Software Engineering – ESC501

- Prof. Poulami Dutta

Intended Learning Outcomes (ILOs)

- Explain what is a life-cycle model.
- Explain what problems would occur if no life-cycle model is followed.
- Identify the different software life-cycle models.
- Identify the different phases of the classical waterfall model.
- Identify the activities undertaken in each phase.
- Identify the shortcomings of the classical waterfall model.
- Identify the phase-entry and phase-exit criteria of each phase.

Software Life Cycle

- Software Life Cycle:
  - series of identifiable stages that a software product undergoes during its life time:
    - Feasibility study,
    - Requirements analysis and specification,
    - Design,
    - Coding,
    - Testing,
    - Maintenance.

Life Cycle Model

- A software life cycle model (or process model):
  - a descriptive and diagrammatic model of software life cycle
  - identifies all the activities required for product development,
  - establishes a precedence ordering among the different activities,
  - divides life cycle into phases.

Life Cycle Model (CONT.)

- Several different activities may be carried out in each life cycle phase.
  - For example, the design stage might consist of:
    - structured analysis activity followed by
    - structured design activity.

Why Model Life Cycle ?

- A written description:
  - forms a common understanding of activities among the software developers.
  - Helps in identifying inconsistencies, redundancies, and omissions in the development process.
  - helps in tailoring a process model for specific projects.

Why Model Life Cycle ? (Contd.)

- Processes are tailored for special projects.
  - A documented process model
    - helps to identify where the tailoring is to occur.

Life Cycle Model (CONT.)

- The development team must identify a suitable life cycle model and then adhere to it.
  - Primary advantage of adhering to a life cycle model:
    - helps development of software in a systematic and disciplined manner.

Life Cycle Model (CONT.)

- When a program is developed by a single programmer ---
  - he has the freedom to decide his exact steps.
- When a software product is being developed by a team:
  - there must be a precise understanding among team members as to when to do what,
  - otherwise it would lead to chaos and project failure.

Life Cycle Model (CONT.)

- A software project will never succeed if:
  - one engineer starts writing code,
  - another concentrates on writing the test document first,
  - yet another engineer first defines the file structure,
  - another defines the I/O for his portion first.

Life Cycle Model (CONT.)

- A life cycle model:
  - defines entry and exit criteria for every phase.
  - A phase is considered to be complete:
    - only when all its exit criteria are satisfied.

Life Cycle Model (CONT.)

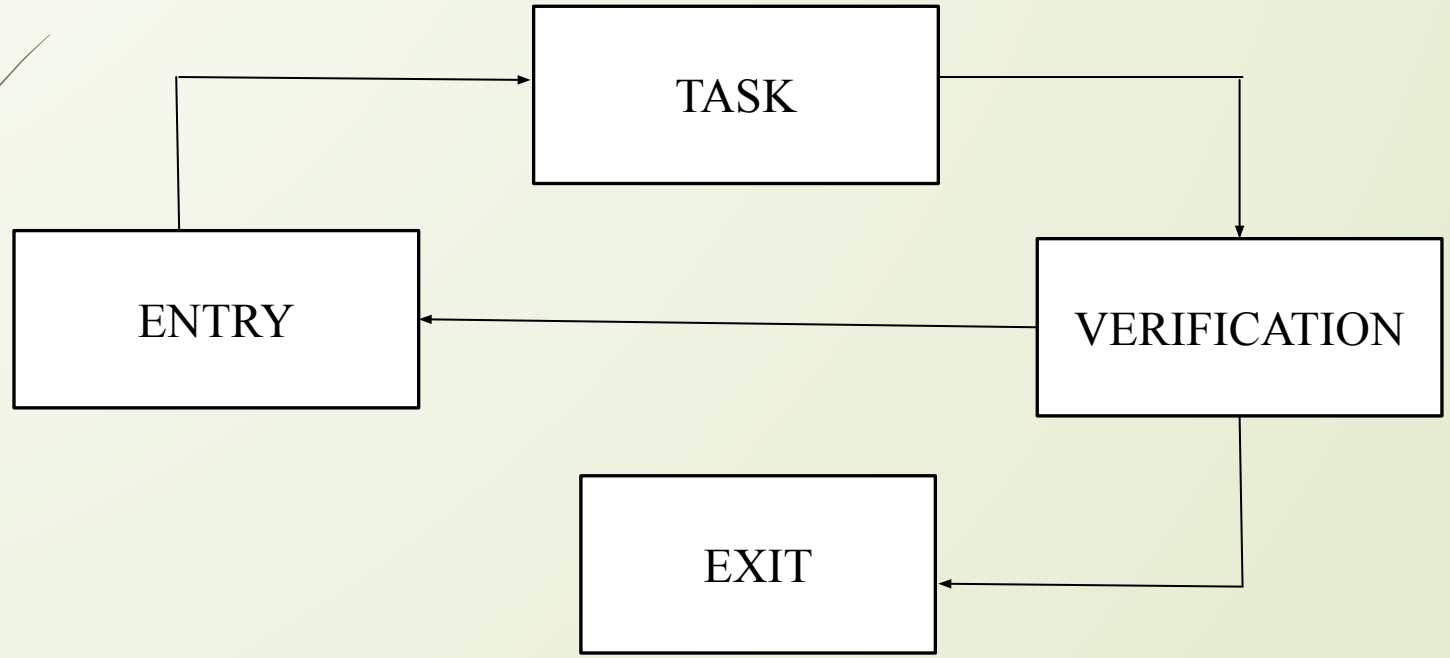
- The phase exit criteria for the software requirements specification phase:
  - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.
- A phase can start:
  - only if its phase-entry criteria have been satisfied.

Life Cycle Model (CONT.)

- It becomes easier for software project managers:
  - to monitor the progress of the project.
- When a life cycle model is adhered to,
  - the project manager can at any time fairly accurately tell,
    - at which stage (e.g., design, code, test, etc.) the project is.
  - Otherwise, it becomes very difficult to track the progress of the project.
    - the project manager would have to depend on the guesses of the team members.

Life Cycle Model (CONT.)

- This usually leads to a problem:
  - known as the 99% complete syndrome, which is in reality far from complete.



Different software life cycle models

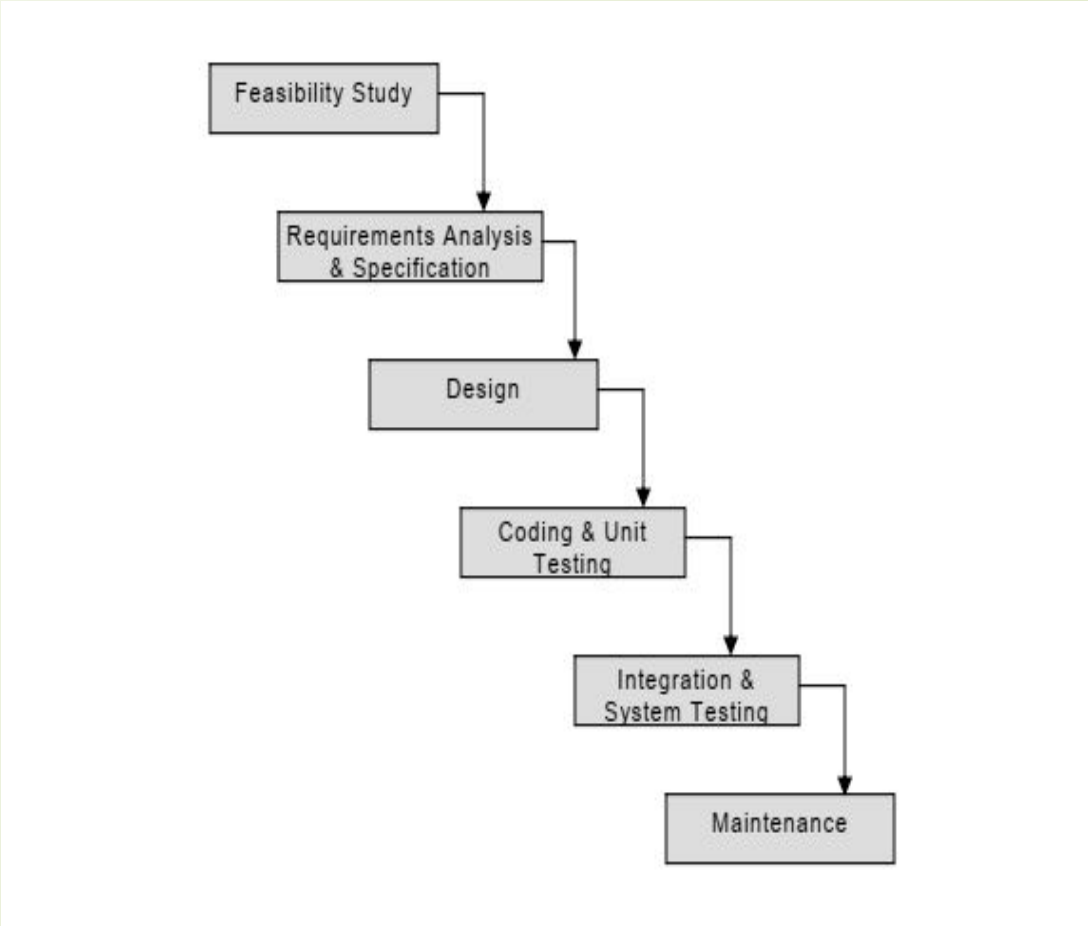
- A few important and commonly used life cycle models are as follows:
  - Classical Waterfall Model
  - Iterative Waterfall Model
  - Prototyping Model
  - Evolutionary Model
  - Spiral Model

Classical Waterfall Model

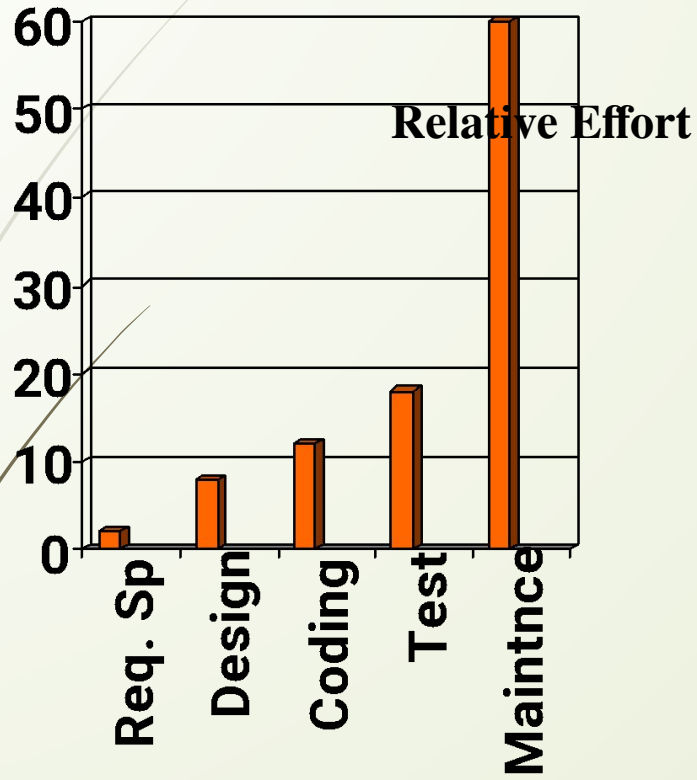
- Classical waterfall model divides the life cycle into the following phases:
  - feasibility study,
  - requirements analysis and specification,
  - design,
  - coding and unit testing,
  - integration and system testing,
  - maintenance.

Classical Waterfall Model (contd.)

- Not a practical model in the sense that it can not be used in actual software development projects.
- Considered to be a *theoretical way of developing software*.
- All other life cycle models are essentially derived from the classical waterfall model.



Relative Effort for Phases



- Phases between feasibility study and testing
  - known as development phases.
- Among all life cycle phases
  - maintenance phase consumes maximum effort.
- Among development phases,
  - testing phase consumes the maximum effort.

Classical Waterfall Model (CONT.)

- **Most organizations usually define:**
  - standards on the outputs (deliverables) produced at the end of every phase
  - entry and exit criteria for every phase.
- **They also prescribe specific methodologies for:**
  - specification,
  - design,
  - Coding,
  - testing,
  - project management, etc.

Classical Waterfall Model (CONT.)

- **The guidelines and methodologies of an organization:**
  - called the organization's software development methodology.
- **Software development organizations:**
  - expect fresh engineers to master the organization's software development methodology.

Feasibility Study

- Main aim of feasibility study: determine whether developing the product:
  - financially worthwhile
  - technically feasible
- First roughly understand what the customer wants by visiting the client site:
  - different data which would be input to the system,
  - processing needed on these data,
  - output data to be produced by the system,
  - various constraints on the behaviour of the system.

Activities during Feasibility Study

- Work out an overall understanding of the problem.
- Formulate different solution strategies.
- Examine alternate solution strategies in terms of:
  - resources required,
  - cost of development, and
  - development time.

Activities during Feasibility Study

- Perform a cost/benefit analysis:
  - to determine which solution is the best
    - Depending on customer budget
    - Technical expertise of the development team
  - You may determine that none of the solutions is feasible due to:
    - high cost,
    - resource constraints,
    - technical reasons.

CASE STUDY

- **MINING Company**

Requirements Analysis and Specification

The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem:

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?
- What exactly are the data input to the system and what exactly are the data output by the system?
- What are the likely complexities that might arise while solving the problem?
- If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be?

Requirements Analysis and Specification (Contd.)

- Aim of this phase:
  - understand the exact requirements of the customer,
  - document them properly.
- Consists of two distinct activities:
  - requirements gathering and analysis
  - requirements specification.

Goals of Requirements Analysis

- Collect all related data from the customer:
  - analyze the collected data to clearly understand what the customer wants,
  - find out any inconsistencies and incompleteness in the requirements,
  - resolve all inconsistencies and incompleteness.

Requirements Gathering

- Gathering relevant data:
  - usually collected from the end-users through interviews and discussions.
  - For example, for a business accounting software:
    - interview all the accountants of the organization to find out their requirements.

Requirements Analysis (CONT.)

- The data you initially collect from the users:
  - would usually contain several contradictions and ambiguities
  - each user typically has only a partial and incomplete view of the system.
- Ambiguities and contradictions:
  - must be identified
  - resolved by discussions with the customers.
- Next, requirements are organized:
  - into a Software Requirements Specification (SRS) document.

Parts of a SRS document

- **Functional requirements** of the system - are those that refer to the functionality of the system, i.e., what services it will provide to the user.
- **Non-functional requirements** of the system - pertain to other information needed to produce the correct system and are detailed separately. Nonfunctional requirements deal with the characteristics of the system which cannot be expressed as functions - such as the maintainability of the system, portability of the system, usability of the system, etc.
- **Goals of implementation** - The goals of implementation part documents provide some general suggestions regarding development. These suggestions guide trade-off among design goals. The goals of implementation section might document issues such as revisions to the system functionalities that may be required in the future, new devices to be supported in the future, reusability issues, etc.

Documenting Functional Requirements

- **Example: Withdraw Cash from ATM**
- **R1: withdraw cash**
- **Description**: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash; otherwise it generates an error message.
- **R1.1:** select withdraw amount option
  - Input: "withdraw amount" option
  - Output: user prompted to enter the account type
- **R1.2**: select account type
  - Input: user option for selecting the account number
  - Output: prompt to enter amount
- **R1.3:** get required amount
  - Input: amount to be withdrawn in integer values greater than 100 and less than 10,000 in multiples of 100.
  - Output: The requested cash and printed transaction statement.

CASE STUDY

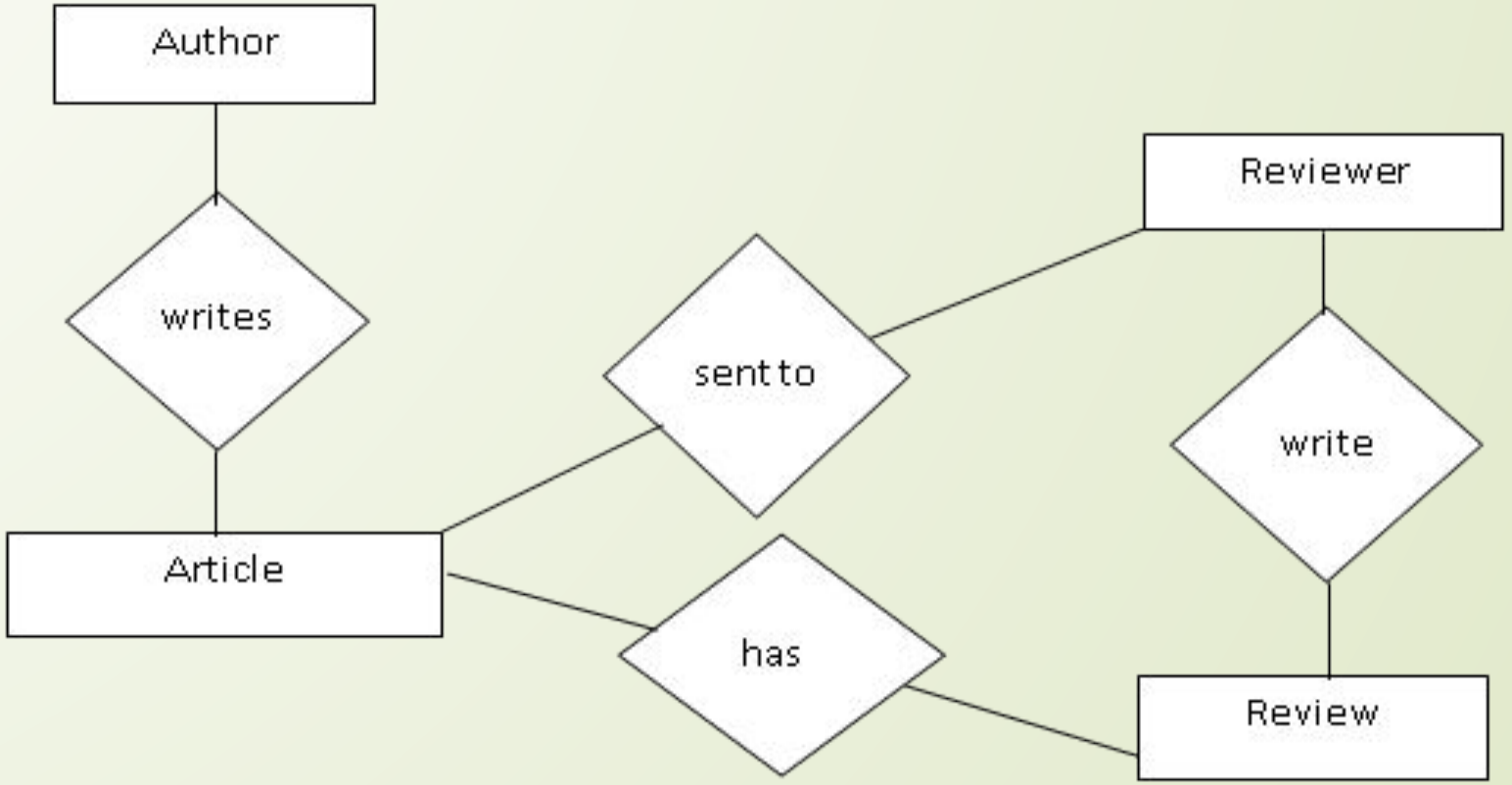
- **Web-Publishing System** for a local editor of a regional historical society.
  - **Scope**: to maximize the editor's productivity by providing tools to assist in automating the article review and publishing process.
- **Functional Requirements:**
  - The Reader chooses how to search the Web site. The choices are by **Author, by Category, and by Keyword**.
    - If the search is by **Author**, the system creates and presents an alphabetical list of all authors in the database. In the case of an article with multiple authors, each is contained in the list.
      - The Reader selects an author.
      - The system creates and presents a list of all articles by that author in the database.
      - The Reader selects an article.
      - The system displays the Abstract for the article.
      - The Reader selects to download the article or to return to the article list or to the previous list.

CASE STUDY (CONTD.)

- **Functional Requirements:**
  - if the Reader selects to search by **category**, the system creates and presents a list of all categories in the database.
    - The Reader selects a category.
    - The system creates and presents a list of all articles in that category in the database.
  - If the Reader selects to search by **keyword**, the system presents a dialog box to enter the keyword or phrase.
    - The Reader enters a **keyword** or phrase.
    - The system searches the Abstracts for all articles with that keyword or phrase and creates and presents a list of all such articles in the database.

CASE STUDY (CONTD.)

- **Non-Functional Requirements:** The logical structure of the data to be stored in the internal Article Manager database.



Data Item	Type	Description
Name	Text	Name of principle author
Email Address	Text	Internet address
Article	Pointer	Article entity

Data Item	Type	Description
Article	Pointer	Article entity
Reviewer	Pointer	Reviewer entity
Date Sent	Date	Date sent to reviewer
Returned	Date	Date returned; null if not returned
Contents	Text	Text of review

Data Item	Type	Description
Name	Text	Name of principle author
ID	Integer	ID number of Historical Society member
Email Address	Text	Internet address
Article	Pointer	Article entity of
Num Review	Integer	Review entity
History	Text	Comments on past performance
Specialty	Category	Area of expertise

CASE STUDY (CONTD.)

Author – Review – Reviewer Data Entities

Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.
- In technical terms:
  - during design phase, software architecture is derived from the SRS document.
- **Two design approaches**:
  - traditional approach,
  - object oriented approach.

Traditional Design Approach

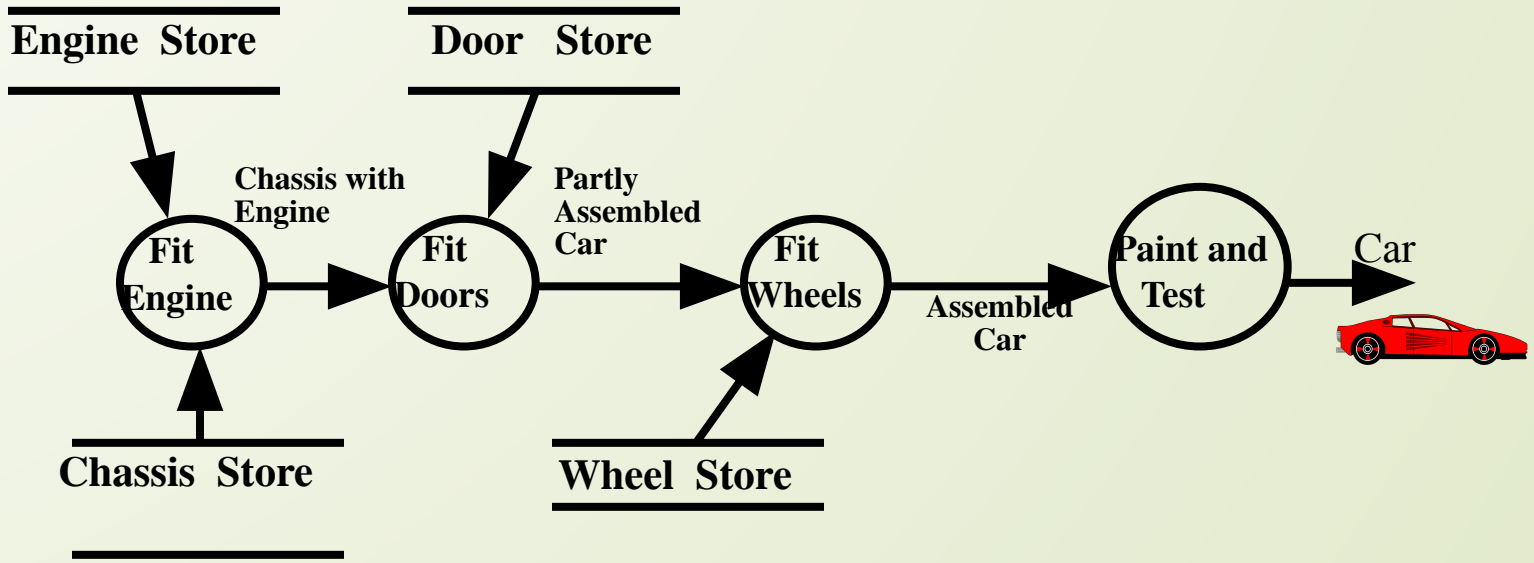
- Consists of two activities:
  - Structured analysis
  - Structured design

Structured Analysis Activity

- Identify all the functions to be performed.
- Identify data flow among the functions.
- Decompose each function recursively into sub-functions.
  - Identify data flow among the sub functions as well.

Structured Analysis (CONT.)

- Carried out using Data flow diagrams (DFDs).
- After structured analysis, carry out structured design:
  - architectural design (or high-level design)
  - detailed design (or low-level design).



Structured Design

- High-level design:
  - decompose the system into *modules*,
  - represent invocation relationships among the modules.
- Detailed design:
  - different modules designed in greater detail:
    - data structures and algorithms for each module are designed.

Object Oriented Design

- First identify various objects (real world entities) occurring in the problem:
  - identify the relationships among the objects.
  - For example, the objects in a pay-roll software may be:
    - employees,
    - managers,
    - pay-roll register,
    - departments, etc.

Object Oriented Design (CONT.)

- Object structure
  - further refined to obtain the detailed design.
- OOD has several advantages:
  - lower development effort,
  - lower development time,
  - better maintainability.

Implementation (Coding & Unit Testing)

- Purpose of implementation phase (aka coding and unit testing phase):
  - translate software design into source code.
  - each module of the design is coded,
  - each module is unit tested
    - tested independently as a stand alone unit, and debugged,
  - each module is documented.

Implementation (Coding & Unit Testing) (Contd.)

- The purpose of unit testing:
  - test if individual modules work correctly.
- The end product of implementation phase:
  - a set of program modules that have been tested individually.

Integration and System Testing

- Different modules are integrated in a planned manner:
  - modules are almost never integrated in one shot.
  - Normally integration is carried out through a number of steps.
- During each integration step,
  - the partially integrated system is tested.

Integration and System Testing (Contd.)

- After all the modules have been successfully integrated and tested:
  - system testing is carried out.
- Goal of system testing:
  - ensure that the developed system functions according to its requirements as specified in the SRS document.

Integration and System Testing (Contd.)

- System testing usually consists of three different kinds of testing activities: *f*
  - **α – testing**: It is the system testing performed by the development team.
  - **β – testing**: It is the system testing performed by a friendly set of customers.
  - **Acceptance testing**: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Maintenance

- **Maintenance of any software product:**
  - requires much more effort than the effort to develop the product itself.
  - development effort to maintenance effort is typically 40:60.

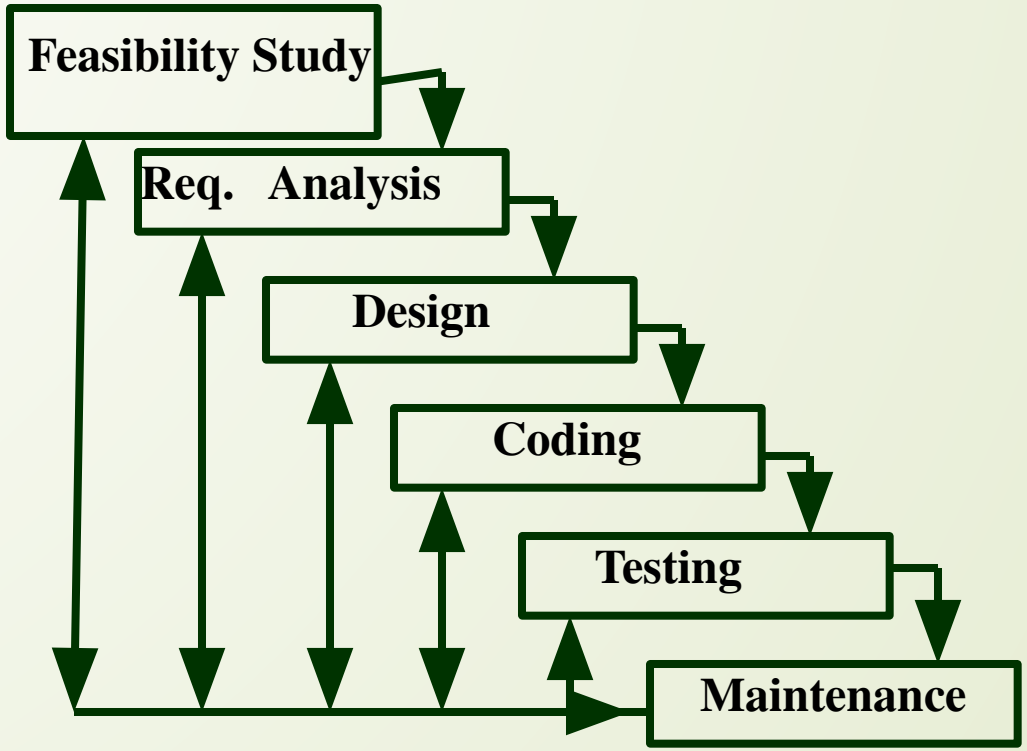
Maintenance (Contd.)

- **Corrective maintenance:**
  - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance:**
  - Improve implementation of the system
  - Enhance functionalities of the system.
- **Adaptive maintenance:**
  - Port software to a new environment,
    - e.g. to a new computer or to a new operating system.

Shortcomings of the classical waterfall model

- Classical waterfall model is idealistic:
  - assumes that no defect is introduced during any development activity.
  - in practice:
    - defects do get introduced in almost every phase of the life cycle.
- Defects usually get detected much later in the life cycle:
  - For example, a design defect might go unnoticed till the coding or testing phase.
- Once a defect is detected:
  - we need to go back to the phase where it was introduced
  - redo some of the work done during that and all subsequent phases.
- Therefore we need feedback paths in the classical waterfall model.

Iterative Waterfall Model



Iterative Waterfall Model (Contd.)

- Errors should be detected
  - in the same phase in which they are introduced.
- <u>For example:</u>
  - if a design problem is detected in the design phase itself,
    - the problem can be taken care of much more easily
    - than say if it is identified at the end of the integration and system testing phase.

Phase containment of errors

- Reason: rework must be carried out not only to the design but also to code and test phases.
- The principle of detecting errors as close to its point of introduction as possible:
  - is known as phase containment of errors.
- Iterative waterfall model is by far the most widely used model.
  - Almost every other model is derived from the waterfall model.


THANK
YOU