Software Process Models

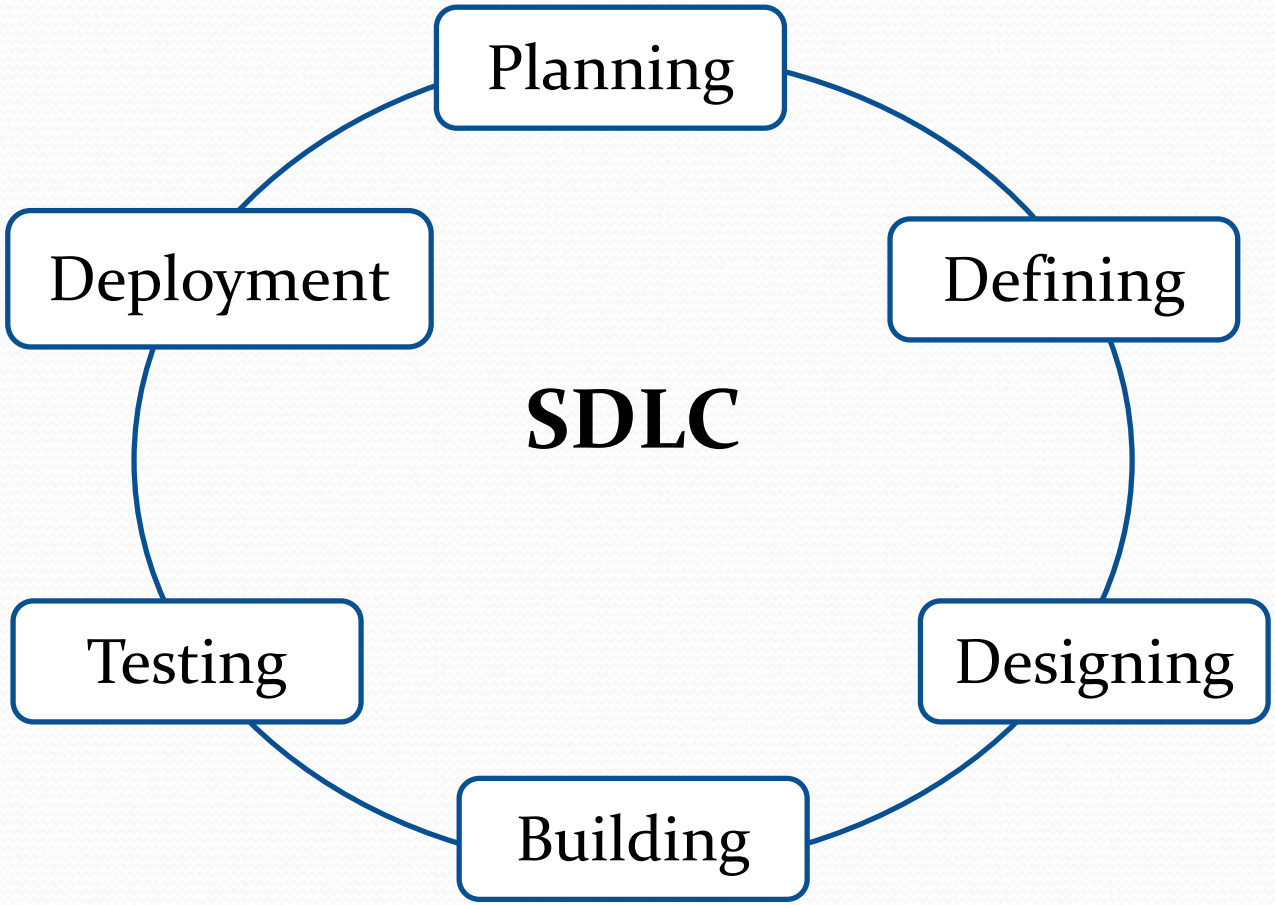
Outline

- Software process models/Software development life cycle (SDLC)
- The waterfall model
- The V-model
- The Spiral model
- Prototyping model
- The incremental and iterative models

Process and Software Process

- A **process** is a series of steps involving activities, constraints, and resources that produce an intended output of some kind
- **Software processes** are activities involved in producing a software system
- **Software process models** are abstract representations of these processes
- **Software development life cycle** is a structured plan for organizing the development of a software product, also called a software development process

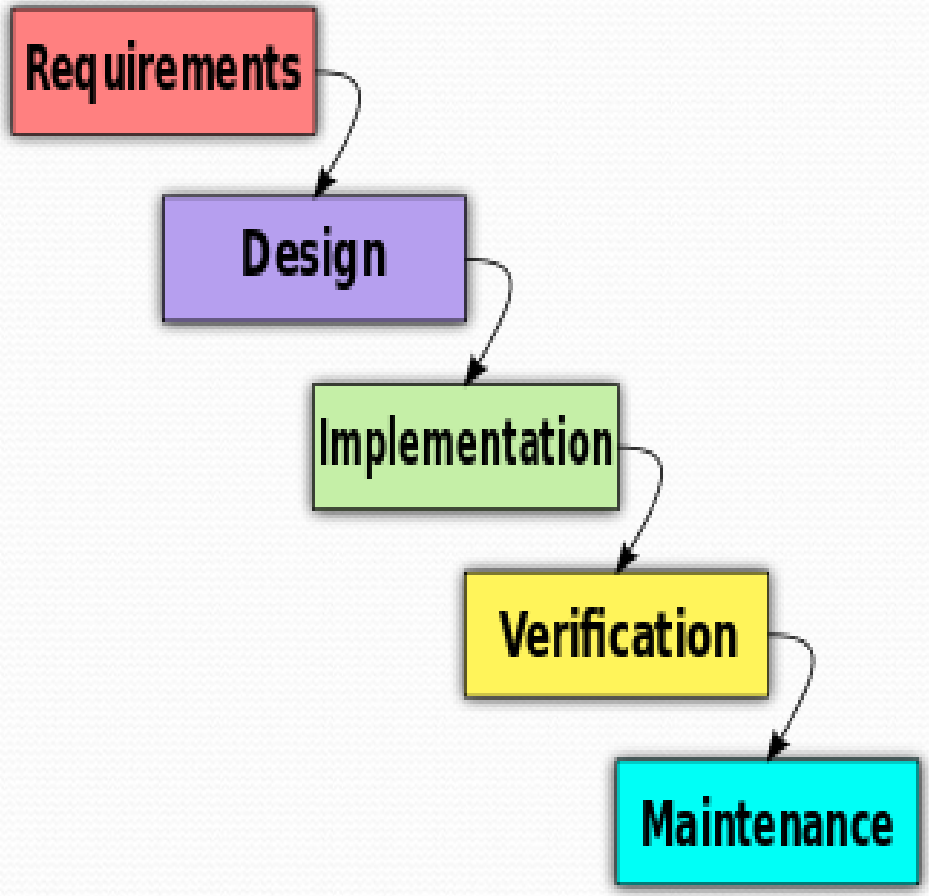
SDLC – Typical stages



The Waterfall model

- The classical model in SE (1970)
- Emphasize planning in early stage
- This model visualizes the software development process as a linear sequence of phases
- Non overlapping phases
- Intensive documentation

Waterfall Model Phases



Requirements Analysis phase

- Activities:
  - Meet with customer to understand requirements
  - Misinterpretation at this phase is costly later
  - Establish the system's services, constraints, and goals
  - All requirements are defined as a system specification
  - Requirement engineering process proposes activities, techniques in capturing and representing requirements
  - Outputs:
    - SRS – Software Requirements Specification Documentation (SRSD)
      - Template of SRS (IEEE or organizational template)

System and Software Design Phase

- Describes the fundamental system abstractions in term of modules, components and their relationships
- **System Design-Architectural Design**
  - Partition the requirements to hardware or software. Establish system architecture – inter-relation between modules – external interfaces and tools used.
  - Outputs:
    - ADD – Architectural (System) Design Doc (SDD)
      - IEEE template for ADD

System and Software Design Phase (contd.)

- **Detailed Design**
  - Description of the internal working of each software component
  - Develop data structures (e.g. database schema in DB), Algorithm details, UI for each component
  - For example: GUI (clients), database (server), application logic and business rules (middleware)
- Outputs:
  - DDD – Detailed Design Doc

Implementation Phase

- Design is translated into program code units.
  - E.g: Front end developer is responsible for client side programming (GUI, application logic). Back end developer is responsible for server side programming (database access and data processing)
- Outputs:
  - Program code

Testing Phase

- The individual program units and the integrated whole are methodically verified or tested to see that requirements(first step) are met
- After testing the software system is delivered to customer
- Outputs:
  - System Test Report, user manual, final system (coded and tested software)

Operation and maintenance

- Final Phase – acceptance testing
- The system is installed and put into practical use
- Visit client for routine maintenance: correcting errors/bugs
- New requirements discovered due to changing needs of customer!!
- Feedback loop in model can accommodate corrections

The waterfall model - advantages

- Simple and easy to manage and implement
- Good habits. Process visibility
  - Define-before-design, Design-before-code, etc.
- Documents are produced at each phase
  - Help to monitor the progress against the development plan
- It fits with other engineering process models
- There is an emphasis on documentation which keeps all knowledge in a central repository and can be referenced easily by new members joining the team

The waterfall model - problems

- Difficulty of accommodating change after the process is underway
- The following phase should not start until the previous phase has finished
- In practice, these stages overlap and feed information to each other
  - During design, problems with requirements are identified
  - During coding, design problems are found and so on…
- Due to cost of producing and approving documents, iterations are costly and involve significant costly rework
- Final product delivered late on in process

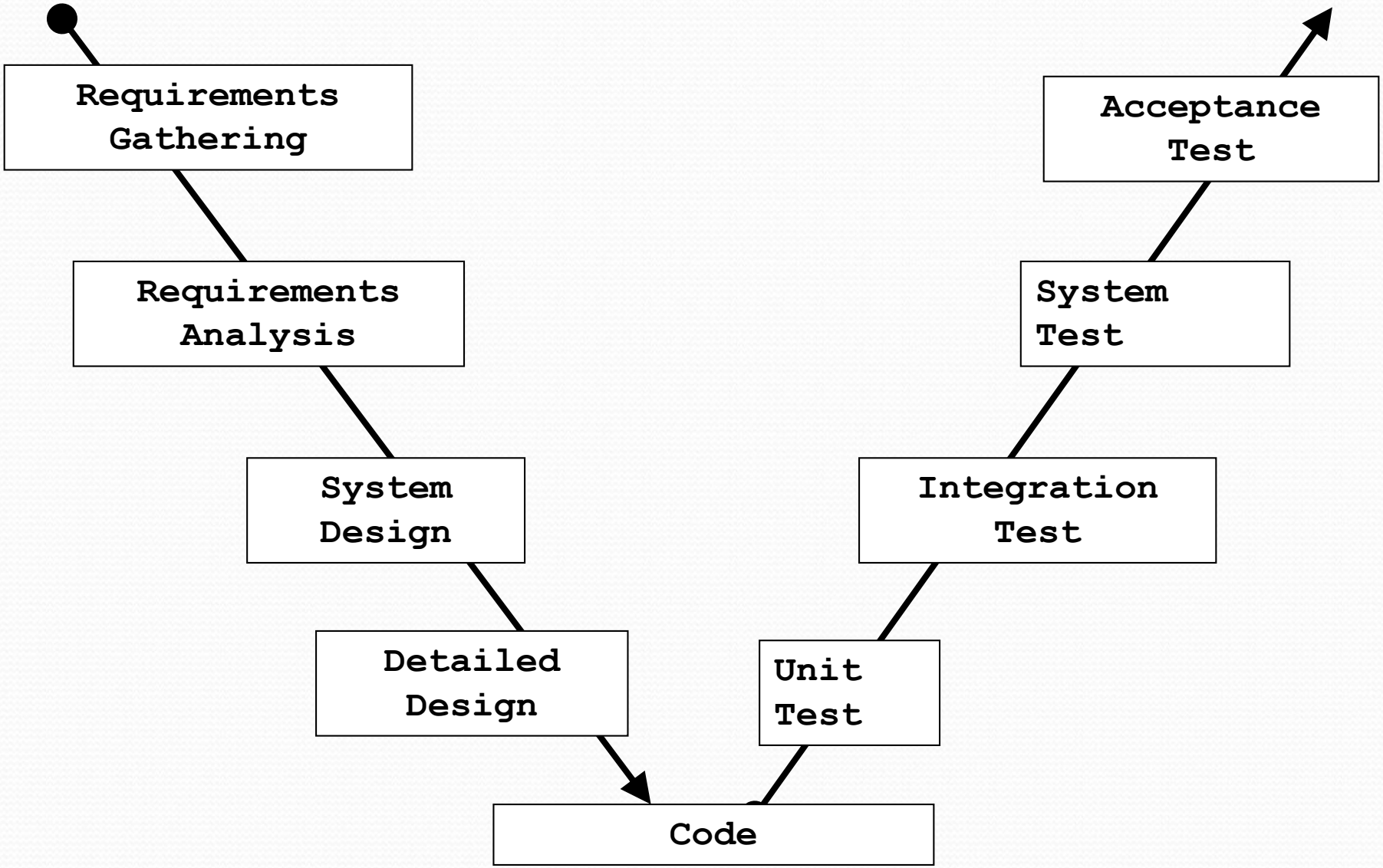
The waterfall model - problems

- Tests are only carried out at the end – this could mean a compromise if time or budgetary constraints exist
- Tend to test the system as a whole rather than systematically progressing from Unit testing to System testing
- After a small number of iterations it is normal to freeze development; problems are left for later resolution; may mean system won't do exactly what the user wants

The waterfall model

- Suitable within the projects:
  - Requirements are very well documented, clear and fixed (unlikely to change)
  - Product definition is stable
  - Technology is understood and is not dynamic.
  - Ample resources with required expertise are available to support the product.
  - The project is short.

V-Model


Requirements Gathering
Requirements Analysis
System Design
Detailed Design
Code
Unit Test
Integration Test
System Test
Acceptance Test

V-Model

- A variation of the waterfall model, it's also called Verification & Validation model as it emphasizes testing at different phases. The testing procedures are developed early in the life cycle before any coding is done, after each phase preceding implementation
- Verification: checking if the system conforms to its specification (doing things right).
- Validation: checking if the system meets the user expectations (doing right thing).
- In a diagram of the V-Model, the V proceeds down and then up

V-Model

- The model highlights the existence of different levels of testing and depicts the way each relates to a different development phase
- Once the implementation is finished testing begins, starting from unit testing and moving up one test level at a time until the acceptance testing phase is completed
- If problems are found during verification and validation, the left side of the V can be re-executed before testing on the right side is re-enacted

Testing types

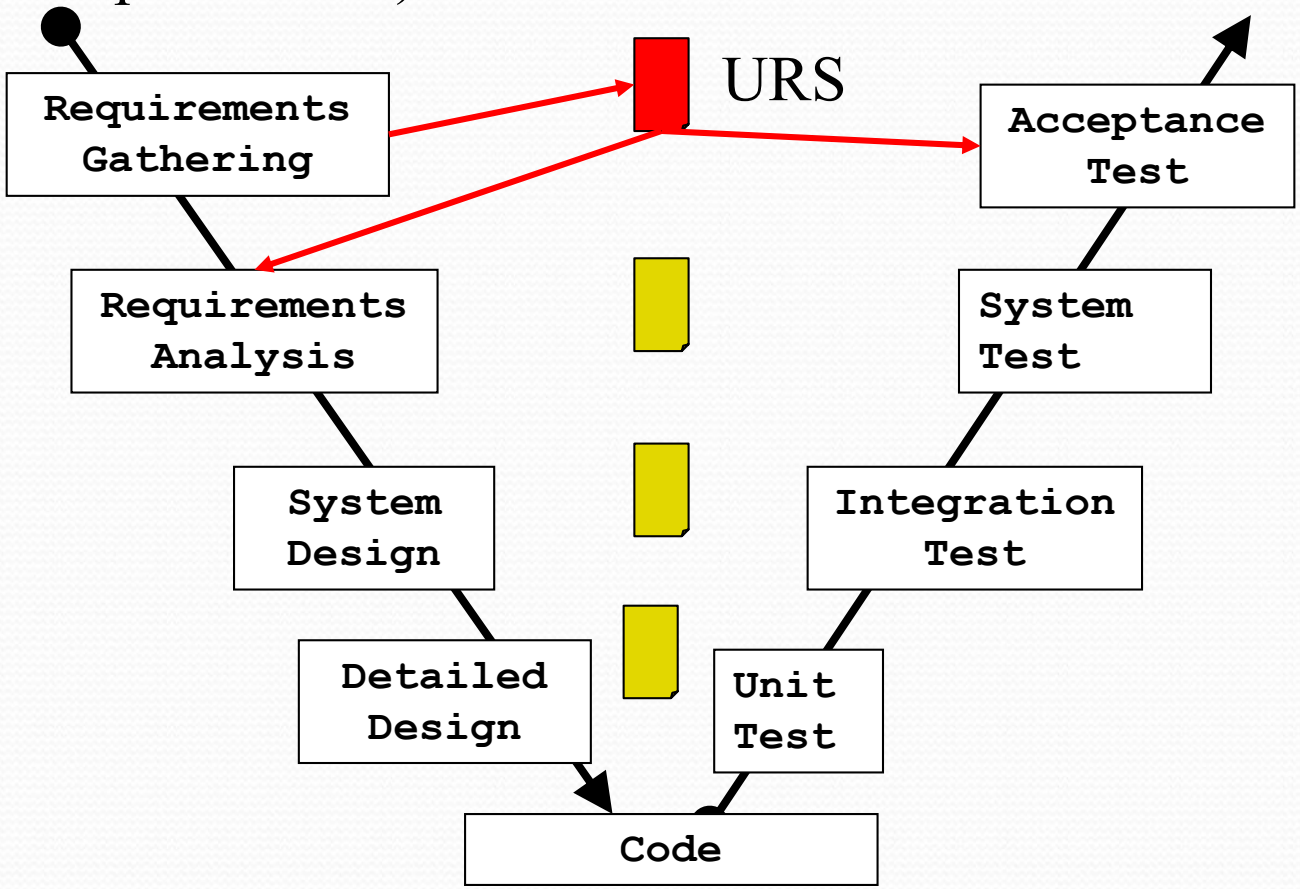
- Unit testing
  - An individual unit of software is tested to ensure that it works correctly.
- Integration testing
  - Two or more units are tested together to ensure that they interoperate correctly.
- System testing
  - The entire software system is tested to make sure that it works correctly and that it meets/solves the user's needs/problem.

Testing types

- Acceptance testing
  - The entire software system is tested at the customer environment (field testing or alpha and beta testing) to make sure that it meets the user's needs

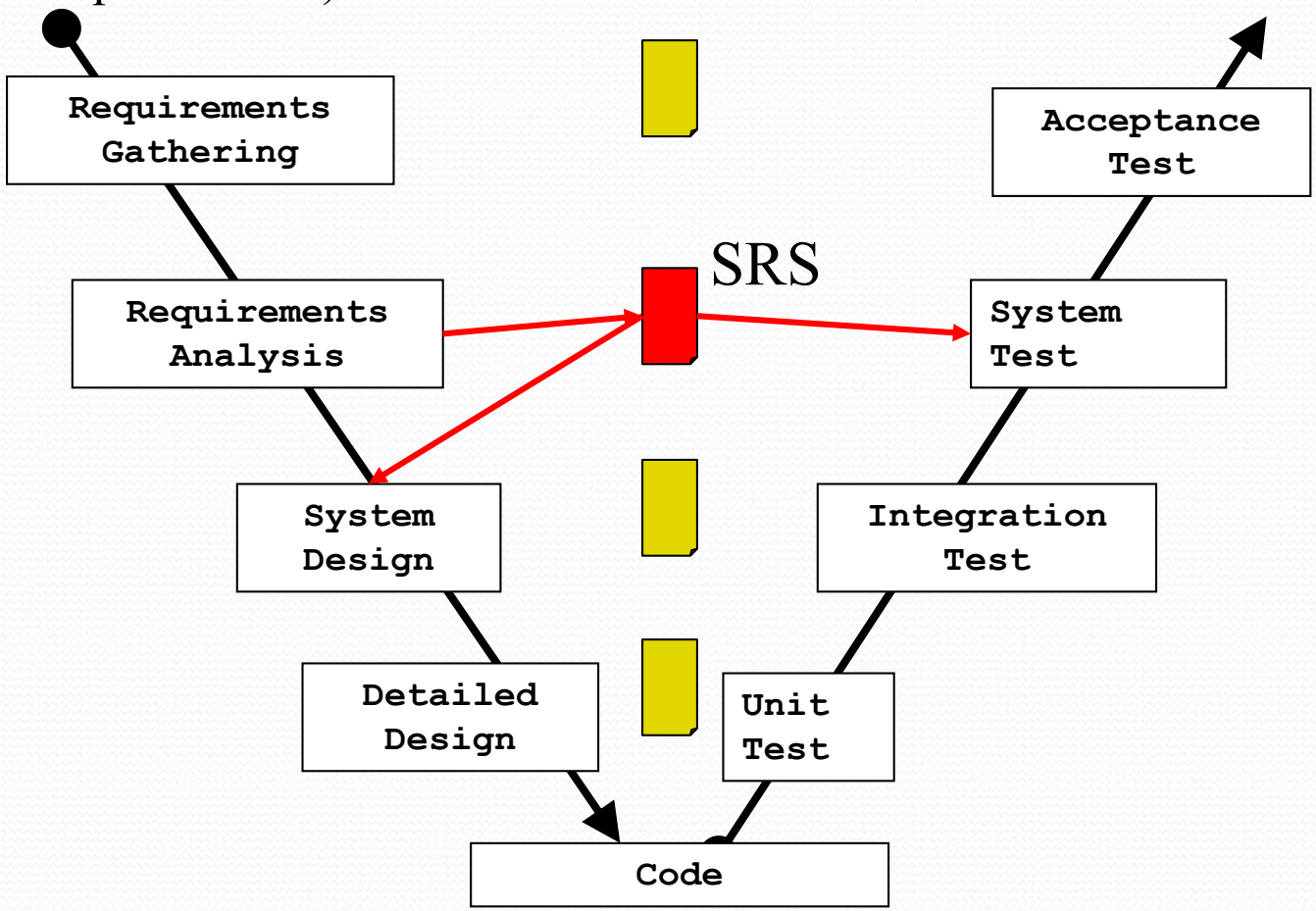
User Requirements

Requirements Gathering produces the User Requirements Specification (URS), which is both the input to Analysis, and the basis for Acceptance Testing (produce Acceptance testing plan and procedures).



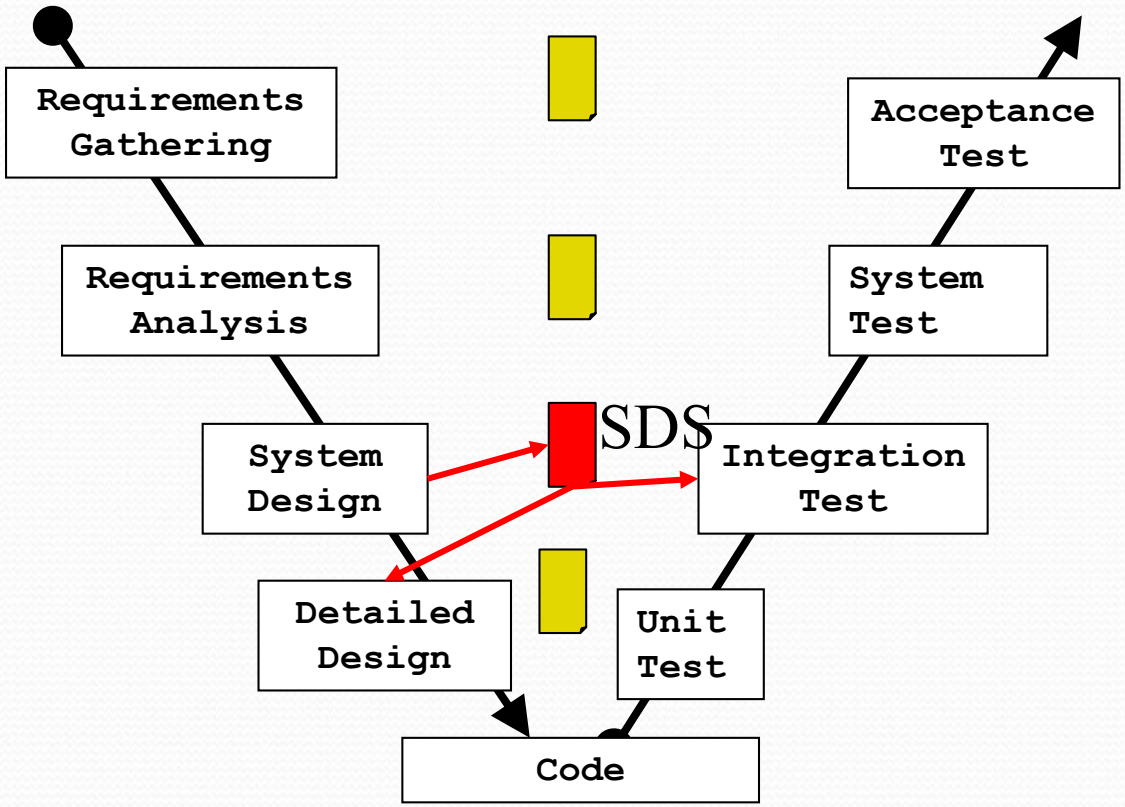
Requirements Analysis

Requirement Analysis produces the System Specification (SS) – also know as the Software Requirements Specification (SRS) – which is both the input for Software Design, and the basis for System Testing (produce system testing plans and procedures)



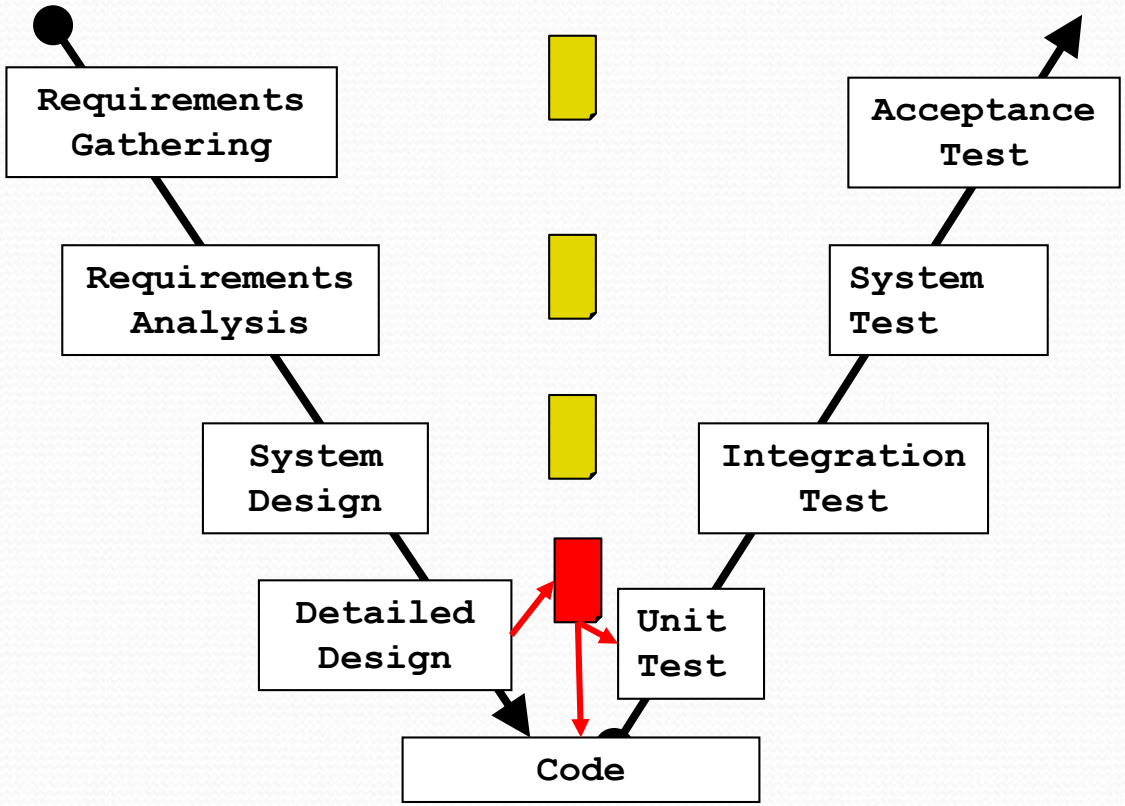
System Design

System Design produces the System Design Specification (SDS), which is both the input for the detailed design phase, and the basis for Integration Testing.



Detailed Design

The Detailed Design specifications contain all the information about the detailed design and from this the unit tests for the code are derived (black box tests).



V-Model Strengths

- Simple and easy to manage due to the rigidity of the model
- Encourages V&V at all phases: each phase has specific deliverables and a review process
- Higher chance of success over the waterfall model due to the early development of test plans during the life cycle
- Works well for small projects where requirements are very well understood

V-Model Weaknesses

- No working software is produced until late during the life cycle
- Not flexible model, adapt to change is expensive
- Poor model for long and large projects

Spiral Model

- Risk-driven software process model
- Suggested by Barry Boehm (1988), it supports iterative development and puts a special emphasis on risk management (through iterative risk analysis)
  - A risk is a probability that some adverse circumstances will occur
- The software process is represented as a spiral rather than as a sequence of activities with some backtracking

Spiral Model

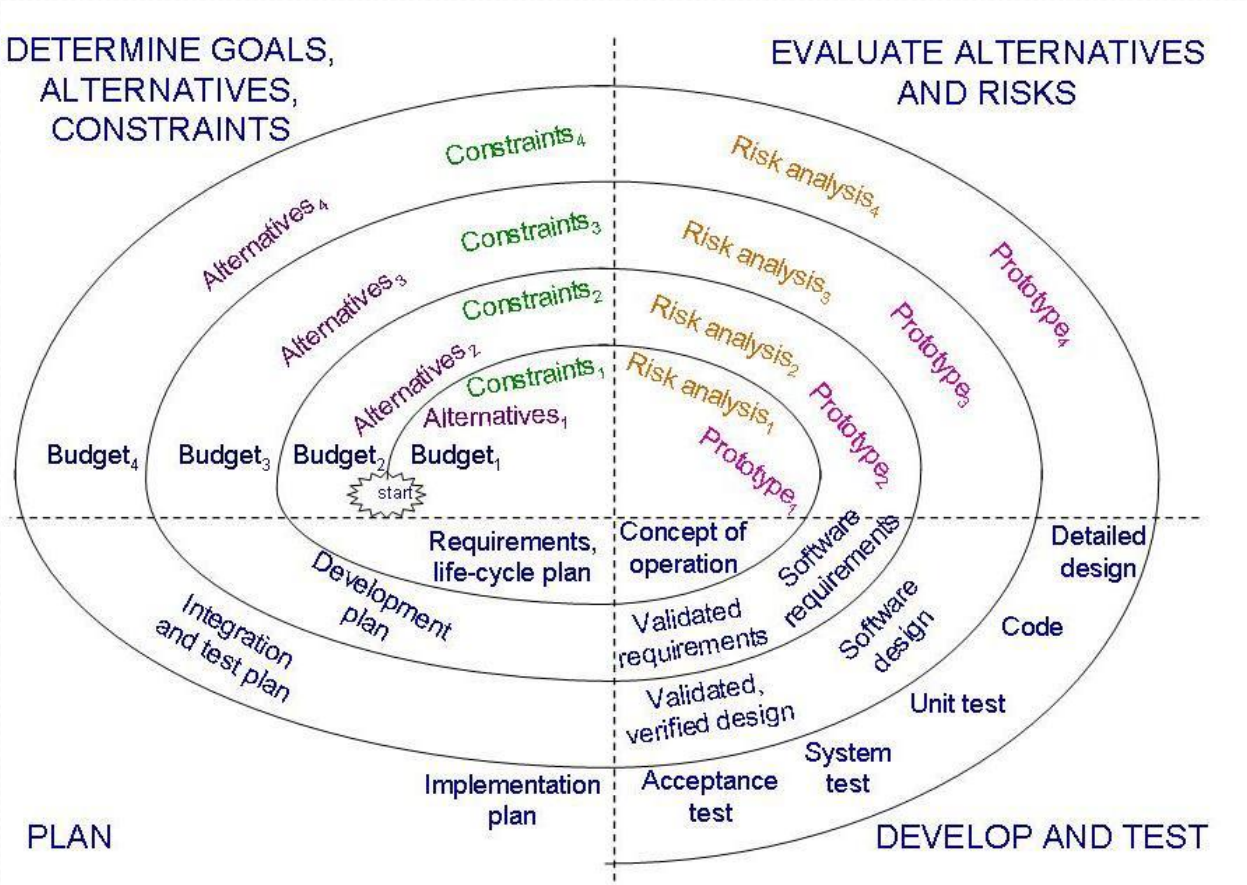


Figure 2.10 the spiral model.

(Pfleeger and Atlee, Software Engineering: Theory and Practice)

Spiral Model

- Each loop in the spiral represents a phase in the software process
  - E.g.: The innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design and so on
- Each loop is split into four major sectors
  - Determine goals, alternatives and constraints/Objective setting
  - Evaluate alternatives and risks
  - Develop and test
  - Plan

Spiral Model Sectors

- Objective setting
  - Specific objectives for the phase are identified, constraints identified, detailed management plan, project risks identified and alternative strategies planned
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks. E.g.: if there is a risk that the requirements are inappropriate, a prototype system may be developed
- Development and validation
  - A development model for the system is chosen which can be any of the generic models, (e.g., throw away prototyping model used if user interfaces risks are dominant)
- Planning
  - The project is reviewed and the next phase of the spiral is planned

Spiral Model

- Advantages
  - Estimates get more realistic as work progresses, as important issues are discovered earlier
  - More able to cope with change (change is considered as risks)
  - Software engineers/programmers can get working earlier (as may need prototype)
- Weakness
  - It is complicated and requires attentive and knowledgeable management to successfully apply the model

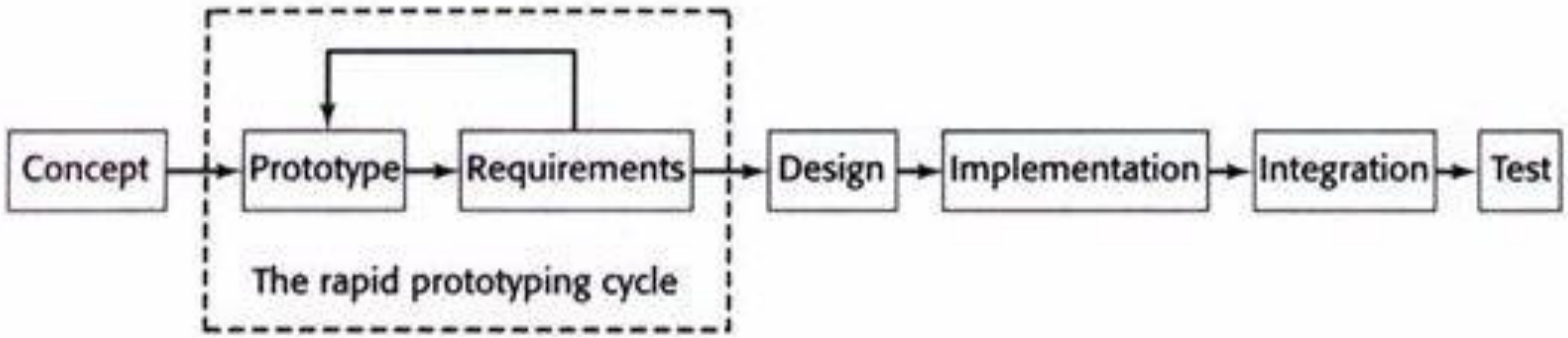
Spiral model

- Works well for large and complex projects
- When the cost and risks are important
- When users are not sure about their requirements

Prototyping model

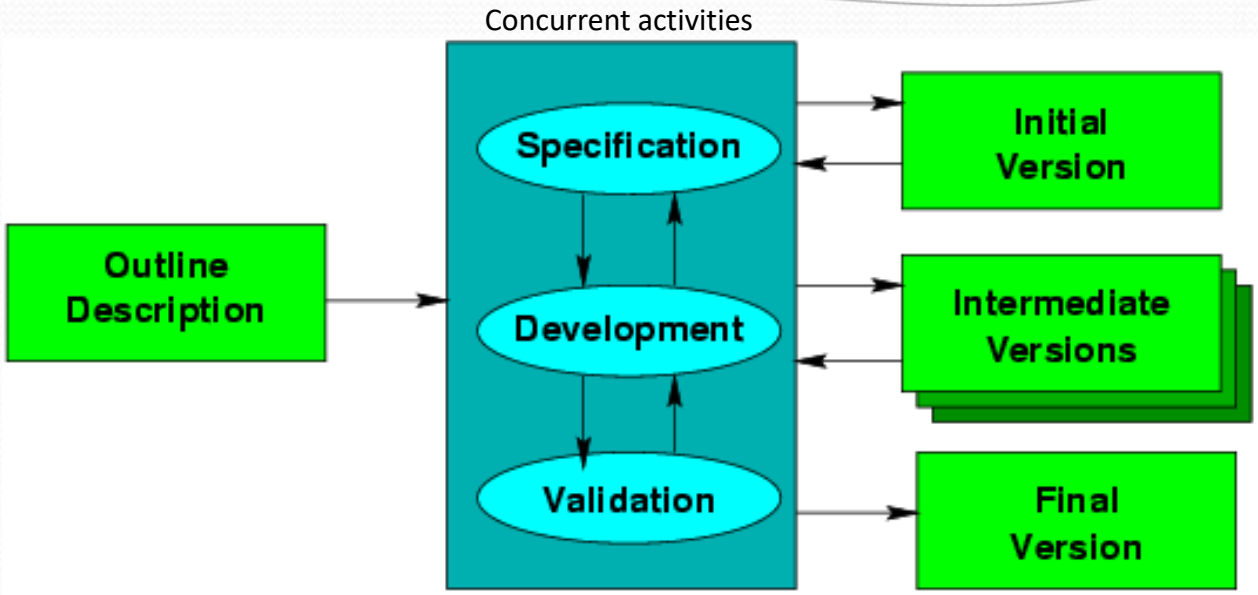
- Allows repeated investigation of the requirements or design
- Reduces risk and uncertainty in the development
- Helps with the elicitation and validation of system requirements
- Can be used to explore particular software solutions and to support user interface design
- Two models
  - Throw away prototyping
  - Evolutionary prototyping

Throw-away prototyping



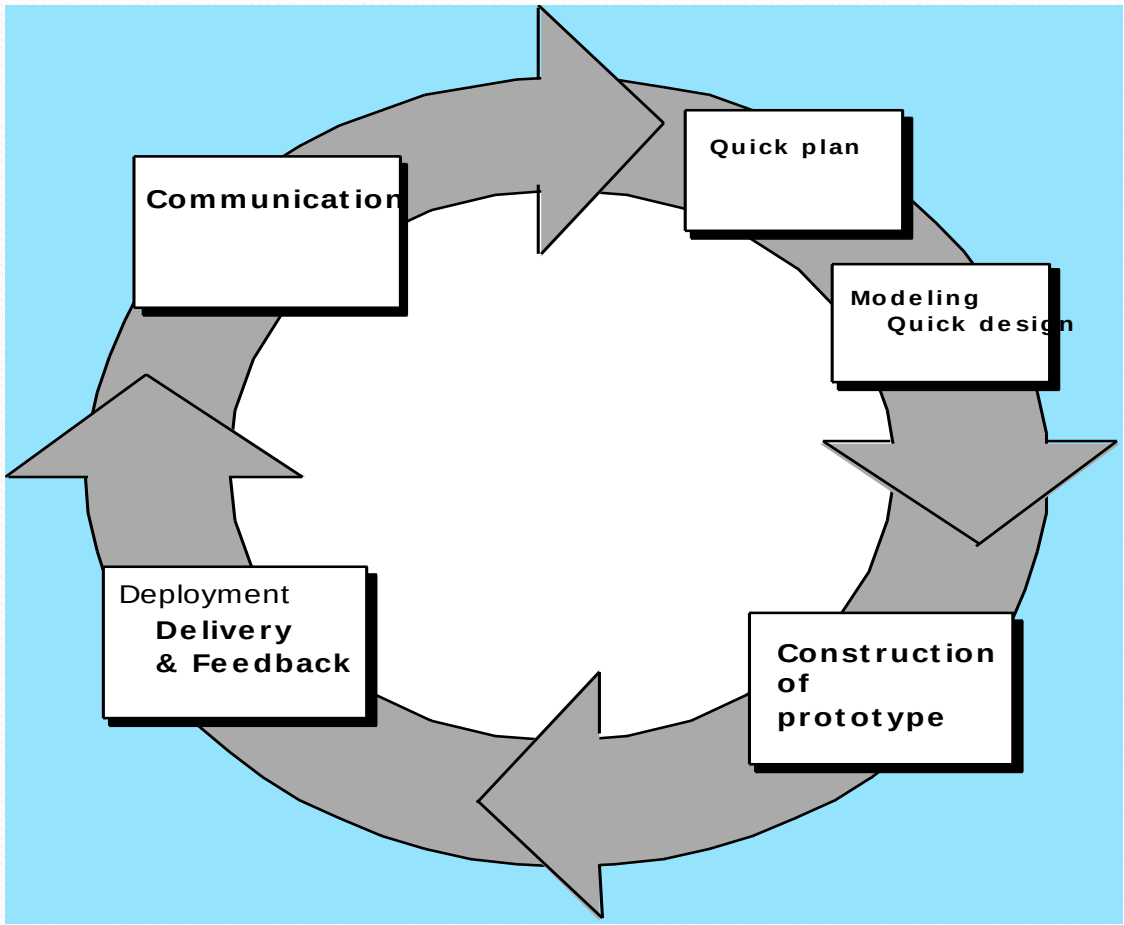
•The objective of throw-away prototyping is to ensure that the system requirements are validated and that they are clearly understood.
•The throw-away prototype is NOT considered part of the final system.
•Time putting together the throw away prototypes is lost unlike the evolutionary approach.

Evolutionary Development



- Evolutionary *Iterative* Development
  - enables software engineers to develop increasingly more complex versions of the software
- Based on the idea of:
  - developing an initial, low-functionality, high-quality implementation for the user
  - Receiving user comment and refining it through many versions until an adequate system has been developed
  - The software ***evolves*** over time (iterations)

Evolutionary Prototyping



- An approach to system development where an initial prototype is produced and refined through a number of stages to the final system

Evolutionary Prototyping

- Begins with communication – software engineer and the customer meet to define the overall objectives for the software and outline where further definition is required
- A quick plan means that the iteration is designed to run in a short space of time
- The quick design focuses on a representation of those aspects of the software that will be visible to the customer
- Then follows the prototype
- This is then evaluated and feedback given by the customer. The requirements are then refined

Prototyping advantages

- Accelerated delivery of the system
  - Rapid delivery and deployment are sometimes more important than functionality or long-term software maintainability
- User engagement with the system
  - Not only is the system more likely to meet user requirements, they are more likely to commit to the use of the system

Prototyping problems

- The customer can believe that the prototype is a fully working version of the software, but without realizing that no attention has been paid to either quality or maintainability. They may not be willing to wait for these goals to be achieved in the end
- The developer will take shortcuts to get the prototype working quickly, so they can do in an inappropriate manner. It may lead to not quality code, incomplete analysis
- Non-functional requirements may be omitted as the prototype mainly focus on functionalities

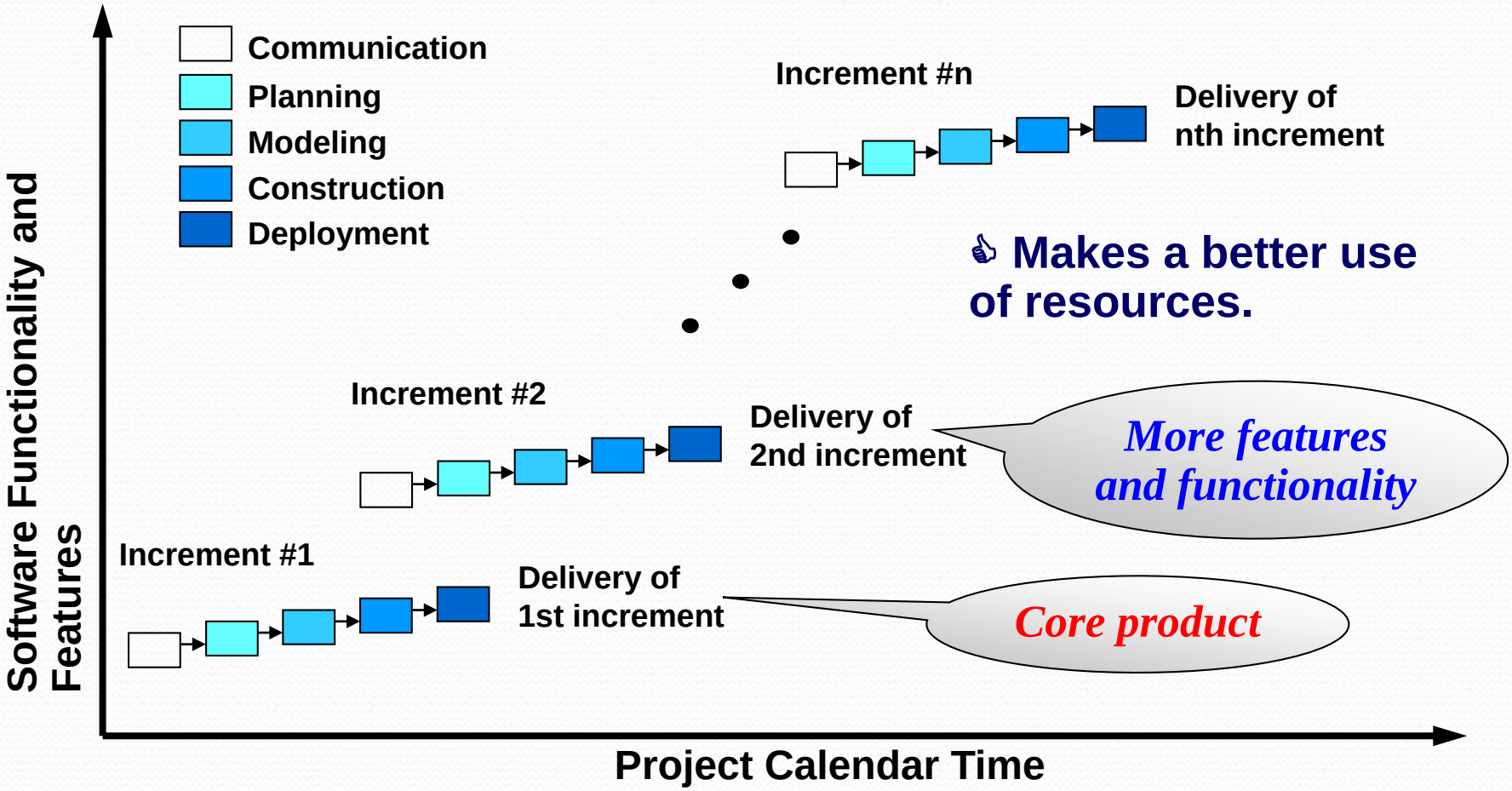
Incremental Process Models

- Used in situations where the initial software requirements are reasonably well-defined, but the overall scope of development is not clear yet
- There may be a need to provide a limited set of software functionality to users quickly, which will be refined in later releases
- Thus, a process model that produces the software in increments is much more suitable

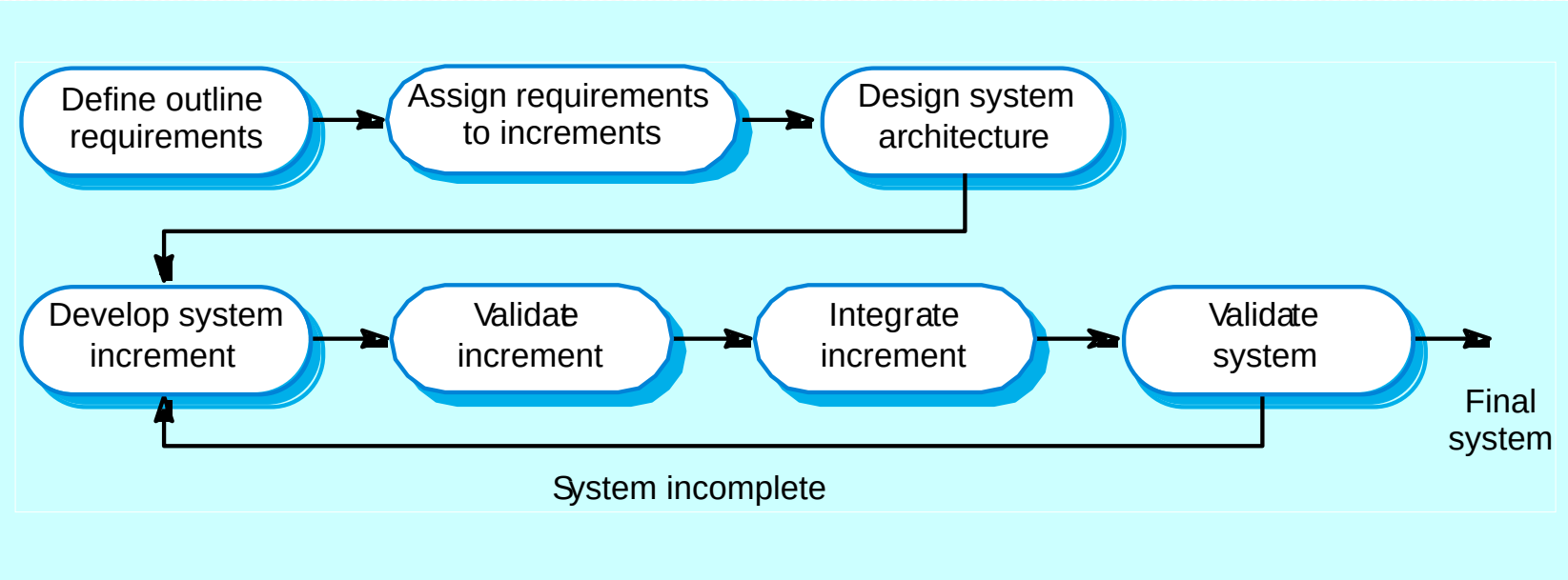
Incremental Development

- Incremental Development is a practice where the system functionalities are sliced into increments (small portions). In each increment, a slice of functionality is delivered by going through all the activities of the software development process, from the requirements analysis to the deployment.

The Incremental Model



Incremental development



more than one increment of the software development cycle may be in progress at the same time

Incremental Development - Advantages

- System functionality is available earlier – first increment satisfies most critical requirements
- Early increments act as a prototype to help elicit requirements for later increments
- The highest priority system services tend to receive the most testing as they are delivered first
- Lower risk of overall project failure

Incremental Development - Problems

- The process is not visible
- System structure tends to degrade as new increments are added
- Most systems require a set of basic facilities that are used by different parts of the system, and due to the fact that requirements are not defined in detail until an increment is to be implemented it can be difficult to identify common facilities needed by all increments
- There is no complete system specification until the final increment is specified. This requires a new form of contract between customer and developer

Incremental summary

- Incremental Development is often used together with the Evolutionary practice of Iterative Development in software development. This is referred to as Iterative and Incremental Development (IID)
- IDD is the core of the agile methods
  - Modern methods recommend 1-6 weeks minimum iteration
  - The resulting software from each iteration is a subset of the final system and not just a prototype.