SOFTWARE ENGINEERING

WHAT IS SOFTWARE?

- The product that software professionals build and then support over the long term.
- Software encompasses:
    (1) instructions (computer programs) that when executed provide desired features, function, and performance;
    (2) data structures that enable the programs to adequately store and manipulate information and
    (3) documentation that describes the operation and use of the programs.
- Software, when made for a specific requirement is called software product.

SOFTWARE PRODUCTS

Generic products

- Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
- Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

Customized products

- Software that is commissioned by a specific customer to meet their own needs.
- Examples – embedded control systems, air traffic control software, traffic monitoring systems.

SOFTWARE APPLICATIONS

- System software: such as compilers, editors, file management utilities
- Application software: stand-alone programs for specific needs.
- Engineering/scientific software: Characterized by "number crunching" algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc
- Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

WHAT IS SOFTWARE ENGINEERING?

- **Software** is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**
- **Engineering** on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering *is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.*

- The outcome of software engineering is an efficient and reliable software product.

WHAT IS SOFTWARE ENGINEERING?

Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.

Software Engineering is a collection of techniques, methodologies and tools that help with the production of

- a high quality software system
- with a given budget
- before a given deadline

while change occurs.

- The seminal definition:

[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

- The IEEE definition:

  Software Engineering:

  (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

  (2) (2) The study of approaches as in (1).

- We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines.
- A small program can be written without using software engineering principles.
- But if one wants to develop a large software product, then software engineering principles are absolutely necessary to achieve a good quality software cost effectively.
- Software engineering principles use two important techniques to reduce problem complexity:

  - abstraction

  - decomposition.

ABSTRACTION

- The principle of abstraction implies that a problem can be simplified by omitting irrelevant details.
- In other words, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and suppress other aspects that are not relevant for the given purpose.
- Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on.
- Abstraction is a powerful way of reducing the complexity of the problem.

DECOMPOSITION

- In this technique, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one.
- However, in this technique any random decomposition of a problem into smaller parts will not help.
- The problem has to be decomposed such that each component of the decomposed problem can be solved independently and then the solution of the different components can be combined to get the full solution.
- A good decomposition of a problem should minimize interactions among various components.
- If the different subcomponents are interrelated, then the different components cannot be solved separately and the desired reduction in complexity will not be realized.

WHY SHOULD YOU LEARN SOFTWARE ENGINEERING?

- Software Engineering helps to apply theoretical knowledge of Computer Science for building high-quality software products for various applications.
- As there is a huge demand for software in every industry, the demand for qualified Software Engineers is also growing high.
- Therefore learning Software Engineering is very helpful to get a job and for knowledge purposes as well.

WHY IS SOFTWARE ENGINEERING REQUIRED?

- Software Engineering is required in every industry, business, and organization to develop software for various applications like supply-chain management, stock management, employees management, accounts management, etc.

- It is becoming an essential part of every company or organization in various domains for efficient business management.

APPLICATIONS OF SOFTWARE ENGINEERING

Following are the different Software Engineering applications:

- Software Development for various domains
- To perform various operations on the software like testing
- Maintenance of various software products
- To apply the knowledge, practices, and technologies to build high-quality software products that enhance productivity in every industry

WHY SOFTWARE ENGINEERING? SOFTWARE CRISIS & ITS SOLUTION:

What was the Software Crisis?

- It was in the late 1960s when many software projects failed.
- Many software became over budget. Output was an unreliable software which is expensive to maintain.
- Larger software was difficult and quite expensive to maintain.
- Lots of software not able to satisfy the growing requirements of the customer.
- Complexities of software projects increased whenever its hardware capability increased.
- Demand for new software increased faster compared with the ability to generate new software.

All the above issues lead to 'Software Crisis.'

The Solution

- Solution was to the problem was transforming unorganized coding effort into a software engineering discipline. These engineering models helped companies to streamline operations and deliver software meeting customer requirements.
- The late 1970s saw the widespread uses of software engineering principles.
- In the 1980s saw the automation of software engineering process and growth of (CASE) Computer-Aided Software Engineering.
- The 1990s have seen an increased emphasis on the 'management' aspects of projects standard of quality and processes just like ISO 9001

WHY SOFTWARE ENGINEERING IS POPULAR?

Here are important reasons behind the popularity of software engineering:

- **Large software** – as the size of the software becomes large, software engineering helps you to build software.
- **Scalability-** If the software development process were based on scientific and engineering concepts, it is easier to re-create new software to scale an existing one.
- **Adaptability**: Whenever the software process was based on scientific and engineering, it is easy to re-create new software with the help of software engineering.
- **Cost-** Hardware industry has shown its skills and huge manufacturing has lower the cost of the computer and electronic hardware.
- **Dynamic Nature**- Always growing and adapting nature of the software. It depends on the environment in which the user works.
- **Quality Management**: Offers better method of software development to provide quality software products.

RELATIONSHIP OF SOFTWARE ENGINEERING WITH OTHER DISCIPLINES

Here, how software engineering related to other disciplines:

- Computer Science: Gives the scientific foundation for the software as electrical engineering mainly depends on physics.
- Management Science: Software engineering is labor-intensive work which demands both technical and managerial control. Therefore, it is widely used in management science.
- Economics: In this sector, software engineering helps you in resource estimation and cost control. Computing system must be developed, and data should be maintained regularly within a given budget.
- System Engineering: Most software is a component of a much larger system. For example, the software in an Industry monitoring system or the flight software on an airplane. Software engineering methods should be applied to the study of this type of systems.

CHALLENGES OF SOFTWARE ENGINEERING

Here are some critical challenges faced by software engineers:

- In safety-critical areas such as space, aviation, nuclear power plants, etc. the cost of software failure can be massive because lives are at risk.
- Increased market demands for fast turnaround time.
- Dealing with the increased complexity of software need for new applications.
- The diversity of software systems should be communicating with each other.

ATTRIBUTES FOR SOFTWARE PRODUCTS

Product Characteristics	**Description**
Maintainability	The software should evolve to meet the changing demands of the clients.
Dependability	Dependability includes various characteristics. Dependable software should never cause any physical or economic damage at the time of system failure.
Efficiency	The software application should overuse system resources like memory and processor cycle.
Usability	The software application should have specific UI and documentation.

CHARACTERISTICS OF GOOD SOFTWARE

Any software should be judged by what it offers and what are the methods which help you to use it.

Every software must satisfy the following attributes:

- Operational
- Transitional
- Maintenance

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

SUMMARY

- Software engineering is a process of analyzing user requirements and then designing, building, and testing software application which will satisfy that requirements
- Important reasons for using software engineering are: 1) Large software, 2) Scalability 3) Adaptability 4) Cost and 5) Dynamic Nature.
- In late 1960s many software becomes over budget. Therefore it offers unreliable software which is expensive to maintain.
- The late 1970s saw the widespread uses of software engineering principles.
- Software engineering concept 1) Computer Science 2) Management Science 3) System engineering and 4) Economics
- Increased market demands for fast turnaround time is the biggest challenges of software engineering field.
- 1)Maintainability, 2) Dependability, 3) Efficiency and, 4) Usability are the most important attributes of software products.
- Three most important characteristics of good software are 1) Operational 2)Transitional 3)Maintenance.

SDLC: PHASES & MODELS OF SOFTWARE DEVELOPMENT LIFE CYCLE

- SDLC stands for Software Development Life Cycle and is also referred to as the Application Development life-cycle.
- **SDLC** is a systematic process for building software that ensures the quality and correctness of the software built.
- SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software.
- Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase.

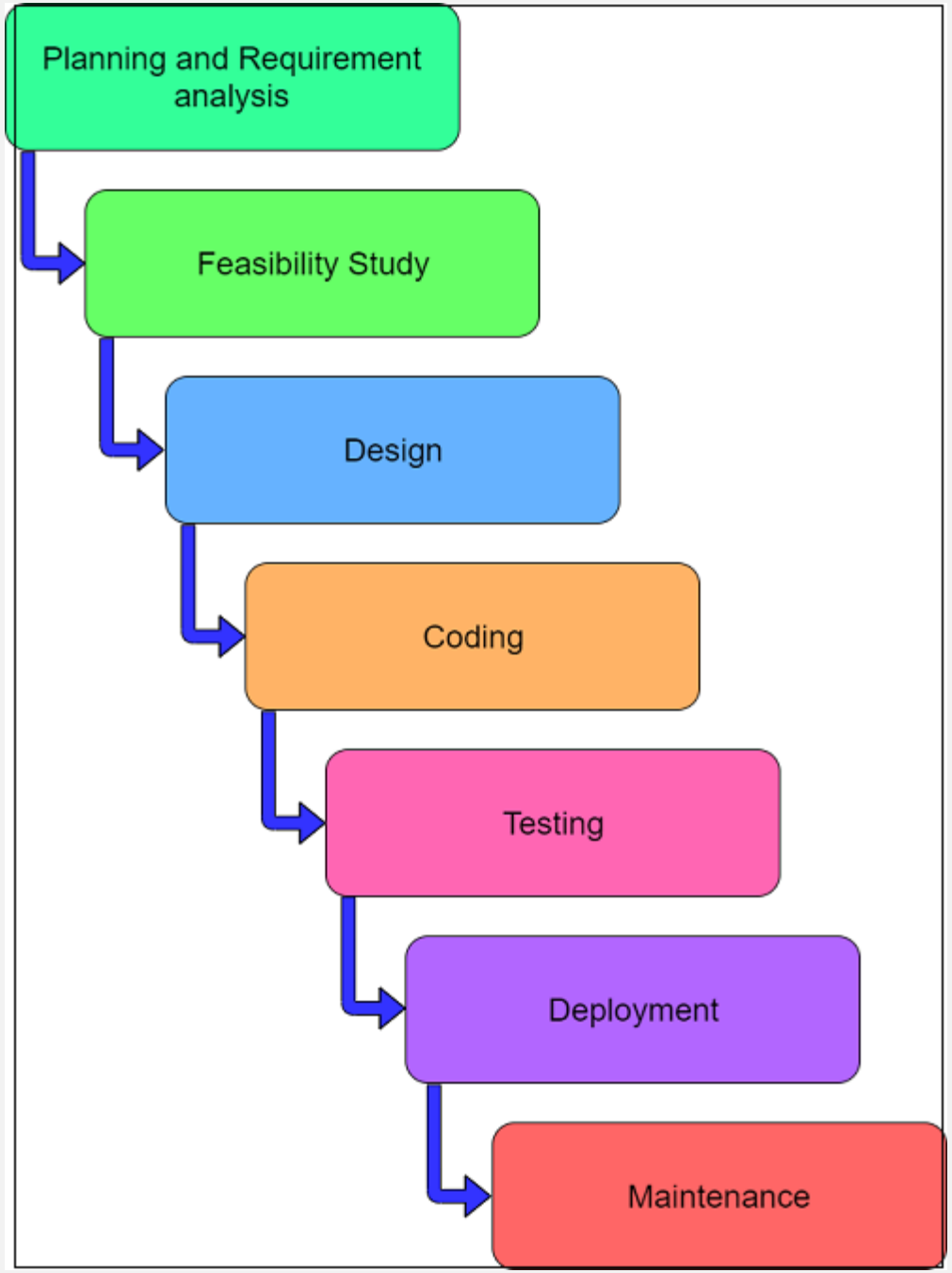
WHY SDLC?

Here, are prime reasons why SDLC is important for developing a software system.

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

SDLC PHASES
THE ENTIRE SDLC PROCESS DIVIDED INTO THE FOLLOWING STAGES:

- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:
- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:


Planning and Requirement analysis
Feasibility Study
Design
Coding
Testing
Deployment
Maintenance

PHASE 1: REQUIREMENT COLLECTION AND ANALYSIS:

- The requirement collection is the first stage in the SDLC process.
- It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry.
- Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.
- This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.
- Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

PHASE 2: FEASIBILITY STUDY

Once the requirement analysis phase is completed the next SDLC step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- Economic: Can we complete the project within the budget or not?
- Legal: Can we handle this project as cyber law and other regulatory framework/compliances.
- Operation feasibility: Can we create operations which is expected by the client?
- Technical: Need to check whether the current computer system can support the software
- Schedule: Decide that the project can be completed within the given schedule or not.

PHASE 3: DESIGN

- In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.
- This design phase serves as input for the next phase of the model.
- There are two kinds of design documents developed in this phase:
  - High-Level Design (HLD)
  - Low-Level Design(LLD)

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

PHASE 4: CODING

- Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language.
- In the coding phase, tasks are divided into units or modules and assigned to the various developers.
- It is the longest phase of the Software Development Life Cycle process.
- In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

PHASE 5: TESTING

- Once the software is complete, and it is deployed in the testing environment.
- The testing team starts testing the functionality of the entire system.
- This is done to verify that the entire application works according to the customer requirement.
- During this phase, QA and testing team may find some bugs/defects which they communicate to developers.
- The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

PHASE 6: INSTALLATION/DEPLOYMENT

- Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts.
- Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

PHASE 7: MAINTENANCE

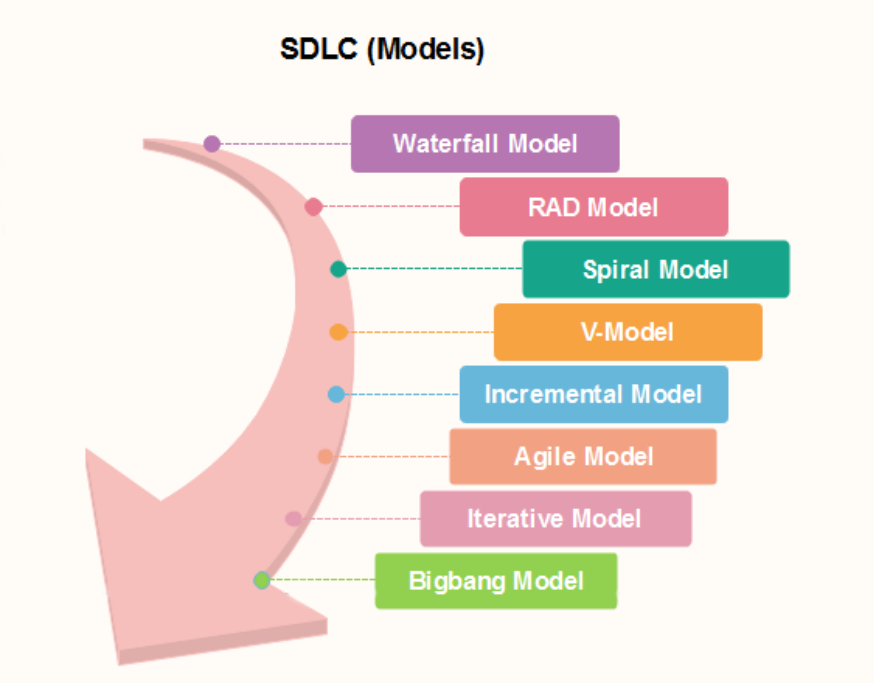
Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

DIFFERENT SOFTWARE LIFE CYCLE MODELS

Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

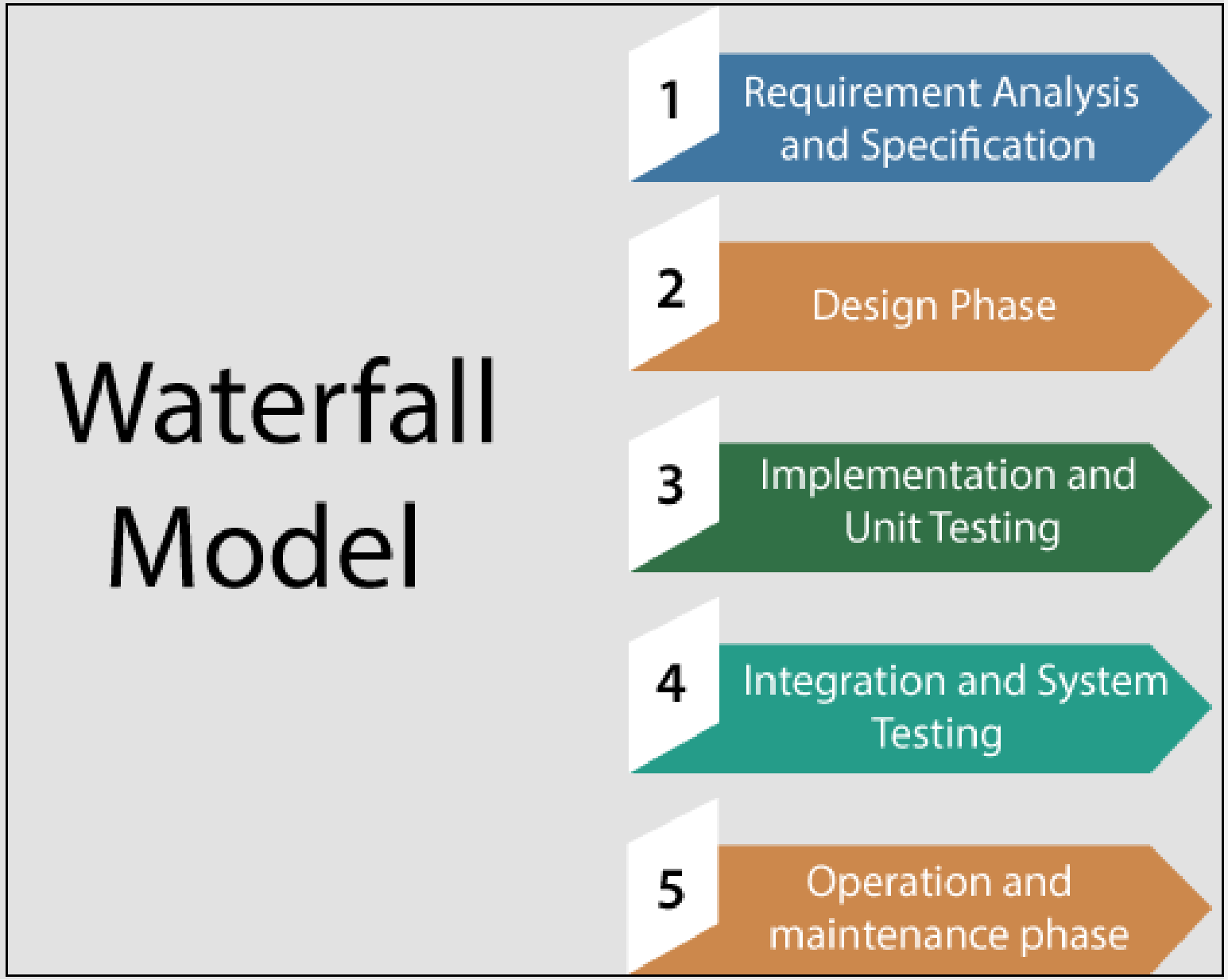


WATERFALL MODEL

- Winston Royce introduced the Waterfall Model in 1970.
- The most popular model is the waterfall model. This model has five phases:
  - Requirement analysis and specification
  - Design
  - Implementation & unit testing
  - Integration and system testing phase
  - Operation and maintenance

  These phases often occur one after the other. The developer should complete each step before the next phase starts.

  This model is called the "Waterfall Model" because its diagrammatic representation resembles a waterfall.


Waterfall Model
1
Requirement Analysis and Specification
2
Design Phase
3
Implementation and Unit Testing
4
Integration and System Testing
5
Operation and maintenance phase

Requirement analysis and specification phase:-

- This phase aims to understand and properly document the exact requirements of the customer.
- This task is usually carried out in collaboration with the customer, as the aim is to record all features, functionality, and configuration specifications for the application.
- The requirements describe the “what” of the system, not the “how” of the system. This process produces a large file, written in a natural language describes what the program will do without explaining how it will be accomplished.
- The resultant document is known as a software requirement specification (SRS) document.
- The SRS document may act as a contract between the developer and the customer. If the developer fails to implement a full set of requirements, it may fail to implement the contracted set of requirements.

Design phase:-

- The SRS document is produced in the previous phase, which contains the exact requirements of the customer.
- The purpose of this step is to convert the requirement specification into a structure that is suitable for implementation in some programming languages.
- Hence, overall software architecture is defined, and the high level and detailed design work are performed.
- This work is documented and known as a software design description (SDD) document. The information included in the Software Design Description should be sufficient to start the coding phase.

Implementation and unit testing phase:-

- During this phase, the design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because of all the information needed by the software developers contained in the SDD.
- During testing, the primary activities are centered around the examination and modification of the code. Small modules were initially tested in isolation from the rest of the software product.
- There are problems associated with the evaluation of unit tests in isolation. How do we run a module without anything to label it, to call it, or, possibly, to intermediate output values obtained during execution?
- These problems are resolved at this point, and the modules are checked after writing some overhead code.

Integration and system testing:-

- This is a very important phase. Good testing can help to deliver higher quality software products, more happy customers, lower maintenance costs, and more accurate and reliable performance.
- It is a very expensive activity and consumes one-third to at least one half the value of a typical development project.
- As we know, the purpose of unit testing is to determine whether each independent module is implemented correctly.
- It gives very little chance to determine if the interface between modules is also correct and, therefore, an integration testing is performed. System testing involves the testing of the entire system, whereas software is a part of the system.
- This is essential to build confidence in the developers before the software is delivered to the customer or released in the market.

Operation and maintenance phase:-

- Software maintenance is a challenge that every team has to face when the software is deployed and installed to the customer's site.
- Therefore, the release of software inaugurates the operation and maintenance phase of the life cycle. The time and effort required to maintain the operating program after launch are very important.
- Despite the fact it is a very challenging task, it is routinely the poorly managed headache that nobody wants to face.
- Software maintenance is a very broad activity that includes error correction, enhancement of capabilities, and optimizations.
- The purpose of this phase is to maintain the value of the software over time.
- This stage can last from 5 to 50 years, while development can last from 1 to 3 years. It is easy to reinforce the model "define before design" and "design before code" design.
- This model sometimes expects complete and accurate requirements, which is unrealistic. There is also no risk assessment involved.

Operation and maintenance phase:-

- Software maintenance is a challenge that every team has to face when the software is deployed and installed to the customer's site.
- Therefore, the release of software inaugurates the operation and maintenance phase of the life cycle.
- The time and effort required to maintain the operating program after launch are very important.
- Despite the fact it is a very challenging task, it is routinely the poorly managed headache that nobody wants to face.
- Software maintenance is a very broad activity that includes error correction, enhancement of capabilities, and optimizations.
- The purpose of this phase is to maintain the value of the software over time. This stage can last from 5 to 50 years, while development can last from 1 to 3 years.
- It is easy to reinforce the model "define before design" and "design before code" design. This model sometimes expects complete and accurate requirements, which is unrealistic. There is also no risk assessment involved.

When to use SDLC Waterfall Model

Waterfall model can be used when

- Requirements are not changing frequently
- Application is not complicated and big
- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable
- Resources are available and trained

Advantages and Disadvantages of Waterfall-Model

Advantages	Dis-Advantages
•Before the next phase of development, each phase must be completed	•Error can be fixed only during the phase
•Suited for smaller projects where requirements are well defined	•It is not desirable for complex project where requirement changes frequently
•They should perform quality assurance test (Verification and Validation) before completing each stage	•Testing period comes quite late in the developmental process
•Elaborate documentation is done at every phase of the software's development cycle	•Documentation occupies a lot of time of developers and testers
•Project is completely dependent on project team with minimum client intervention	•Clients valuable feedback cannot be included with ongoing development phase
•Any changes in software is made during the process of the development	•Small changes or errors that arise in the completed software may cause a lot of problems