Advanced Software Engineering
(CS6401)

Autumn Semester (2024-2025)

Dr. Judhistir Mahapatro
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Software Life Cycle Models
(Lecture-6)

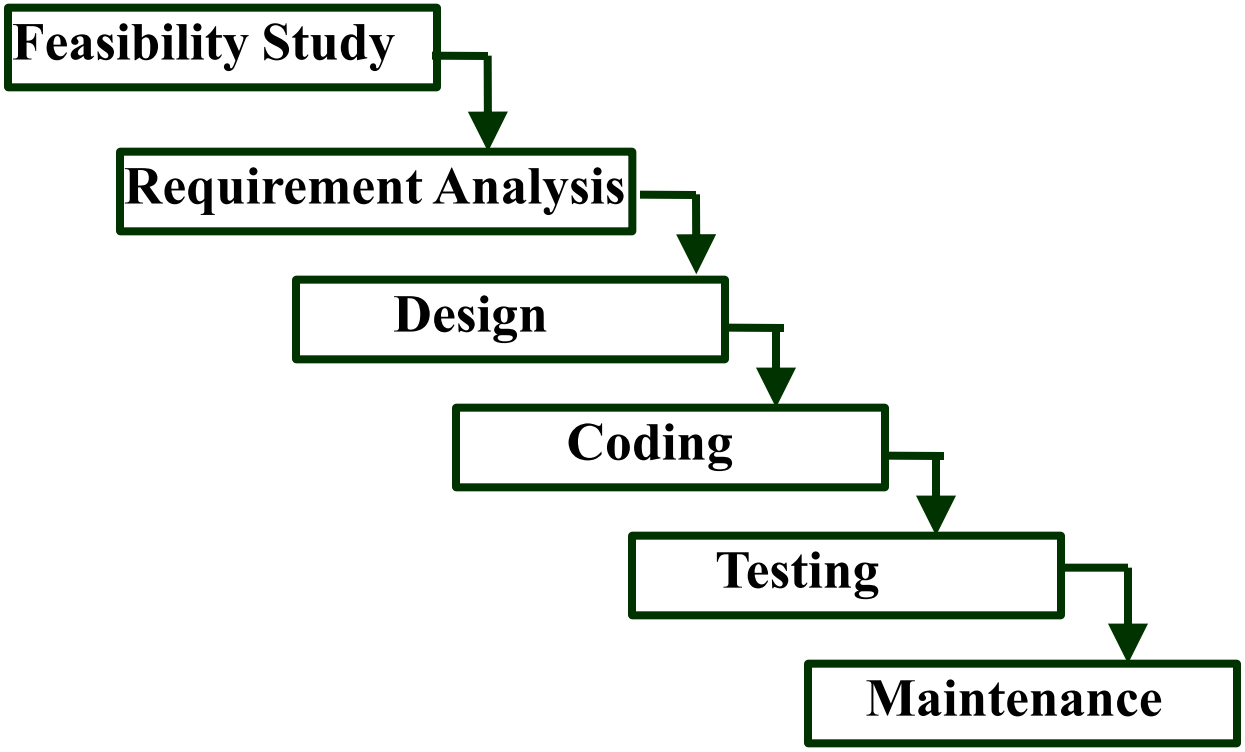
Topics covered in Previous Lecture:

- Phases of Software Life Cycle Model
- Problem Definition Document
- Feasibility study report
- Software Design Document

Classical Waterfall Model

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

Classical Waterfall Model



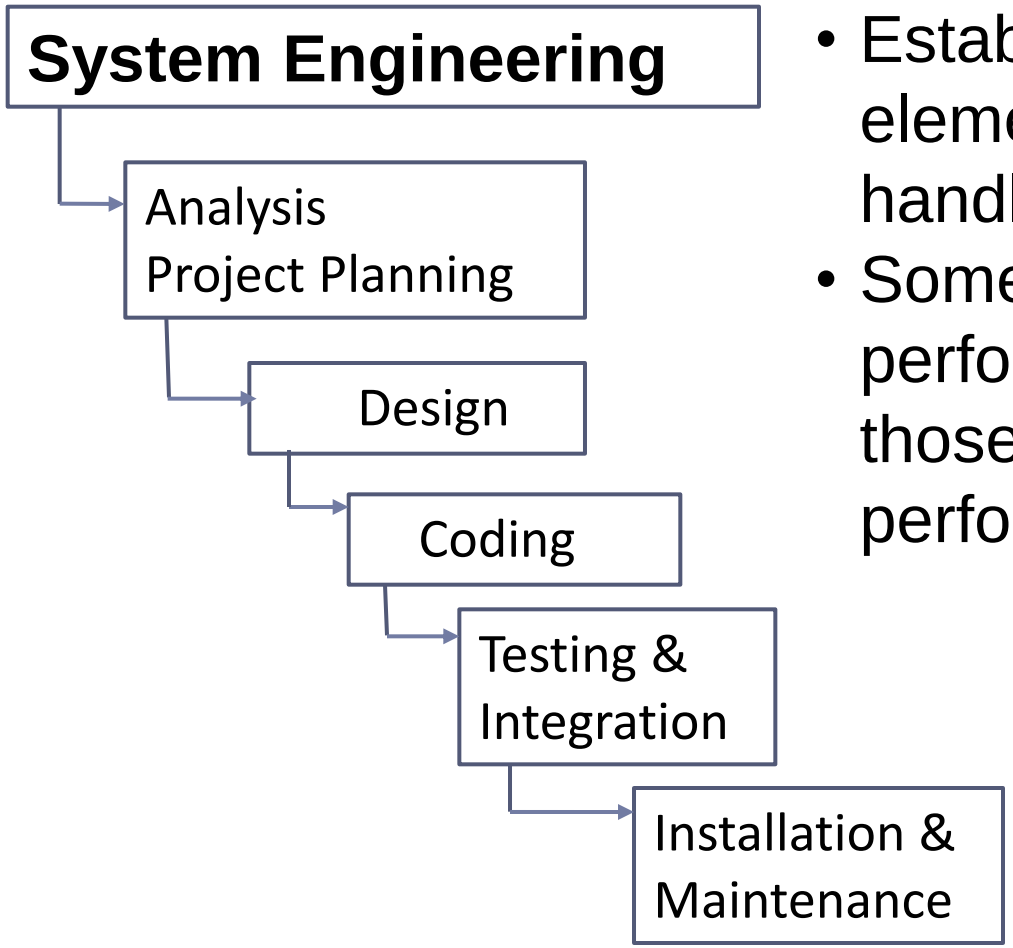
Waterfall model for Development

- Here, steps are arranged in linear order
  - A step takes input from previous step, gives output to next step (if any)
  - Exit criteria of a step must match with entry criteria of the succeeding step
- It follows specify, design, build sequence that is intuitively obvious and appears natural

Waterfall model for Development

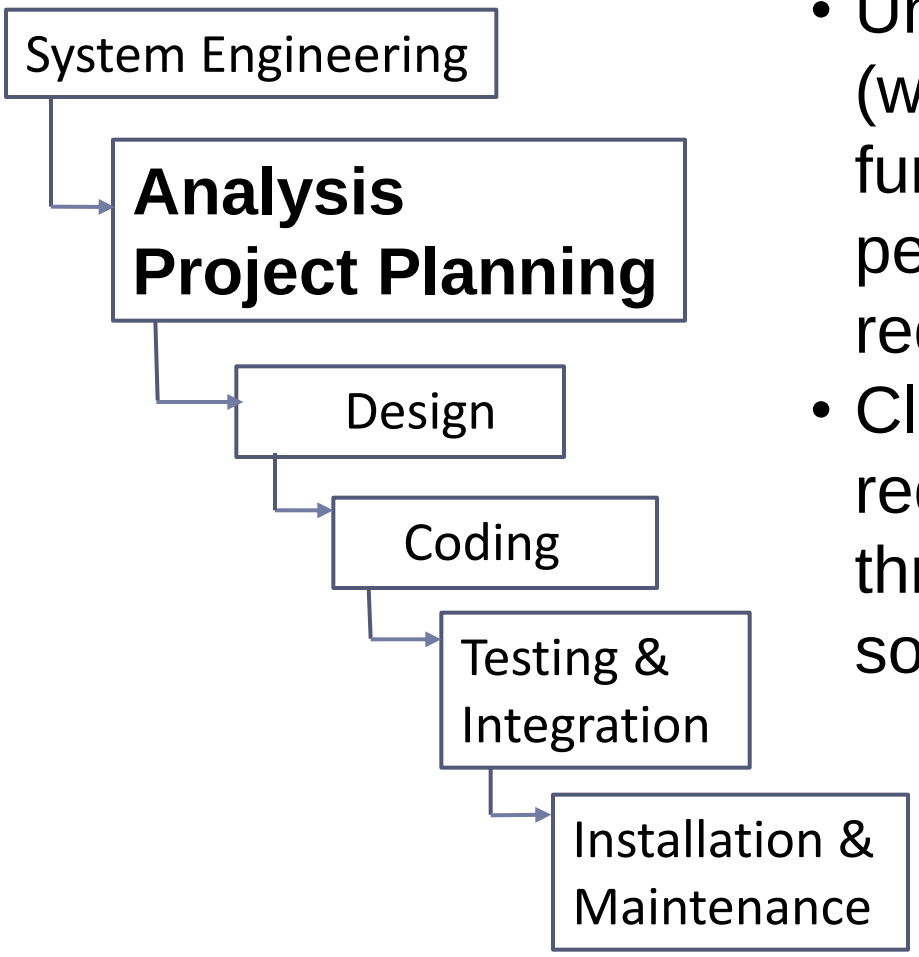
- It produces many intermediate deliverables, usually documents
  - Standard formats defined for these documents
  - Act as baseline used as reference (for contractual obligations of user, for maintenance)
  - Important for quality assurance, project management, etc.

Waterfall model



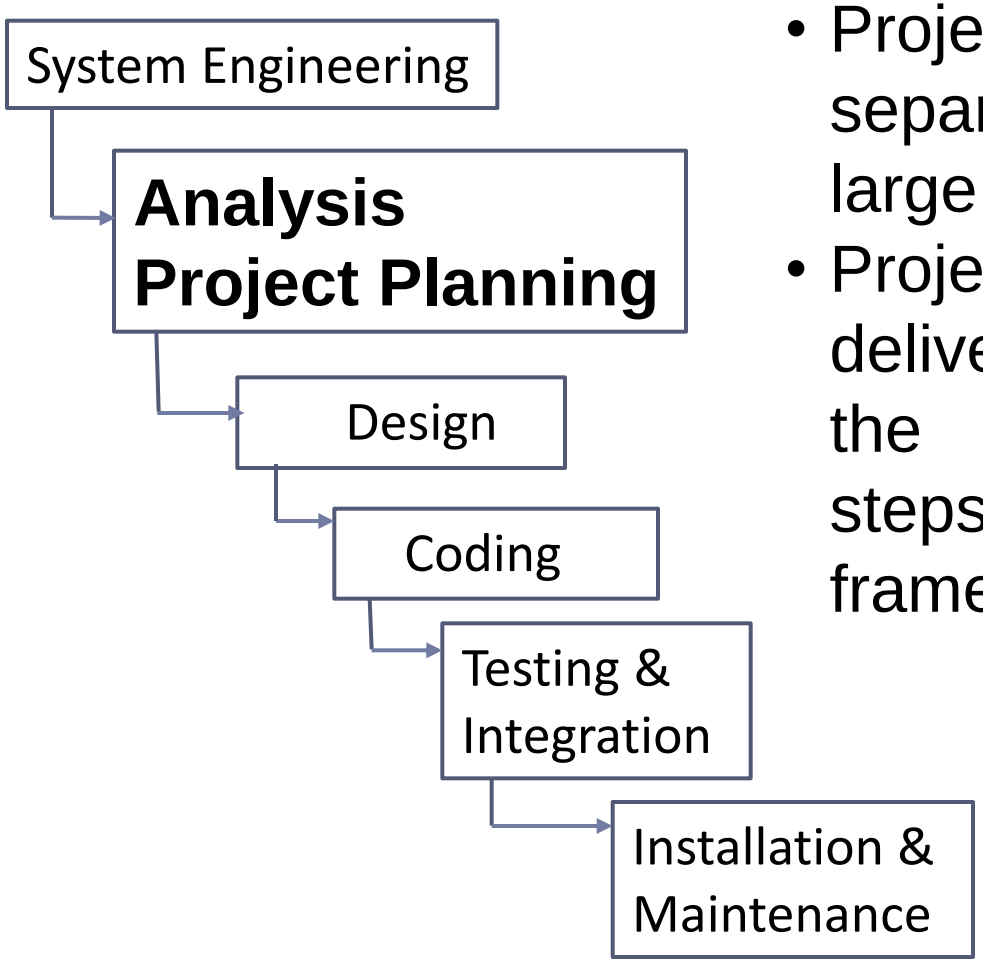
- Try to understand overall problem
- Establish requirements for all elements of the system, are to be handled through the software
- Some functions can not be performed by the software and those functions need to be performed manually
  - When the requirements are clearly defined, waterfall model is the ideal model for development

Waterfall model



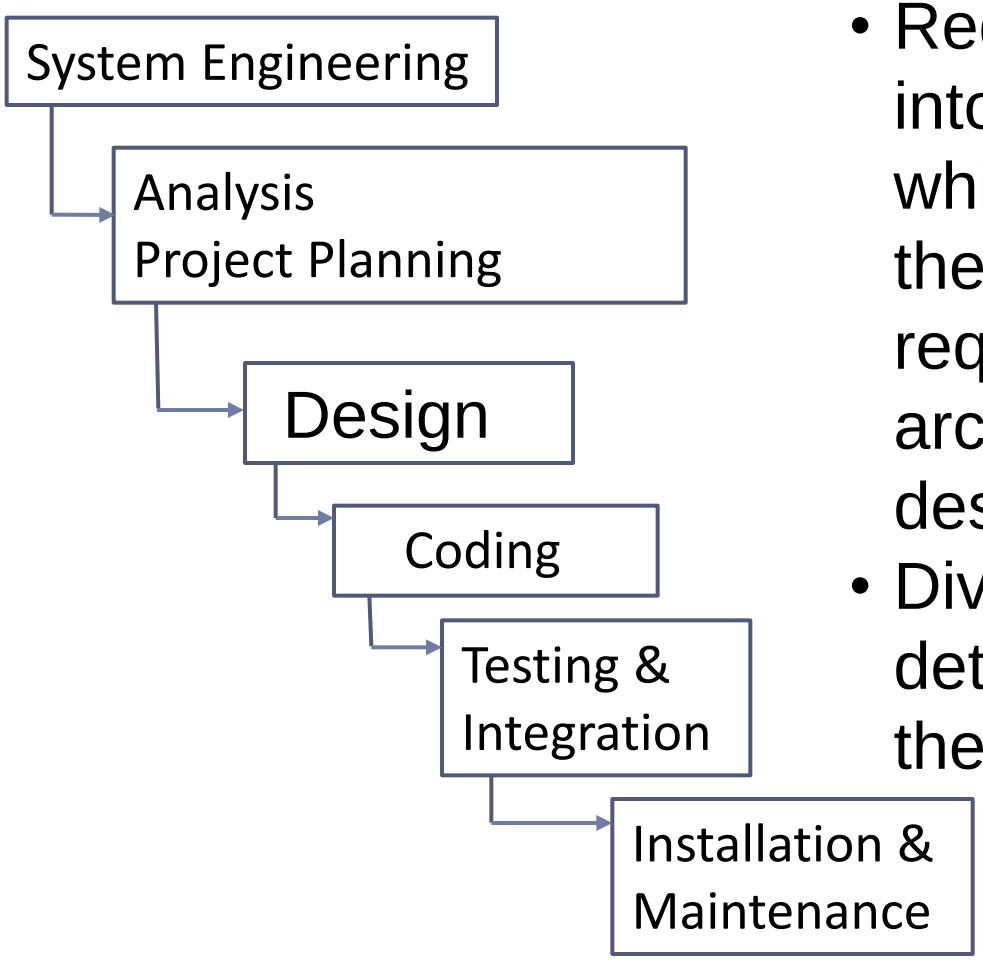
- Understand Information domain (what data is to be used), what function is to be performed, performance and interfacing requirement.
- Clearly define/identify the requirements which are to be met through the development of the software

Waterfall model



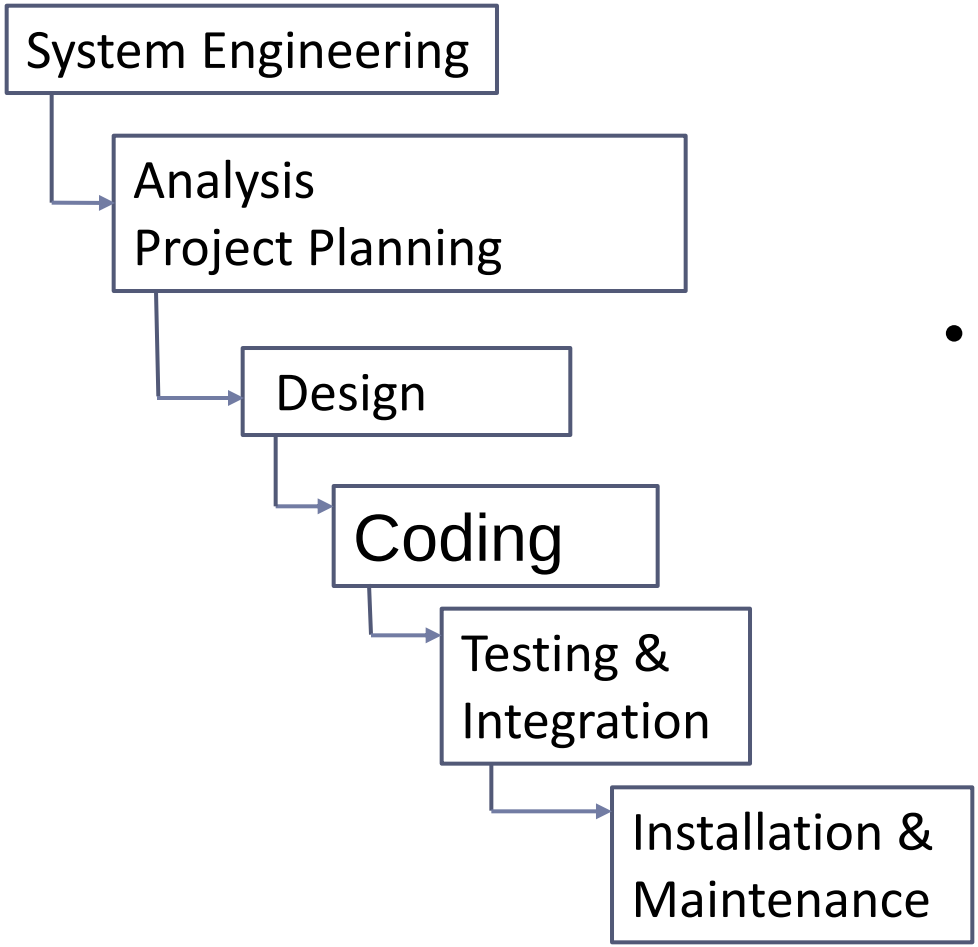
- Project planning can be made as separate step if the project is large
- Project planning-what are deliverables, how we will carry out the project, what are different steps, what would be the time frame and resources allotted

Waterfall model



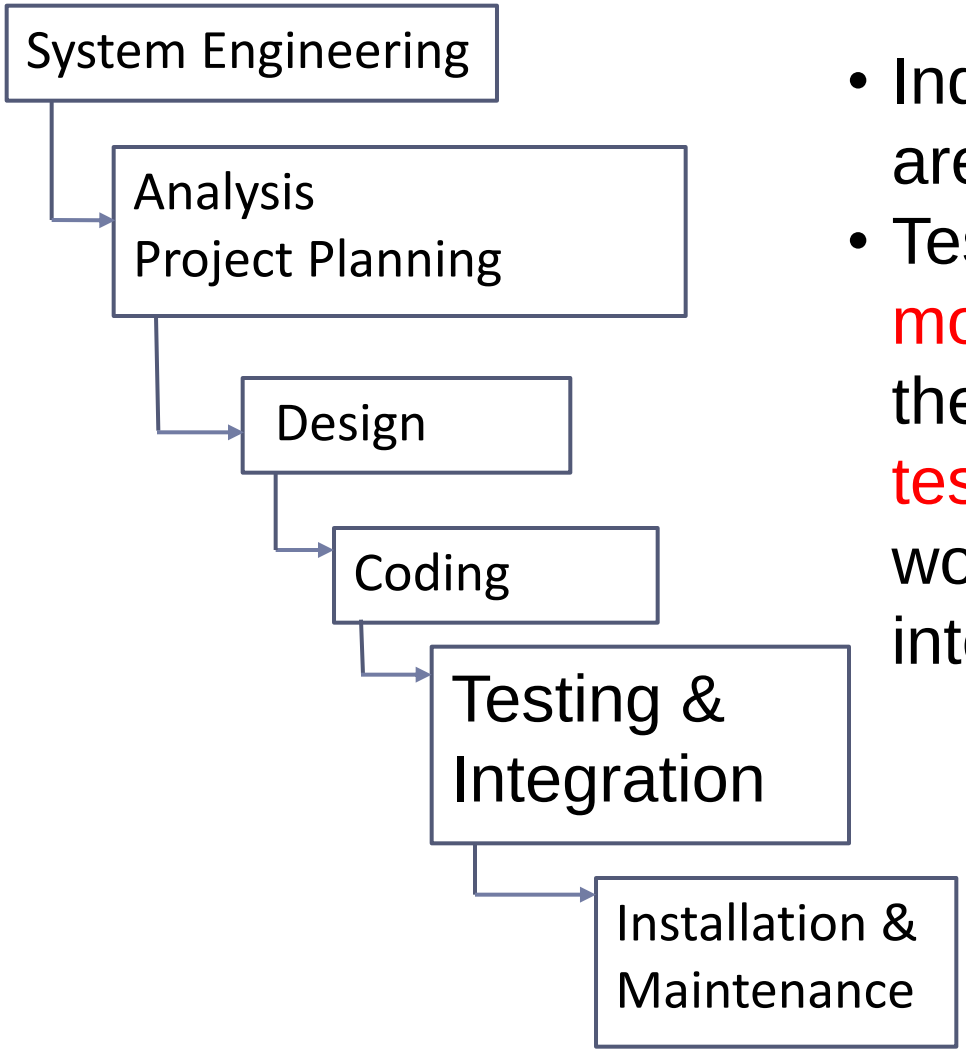
- Requirements are translated into implementation framework, which will be implemented in the next phase. (Translate the requirement into software architecture, prepare database design etc.)
- Divided into high level and detailed design depending on the complexity of the task

Waterfall model



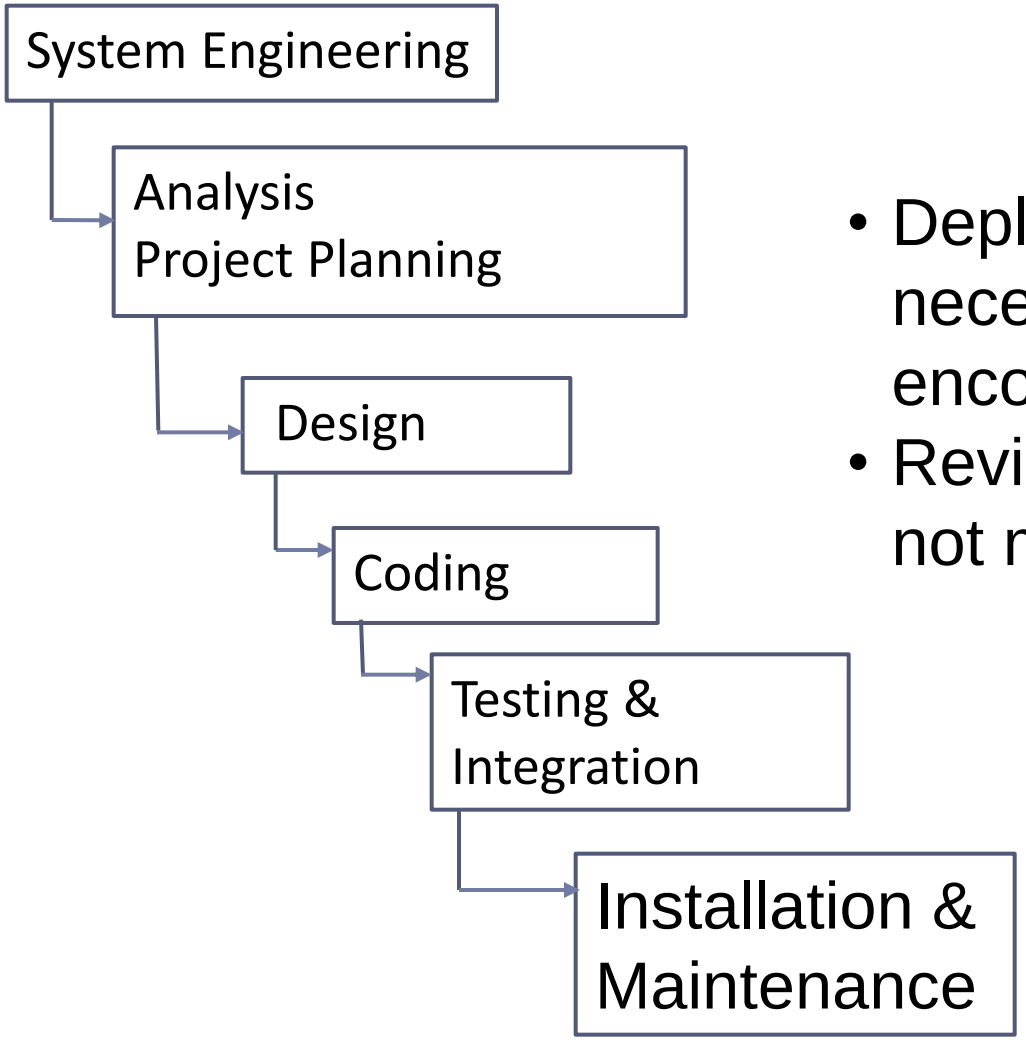
- Its all about Programming.

Waterfall model



- Individual pieces of software are tested
- Test first the logic of individual module and when we put all of them together then we want to test interfaces so that they work collectively that they are intended to work

Waterfall model



- Deployment and make necessary changes if error encountered
- Review it if performance is not met

Deliverables in waterfall model

- **Project plan** (it may be modified at later time) and feasibility report (defines cost effectiveness of the whole project, whether it is good investment we are making, giving kind of benefit we expect from the software)
- **Requirements document** (SRS: Software Requirement Specification)
  - It is an important baseline and defines the functions that the software will perform for us.
  - This SRS output that comes from the analysis step

Deliverables in waterfall model

- **System design document**: It defines the different components of the software and how they will be implemented

- **Detailed design document**: specify individual units which will be implemented by different programmers

Deliverables in waterfall model

- **Test plans and reports**
  - Test plans are made in advance and test plans are carried out as soon as the software is ready
  - What input should be given to the system and what outputs are expected
  - Whether the expected outputs are correct and the performance is as expected
- **Source code**
  - It is in the form of executable unlike other deliverables that comes in the form of documents

Deliverables in waterfall model

- **Software manuals** (user manuals, installation manuals)
- **Review Reports** (after each step)
  - Identify the shortcomings, defects are detected and were properly addressed
  - A catalogue of how the development went through
  - It is an important input to the management and development team

Cost/Effort distribution

- Accumulated cost (from system engineering to installation/maintenance phase) increases dramatically as programmers, operators, technical writers and computer time have committed. (we need to fix the errors as early as possible because everyone has spent enough time)
- Detailed design, Implementation and maintenance phases requires more cost
- Cost of discovering and fixing errors also increases with steps

Shortcomings of Waterfall model

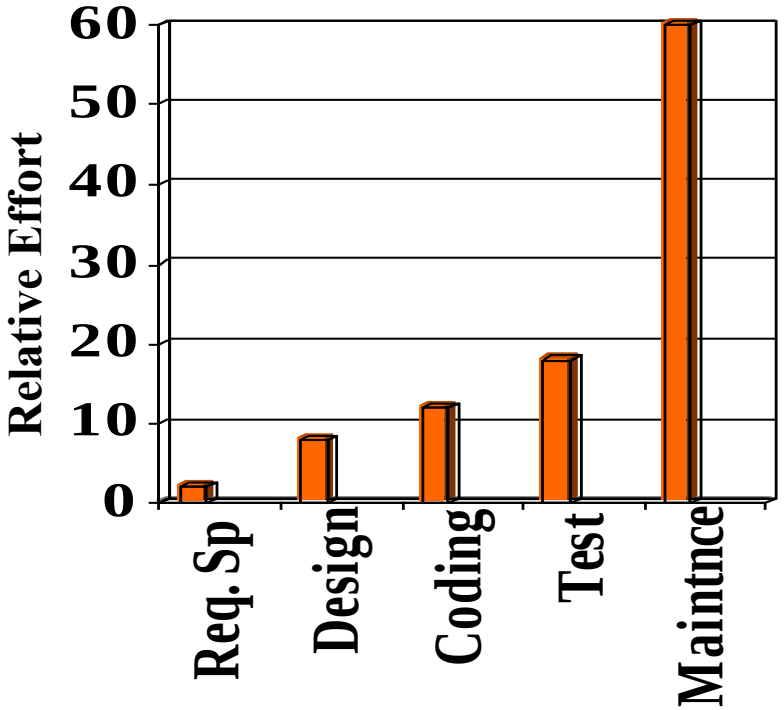
- Requirements may not be clearly known, especially for applications not having existing (manual) counterpart
  - Railway reservation: manual system is existed, so SRS can be defined
  - Online-container management for railways-new
  - Knowledge management for a central bank-new

Shortcomings of Waterfall model

- It assumes that the requirements are stable but Requirements change with time during project life cycle itself
  - Users may find a solution of little use
  - Better to have development in parts in smaller increments and get the feedback from user; this is common for packages, system software
- Considered documentation-heavy: so much documentation may not be required for all type of projects.

Relative Effort for Phases

- Phases between feasibility study and Maintenance
  - 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,
  - 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:
  - 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
  - important requirements of the customer need to be understood
  - screen layouts in GUI, specific algorithms required and data base schema to be used are ignored.
- Formulate different solution strategies
  - high-level solution schemes to the problem are determined
    - For example, client-server framework or standalone framework to be explored

Activities during Feasibility Study

- 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.
  - You may determine that none of the solutions is feasible due to:
    - high cost
    - resource constraints
    - technical reasons

Requirements Analysis and Specification

- 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:
  - analyse the collected data to clearly understand what the customer wants,
  - find out any inconsistencies and incompleteness in the requirements,
  - resolve all inconsistencies (requirements contradicts) and incompleteness (parts of requirement omitted).

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.

Requirements Analysis (CONT.)

- Ambiguities and contradictions:
  - must be identified
  - resolved by discussions with the customers.
- Next, requirements are organized:
  - into a Software Requirements Specification (SRS) document.
- Engineers doing requirements analysis and specification:
  - are designated as analysts.

Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.

Design

- 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

- Purpose of implementation phase (aka coding and unit testing phase):
  - translate software design into source code.

Implementation

- During the implementation phase:
  - each module of the design is coded
  - each module is unit tested
    - tested independently as a standalone unit, and debugged,
  - each module is documented.

Implementation (CONT.)

- 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

System Testing

- 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.

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 (CONT.)

- **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.

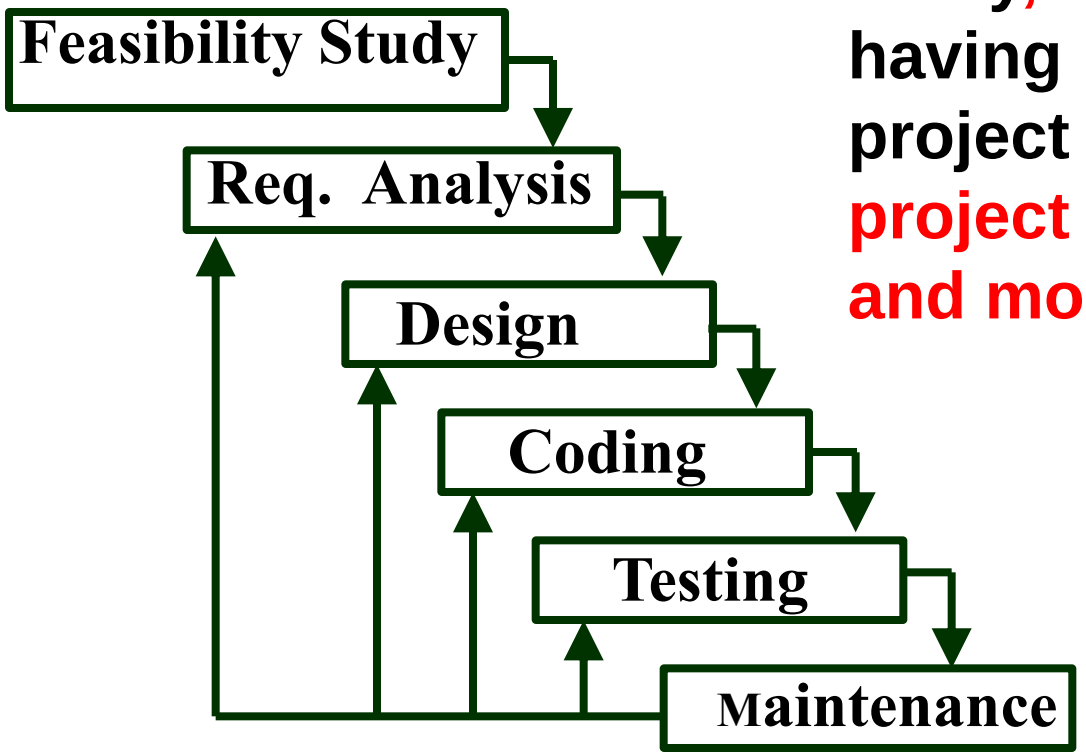
Iterative Waterfall Model

- 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.

Iterative Waterfall Model (CONT.)

- 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 (CONT.)



- **No path to the feasibility study, because once a team having accepted to take up a project does not give up the project easily due to legal and moral reasons.**

Iterative Waterfall Model (CONT.)

- Errors should be detected
  - in the same phase in which they are introduced.

- For example:
  - 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.

Iterative Waterfall Model (CONT.)

- Irrespective of the life-cycle model actually followed:
  - the documents should reflect a classical waterfall model of development,
  - comprehension of the documents is facilitated.

Iterative Waterfall Model (CONT.)

- Metaphor of mathematical theorem proving:
  - A mathematician presents a proof as a single chain of deductions,
    - even though the proof might have come from a convoluted set of partial attempts, blind alleys and backtracks.

Phase overlap

- In practice the activities of different phases overlap due to two main reasons:

  Reason 1:

  - Some errors still escape detection and are detected in a later phase.
  - Some already completed phases to be reworked
  - Activities pertaining to a phase do not end at the completion of the phase due to rework, actually overlap with other phases.

Phase overlap

Reason 2:

- If strict phase transitions are maintained
  - Team members who complete their work early would idle waiting for the completion of the phase (called blocking state). Therefore, phases are allowed to overlap.
- Team member who completes his assignment is allowed to proceeds to start the work for next phase.

Shortcomings of iterative waterfall model

- Difficult to accommodate change request:
  - Once requirements have been frozen, this model provides no scope for modifications to the requirements.
  - Requirement changes arise due to a variety reasons – requirements were not clear to the customer, misunderstood, business process of the customer may have changed after the SRS was signed off.
  - The assumption made in this model that the methodical requirements gathering and analysis alone would comprehensively and correctly identify all the requirements by the requirements phase is flawed.

Shortcomings of iterative waterfall model

- Incremental delivery not supported
  - By the time software is delivered, installed, and become ready for use, the customer's business process might have changed
- Phase overlap not supported:
  - Strict adherence to waterfall model creates blocking states.
- Limited customer interactions
  - Interaction only at the start of the project and at completion

Shortcomings of iterative waterfall model

- Error correction unduly expensive
  - Validation is delayed till the complete development
  - Defects that are noticed at the time of validation
- Heavy Weight
  - Significant portion of the time of the developers is spent in preparing documents and revising them as changes occur over the life cycle.
- No support for risk handling and code reuse

Prototyping Model

- Before starting actual development,
  - a working prototype of the system should first be built.
- A prototype is a toy implementation of a system:
  - limited functional capabilities,
  - low reliability,
  - inefficient performance.

Prototyping model

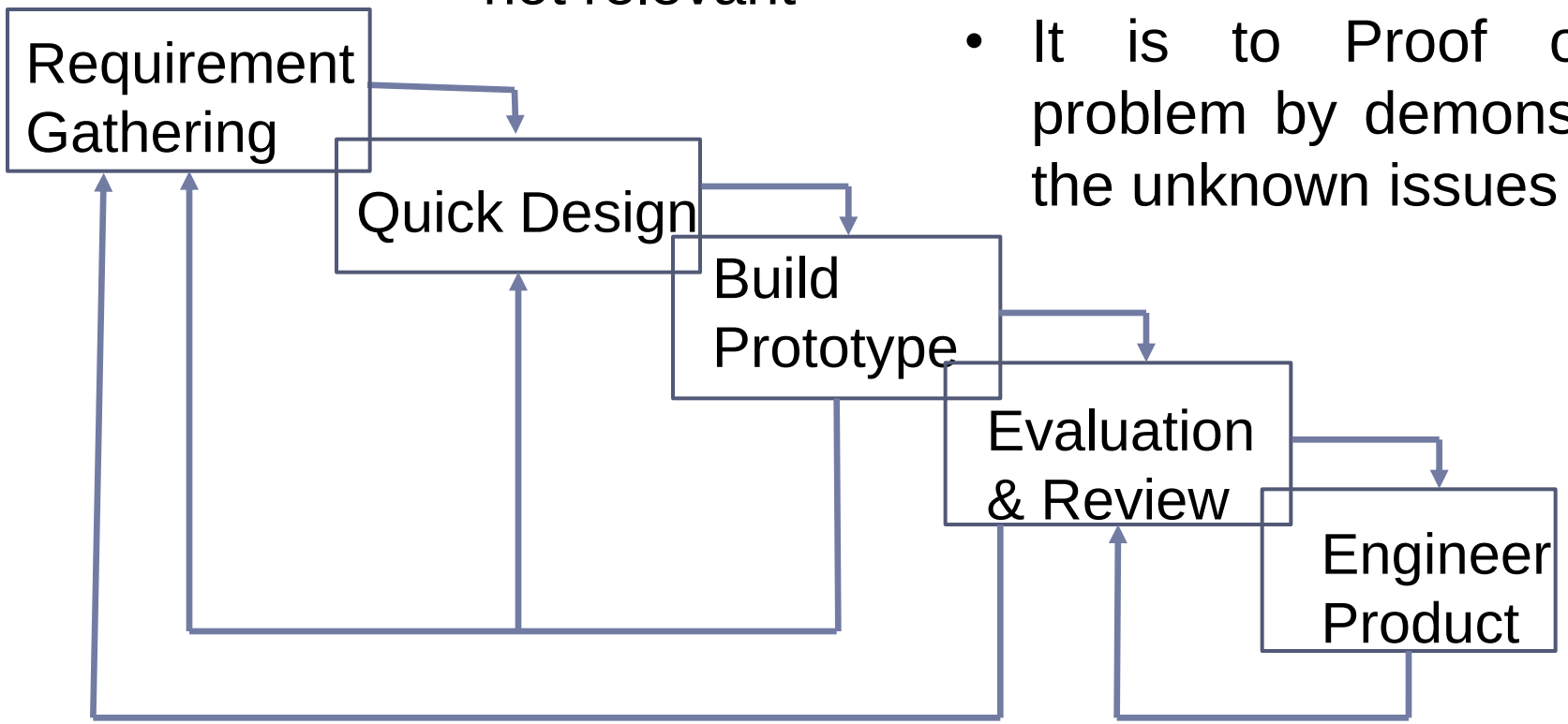
- When the customer or the developer is not sure (we go for a prototype first when the customer is not aware of what his/her requirements)
  - of requirements (inputs, outputs)
  - of algorithms, efficiency (build a system which will give high response time and what techniques will give that kind of guaranteed response), human machine interaction
- A throwaway prototype built from currently known user needs
- Working or even paper prototype

Prototyping…

- Quick design focuses on aspects visible to user
  - features clearly understood need not be implemented
- Prototype is tuned to satisfy customer needs
  - Many iterations may be required to incorporate changes and new requirements
- Final product follows usual define-design-build-test life cycle

Prototyping..

- Doing prototype, we don't care about the documentation, maintainability issues, even don't consider performance issue if they are not relevant
- It is to Proof of the problem by demonstrating the unknown issues



Reasons for developing a prototype

- Illustrate to the customer:
  - input data formats, messages, reports, or interactive dialogs.
- Examine technical issues associated with product development:
  - Often major design decisions depend on issues like:
    - response time of a hardware controller,
    - efficiency of a sorting algorithm, etc.

Prototyping Model (CONT.)

- The third reason for developing a prototype is:
  - it is impossible to "get it right" the first time,
  - we must plan to throw away the first product
    - if we want to develop a good product.

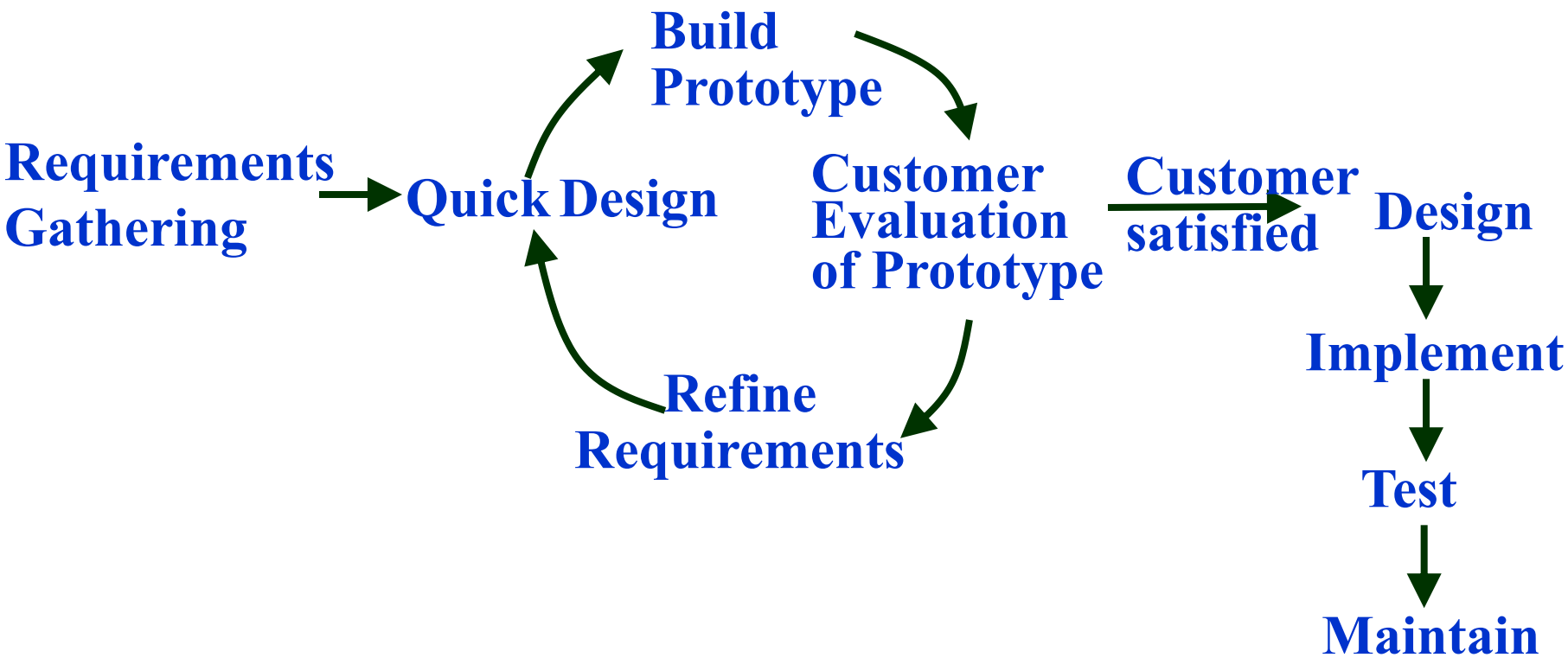
Prototyping Model (CONT.)

- Start with approximate requirements.
- Carry out a quick design.
- Prototype model is built using several short-cuts:
  - Short-cuts might involve using inefficient, inaccurate, or dummy functions.
    - A function may use a table look-up rather than performing the actual computations.

Prototyping Model (CONT.)

- The developed prototype is submitted to the customer for his evaluation:
  - Based on the user feedback, requirements are refined.
  - This cycle continues until the user approves the prototype.
- The actual system is developed using the classical waterfall approach.

Prototyping Model (CONT.)



Prototyping Model (CONT.)

- Requirements analysis and specification phase becomes redundant:
  - final working prototype (with all user feedbacks incorporated) serves as an animated requirements specification.
- Design and code for the prototype is usually thrown away:
  - However, the experience gathered from developing the prototype helps a great deal while developing the actual product.

Prototyping Model (CONT.)

- Even though construction of a working prototype model involves additional cost--- overall development cost might be lower for:
  - systems with unclear user requirements,
  - systems with unresolved technical issues.

- Many user requirements get properly defined and technical issues get resolved:
  - these would have appeared later as change requests and resulted in incurring massive redesign costs.

Strengths

- Appropriate for projects that suffer from technical and requirements risks.
- Constructed prototype helps overcome these risks.

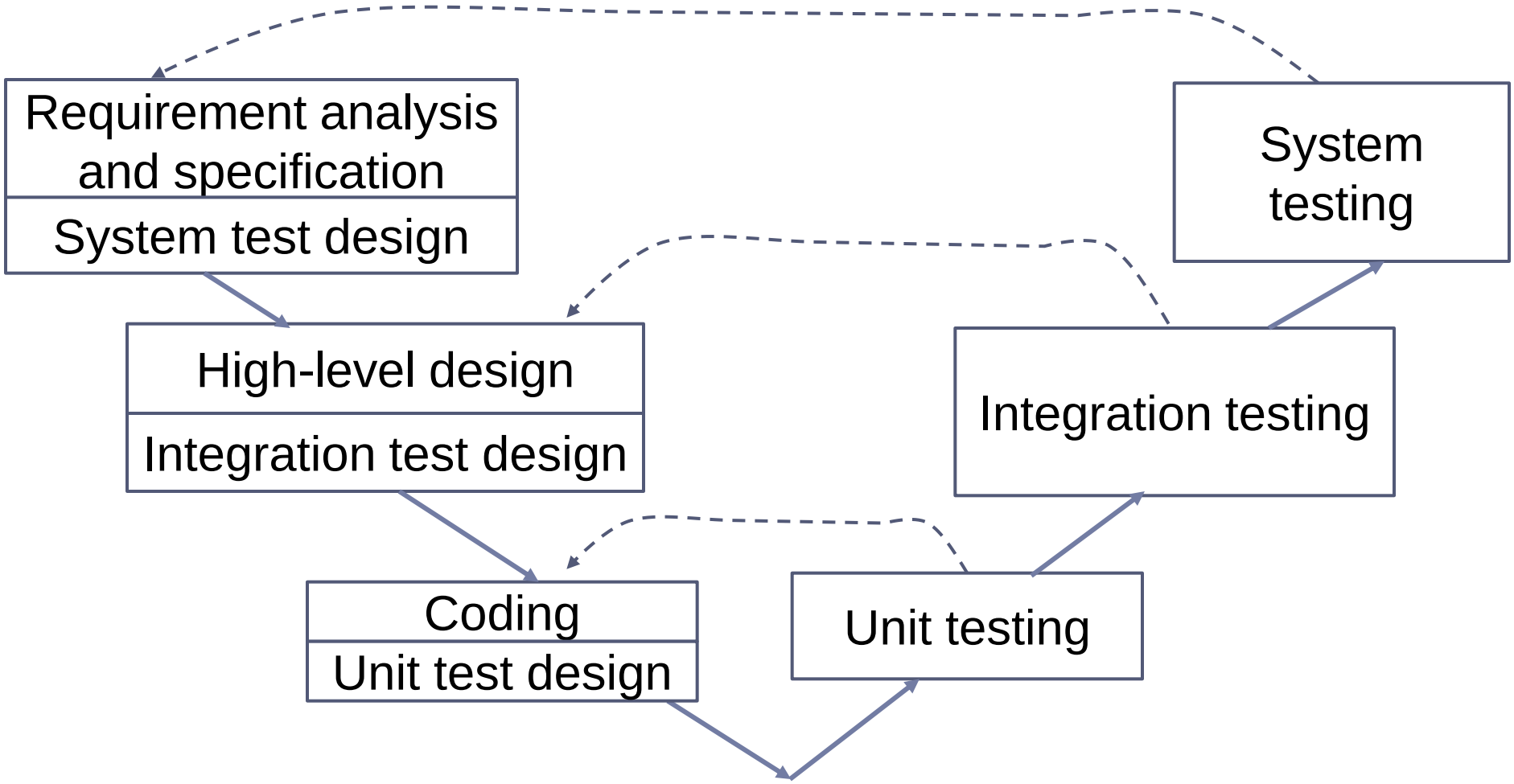
Limitations of prototyping

- Costumer may want the prototype itself! (happy with the prototype)
- Developer may continue with implementation choices made during prototyping
- May not give required quality and performance (when continue with the prototype)
- Good tools need to be acquired for quick development
- May increase project cost
- Would not be appropriate for projects for which the risks can only be identified after the development is underway.

V-Model

- It is a variant of waterfall model
- Validation and Verification activities are carried out throughout the development life cycle (considerably reduce the number of bugs in the work products)
- Use in projects concerned with the development of safety-critical software that are required to have high reliability
- Two phases:
    1. development phases (left half of the model)
    2. validation phases (right half of the model)

V-Model


Requirement analysis and specification
System test design
High-level design
Integration test design
Coding
Unit test design
Unit testing
Integration testing
System testing

V-model

- In each development phase, along with the development of a product, test case design and the plan for testing the product are carried out, whereas the actual testing is carried out in the validation phase.
  - This validation plan created during the development phases is carried out in the corresponding validation phase.
- In the validation phase, three different steps of testing during the validation phase are performed to detect defects that arise in the corresponding phases of software development

V-model versus waterfall model

- Iterative waterfall model where testing activities are confined to the testing phase only, whereas, in the V-model testing activities are spread over the entire life cycle.

Advantages of V-model

- Testing activities are carried out in parallel with the development activities. This model usually leads to a shorter testing phase and an overall faster product development as compared to the iterative model.

- Test cases are designed before the schedule pressure built up and leads to the quality of test cases.

Advantages of V-model

- Test team is reasonably kept occupied throughout the development cycle. This leads to more efficient manpower utilization.

- Test team is associated with the project from the beginning.
  - They build up a good understanding of the development artifacts, and this helps them to carry out effective testing of the software.

Disadvantage of V-model

- Being a variant of the classical waterfall model, this model inherits most of the weaknesses of the waterfall model.

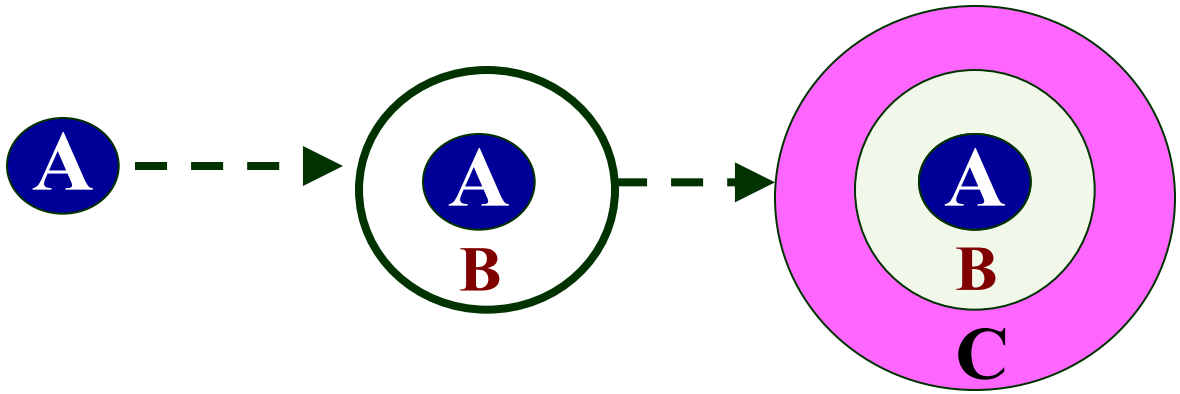
Incremental Development Model

- Incremental model (aka successive versions):
  - The system is broken down into several modules which can be incrementally implemented and delivered.
- First develop the core modules of the system.
- The initial product skeleton is refined into increasing levels of capability:
  - by adding new functionalities in successive versions.

Incremental Development Model

- Successive version of the product:
  - functioning systems capable of performing some useful work.
  - A new release may include new functionality:
    - also existing functionality in the current release might have been enhanced.

Incremental Development Model



- A,B,C are modules of a software product that are incrementally developed and delivered.

Incremental Development Model

- After the requirements gathering and specification, the requirements are split into several versions.

- Starting with the core version (version 1), in each successive increment, the next version is constructed using an iterative waterfall model of development and deployed at the customer site, the full software is deployed.

- At any time, plan is made only for the next increment and no long-term plans are made.

Advantages of Incremental Development Model

- Saves the developing organization from deploying large resources and manpower for a project in one go.
- Users get a chance to experiment with a partially developed system:
  - much before the full working version is released,
- Helps finding exact user requirements:
  - much before fully working system is developed.
- Core modules get tested thoroughly:
  - reduces chances of errors in final product.

Disadvantages of Incremental Development Model

- Often, difficult to subdivide problems into functional units:
  - which can be incrementally implemented and delivered.
  - It is useful for very large problems
    - where it is easier to find modules for incremental implementation.

Evolutionary Model

- Many organizations use a combination of iterative and incremental development:
  - a new release may include new functionality
  - existing functionality from the current release may also have been modified.

Evolutionary Model

- In incremental development model, complete requirements are first developed and SRS document prepared.

- In contrast, in evolutionary model,
  - The requirements, plan, estimates and solution evolve over the iterations, rather than fully defined and frozen in a major up-front specification effort before the development iterations begin.
  - Referred to as ***design a little, build a little, test a little, deploy a little*** model

Advantages of Evolutionary Model

- Training can start on an earlier release
  - customer feedback taken into account
- Markets can be created:
  - for functionality that has never been offered.
- Frequent releases allow developers to fix unanticipated problems quickly.
- The change requests after delivery of the complete software substantially reduced.
- Handling change requests are easier.

Disadvantages of Evolutionary Model

- Feature division into incremental parts can be non-trivial
  - For small projects, it is difficult to divide the required features into several parts
  - Even for larger projects, expert would require mode efforts to plan the incremental deliveries because features are intertwined and dependant on each other

- Ad hoc design, only attention is paid to current increment therefore less attention towards the maintainability and optimality.

Spiral Model

- Proposed by Barry Boehm in 1988.
- Each loop of the spiral represents a phase of the software process:
  - the innermost loop might be concerned with system feasibility,
  - the next loop with system requirements definition,
  - the next one with system design, and so on.
- There are no fixed phases in this model, the phases shown in the figure are just examples.

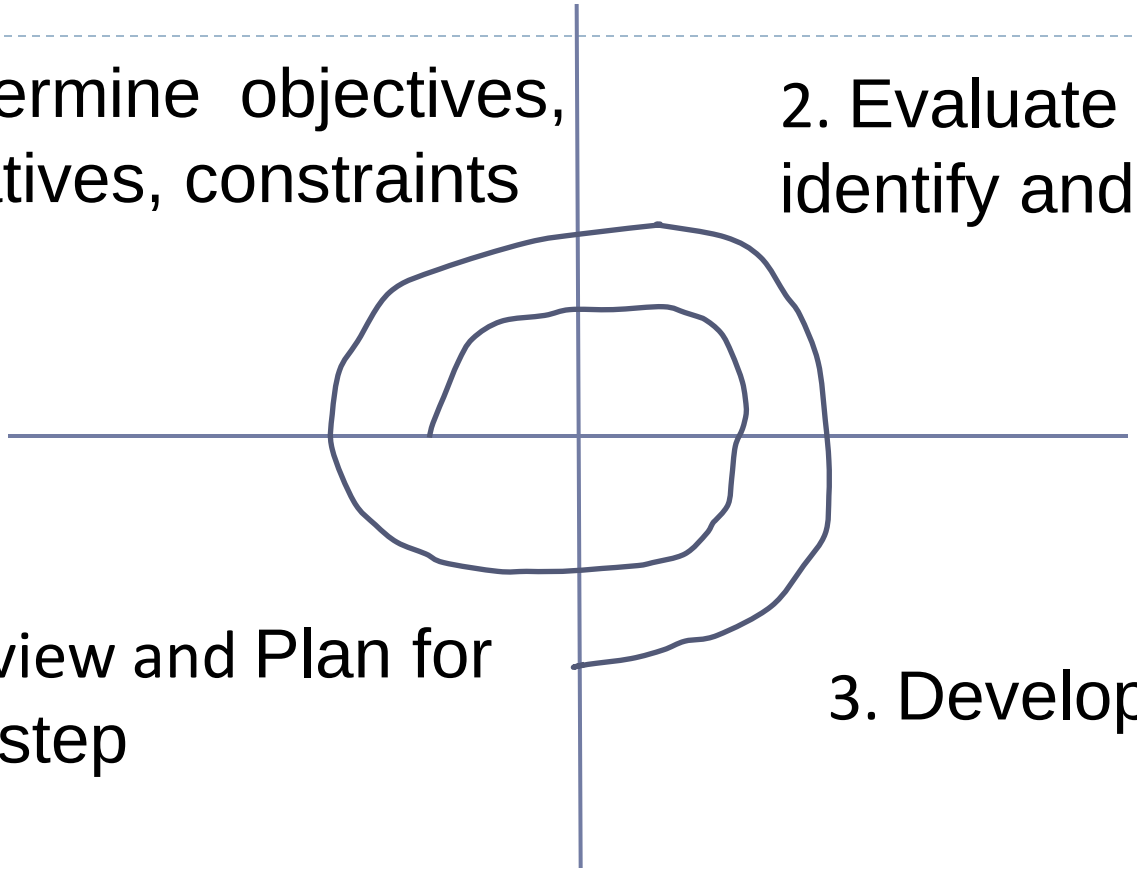
Spiral Model (CONT.)

- The team must decide:
  - how to structure the project into phases.
- Start work using some generic model:
  - add extra phases for specific projects or when problems are identified during a project.
- Each loop in the spiral is split into four sectors (quadrants).

Spiral Model

1. Determine objectives, alternatives, constraints

2. Evaluate alternatives, identify and handle risks



4. Review and Plan for next step

3. Develop the software

Spiral Model

- Activities are organized in a Spiral having many cycles.
- Four quadrants in each cycle
- Identifying the risks in the alternatives and handling the risks
- Prototyping, simulations, benchmarking may be done to resolve uncertainties/risks

Spiral Model

- Development step depends on remaining risks; e.g.,
  - Do prototype for user interface risks;
  - Use basic waterfall model when user interface and performance issues are understood but only development risks remains

- Risk driven: allows us mix of specification-oriented, prototype-oriented, simulation based or any other approach.

Spiral Model

- Risk perception of what kind of user interface we should build then we do prototyping (requirement risk) and
  - if user interfaces and performance is well understood then the risk just may be the development risk in that case we may decide to follow the waterfall model.

- It allows us to mix different models.

Objective Setting (First Quadrant)

- Identify objectives of the phase
- Examine the risks associated with these objectives.
  - Risk: any adverse circumstance that might hamper successful completion of a software project.
- Find alternate solutions possible.

Risk Assessment and Reduction (Second Quadrant)

- For each identified project risk,
  - a detailed analysis is carried out.
- Steps are taken to reduce the risk.
- For example, if there is a risk that the requirements are inappropriate:
  - a prototype system may be developed.

Spiral Model (CONT.)

- Development and Validation (Third quadrant):
  - develop and validate the next level of the product.
- Review and Planning (Fourth quadrant):
  - review the results achieved so far with the customer and plan the next iteration around the spiral.
- With each iteration around the spiral:
  - progressively more complete version of the software gets built.

Spiral Model as a meta model

- Subsumes all discussed models:
  - a single loop spiral represents waterfall model.
  - uses an evolutionary approach --
    - iterations through the spiral are evolutionary levels.
  - enables understanding and reacting to risks during each iteration along the spiral.
  - uses:
    - prototyping as a risk reduction mechanism
    - retains the step-wise approach of the waterfall model.

Comparison of Different Life Cycle Models

- Iterative waterfall model
  - most widely used model.
  - But, suitable only for well-understood problems.
- Prototype model is suitable for projects not well understood:
  - user requirements
  - technical aspects

Comparison of Different Life Cycle Models (CONT.)

- Evolutionary model is suitable for large problems:
  - can be decomposed into a set of modules that can be incrementally implemented,
  - incremental delivery of the system is acceptable to the customer.
- The spiral model:
  - suitable for development of technically challenging software products that are subject to several kinds of risks.