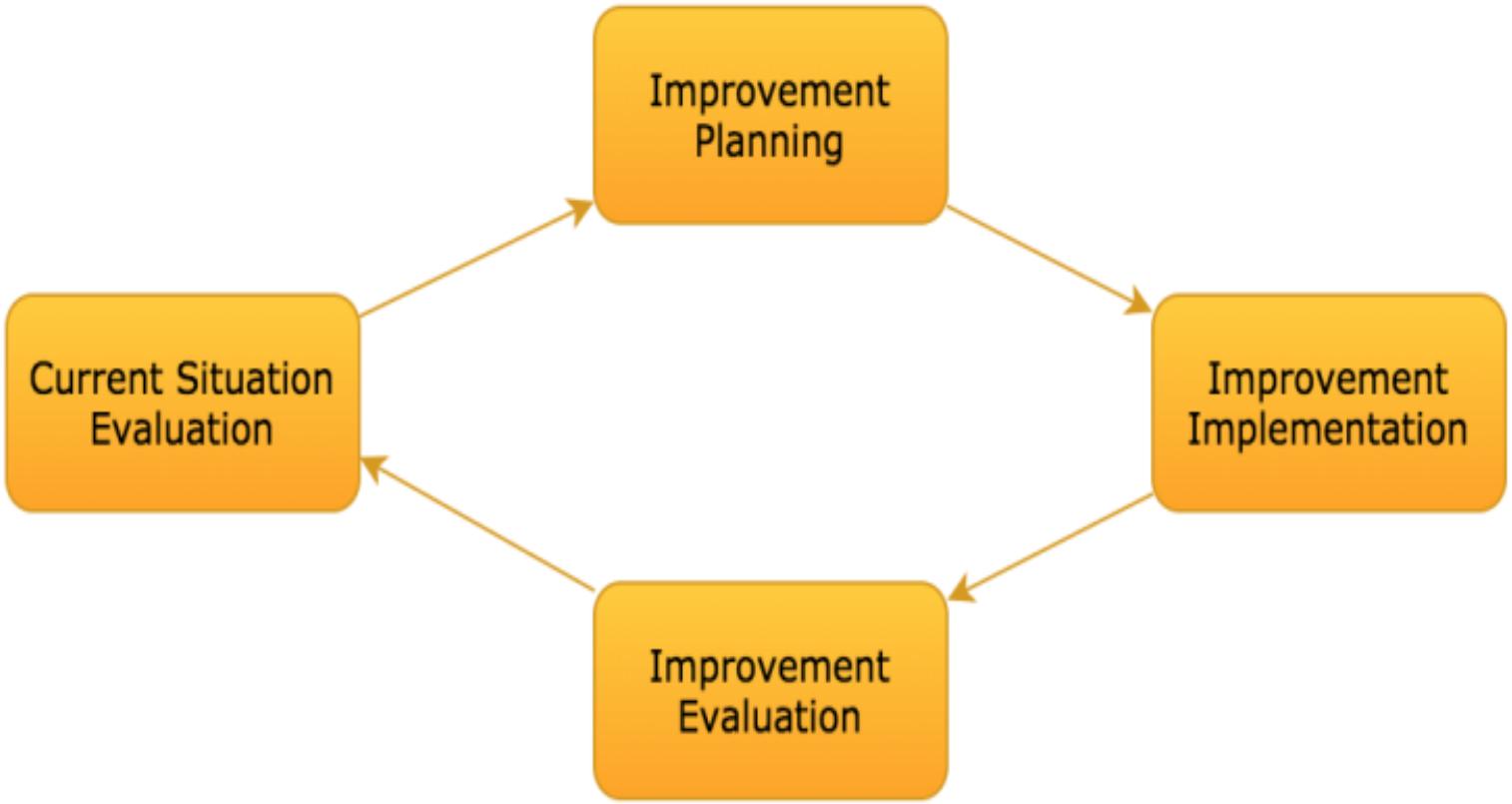
- Software process improvement
  &
  SDLC

- Software Process Improvement (SPI)

- Set of tasks,tools, and techniques

- to plan and implement improvement activities to achieve specific goals such as increasing development speed, achieving higher product quality or reducing costs.

SPI cycle


Improvement Planning
Improvement Implementation
Improvement Evaluation
Current Situation Evaluation

SPI-Current Situation Evaluation

- Initial phase of the process
- To assess the current situation of the software process
- Eliciting the requirements from the stakeholders, analyzing the current artifacts and deliverables, and identifying the inefficiencies from the software process.
- The elicitation can be conducted through different techniques. For example, individual interviews, group interview, use-case scenarios, and observations.

- SPI-Improvement Planning

- The gap between the current level and the target level should be planned in terms of a set of activities to reach that target.

- SPI-Improvement Implementation
- The planned activities are executed and it puts the improvements into practice .

SPI-Improvement Evaluation

- What is cannot be measured cannot be improved.
- to compare the rate of actual change against its planned change.

Why are Companies Seeking SPI

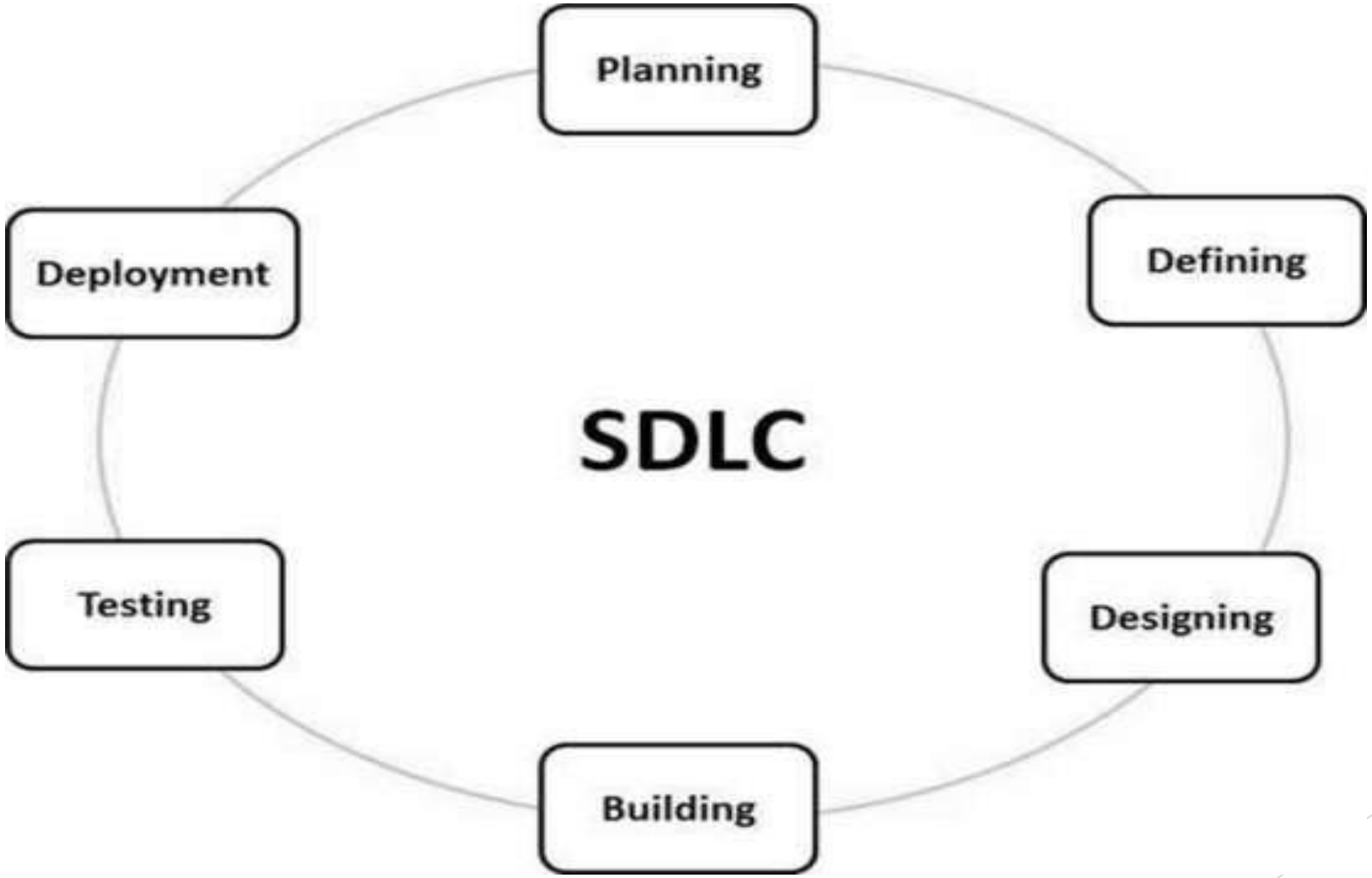
- Standardization and Process consistency
- Cost Reduction
- Competitive Edge
- Meeting targets and reduce time to market
- Improve customers satisfaction

Software Development Life Cycle (SDLC)

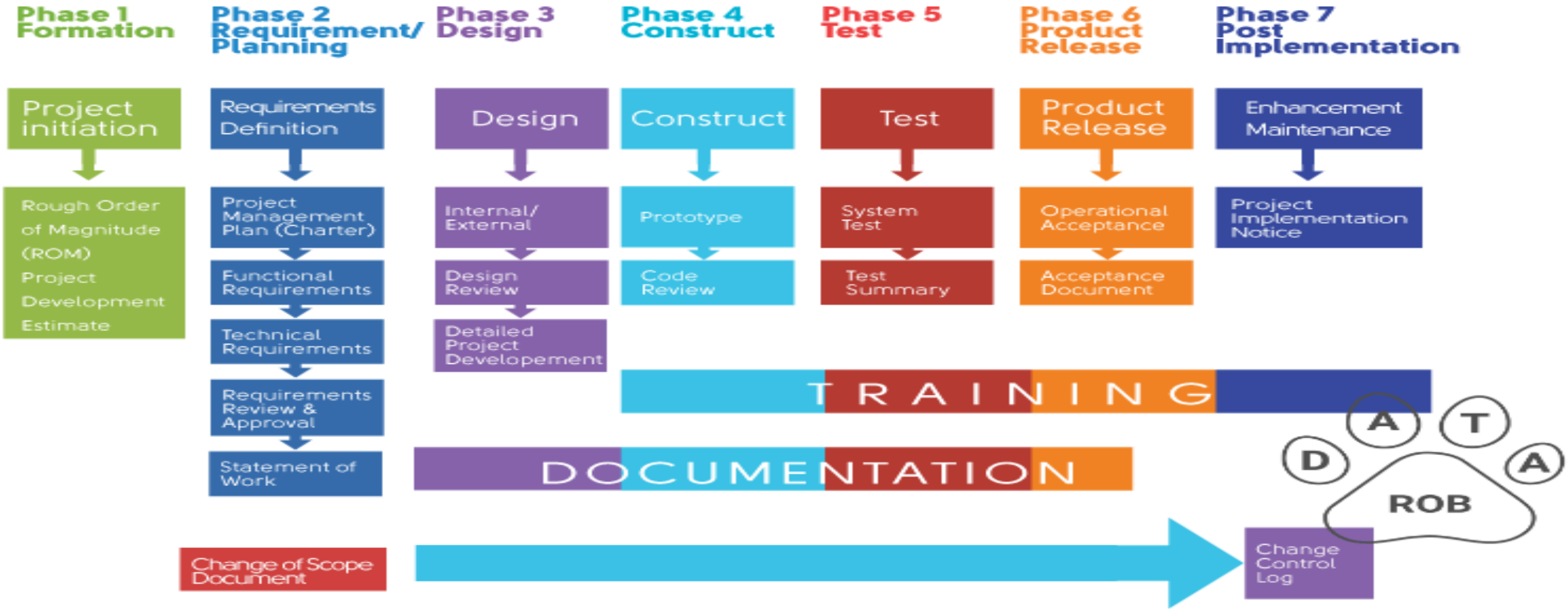
- It is a process used by the software industry
- To design, develop and test high quality software's.
- The SDLC framework provides a systematic process for building software that ensures the quality and correctness of the software built.
- It aims to produce high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
- The SDLC framework consists of detailed plans describing how to develop, maintain and replace specific software.

- Every phase of the SDLC framework has its own process and deliverables that feed into the next phase.
- A typical SDLC framework used for developing a software application might include some version or subset of the following activities:

SDLC


Planning
Defining
Designing
Building
Testing
Deployment
SDLC

SDLC PHASES


Phase 1 Formation
Phase 2 Requirement/ Planning
Phase 3 Design
Phase 4 Construct
Phase 5 Test
Phase 6 Product Release
Phase 7 Post Implementation
Project initiation
Requirements Definition
Design
Construct
Test
Product Release
Enhancement Maintenance
Rough Order of Magnitude (ROM) Project Development Estimate
Project Management Plan (Charter)
Functional Requirements
Technical Requirements
Requirements Review & Approval
Statement of Work
Internal/ External
Design Review
Detailed Project Developement
Prototype
Code Review
System Test
Test Summary
Operational Acceptance
Acceptance Document
Project Implementation Notice
TRAINING
DOCUMENTATION
DATA ROB
Change of Scope Document
Change Control Log

- System Initiation/Planning: where do systems come from? In most situations, new feasible systems replace or supplement existing information processing mechanisms whether they were previously automated, manual, or informal.
- Requirement Analysis and Specification: identifies the problems new software system is supposed to solve, its operational capabilities, its desired performance characteristics, and the resource infrastructure needed to support system operation and maintenance.
- Functional Specification or Prototyping: identifies and potentially formalizes the objects of computation, their attributes and relationships, the operations that transform these objects, the constraints that restrict system behaviour, and so forth.
-

- Partition and Selection (Build vs. Buy vs. Reuse): given requirements andfunctional specifications, divide the system into manageable pieces thatdenote logical subsystems, then determine whether new, existing, orreusable software systems correspond to the needed pieces.
- Architectural Design and Configuration Specification: defines the interconnection and resource interfaces between system subsystems, components, and modules in ways suitable for their detailed design and overall configuration management.
- Detailed Component Design Specification: defines the procedural methods through which the data resources within the modules of a component are transformed from required inputs into provided outputs.
- Component Implementation and Debugging: codifies the preceding specifications into operational source code implementations and validates

- Software Integration and Testing: affirms and sustains the overall integrity of the software system architectural configuration through their basic operation.verifying the consistency and completeness of implemented modules, verifying the resource interfaces and interconnections against their specifications, and validating the performance of the system and subsystems against their requirements.
- Documentation Revision and System Delivery: packaging and rationalizing recorded system development descriptions into systematic documents and user guides, all in a form suitable for dissemination and system support.
- Deployment and Installation: providing directions for installing the delivered software into the local computing environment, configuring operating systems parameters and user access privileges, and running diagnostic test cases to assure the viability of basic system operation.

- Training and Use: providing system users with instructional aids and guidance for understanding the system's capabilities and limits in order to effectively use the system.
- Software Maintenance: sustaining the useful operation of a system in its host/target environment by providing requested functional enhancements, repairs, performance improvements, and conversions.

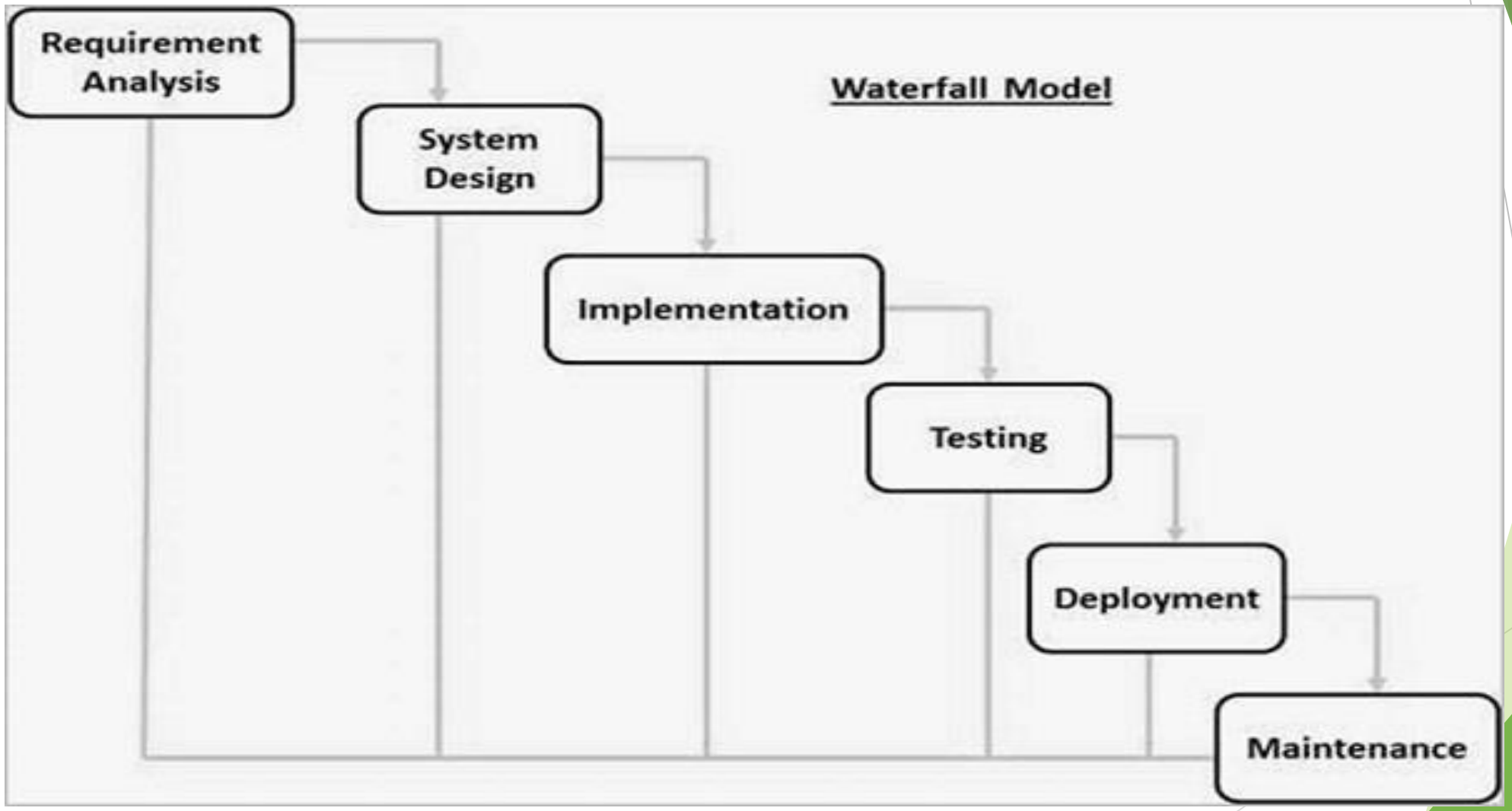
SDLC MODELS

- SDLC –
- Descriptive or Prescriptive
- A software life cycle model can be either a descriptive or prescriptive characterization of how software is or should be developed.
- A descriptive model describes the history of how a particular software system was developed.
- Descriptive models may be used as the basis for understanding and improving software development processes, or for building empirically grounded prescriptive models.
- A prescriptive model prescribes how a new software system should be developed.
- Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order.

PRESCRIPTIVE PROCESS MODELS

- Waterfall Model
- Evolutionary model
  - Increment Model
  - Spiral Mode
- Prototype Model
- RAD
- Agile Model
- Component Based development

WATERFALL MODEL



WATERFALL MODEL

- First Process Model
- Linea r-sequential life cycle model.
- Each phase must be completed before the next phase can begin
- The outcome of one phase act as the input of another phase.
- There is no overlapping in the phases.
- Outcome of one phase acts as the input for the next phase .

- When?

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

- Requirements analysis and definition: The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.(SRS)
- System and software design: The systems design process allocates the requirements to either hardware or software systems.
- It establishes an overall system architecture.
- Software design involves identifying and describing the fundamental software system abstractions and their relationships.
- Implementation and unit testing: During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

- Integration and system testing: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- Operation and maintenance: Normally, this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors that were not discovered in earlier stages of the life cycle, improving the implementation of system units, and enhancing the system's services as new requirements are discovered.

Advantageous

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Dis Advs

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.