):
  - Unit testing (as independent standalone module)

    Purpose of unit testing:
    - To confirm that each module works correctly as a standalone module
  - Documentation
- Coding standards address aspects like:
  - Layouts
  - Comments
  - Naming convention
  - Guidelines for lines per module
- End product:
  - Set of independently tested modules
- Responsible person: Developer/Programmer

# Testing (1/2)

- The testing phase can be further split into the following sub-phases

I. Integration testing

Purpose of integration testing:

- To confirm that the interfaces between module works correctly i.e. the modules work well when the are combined together to build up the system.

II. System testing

Purpose of system testing:

- To confirm that the entire software system works as desired and that all the desired functionality is present in the system that has been built.
- Usually done with the help of the system test plan.

# Testing (2/2)

## III. Acceptance testing

Purpose of acceptance testing:

- To confirm that the software system built satisfies the user defined acceptance criteria that would establish that the built software is acceptable by the client.
- This may be done at the client site prior to actual handover of the software system.

# Integration Testing

- Modules are integrated in planned manner
- During each integration step, partially integrated system is tested
- When ALL modules are integrated (and tested), the system is ready for system testing
- Types of Integration testing
  - Top-down integration
  - Bottom-up integration
  - Regression Testing
  - Smoke Testing

# Validation Testing (1/2)

- Objective of Validation testing
  - To confirm that the developed system meets the requirements (as specified in SRS document in the section called validation criteria)
- Both plan and procedure are designed to ensure that
  - all functional requirements are satisfied.
  - all behavioral characteristics are achieved
  - all performance requirements are attained
  - documentation is correct
  - Human-engineered and other requirements are met (e.g. transportability, compatibility error recovery, maintainability)

# Validation Testing (2/2)

- Types of validation testing
  - Configuration review
  - Alpha testing
  - Beta testing

# System Testing(1/2)

- Objective of System Testing
  - To confirm that the developed system meets the requirements (as specified in SRS document)
- Carried out according to the system test plan

  NOTE: System test plan is prepared during requirements phase (include test cases and expected results)
- Types of System testing
  - Recovery testing
  - Security testing
  - Stress testing
  - Performance testing

# System Testing (2/2)

- Final output of the test phase: Test report and Tested System

# Acceptance Testing

- Objective of Acceptance Testing
  - To confirm that the developed system is acceptable by the client as meeting his/her requirements (as specified through the acceptance criteria)
- Carried out according to the Acceptance test plan using the acceptance criteria specified or agreed upon by the client

  NOTE: Acceptance criteria has also to be signed off by the client
- Final output of the test phase: The client accepts the product and pays up!!!

# Maintenance Phase

- Maintenance phase involves performing one or more of the following activities:
  1. Corrective maintenance:
  - Fixing errors not detected during the development phase
  2. Adaptive maintenance:
  - Improving implemented system e.g. performance
  - Enhancing functionalities e.g. new requirements
  - Porting software to new environment (as needed)
- Maintenance phase consumes far more effort than the development phases (40:60)

# Project Management Process

- Proper management of the project is an integral part of software development and it involves the planning, monitoring and control of the people, process and events that occur as software evolves from a preliminary concept to an operational implementation.
- To meet the cost, quality and schedule objectives
  - resources have to be properly allocated to each activity of the project
  - progress of different activities has to be monitored
  - corrective actions have to be taken, if needed.

# Project Management Process

- The activities in project management can be broadly grouped into three categories
  - Planning
  - Monitoring and control (aka project tracking and control)
  - Termination analysis

# Who Does It?

- Everyone “manages” to some extent but the scope of management activities varies with the person doing it i.e.
  - A software engineer manage their day-to-day activities, planning, monitoring and controlling technical tasks.
  - Project managers plan, monitor and control the work of a team of software engineers.
  - Senior managers coordinate the interface between the business and the software professionals.

# Project Planning (1/2)

- Project management begins with planning, which is perhaps the single largest responsibility of the project management.
- The goal of this phase is to develop a *plan* for software development following which the objectives of the project can be met *successfully* and *efficiently*.
- The project plan provides the fundamental basis for project management.

# Project Planning (2/2)

- The activities of project planning are:
  - Cost estimation
  - Schedule and milestone determination
  - Project staffing
  - Quality control
  - Monitoring and Control
- The work product produced as a result of project planning is the ***Project Plan***

# The Project Plan

- The project plan is usually produced before the development activity begins and is **updated** as development proceeds and data about the progress of the project becomes available.
- The plan
  - Defines the process and tasks to be conducted
  - The people who will do the work
  - The mechanisms of assessing risks, controlling change and evaluating quality.

# Project Team Assignment

Prepare a project plan for your respective projects. Identify the milestones and the deliverables associated with those milestones.

Submit by: 19th January 2012 12:00 hrs.

# Cost Estimation (1/2)

- For a given set of requirements it is desirable to know or estimate
  - How much it will cost to develop the software to satisfy the given requirements?
  - How much time the development will take?
- These estimates are needed
  - For bidding for software projects
  - To enable the client or developer to perform the cost benefit analysis

# Cost Estimation (2/2)

- These estimates are needed (Continued)
  - For determining the staffing level for a project during different phases
  - For project monitoring and control

    NOTE: For a software development project, detailed and accurate cost and schedule estimates are essential prerequisites for managing the project.

# Cost of a Project (1/2)

- Cost in a project is due to the requirements for
  - Software
    - Tools
    - Compilers etc.
  - Hardware
    - Computer time
    - Terminal time
    - Memory
    - Dedicated hardware boards etc.
  - Human Resources

# Cost of a Project (2/2)

- Cost is often determined in terms of effort that is expressed in person months.

  NOTE:

  1. Most cost estimation procedures focus on cost of the human resources needed since it is the bulk of the cost.
  2. By properly including “overheads” (cost of hardware, software, office space, etc.) in the cost per person month, most costs for a project can be incorporated by using person month as the basic measure

# Cost Estimation – Problems (1/2)

- The accuracy of the estimate depends on
  - The amount of *reliable* information available about the final product.

  NOTE: The cost estimations done with uncertainty about the actual specifications of the final product can be off by a factor of four from the actual cost.
- The accuracy of the cost estimates to the actual cost will depend on the effectiveness and accuracy of the cost estimation procedure (models) and the process (how predictable it is).

# Cost Estimation – Problems (2/2)

NOTE: Achieving a cost estimate after the requirements have been specified within 20% of the actual cost incurred is termed as a good cost estimate.

# Problem Size and Program Size (1/2)

- In context of software development, words “problem size” and “program size” are often used interchangeably
- What is meant by program size?
  - It is an indicator of the amount of effort and time required to develop the program
  - It indicates the “development complexity”

  NOTE: It is not:
  - Number of bytes of source code
  - Number of bytes of executable code

# Problem Size and Program Size (2/2)

- Estimating problem size is fundamental to estimating effort, cost and duration for a planned project
- Several metrics are available for measuring problem size:
  - LOC (Lines Of Code)
  - FP (Function Point)

# Parameters Affecting Cost

- The cost of a project is a function of many parameters
  - Size of the project – primary factor controlling the cost
  - Programmer ability
  - Experience of the developers in the area
  - Complexity of the project
  - Reliability requirements

  NOTE: The goal of a cost model is to determine which of these many parameters have a “significant” effect on the cost and then establish the relationships between the cost and these parameters.

# Project Estimation Techniques (1/2)

- During project planning, one needs to estimate these parameters:
  - Project size
  - Effort required to develop software
  - Project cost
  - Project duration

Project estimation techniques:

- Empirical estimation techniques
  - Use educated guess based on past experience e.g. Expert judgement; Delphi

# Project Estimation Techniques (2/2)

Project estimation techniques (Continued):

- Heuristic estimation techniques
  - Assume that project parameters can be modeled using mathematical expressions e.g. Constructive Cost Model (COCOMO), Function Point (FP)
- Analytical estimation techniques
  - Are based on certain basic assumptions regarding the project e.g. Halstead's metric

# Reading Assignment

Find out the details about the *Delphi* estimation technique and the *Halstead's metric* for project estimation.

Complete by: 26th January 2012 17:00 hrs.

# Constructive Cost Model (COCOMO) (1/2)

- Resource estimates can depend on many different factors (multivariable models).
- The COCOMO model starts with an initial estimate determined by using the static single variable model equations (depends on size) and then adjusting the estimates based on other variables.
- The model implies that size is the primary factor for cost and other factors have lesser effect.

# Constructive Cost Model (COCOMO) (2/2)

- The model estimates the total effort in terms of person months of the technical staff.

  NOTE: The effort estimate includes development, management and support tasks but does not include the cost of the secretarial and other staff that might be needed in an organization.

# Basic Steps in COCOMO Model (1/2)

- Obtain an initial estimate of the development effort from the estimate of thousands of delivered lines of source code (KDLOC)

$$E_i = a * (KDLOC)^b$$

- Determine a set of 15 multiplying factors from different attributes of the project called the cost driver attributes
- Adjust the effort estimate by multiplying the initial estimate with all the multiplying factors together called the effort adjustment factor (EAF).

$$E_a = (EAF) * E_i$$

# Basic Steps in COCOMO Model (2/2)

NOTE: The 15 attributes are called the *cost driver* attributes that determine the 15 multiplying factors. They depend on product, computer, personnel and technology attributes.

# Classification of Projects in COCOMO

- Projects are classified into:
  - Organic – projects in an area in which the organization has considerable expertise and requirements are less stringent e.g. simple data processing system. a = 3.2 and b = 1.05
  - Embedded – projects are ambitious and novel and the organization has little or no prior experience in those areas and there are stringent requirements to be met e.g. embedded avionics system. a = 2.8 and b = 1.20
  - Semidetached – projects that fall between the above two categories e.g. new operating system, database management system (DBMS). a = 3.0 and b = 1.12

# Cost Driver Attributes (1/3)

- Product Attributes
  - Reliability
  - Database size
  - Complexity
- Computer Attributes
  - Execution time
  - Storage requirement
  - Virtual memory volatility
  - Turnaround time

# Cost Driver Attributes (2/3)

- Project attributes
  - Modern programming practices
  - Use of SW tools
  - Development schedule
- Personnel Attributes
  - Analyst capability
  - Application experience
  - Programmer capability
  - Virtual machine experience
  - Programming language experience

# Cost Driver Attributes (3/3)

- Each cost driver has a rating scale and for each rating a multiplying factor is provided.
- Multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF)
- The final effort is obtained by multiplying the initial estimate by the EAF

$$E = (EAF) * E_i$$

# Phase Effort Estimation (1/2)

- The final effort estimate

$$E = (EAF) * E_i$$

  gives the overall cost of the project (in person months)

- For planning and monitoring purposes, estimates of the effort required for the different phases is also desirable.
- In COCOMO, effort for a phase is considered as a defined percentage of the overall effort.

# Phase Effort Estimation (2/2)

NOTE: The percentage of the total effort spent in a phase varies with the type (**organic**, **semidetached** and **embedded**) and size (**small** (<=2KDLOC), **intermediate** (~8 KDLOC), **medium** (~32KDLOC), **large** (>=128KDLOC)) of the project.

- An estimate of percentages for project sizes different from the one specified can be obtained by linear interpolation.

# Project Complexity

- COCOMO provides three levels of models of increasing complexity viz.
  - Basic
  - Intermediate
  - Detailed

  NOTE: The detailed model is the most complex. It has different multiplying factors for the different phases for a given cost driver. The set of cost drivers applicable to a system or module is also not the same as the drivers for the system level.

# Project Scheduling (1/2)

- Schedule estimate and staff requirement estimation may be the most important activities after cost estimation.

  NOTE: Both are related, if phase wise cost is available.

- The goal of schedule estimation is to determine the total duration of the project and the duration of the different phases.

# Project Scheduling (2/2)

NOTE: Though there is a relationship between the person-month cost and the project duration, it is not obtained by dividing the total effort by the average staff size.

# Average Duration Estimation

- Single variable models can be used to determine the overall duration of the project.
- Generally, schedule is modeled as depending on the total effort (which in turn depends on size).

  IBM Federal Systems Division uses total duration D (in calendar months) as

  $$D = 4.1\ E^{0.36}\ \text{(months)}$$

  COCOMO model, for organic type

  $$D = 2.5\ E^{0.38}\ \text{(months)}$$

  NOTE: For other projects the constants vary slightly.

# Phase Duration Estimation

- The duration or schedule of the different phases is obtained in the same manner as in effort distribution i.e. by using the percentages for the different phases.

  NOTE: In COCOMO, sometimes, the detailed design, coding and unit testing phases are combined into one “programming phase”.

# Project Scheduling & Milestones

- Once we have the estimates of the effort and time requirement for the different phases, a schedule for the project can be prepared.
- This schedule will be used to monitor the progress of the project.
- A conceptually simple and effective scheduling technique is the Gantt chart, which uses a calendar-oriented chart to represent the project schedule.

# Software Development Life Cycle (1/2)

- Software life cycle:
  - Series of identifiable phases that a software undergoes during its lifetime
- Life cycle phases:
  - Feasibility
  - Requirements specifications
  - Design
  - Coding & Unit testing
  - Integration
  - Testing
  - Maintenance

# Software Development Life Cycle (2/2)

- Life cycle activities:
  - Each life cycle phase consists of several activities e.g. project management activities, developmental activities, etc.
- Entry criteria for each phase:
  - Preconditions for initiating a phase
- Exit criteria for each phase
  - Preconditions for concluding a phase

# Software Life Cycle Models: Comments (1/2)

- Large development teams cannot function without **formal definitions of work** e.g. work breakdown structure
- Defining life cycle model encourages development of software in *systematic* and *disciplined* manner
- Entry and exit criteria associated with the phases help in:
  - Better monitoring and controlling of projects
  - Better reporting of project progress e.g. no 90% complete syndrome

# Software Life Cycle Models: Comments (2/2)

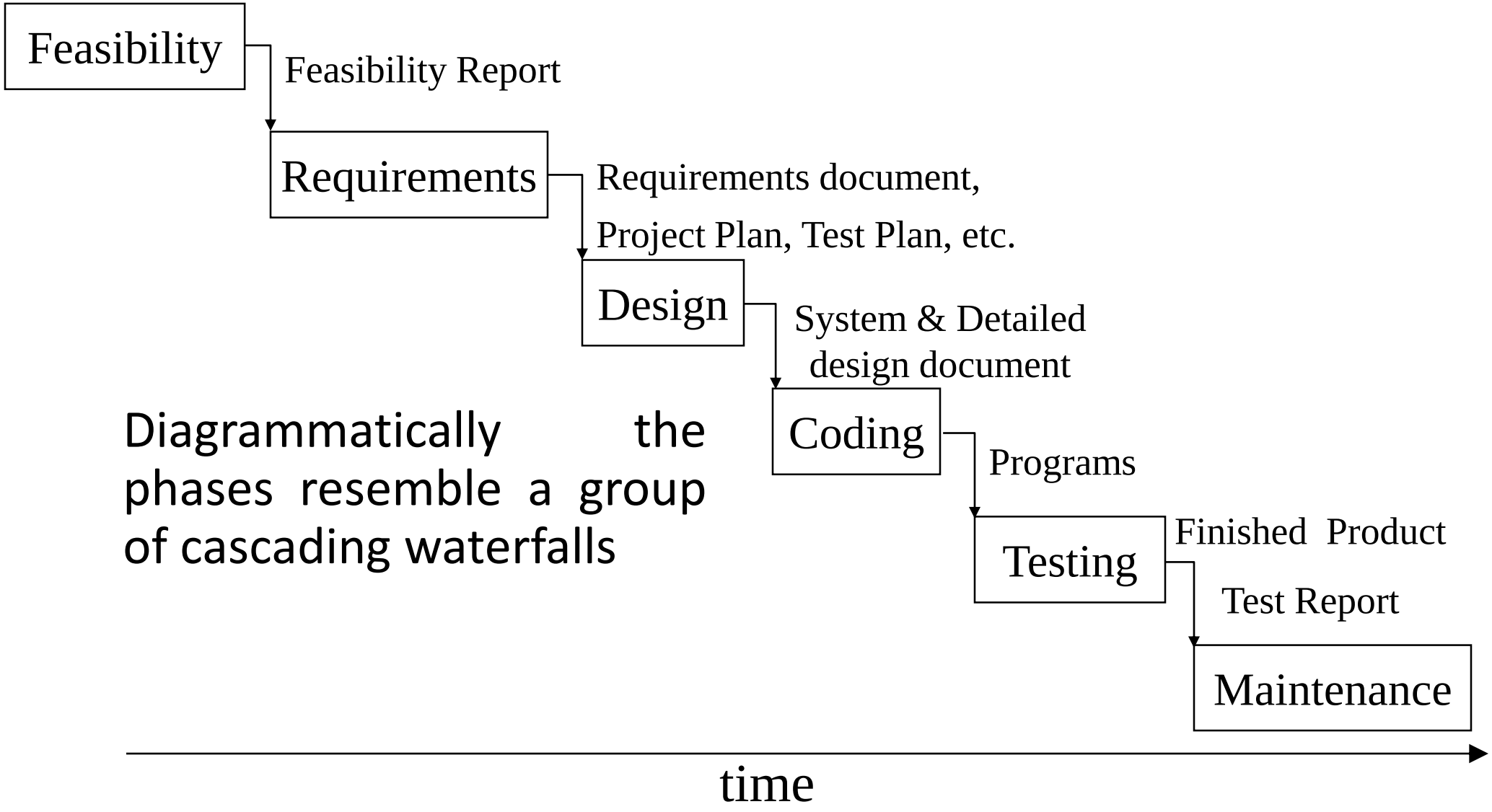
- Some well known life cycle models are:
  - Classical waterfall model
  - Iterative waterfall model
  - Prototyping model
  - Evolutionary model
  - Spiral model

# Group Assignment

Assess the merits and demerits of each development life cycle model and select one development life cycle model for your project. Justify your selection.

**Submit by: 1st February 2012 12:00 hrs.**

# Waterfall Model (1/3)

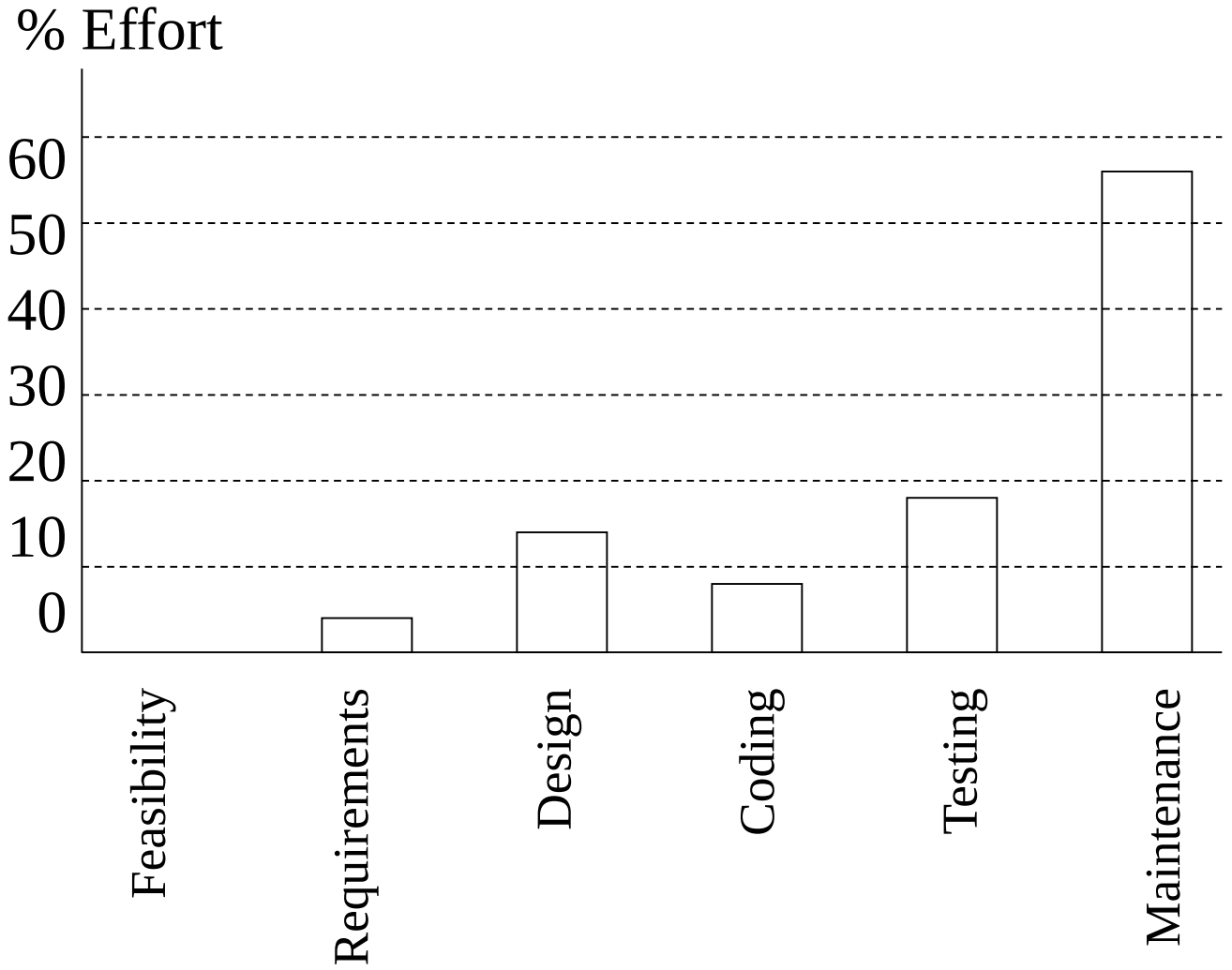



Diagrammatically the phases resemble a group of cascading waterfalls

# Waterfall Model (2/3)

Each phase requires relatively different amount of effort

Note: Maintenance and Testing phases consume sizable effort



# Waterfall Model (3/3)

- Waterfall model:
  - Most easy to understand model
- Too simplistic
  - Well suited for only well-understood problems
- Not very good for improving “product visualization”

  Note: Without adequate product visualization, it is very difficult to manage software development projects i.e. if it is difficult to visualize, it will be difficult to manage

# Waterfall Model - Work Products (1/2)

Some of the work products of the waterfall model are:

- Systems Requirements Specification Document
- Project Plan
- System Design Document
- Detailed Design Document
- Test Plan (with test cases)
- Test Report
- Final Code
- Software manuals (e.g. User manual, Installation manual, etc.)
- Review Reports

# Waterfall Model - Work Products (2/2)

NOTE: Any other document that enables the project team to achieve its project objectives, should be maintained as one of the work products of the project.

# Waterfall model – Limitations (1/2)

- Real projects rarely follow the sequential flow that the model proposes.
- The model requires that all the requirements are stated explicitly.
- Changes can cause confusion as the project proceeds.
- The model has difficulty in accommodating the natural uncertainty that exists in many projects.

# Waterfall model – Limitations (2/2)

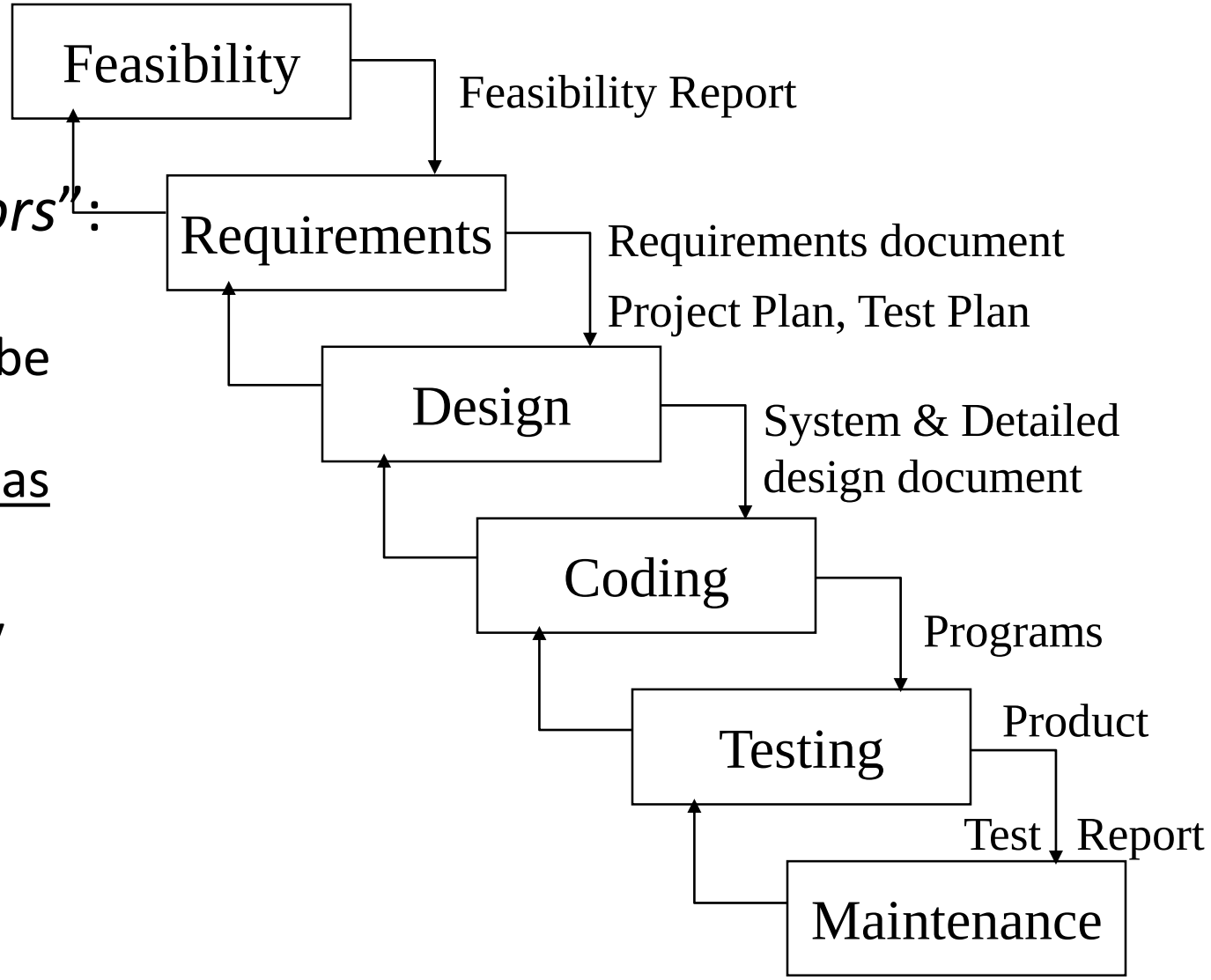
- The working versions of the programs are available only late in the project time span, hence the customer must have patience.
- A major blunder, if undetected until working program is reviewed, can be disastrous.

# Iterative Waterfall Model (1/2)

Principle of "*Phase containment of errors*":

- If and when errors occur, they should be detected (and corrected) as early as possible

Note: This dramatically reduces rework



# Iterative Waterfall Model (2/2)

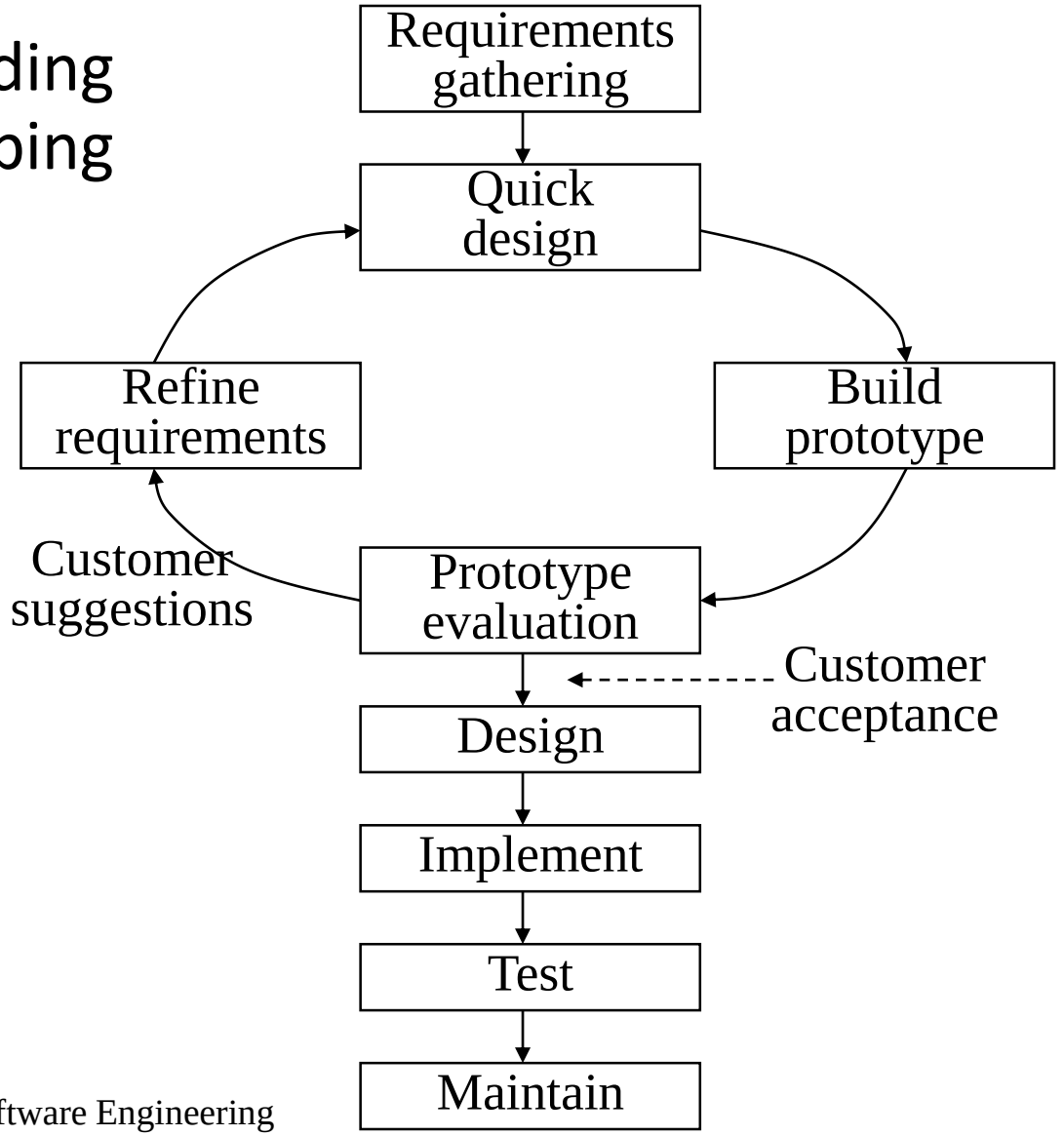
- In real life, defects are likely to be introduced during each phase of development

  Note:

  - It is desirable to minimize (if not eliminate) human errors during software development
  - These defects get detected (later) as the development progresses
  - Detected defects are then corrected by going back to the appropriate phase (where the defect got introduced)
  - This process can become time consuming and costly
- Hence it is desirable to detect the defects as early as possible, preferably in the same phase itself i.e. before moving on to the next phase.

# Prototyping Model (1/6)

- This approach suggests building a prototype before developing the actual software



# Prototyping Model (2/6)

## Approach to prototyping:

- Prototypes are built using several shortcuts
  - Limited functional capabilities
  - Low reliability
  - Inefficient performance

  Note: Prototype usually turns out to be a very crude version of the proposed system
- Prototyping may involve several iteration (until receiving customer acceptance)
- After acceptance of the prototype, the development can be done using a simple waterfall model

# Prototyping Model (3/6)

Note:

- Requirements gathering phase includes prototyping
- Prototype code may be thrown away
- Time/effort spent on prototype is worth it
  - SRS gets supplemented with prototype
  - Experience in building prototype is carried forward to development

## Advantages of prototyping

- Provides a good mechanism for understanding the customer requirements

# Prototyping Model (4/6)

- Illustrates data formats, messages, reports, interactive dialogues
- Specially useful for GUI (Graphic User Interface) development

- Facilitates critical examination of technical issues associated with software development
  - Performance related issues
    e.g. response times
    e.g. efficiency of algorithms
- Reduces wasted development effort
  - Difficulty of “getting a product right first time”
  - Inevitable discarding of initial product

# Prototyping Model (5/6)

Situations warranting prototyping:

- User requirements are not well understood
- Technical aspects are not well understood

Problems with Prototyping model

- Usually the prototype is built without adequately considering overall software quality or long-term maintainability.
- Lack of appreciation that Prototype building is very different from product building.

# Prototyping Model (6/6)

## Problems with Prototyping model (Continued)

- Compromises on the choices like operating system or programming language, are often made during prototyping.
- Customer demands that the prototype be “quickly” converted to a working product.
- The prototype is confused to be the actual product.

# Rapid Application Development (RAD) Model (1/2)

- Incremental software development process model emphasizing an extremely short development cycle.
- It is an “high speed” adaptation of the waterfall model, using component based construction.
- Enables a development team to create a “fully functional system” within very short time periods, provided the requirements are well understood and project scope is constrained.
- RAD process works to reuse existing program components (when possible) or create reusable components (when necessary).

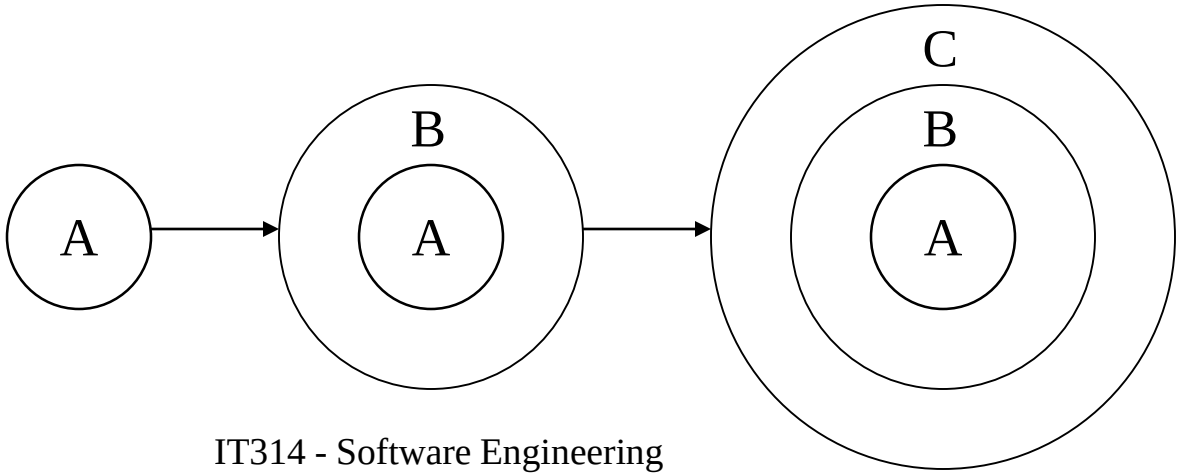
# Rapid Application Development (RAD) Model (2/2)

- Problems with RAD model
  - For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams
  - RAD requires equal commitment from the developers and customers towards the rapid development process.
  - RAD cannot be applied to projects that cannot be properly modularized.
  - RAD is not appropriate for areas using new technology (high technical risk).
  - RAD requires that the requirements are “well understood” and that the project scope is “constrained”.

# Evolutionary Model (1/3)

- This model is also known as SUCCESSIVE VERSIONS model
- It involves:
  - Breaking a system down into several modules (or functionalities) such that
  - These modules can be delivered in an incremental fashion
- Developer initially develops the core module and then refines it by incrementally adding new functionalities

  Note: Each successive version of the product is a working version



# Evolutionary Model (2/3)

- Advantage:
  - User gets an opportunity to experiment/use the partial system much before the fully developed version is released
  - Helps in eliciting requirements
  - Core module gets tested very thoroughly (since it gets tested at the time of each release)
  - Entire resource requirements need not be committed to the project at the same time
- Disadvantages:
  - It is difficult to break down a system into functional units that can be implemented in an incremental fashion

# Evolutionary Model (3/3)

Situations for using evolutionary model:

- Useful only in case of very large systems where:
  - it is easy to identify modules which can be implemented in incremental fashion
  - incremental delivery is acceptable to the customer
- Examples of Evolutionary Models
  - The incremental Model
  - The Spiral Model
  - The Concurrent development model

# Customer Reaction (1/2)

Evolutionary model and Waterfall model

- Initially, the customer confidence is high irrespective of the type of development approach (waterfall or evolutionary)
- Reactions to lengthy monolithic development process (e.g. waterfall model):
  - Technical jargons are used
  - Delays are announced
  - Customer confidence drops off as there is no “visible working product”

# Customer Reaction (2/2)

- Reactions to lengthy monolithic development process (e.g. waterfall model) (continued):
  - Results in customer resentment and resistance
    - May manifest into sabotage or scapegoating,
- Reactions to Evolutionary model:
  - In early stages, customer reactions are of the type “this is not what I want”
  - Incremental delivery, however, provides a much more satisfying experience i.e. helps in overcoming initial “loss of confidence”

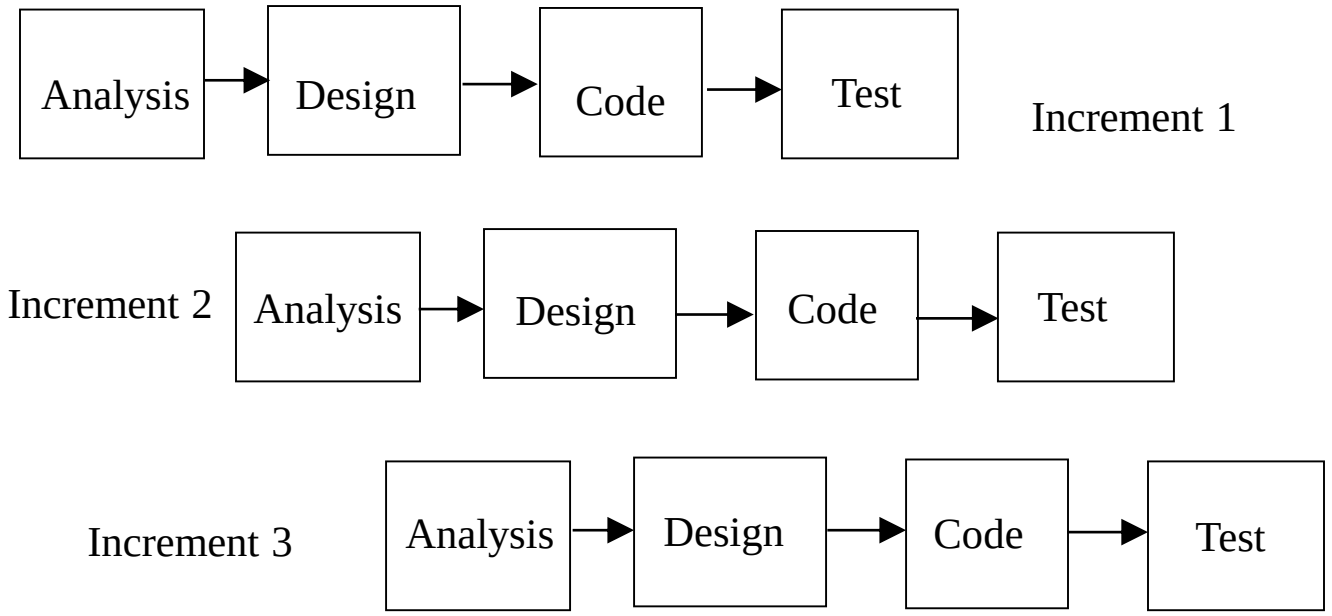
# Any Questions?

# Incremental Model (1/3)

- Combines elements of the waterfall model with the iterative philosophy of prototyping.
- It applies the linear sequences in a staggered fashion as calendar time progresses.
- Each linear sequence produces a deliverable “increment” of the software.
- The process is repeated following the delivery of each increment, until the complete product is produced.

  NOTE: The first increment is often the *core* product. The client can use the *core* product

# Incremental Model (2/3)




Incremental Model - The Time sequence of development

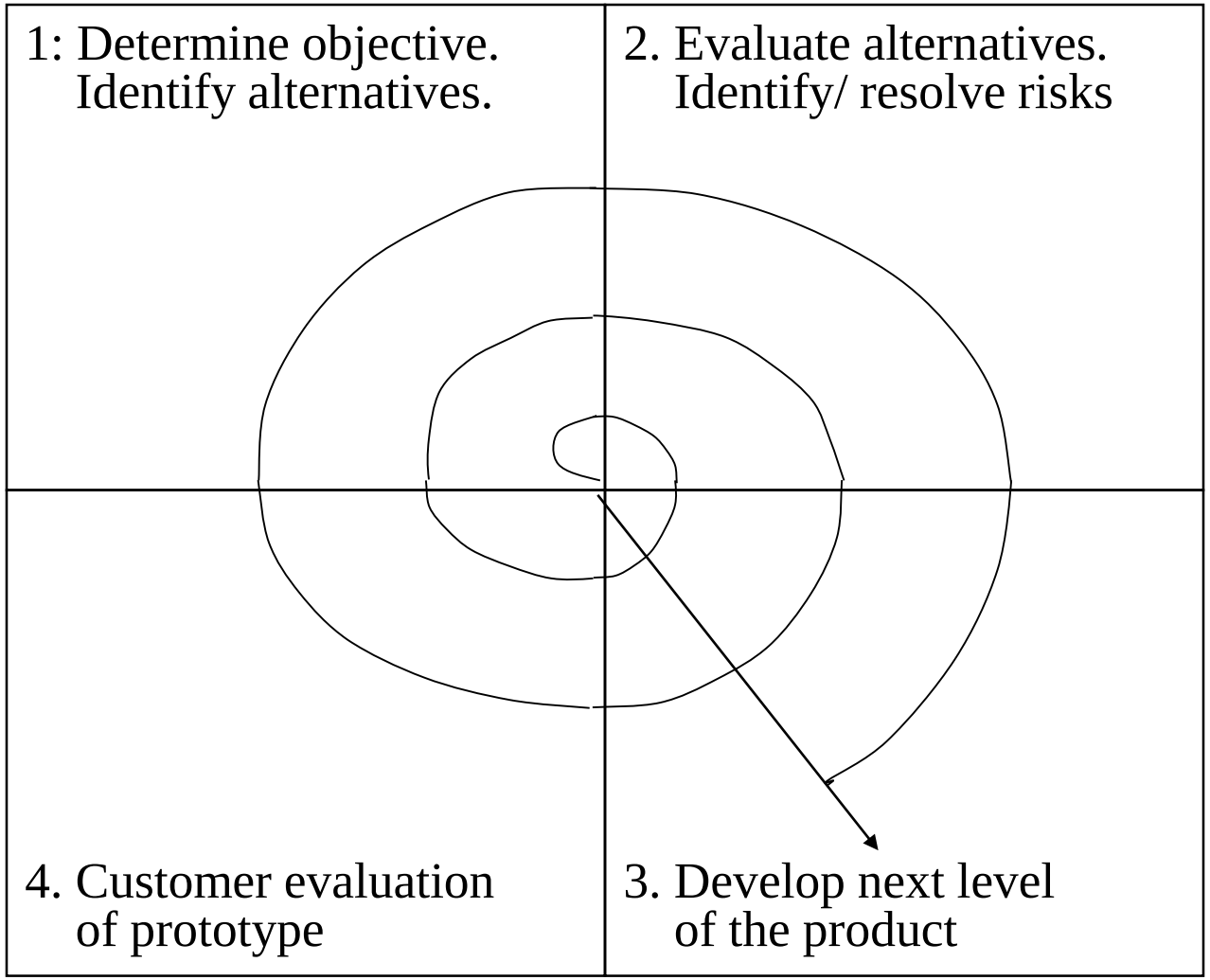
Calendar Time

NOTE: Feedback from usage of the core product is used to modify the core product to better meet the needs of the customer. Other features are added later.

# Incremental Model (3/3)

- Advantages of Incremental Model
  - Useful when staffing is unavailable for a complete implementation by the business deadline of the project
  - Early increments can be implemented by fewer people, and depending on the acceptability of the core product, more resources can be added to implement subsequent stages.
  - Increments can be planned to manage technical risks.
- Disadvantages of Incremental Model
  - Agreement on the core product is not easy
  - The process can be time consuming
  - Depends on the client agreeing for it.

# Spiral Model (1/5)




Note: RADIUS indicates cost
ANGLE indicates progress

# Spiral Model (2/5)

Four Quadrants of Spiral Model

1. Identify objectives of the product and identify alternative solutions.
2. Evaluate alternative solutions. Identify potential risks. Resolve risks by building prototype
3. Develop next level of product. Verify this product
4. Evaluation of the product by customer.

# Spiral Model (3/5)

- Plan the next iteration around the spiral (if required)

NOTE:

- Radius of the spiral (at any point) indicates the cost incurred on the project (so far)
- Angular dimension represents progress
- Spiral model can be viewed as meta model
  - Single loop spiral model represents a Waterfall model

# Spiral Model (4/5)

- Spiral model … meta model (Continued)
  - Looping around each cycle represents evolutionary model
    - More complete version of the product gets built progressively
  - Prototyping is used as a risk reduction mechanism
    - Provides direct support for coping with project risks
    - After final iteration, all risks are resolved and the “requirements” are ready for development
  - Completion of the final product is done according to the Waterfall Model

# Spiral Model (5/5)

NOTE: Spiral model subsumes all the models discussed earlier

- Situations for using spiral model
  - Development of technically challenging products which are prone to several risks.

# Concurrent Development Model (1/3)

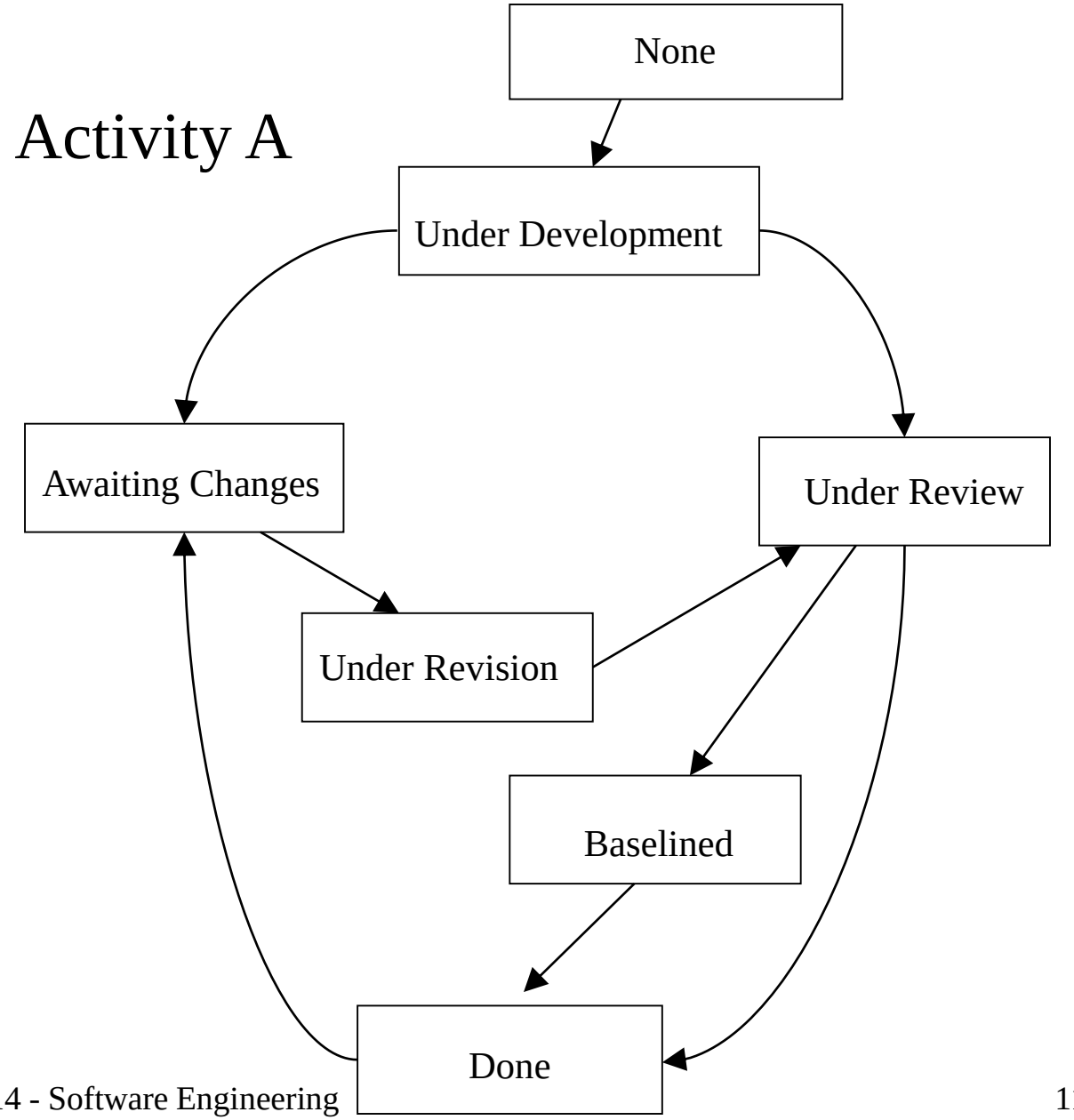
- Typically in a large project, there are personnel on the project involved in activities typically associated with many phases of development simultaneously.
- The model can be represented schematically as a series of major technical activities, tasks and their associated states
- Some of the States Possible
  - Under development
  - Awaiting Changes
  - Under Revision

# Concurrent Development Model (2/3)

- Some of the States Possible (Continued)
  - Under Review
  - Baselined
  - Done
- It defines a series of events that will trigger transition from one state to another for each of the SE activities

# Concurrent Development Model (3/3)

Schematic representation of an activity (A) with concurrent development process model



# Recap: Software Life Cycle Models

- Classical Waterfall Model
- Iterative Waterfall Model
- Prototyping Model
- Rapid Application Development (RAD) model
- Evolutionary Models
  - Incremental Model
  - Spiral Model
  - Concurrent Development Model
  - Schematic representation of an activity (A) with concurrent development process model