CHAPTER – 1

INTRODUCTION

HISTORY OF SOFTWARE ENGINEERING:

The term software engineering first appeared in the late 1950's and early 1960's. Programs might have always known about civil, electrical and computer engineering and debated what engineering might mean for software.

The NATO science committee sponsored two conference on software engineering in 1968 and 1969 and many believes these conferences marked the official start of the profession of software engineering. The term software engineering gets importance after software crisis in 1965 to 1985.

Before 1960's, different individual proposed different solutions to problems like: some would suggest problems in management team, some team organization while some would argue for better languages and tools. Based on these facts, it was required to apply engineering approach for management, team organization, tools, theories, methodology and techniques implemented which finally gave rise to software engineering.

SOFTWARE AND SOFTWARE ENGINEERING:

❖ SOFTWARE:

Software are programs that execute within a computer of any size and architecture and provides desired output. Software is a data structures that enable the programs to adequately manipulate information.

Who builds software?

- Software Engineers

Importance of Software:

- Touch every aspects of lives

❖ SOFTWARE ENGINEERING:

Software engineering encompasses a process, a collection of methods (practices) and an array of tools that allows professionals to build a high quality computer software. Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software and study of these applications of engineering to software.

It is important because it enables us to build complex systems in a timely manner and with high quality. Software engineering can be viewed as a layered technology.

ROLE OF SOFTWARE ENGINEERING:

1. To deal with the increasing complex nature of software, software engineering plays an important role.
2. Software engineering researches, designs and develops software systems to meet with clients requirements. Once the system had been fully designed software engineers then test, debug and maintain the system.
3. Generally, software engineering translates vague requirements and drives into precise specification.
4. It derives model of application and understand the behavior of performance.
5. Provides methods to schedule works, operate systems at various levels and obtain the necessary details of software.
6. Promotes interpersonal skills, communication skills and management skills.

SOFTWARE DEVELOPMENTS:

Software development is the process of developing software through successive phases in an orderly way. This process incudes not only the actual writing of code but also the preparation of requirements and objectives the design of what is to be coded and confirmation that what is developed has met objectives.

The complexity of modern systems and computer programs made the need of clearly and orderly development process for software. The orderly phases are:

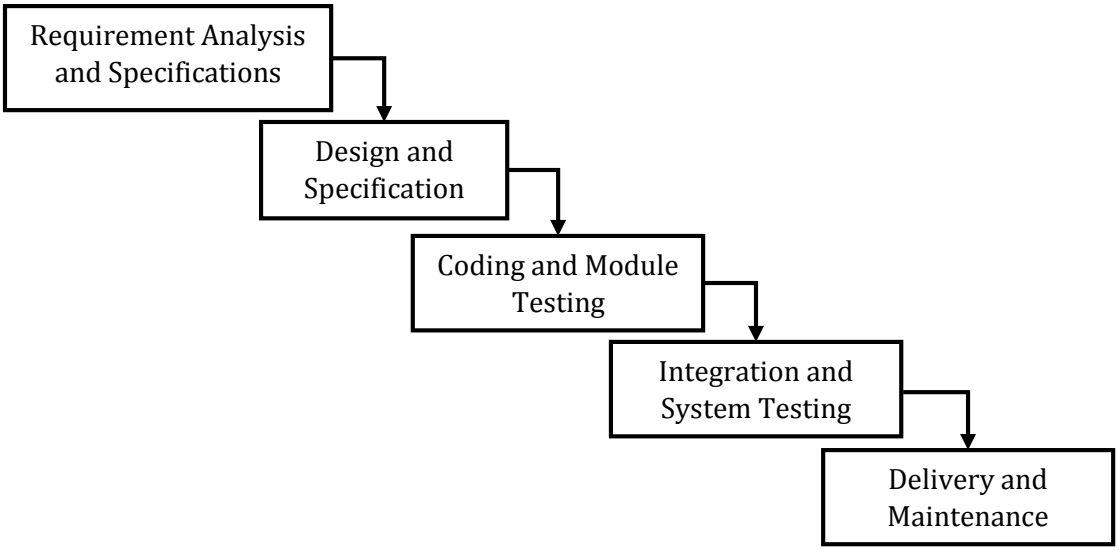

Fig: Waterfall Model

SOFTWARE DEVELOPMENT MODEL:

1. Waterfall Model:

Waterfall model is also known as the traditional or classical model and is popular in most software development and industrial practices. It is a structured process in which the activities are in linear and sequential phases. Similar to the name this model is just like a waterfall flowing downward the phases. The model uses an approach where one phase continues its operation and the previous phase is completed and the processes continuous till the last phase. The common phases of waterfall model are:

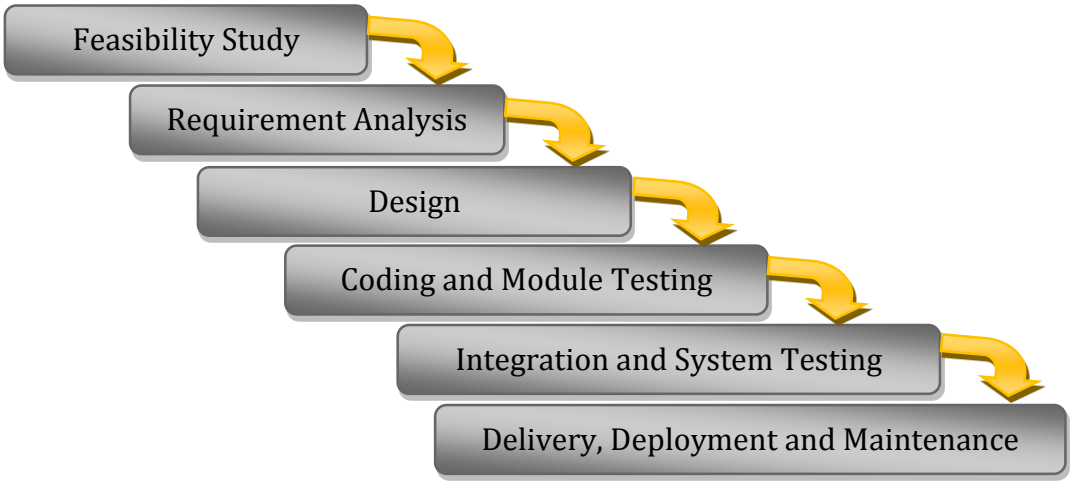

Fig: Waterfall Model

Advantages of Waterfall Model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model - each phase has specific deliverables and a review process.

- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of Waterfall Model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- 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.

2. Spiral Model:

Spiral model is a risk driven process model for the development of software projects. It provides a framework for designing processes including the risk levels associated with them. A spiral model is cyclic model unlike that of the Waterfall Model which is linear programming model. It allows the rapid generation of subsequent phases during the software development phases. It also allows checking the robustness and correctness of the phases. After each cycle a prototype is developed and checked for its robustness and to meet the requirements.

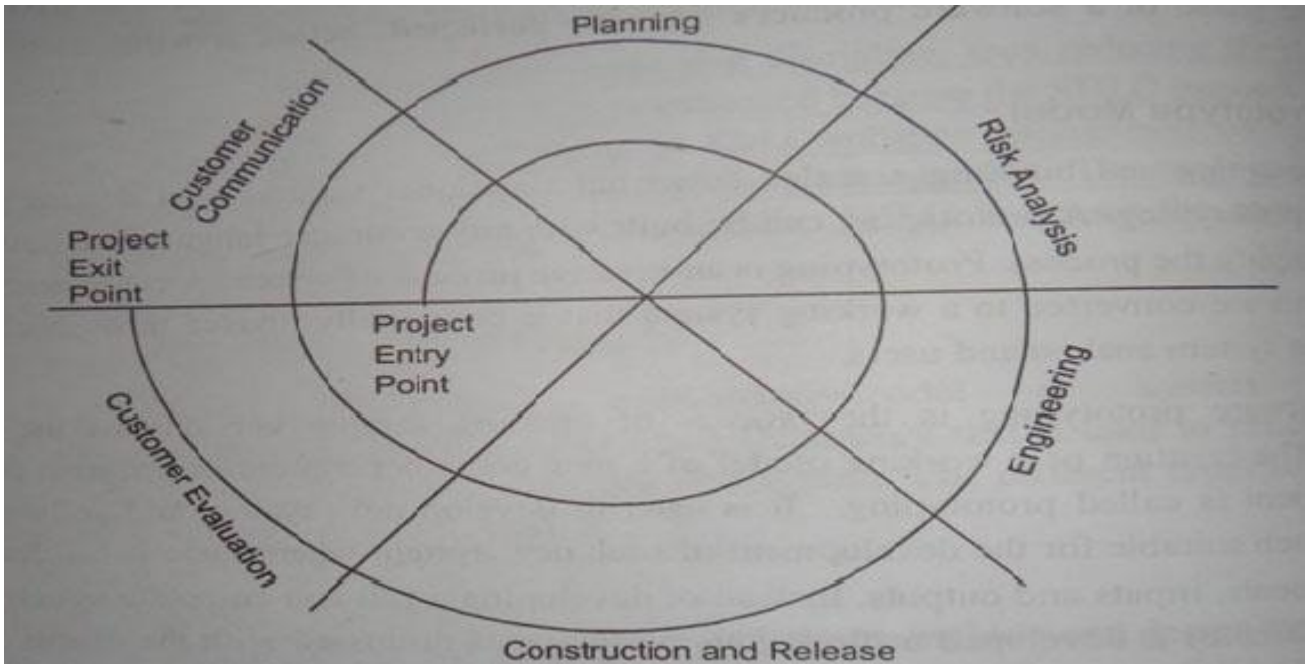

Fig: Spiral Model

Advantages of Spiral Model:

- High amount of risk analysis hence, avoidance of risk is enhanced.

- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional functionality can be added at a later date.
- Software is produced early in the software life cycle.
- Project estimates in terms of schedule, cost etc. become more and more realistic as the project moves forward and loops in spiral get completed.
- It is suitable for high risk projects, where business needs may be unstable. A highly customized product can be developed using this.

Disadvantages of Spiral Model:

- Can be costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.
- It is not suitable for low risk projects.
- May be hard to define objective, verifiable milestones.
- Spiral may continue indefinitely.

3. Prototype or Transformation Model:

A Prototype is a working model (sample) of software with some limited functionality. Prototyping approach of software development tries to implement a sample of the system that shows limited but measure functionality of the proposed system. The goal of the development of Prototyping is to counter the limitations of Waterfall Model. The Prototype displays the functionality of the product under development but may not hold the actual logic of the original software. It enables the customers to understand the system at an early stage of development and may give valuable feedback to the software developer to develop what the costumer actually expect.

Advantages of Prototype Model:

- Increase user involvement in the product even before implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defect can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

Disadvantages of Prototype Model:

- Risk of insufficient requirement analysis owing to too much dependency prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much is not monitored properly.

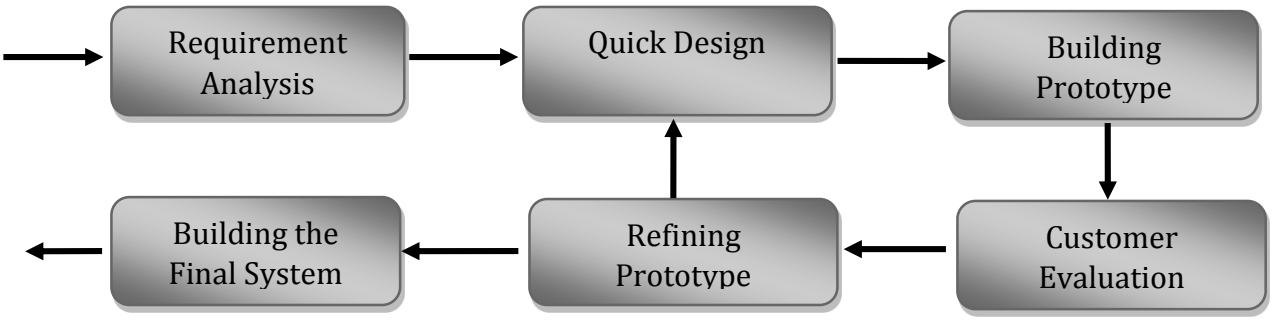

Fig: Prototype Model

4. **Iterative Model:**

A sequence of operation that can run multiple times is called iteration. In computer programming, iterative is used to describe a situation in which a sequence of instructions can be executed multiple times. One complete cycle through the sequence is called iteration. If the sequence is executed repeatedly, it is known as loop. In software development, iterative model uses a sequential planning and development process where an application is developed in small section of sequences called iterations. Each and every iteration are reviewed and analyzed by the development team and the potential end-users insight gained from. The analysis is used to improve and to determine the next step in the development. In software development, iterative model is the most popular model where the loops are used in every component of programs.

5. **The V-Model:**

The V-model represents a software development process which is an extension of waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase to form a v-shape. This model demonstrates the relation between each phases of the development and its associated phase of testing. The horizontals access represents the time taken for completing the project and the vertical access represents definition and the implementation phase of the project.

6. **Rapid Application Development (RAD) model:**

Basic principle of RAD is to create a prototype of the working system in such a way that it reduces the time taken to build a complete package of the system. Since the emphasis of RAD is on reducing the time in development, the initial problem, requirement and decision analysis phases are combined and accelerated. The logical and physical design phases are also treated as a single phase and they are also accelerated. In each of the phases a prototype is created and tested. A

technical feedback of the prototype is analyzed and the system is revised with respect to the requirement and objectives of the system until a final system is placed into operation.

7. The Big Bang Model:

Big Bang Model of software or system development is less used model that is based on the Big Bang Theory. A huge amount of people, money, time and resources are put together, a lot of energy is used and the final system is produced. This model is very simple and there is little or no planning, scheduling or any other formal development process. All the efforts are spend on developing the software and delivering the solution on one row. This model is useful in projects where the requirements are very clear and well understood but the ratio of failure in projects are very high.

8. The Evolutionary Model:

The Evolutionary Model is based on theory of evolution in which a system gradually improves in its functionality, co-operation and the procedure over a considerable period of time. The system may get changes as per requirement in the changing environment of policies, technology and other factors. In the evolutionary model development effort is first made to establish correct requirement definition agreed by all the users in the organization. It is achieved to the applications of gradual processes to evolve a system best suited for a given situation. This process is iterative as it goes to a repetitive process of requirement analysis, designing, testing, prototyping, implementation and evaluation until the users are satisfied.

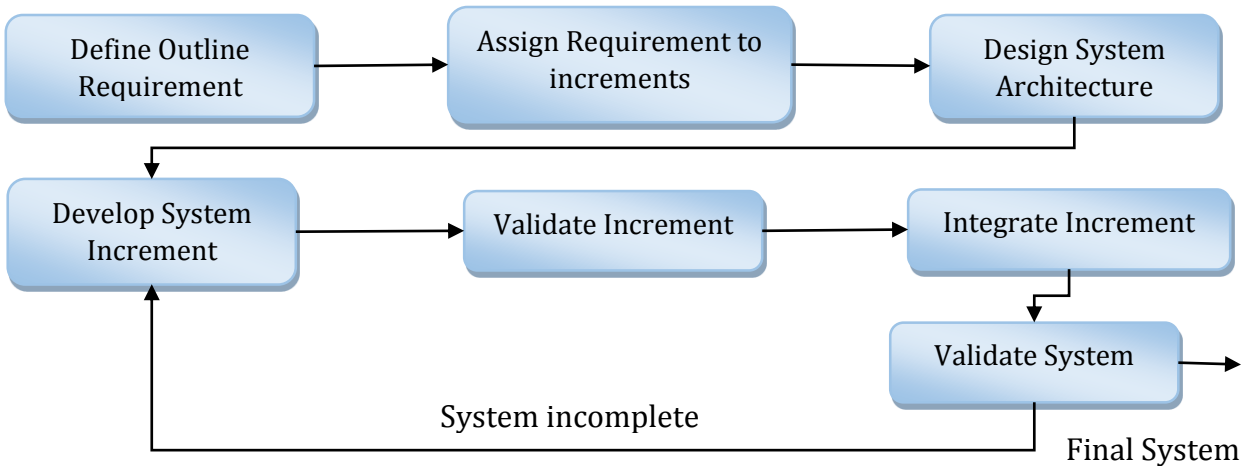

Fig: The Evolutionary Model

ATTRIBUTE OF GOOD SOFTWARE:

Maintainability is "the ease with which changes can be made to satisfy new requirements or to correct deficiencies". Well-designed software should be flexible enough to accommodate future changes that will be needed as new requirements come to light. Since maintenance accounts for nearly 70% of the cost of the software life cycle, the importance of this quality characteristic cannot be overemphasized. Quite often the programmer responsible for writing a section of code is not the one who must maintain it. For this reason, the quality of the software documentation significantly affects the maintainability of the software product.

Correctness is "the degree with which software adheres to its specified requirements". At the start of the software life cycle, the requirements for the software are determined and formalized in the requirements specification document. Well-designed software should meet all the stated requirements. While it might seem obvious that software should be correct, the reality is that this characteristic is one of the hardest to assess. Because of the tremendous complexity of software products, it is impossible to perform exhaustive execution-based testing to insure that no errors will occur when the software is run. Also, it is important to remember that some products of the software life cycle such as the design specification cannot be "executed" for testing. Instead, these products must be tested with various other techniques such as formal proofs, inspections, and walkthroughs.

Reusability is "the ease with which software can be reused in developing other software". By reusing existing software, developers can create more complex software in a shorter amount of time. Reuse is already a common technique employed in other engineering disciplines. For example, when a house is constructed, the trusses which support the roof are typically purchased preassembled. Unless a special design is needed, the architect will not bother to design a new truss for the house. Instead, he or she will simply reuse an existing design that has proven itself to be reliable. In much the same way, software can be designed to accommodate reuse in many situations. A simple example of software reuse could be the development of an efficient sorting routine that can be incorporated in many future applications.

Reliability is "the frequency and criticality of software failure, where failure is an unacceptable effect or behavior occurring under permissible operating conditions". The frequency of software failure is measured by the average time between failures. The criticality of software failure is measured by the average time required for repair. Ideally, software engineers want their products to fail as little as possible (i.e., demonstrate high correctness) and be as easy as possible to fix (i.e., demonstrate good maintainability). For some real-time systems such as air traffic control or heart monitors, reliability becomes the most important software quality characteristic. However, it would be difficult to imagine a highly reliable system that did not also demonstrate high correctness and good maintainability.

Portability is "the ease with which software can be used on computer configurations other than its current one". Porting software to other computer configurations is important for several reasons. First, "good software products can have a life of 15 years or more, whereas hardware is frequently changed at least every 4 or 5 years. Thus good software can be implemented, over its lifetime, on three or more different hardware configurations". Second, porting software to a new computer configuration may be less expensive than developing analogous software from scratch. Third, the sales of "shrink-wrapped software" can be increased because a greater market for the software is available.

DIFFERENCE BETWEEN SOFTWARE ENGINEERING AND SYSTEM ENGINEERING:

1. System Engineers focus more on users and domains, while Software Engineering focus more on implementing quality software.

2. System Engineer may deal with a substantial amount of hardware, software, people, database and other elements involved with that system which is going to be developed but typically software engineers will focus solely on software components.
3. System Engineers may have a broader education (including Engineering, Mathematics and Computer science), while Software Engineers will come from a Computer Science or Computer engineering background.
4. Software engineering is a part of system engineering.

SOME CHALLENGES OF SOFTWARE ENGINEERING:

- The methods used to develop small or medium scale projects are not suitable when it comes to the development of large scale or complex projects.
- Changes in software development are not avoidable.
- Production of high quality software
- Maintenance of software within acceptable cost.
- Informal communications which leads to waste of time, develop the completion of projects in time.
- Verification is difficult.
- Through testing consumes more resources for large projects.

SOFTWARE ENGINEERING ETHICS:

Software engineers shall commit themselves to make the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public software engineers shall adhere (follow) to the following principles:

1. **Public:**

Software engineers should act consistently with the public interest.

2. **Client and Employer:**

Software engineers shall act in a manner that is in the best interests of their client and employer.

3. **Product:**

Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. **Management:**

Software engineers/managers and leaders shall subscribe to and promote and ethical approach to management of software.

5. **Colleagues:**

Software engineers shall be fair to and supportive of their colleagues.