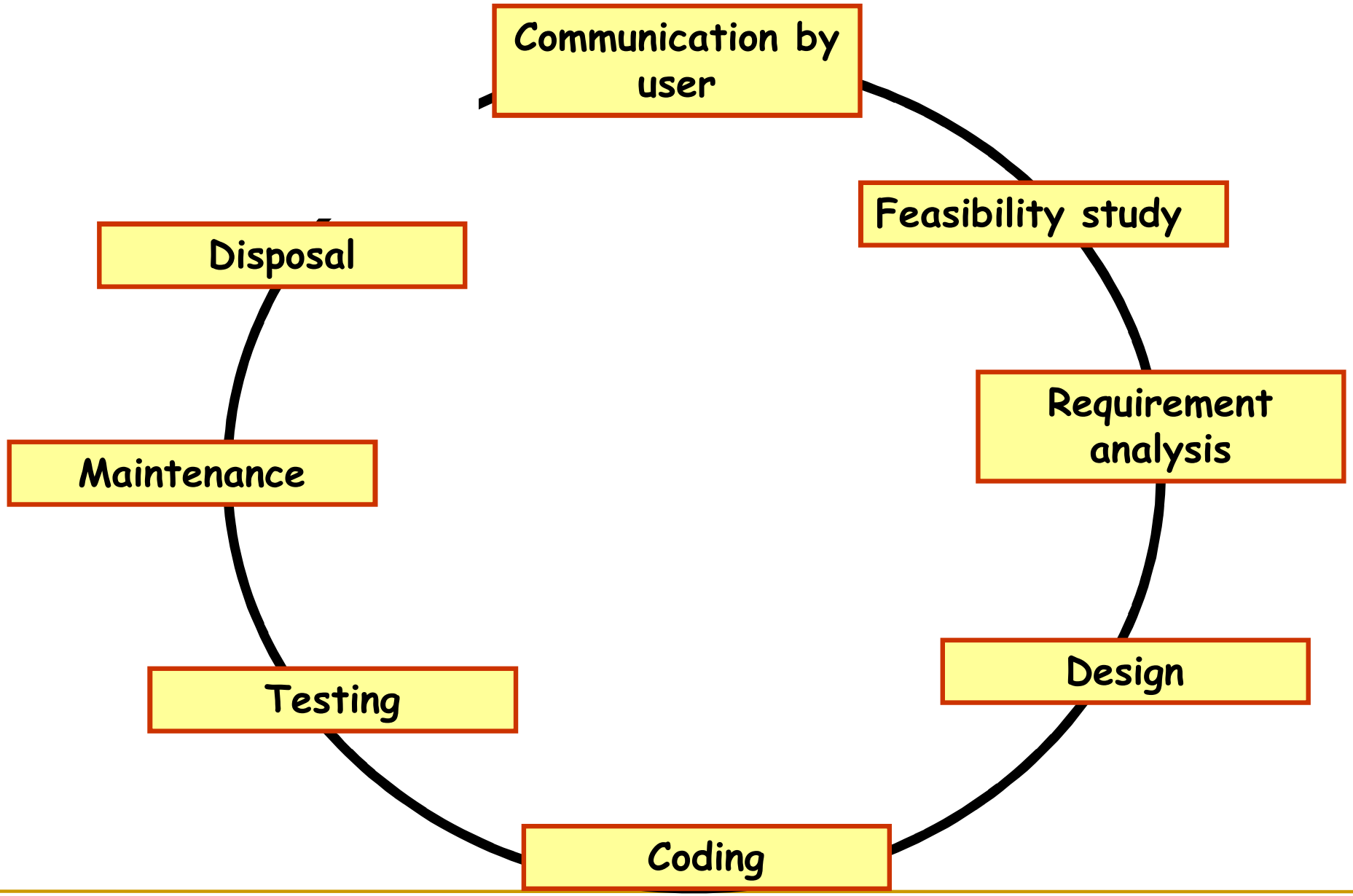
Software Engineering

Life Cycle Models

Life cycle model

- A descriptive and diagrammatic model of software life cycle
  - Divides life cycle into several phases
  - Each phase consists of several activities
  - Identifies activities to be performed in each phases
  - Defines entry and exit criteria for each phase

Life cycle of a software



Why life cycle model?

- Adhering to a life cycle model helps development of software in a systematic and disciplined manner
  - Each team member must understand when to do what
- Becomes easier for project manager to monitor progress of project
  - 'At which stage the project is' is clearly known
  - Avoid "99% complete syndrome"

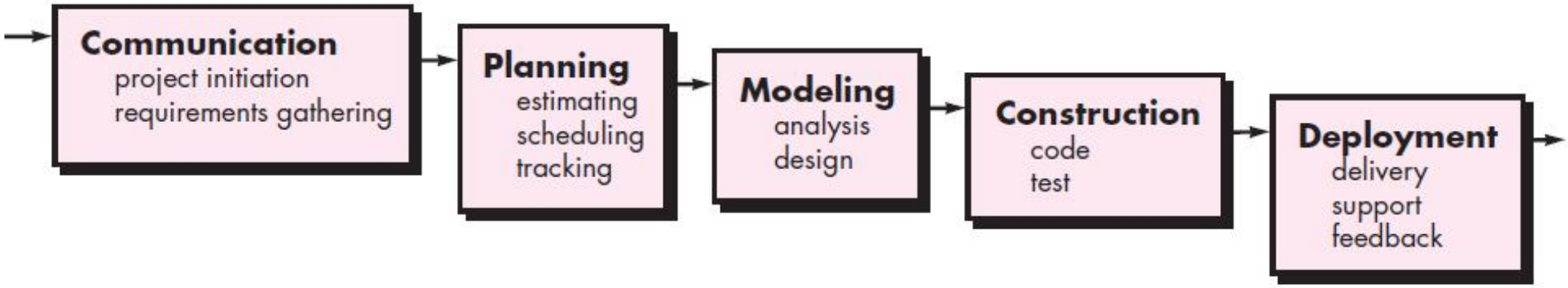
Several life cycle models

- Several have been proposed
  - Classical waterfall model
  - Iterative waterfall model
  - Evolutionary model
  - Spiral model
  - Rapid prototype model
  - Agile model

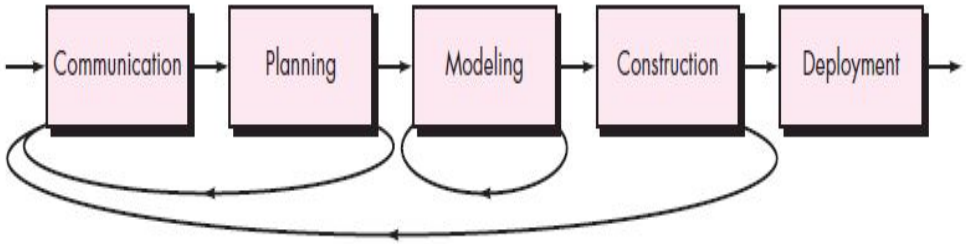
- Development team must identify which model is suitable, and then adhere to it

A Generic Process Framework for SE

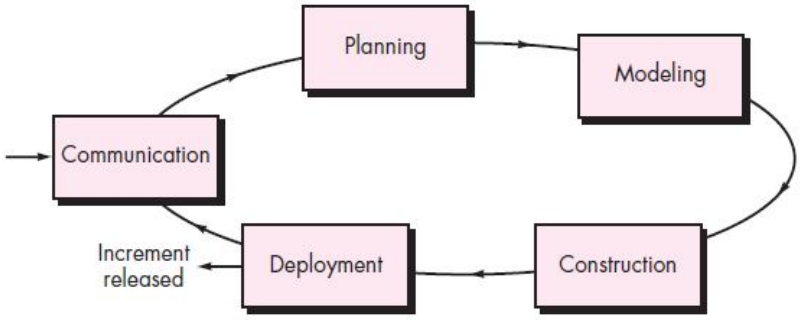
Linear process flow



Iterative process flow



Evolutionary process flow

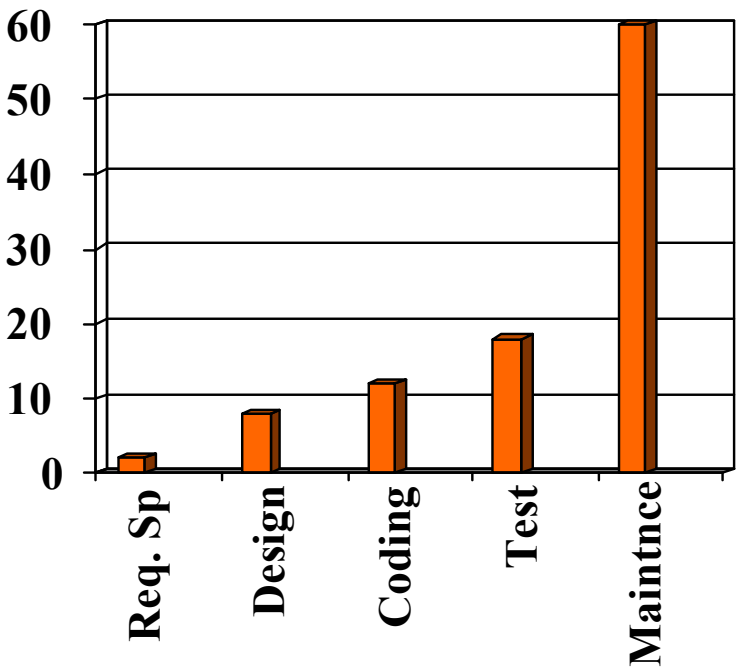


Classical Waterfall model

- Divides software life cycle into following phases
  - Feasibility study
  - Requirements analysis and specification
  - Design (architectural design)
  - Coding and unit testing
  - Testing
  - Maintenance

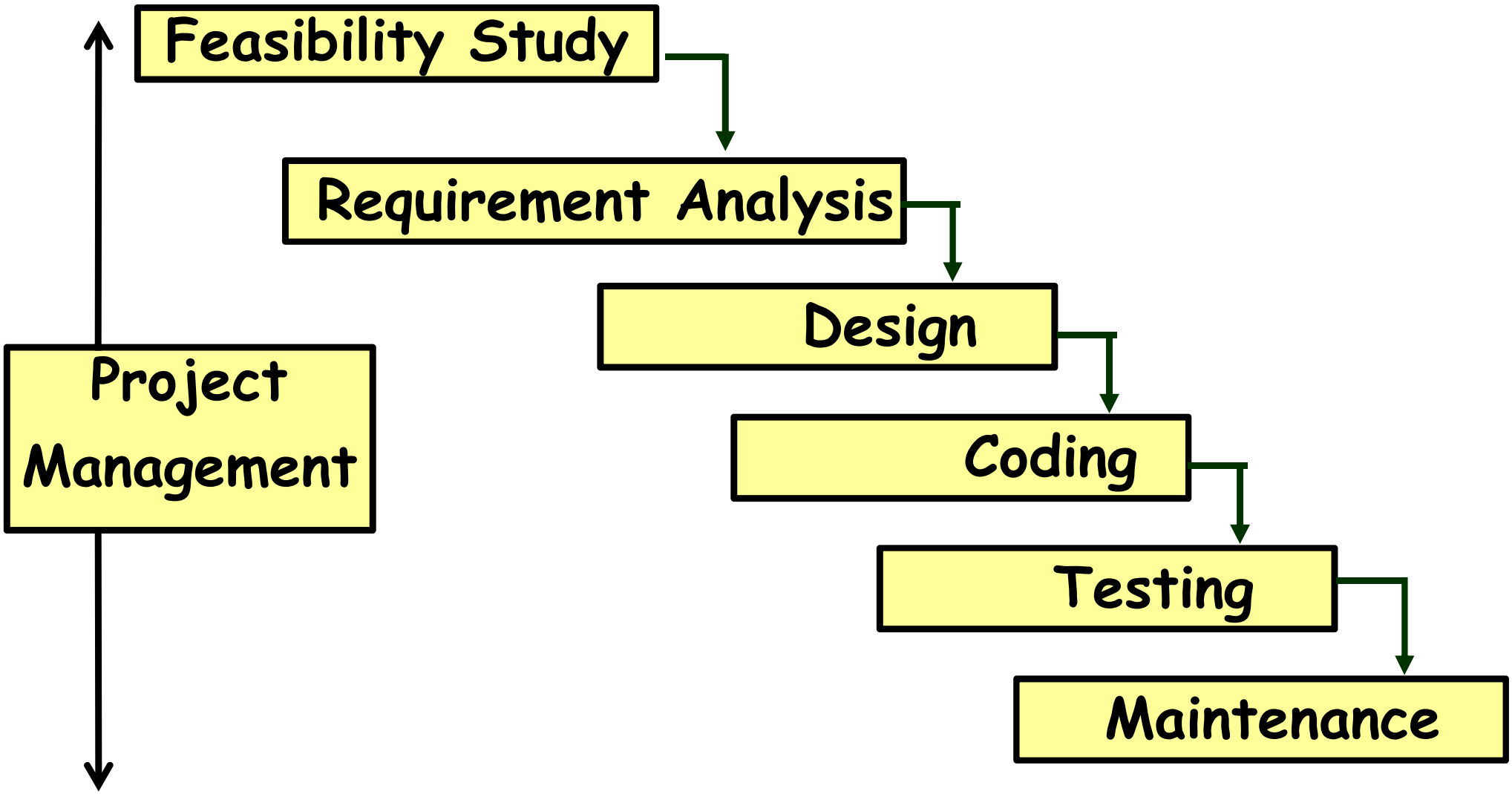
- First 5 phases called development phases
- Complete one phase before starting next one
- Extensive documentation per phase

Relative effort for phases



- As per Zelkowitz, Requirement analysis 10%, Specification 10%, Design 15%, Coding 20% and Testing 45%.
- As per Boehm, Requirements analysis and design 60%, implementation 15% and 25% for testing.
- As per Brooks, Planning 1/3, Coding 1/6; Component tests 1/4, and System test 1/4

Classical Waterfall model



Feasibility study

- Objectives
  - Determine whether developing the product would be financially worthwhile and technically feasible
- Activities
  - Overall understanding of problem – input, output, processing required, constraints
  - Formulate different solution strategies, compare w.r.t. resources, cost and time required
  - Cost / benefit analysis: which solution is best (if any)

Requirements Analysis

- Objectives
  - Understand exact requirements of customer
  - Document these requirements properly

- Activities
  - Requirements gathering and analysis
    - Discussions with client / end-user
    - Resolve inconsistencies and incompleteness in requirements
  - Requirements specification
    - Requirements organized and documented into Software Requirements Specification (SRS) document

Design

- Objective
  - Transform requirements specification into a form suitable for implementation in some programming language
  - Software architecture derived from the SRS document

- Two design approaches
  - Traditional approach – structured analysis & design
  - Object-oriented approach

Implementation

- Also known as `coding and unit testing' phase
- Objectives
  - Translate software design into source code
- Activities: Each module of the design is
  - Coded
  - Unit-tested and debugged
  - Documented
- End product: a set of program modules that have been tested individually

Testing

- System and integration testing (unit testing already carried out in implementation phase)

- Objectives
  - Ensure that the total developed system functions according to its requirements specified in SRS document

- Activities
  - Different modules integrated in a planned manner
  - After all modules are successfully integrated, system testing carried out

Maintenance

- Enhance / debug the software
- May take more effort than all development phases combined

- Types of maintenance
  - Corrective: correct errors which were not discovered during the product development phases
  - Perfective: improve implementation or enhance functionalities
  - Adaptive: port software to a new environment

Implementation of waterfall model

- Most organizations usually define
  - Standards on the outputs (deliverables) produced at the end of every phase
  - Entry and exit criteria for every phase
  - Methodologies for the different stages & activities

- Guidelines and methodologies of an organization called the organization's software development methodology
  - Engineers expected to know and follow this

Classical waterfall model: Applicable when

- Requirements are well documented, clear and fixed
- Technology is not dynamic
- Project is short
- Management provides enough resoureces

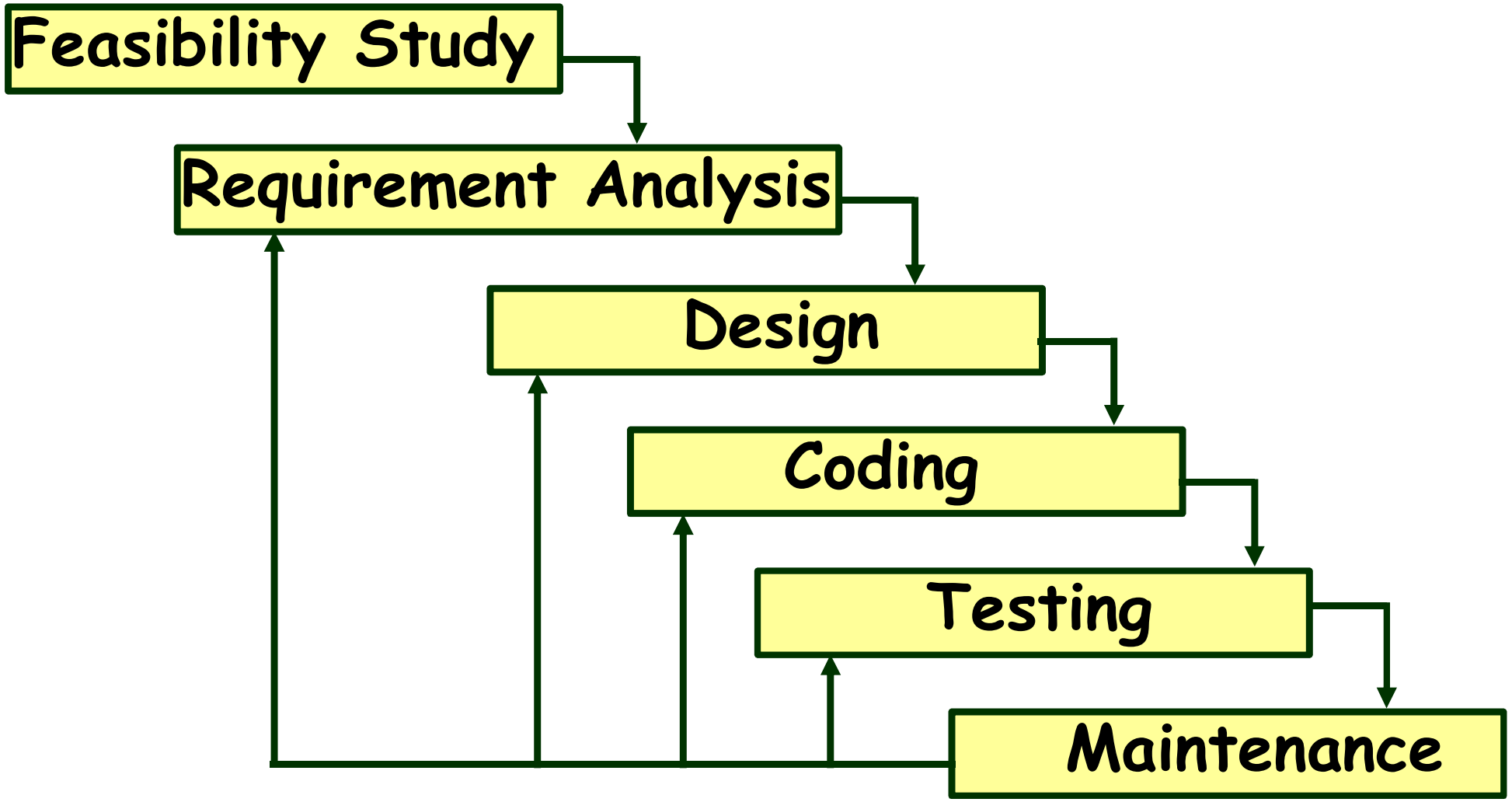
Advantages of classical waterfall model

- Simple
- Easy to understand
- Each stage is clearly defined
- Process action and results are well documented

Drawbacks of classical waterfall model

- High amount of risk and uncertainty
- Not good for complex project
- Very rigid----cannot accommodate any change
- No error correction mechanism
- Efficiency reduces---new phase starts only after previous phase ends

Iterative waterfall model



Phase containment of errors

- Best if errors are detected in the same phase in which they are introduced
- Principle: detect errors as close to its point of introduction as possible
- Technique : Conduct reviews after every milestone

Phase overlap

- Phase containment of errors is always not achieved
- Better human resource utilization – avoid *blocking state*

Waterfall models

- Iterative waterfall model is by far the most widely used model
  - Almost every other life cycle model derived from this
- Importance of classical waterfall model
  - 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: Applicable when

- Major requirements are defined but minor details may evolve over time
- New technologies are learnt by development team
- Experience team of developer

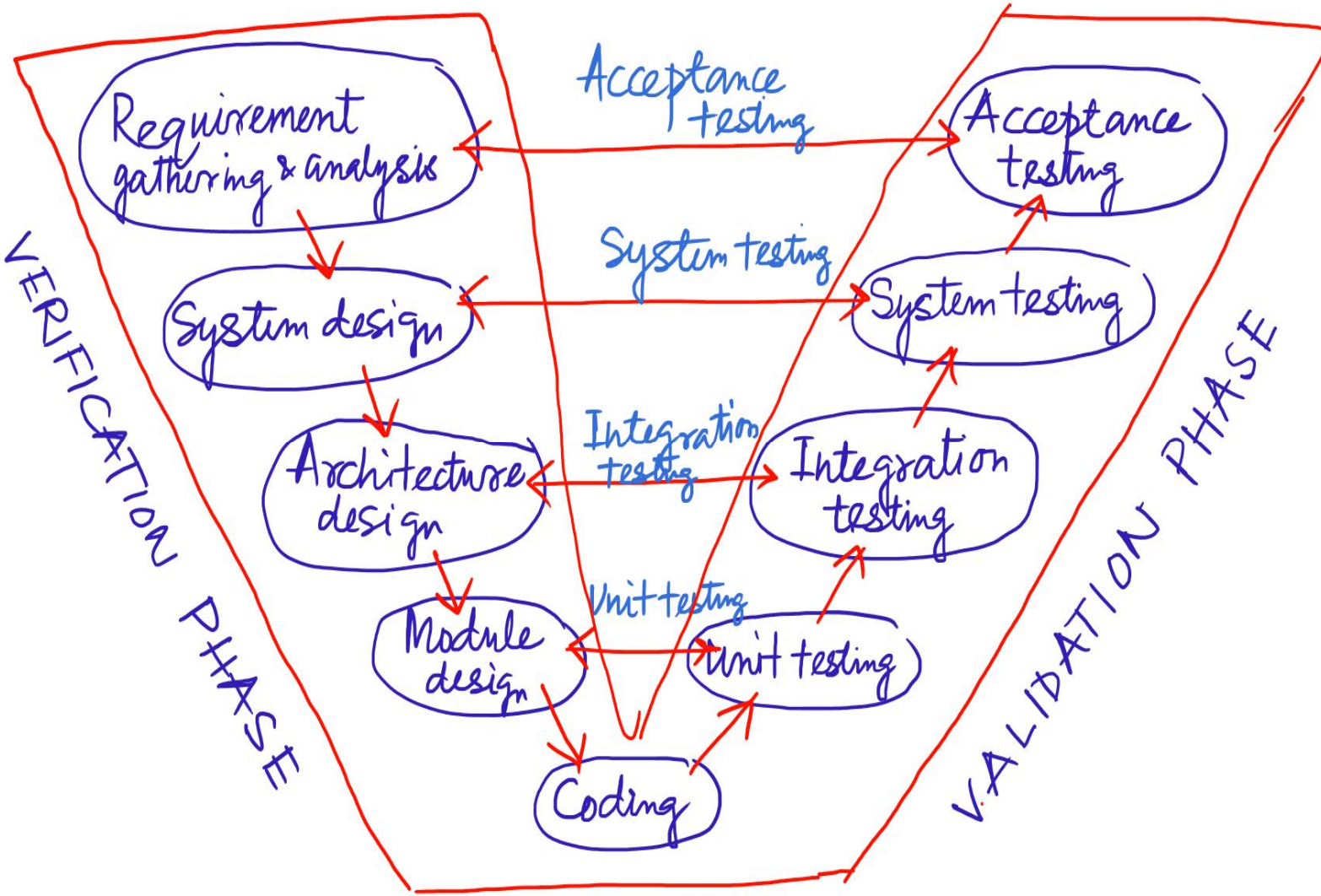
Advantages of iterative waterfall model

- Feedback path allows correcting errors
- Simple to understand and use
- Customer involvement is not required during software development
- Suitable for comparatively large and complex project

Drawbacks of iterative waterfall model

- No scope for any incremental delivery
- No phase overlapping
- No risk handling mechanism
- Limited customer interaction/review/preview

V-Shaped SDLC Model



A variant of the Waterfall that emphasizes the verification and validation of the product.

Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped SDLC Steps

Project and Requirements Planning – allocate resources

Product Requirements and Specification Analysis – complete specification of the software system

Architecture or High-Level Design – defines how software functions fulfil the design

Detailed Design – develop algorithms for each architectural component

Coding – transform algorithms into software

Unit testing – check that each module acts as expected

Integration and Testing – check that modules interconnect correctly

System and acceptance testing – check the entire software system in its environment

Production, operation and maintenance – provide for enhancement and corrections

V-Shaped model: Applicable when

- Medium sized projects
- Requirements are clear
- (Most frequently in) Medical field

Advantages of V-Shaped model

- Simple and understandable
- Focuses on verification and validation in early life cycle---high probability of error free product
- Easy for project managers to track progress due to rigidity
- Each phase is deliverable and can be reviewed

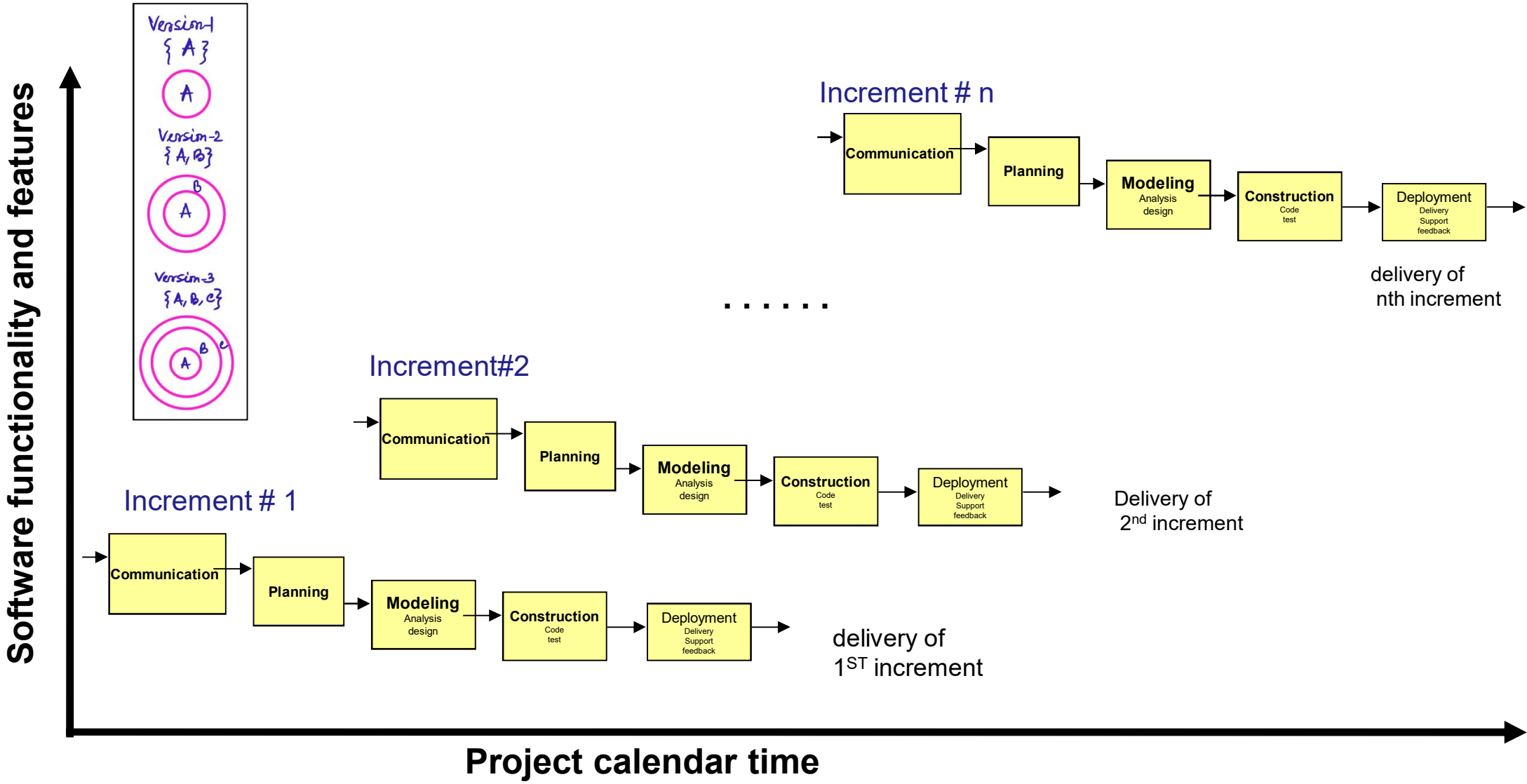
Drawbacks of V-Shaped model

- Does not support iteration—difficult to handle change
- No phase overlapping
- Not good for complex and long project

The Incremental / Evolutionary Model

- Software releases in increments
- 1st increment constitutes Core product
  - Basic requirements are addressed
- Core product undergoes detailed evaluation by the customer
  - As a result, plan is developed for the next increment
  - Plan addresses the modification of core product to better meet the needs of customer
- Process is repeated until the complete product is produced

The Incremental / Evolutionary Model



Evolutionary model: Applicable when

- Very large project where modules are implemented incrementally
- Customer prefer to receive product in increment
- System with stand alone units
- (Mostly)Object oriented design

Advantages of Evolutionary model

- Customer's confidence increases as he gets product constantly from developer
- Reduces error because modules gets tested thoroughly
- Chance to experiment with partially developed software
- Does not need for large resource at a time
- Good when complete version is impossible because of tight market deadline
- Better risk analysis
- Supports environmental change

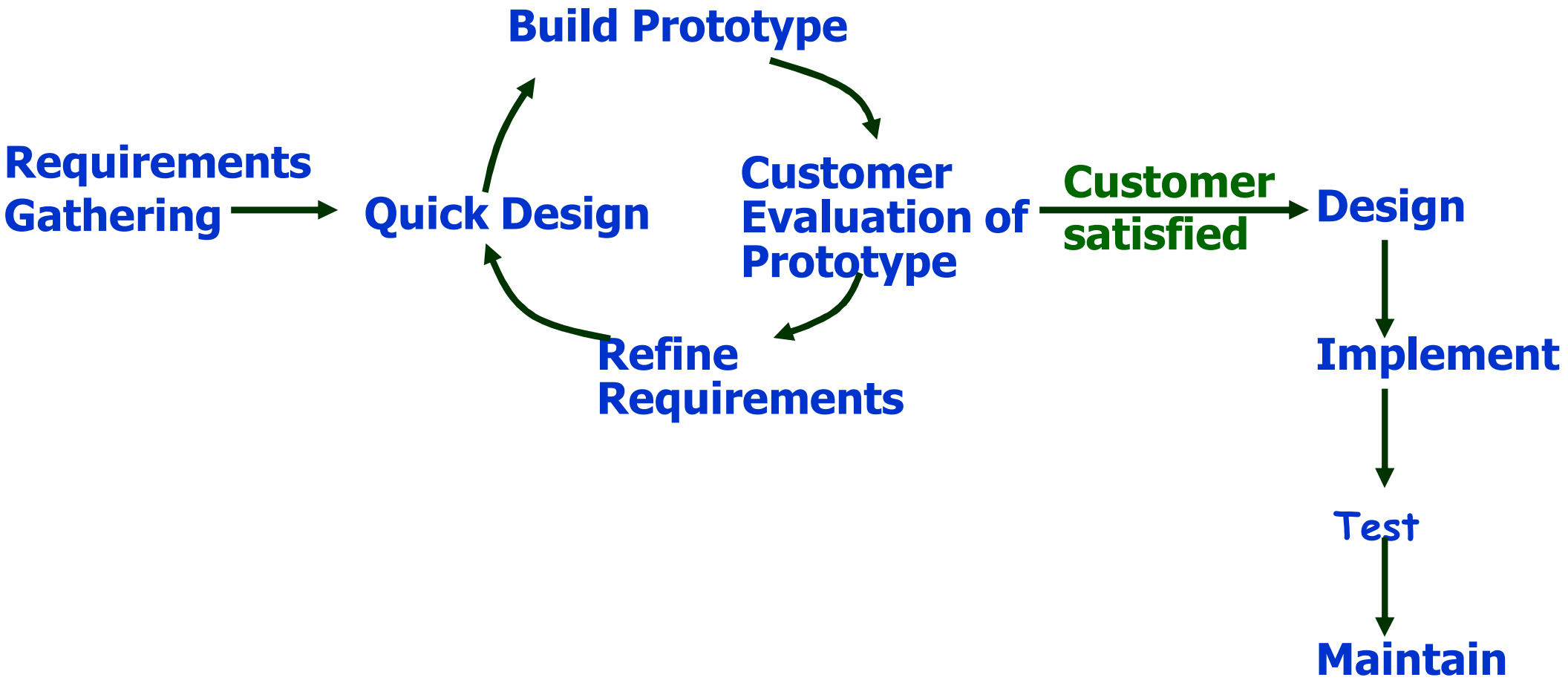
Drawbacks of Evolutionary model

- Sometimes hard to divide into several versions that will be acceptable by the customer

Prototyping Model

- Software of all complex system evolves over time
- Requirements of customers often not clearly understood / known initially – requirements are fuzzy
- Before starting actual development, build a working prototype of the system
  - Illustrate to the customer: data formats, messages / reports / interactive dialog formats
  - Helps to decide technical issues associated with the actual software development
  - Impossible to "get it right" the first time

Rapid prototype model - steps



Rapid prototype model - steps

- Start with approximate requirements
- Carry out a quick design
- Implement the design: use approximations
  - e.g. table lookup instead of computation
- Submit prototype to customer for evaluation
- Based on user feedback, refine requirements

- Cycle continues until customer approves prototype

Rapid prototype model

- Requirements analysis and specification phase becomes redundant
  - Final working prototype (user feedbacks incorporated) serves as an animated requirements specification
- Utilities of rapid prototyping model
  - Design and code for prototype usually discarded, but valuable experience gained
  - Actual system can now be developed using *almost* the classical waterfall approach
  - Preferred for systems with unclear requirements or unresolved technical issues

Types of prototype

- Throwable prototype
- Evolutionary prototype
- Incremental prototype
- Extreme prototype

Rapid prototype model: Applicable when

- Requirement is unclear

Advantages of Rapid prototype model

- Errors are detected at initial stage
- Customer satisfaction exists as customer feels the product at very early stage
- Encourages innovation and flexible design
- **Working model of product**: Prototype may offer early training for future user
- Increase customer involvement
- Reduces overall time and cost

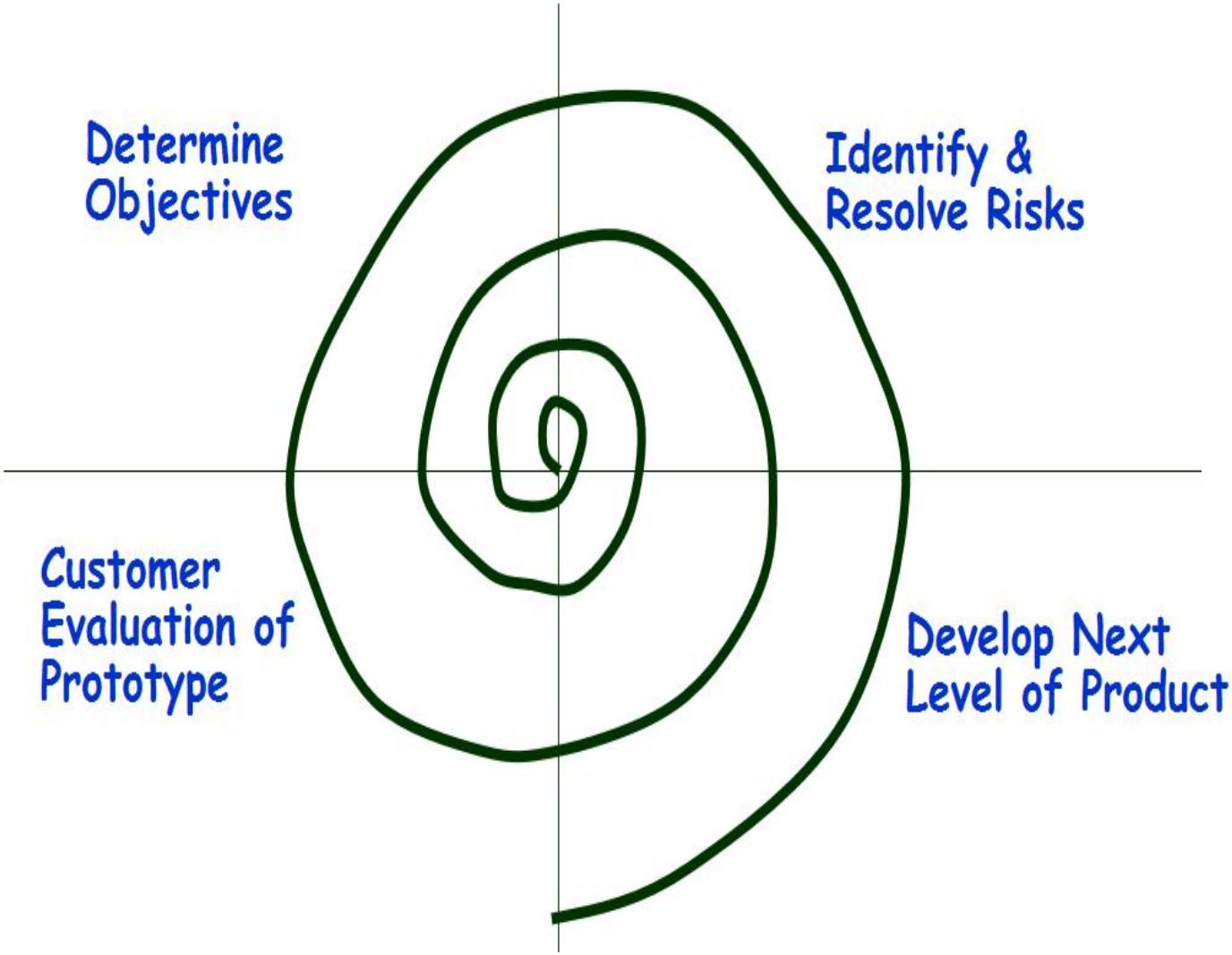
Drawbacks of Rapid prototype model

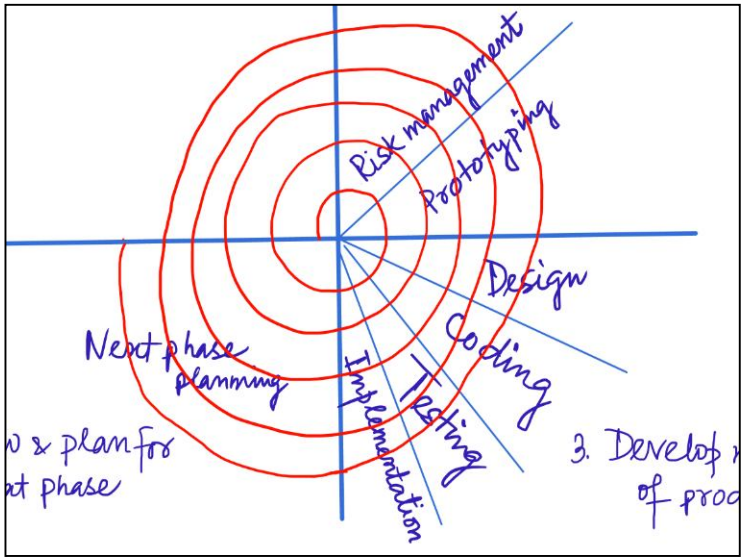
- Slow and time taking
- Cost of building prototype is waste if it is thrown off
- May encourage excessive change request
- Hard to document because requirements of customers are changing
- ***Customer's confusion***
  - After seeing prototype customer may think that product will be delivered soon.
  - Customer may lose interest if he is not happy with initial product.
- Quality of product may drop compared to quality of prototype

Spiral model

- Proposed by Boehm in 1988
- No fixed phases in this model
  - Team must decide how to structure project into phases
- Each loop of the spiral represents a phase of the software development process
  - Innermost loop may be concerned with feasibility study
  - The next loop with system requirements definition,
  - The next one with system design, ...
- Each loop in the spiral split into four quadrants

Spiral model





Quadrants in spiral model

- 1st quadrant: setting objectives
  - Identify objectives of the phase
  - Identify the risks associated with these objectives
  - Risk: any adverse circumstance that may hamper successful completion of the phase / project

- 2nd quadrant: resolving risks
  - Analyse each identified risk, take steps to resolve
  - E.g. if there is a risk that requirements are not well understood, a prototype system may be developed

Quadrants in spiral model (contd.)

- 3rd quadrant: develop
  - develop next level of product / prototype
- 4th quadrant: customer evaluation
  - Review results achieved so far, with customer feedback
  - Plan next iteration around the spiral

Spiral model: Applicable when

- Large and high budget
- Requirement is unclear and complex
- Changes may be requested at any time

Advantages of Spiral model

- ***Useful for mission critical project:*** Support for better risk handling
- Good for large project
- Flexibility in requirement change
- Customer's satisfaction
- User can see the system early
- Extensive use of prototype
- Incremental refinement and incremental release of product around the sprial

Drawbacks of Spiral model

- Most complex management among all other models
- Expensive ----not suitable for small projects.
- Difficulty in time management as number of loops are not fixed
- End of project is not known early
- Spiral may go indefinitely
- Excessive documentation required

Comparison of different models

- Iterative waterfall model
  - Most widely used
  - But, suitable only for well-understood problems
- Rapid prototype model
  - Suitable for projects that are not well understood
- Incremental / Evolutionary model
  - Suitable for large projects which can be decomposed into a set of modules that can be developed incrementally
  - Incremental delivery must be acceptable to customer

Comparison on life cycle models

- Spiral model – a meta model
  - Retains the phase-wise approach of waterfall model
  - A single loop of the spiral may use the waterfall model
  - Uses evolutionary approach – iterations over the spiral are evolutionary levels
  - Prototyping can be used in any of the phases (loops), as a risk reduction mechanism

- Agile model
  - Suitable when requirements will change frequently (which is almost always)

Conclusion

- Traditional SDLC gives heavy importance to
  - Processes & tools
  - Comprehensive documentation
  - Contract negotiation
  - Following a plan