CHAPTER-III
SOFTWARE ENGINEERING

3.1 Overview of Software:
According to IEEE standard Glossary of Software Engineering terminology “software is a collection of computer programs, procedures, rules and associated documentation and data pertaining to the operation of a computer system.” Software consists of various programs used to operate computers and its related devices. A program is a set of instructions that tell a computer what operations to perform. Programs can be built into the hardware itself, or they may exist independently in a form known as software. Hardware describes the physical components of computers and related devices.

There are basically two types of software, System software and Application software. System software are software required by the computer system to run itself. System software includes operating system, device drivers and any program that support application software. System software are often complex and are created using systematic approach called programming systems. Application software is developed to solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decisions. Application software is a simple program that is usually designed, developed, used and maintained by the same person. No systematic approach is required for such type of software.

3.2.1 Evolution of Software Industry:
Today computer software is the single most important technology on the world stage. No one had predicted that software would become indispensible technology in the field of science, business and engineering. It was not perceived that software would enable the creation of new technologies, the extension of existing technology and demise of older technologies. The software is dominating every practical field of human life. The vast industrial field and advent of internet which is the software driven network has changed the world into global village. No one had foreseen that software would become embedded in every walk of life like transportation, communication, education, governance, business, military, entertainment office, factories and the list is endless. The software industry is still growing, changing and future of this industry yet remains unpredictable.

Software is the soul of all IT based systems. It is a set of programs for performing targeted operations. Regardless of its application domain, size, or complexity, computer software will
evolve over time. Change (often referred to as a software maintenance) drives this process and occurs when errors are corrected, when the software is adapted to a new environment, when the customer requests new features or functions, and when the application is reengineered to provide benefit in a modern context.²

3.2.2 The Beginning Era (1955-1965) :
In this era the most crucial development was that new computers were coming out almost every year. Software people had to rewrite all their programs to run on these new machines. Jobs were run by signing up for machine time or by operational staff by putting punched cards for input into the machine's card reader and waiting for results to come back on the printer. The field was so new that the idea of management by schedule was non-existent. Making predictions of a project's completion date was almost impossible. The Computer hardware was application-specific. Scientific and business tasks needed different machines. Hardware vendors gave away systems software for free as hardware could not be sold without software. A few companies sold the service of building custom software but no software companies were selling packaged software.

3.2.3 The Establishing Era (1965-1980) :
This era saw the coming of IBM 360 machines. The job-queue system has been institutionalized. The 360 machine combined scientific and business applications onto one machine. The job control language (JCL) raised a whole new class of problems. The programmer had to write the program in a whole new language to tell the computer and OS what to do. JCL was the least popular feature of the 360. In the middle of this era ‘Structured Programming’ was invented. IBM introduced PL/I to merge all programming languages into one but it failed. The customized applications were still done in-house.

3.2.4 The Micro Era (1980-1999) :
In this era saw dramatically drop in the price of computers thus made ubiquitous computing possible. Now every programmer can have a computer on his desk. The turning point in the evolution of software came in this era with the introduction of user-friendly GUI which replaced previously used JCL technology. The software part of the hardware architecture that the programmer must know about, such as the instruction set, has not changed much since the advent of the IBM mainframe and the first Intelchip. The most-used programming languages
in this era were between 15 and 40 years old. The Fourth Generation Languages never achieved the dream of "programming without programmers" and the idea is pretty much limited to report generation from databases.

3.2.5 New Millennium (1999 to Present ):

This era by far saw the fastest ever growth in the software technology than ever. Software by this era dominates every walk of life. During the later 1990s, Yourdon\(^3\) re-evaluated the prospects of the software professional and suggested the “the rise and resurrection” of the American programmer. As the internet grew in importance, Yourdon’s change of heart proved to be correct. As the twentieth century closed, the focused shifted once more, this time to the impact of the Y2K ‘time-Bomb’.\(^4\) With time, the Y2K problem got fixed efficiently with tireless efforts of software engineers.

As the 2000 progressed competition in the market grew, new technology is rapidly replacing the old technology. Internet, web designing, artificial intelligence, open source, free ware, mobile computing etc these technologies dominated the market. Software became integral part of human life, which resulted in many software companies being established; competition increased and many types of programming language being introduced replacing the older one. This rapid change presents a new challenge for the software engineers.

3.3 Dynamic Mode of Software:

Today, software can be categorized into various distinct categories nature of each category is different from the other which is challenging for the software engineers. Following are the various categories of software used today;

1. System Software: - System Software is a collection of programs written to service other programs. System Software like Compiler, Editor, Assembler etc. process complex but defined, information structure. Other system software like Operating System components, device drivers etc. process large undefined data. The System Software are involved in heavy interaction with hardware: heavy usage of multiple users, concurrent process scheduling, resource sharing, process management, etc.

2. Application software: Application software is developed to solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decisions. Application software is a simple program that is usually designed, developed, used and maintained by the same person. No systematic approach is required for such type of software.
3. **Real-Time software:** Real Time software are those software which monitor/analyzes/controls real world events as they occur. Real time software are made up of four major components data gathering component, analysis component, control component and monitoring component. The data gathering component collects and formats information from an external environment. The analysis component transforms information as required by the application. The control component responds to the external environment. The monitoring component coordinates all other components.

3. **Business Software:** Business information processing is the largest single software application area. Applications in this area (payroll, inventory) restructure existing data in a way that facilitates business operations or management decision making.

4. **Engineering and Scientific Software:** These software basically consist of “number crunching” algorithms. The application area of engineering and scientific software is very vast ranging from geology to space shuttle orbital dynamics, molecular biology to automated manufacturing, weather forecasting to automotive stress analysis and many more. The new modern technologies are taking place of old conventional methods of scientific and engineering software. System simulation, computer-aided design, artificial intelligence and expert systems other interactive applications are some of the examples.

5. **Embedded Software:** Embedded software resides in read-only memory and is used to control products and systems for consumer and industrial markets. Embedded software can perform limited functions (keypad control for a microwave oven) or provide significant function and control capability (digital functions in an automobile such as fuel control, dashboard displays, braking systems).

6. **Personal Computer Software:** The personal computer software market has grown rapidly over the past decade. Word processing, spreadsheets, computer graphics, multimedia, database management, personal and business financial applications are some of the applications.

7. **Artificial Intelligence Software:** Artificial Intelligence software makes use of non-numerical algorithm to solve complex problem that are not liable to computation or straightforward analysis. An active area is Expert systems, also known as knowledge-based systems. Other application areas include pattern recognition, game playing and recently artificial neural network has evolved.

8. **Web-application:** With the rapid growth internet web based application also became popular. The web based applications are in their simple set of hyperlinked pages or complex
interlinked pages written in scripting languages. Web based applications are evolving with growth of e-commerce, social networking and cloud based computing.

9. Ubiquitous Computing: Today with growth of wireless network has led to true distributed computing. The challenge for software engineers is to develop system and application software that will allow small devices, personal computers and large enterprise to communicate over the network.

10. Open Source: Open source is the latest trend of making the source code for system application freely available on the internet so that the customers can make local modification. The challenge for the software engineers here is to write self-explanatory source code and enable both customers and developers to know what changes have been made. Andy Lipman writes about changing era of software as, “We are entering an era characterized by communications among distributed machines and dispersed people, rather than being mostly about a connection between two individuals or between an individual and a machine. The old approach to telephony was about ‘connection to’; the next wave is about ‘connection among’. Napster, Instant Message, Short Message System (SMS), and BlackBerries are examples”.

All the above forms of software and change in their nature present a new challenge to software engineers to develop software which are agile and adaptive in nature. In this ever changing world of technology software engineers play most crucial role as they are the master minds behind the changing nature of software.

3.4.1 Software Engineering:
Software Engineering became apparent in 1960s when the need for systematic approaches to development and maintenance of computer software products was realised. During this decade, third generation computing hardware was invented, and the software techniques of multiprogramming and time-sharing were developed. These capabilities provided the technology for implementation of interactive, multiuser, on-line and real-time computing systems. As the computing systems became larger and more complex, it became apparent that the demand for computer software was growing faster than our ability to produce and maintain it. A workshop was held in Garmisch, West Germany, in 1968, to consider the growing problems of software technology. This workshop and the subsequent one held in Rome, Italy, in 1969, stimulated widespread interest in the technical and managerial processes used to develop and maintain computer software. The term “software engineering”
was first used in these workshops. Since 1968, the applications of digital computers have become increasingly diverse, complex, and critical to modern society. As a result, the field of software engineering has evolved into technological discipline of considerable importance. According to Boehm, software engineering involves “the practical application of scientific knowledge to the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them”\(^6\). The IEEE Standard Glossary of Software Engineering terminology defines software engineering as “The systematic approach to the development, operation, maintenance, and retirement of software”. “Software engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates”. The primary goals of software engineering are to improve the quality of software products and to increase the productivity and job satisfaction of software engineers.

### 3.4.2 Broad View of Software Engineering:

The activities associated with software Engineering can be divided into three generic phases regardless of application area, project size, or complexity.

**Definition phase**: The definition phase focuses on ‘what’ part of the activities. The software developer attempts to identify what information is to be processed, what function and performance are desired, what system behaviour can be expected, what interfaces are to be established, what design constraint sexists, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified.

**Development Phase**: The development phase focuses on ‘how’ part of the activities. During development a software engineer attempts to define how data are to be structured, how function is to implemented, how procedural details are to be implemented, how interfaces are to be characterized, how the design will be translated into a programming language, how testing will be performed.

**Maintenance Phase**: The main focus of ‘maintenance’ phase is on change associated with error correction, adaptations required as software’s environment evolves, and changes due to enhancements brought about by changing customer requirements. Four types of change are encountered during the maintenance phase:
Correction: Corrective maintenance changes the software to correct defects.
Adaptation: Adaptive maintenance results in modification to the software to accommodate changes to its external environment like (CPU, Operating System, business rules etc).
Enhancement: With the passage of time the customer will recognize additional functions that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.
Preventive: Preventive maintenance, often called software reengineering, must be conducted to enable the software to serve the need of its end users because computer software deteriorates due to change. Preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted and enhanced. Many companies pursue software reengineering strategies due the “aging software” concept.

3.4.3 The Software Process:
The software process comprises of all the necessary activities required to develop and maintain the software products. The software process is exhibited in the figure below :-

Figure 3.1 : Broad View of Software Engineering.
A Common Process Framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size and complexity. A number of task sets each a collection of software engineering work tasks, project milestones, software work products and deliverables, and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.

Umbrella activities such as software quality assurance, software configuration management, and measurement overly the process model. Umbrella activities are independent of any framework activity and occur throughout the process.

**3.4.4 Software Project Size**

The size of the software project is the major concern and influences the overall activities of the development and maintenance. The project size determines types of tools and techniques required, the number of programmers required, the duration of the project and the level of management control required for the project. These factors help to determine the type of software engineering model to be used for the project. Yourdon presented following categories of the projects:
Types of Project based on its Size :

1. Trivial Projects : These are the projects or software developed in few days or weeks for personal use by a single programmer having less than 500 lines of code. Often such software are discarded after few months. There is no need of formal software engineering process for trivial projects.

2. Small Projects : A small project contains 1000 to 2000 lines of code developed by a single programmer within a time duration of 1 to 6 months. A systematic software engineering approach may be used for small project with less degree of formalities.

3. Medium Size projects : Medium size projects require 2 to 5 programmers working for the duration of 1 to 2 years resulting in software of 10,000 to 50,000 lines of source code. A formal software engineering principals are used for the medium size projects.

4. Large Projects :Large size projects are complex big software comprising of 50 thousand to 1 lakh lines of code, developed by a team of 5 to 20 programmers in period of 2 to 3 years. These projects essentially needs systematic software engineering activities and proper monitoring by the management.

5. Very Large Projects : Very Large projects are huge projects developed in a period of 4 to 5 years by team of 100 to 1000 programmers resulting in software of 1 million source code instructions.

6. Extremely Large Projects : Extremely large projects are rarely developed. These projects require 10 years of duration with nearly 2000 to 5000 programmers employed resulting in 10 millions of lines of code.

<table>
<thead>
<tr>
<th>Category</th>
<th>No. Of Programmer</th>
<th>Duration</th>
<th>Product Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial</td>
<td>1</td>
<td>1 to 4 wks</td>
<td>500 source lines</td>
</tr>
<tr>
<td>Small</td>
<td>1</td>
<td>1 to 6 months</td>
<td>1K to 2K</td>
</tr>
<tr>
<td>Medium</td>
<td>2 to 5</td>
<td>1 to 2 years</td>
<td>5K to 50K</td>
</tr>
<tr>
<td>Large</td>
<td>5 to 20</td>
<td>2 to 3 years</td>
<td>50K to 100K</td>
</tr>
<tr>
<td>Very Large</td>
<td>100 to 1000</td>
<td>4 to 5 years</td>
<td>1M</td>
</tr>
<tr>
<td>Extremely Large</td>
<td>2000 to 5000</td>
<td>5 to 10 years</td>
<td>1M to 10M</td>
</tr>
</tbody>
</table>

*Figure.3.3 : Categories of Software Products*  
(Source : Yourdon, “Techniques of Program Structure and Design”, Prentice Hall)
3.5.1 SDLC Models (Software Development Life Cycle Models):

A software development process, also known as a software development life cycle (SDLC), is a structure imposed on the development of a software product. Similar terms include software life cycle and software process. It is often considered a subset of systems development life cycle. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Some people consider a lifecycle model a more general term and a software development process a more specific term, (Kevin Reobuck).

The software development life cycle (SDLC) or software process model is a framework defining tasks performed at each step in the software development process. SDLC is a structure followed by a development team within the software organization. It consists of a cycle defines a methodology for improving the quality of software and the overall development process. The development strategy encompassing all the activities required to define, develop, test, deliver, operate and maintain software product is often referred to as Process Model or Software Engineering Paradigm. Different models emphasize different aspects of the life cycle, and no single model is appropriate for all software products. It is important to define a life-cycle model for each software project because the model provides a basis for controlling the various activities required to develop and maintain a software product. The selection of a process model is based on the nature of the project and application, the tools to be used and the controls and deliverables that are required. All software development can be characterized as a problem solving loop in which four distinct stages are encountered: status quo, problem definition, technical development, and solution integration.

3.5.2 The Linear Sequential Model or Waterfall Model:

The Linear Sequential Model also called as “Waterfall Model” or “Classical Life Cycle Model” follows a sequential systematic approach for software development. The development phase begins at requirement specification and progress through analysis, design, coding, testing and maintenance. Following are the steps of waterfall model:

**Feasibility Study**: Before beginning the development process a feasibility study of the project is carried out. Feasibility study deals with determining whether it is feasible to develop the project in terms of finance and technical support. Financial feasibility study determines whether it is financially feasible to undertake the project. Technical feasibility
study determines whether the organization has required technical support needed to build the project. Once the feasibility study is done the development process shift to next phase.

Figure 3.4 : The Linear/Classical/Waterfall Model

**Software Requirement Analysis:** The requirement analysis involves gathering and analyzing different requirement of the software project. All the relevant information of the project is collected. Once the requirements are collected they are analyzed so that any incompleteness or inconsistency in the requirements can be removed. At the end Software Requirement Specification (SRS) document is prepared which contains all the information related to requirement of the project in a systematic manner. This SRS document is further given to the customer for review.

**Design:** The design phase deals with the representation of the requirements mentioned in the SRS document to be designed in a structural form which can be implemented in programming language. The design process translates requirements into a representation of the software that can be assessed for quality before code generation begins. The design is documented and becomes part of the software configuration. Software design can be created using traditional design approach or object oriented design approach.
**Code generation:** Design phase is followed by the coding step where, the design must be translated into a code written in any of the programming language selected. Systematic and detailed design makes code generation easy to accomplish.

**Testing:** Program testing begins after code has been generated. The testing process focuses that all statements have been tested, all errors are uncovered and ensured that defined input will produce actual results that agree with required results. There different methods of testing such unit testing, system testing, black box testing, alpha beta testing etc.

**Maintenance:** Maintenance is the step which covers the majority of the efforts of the development life cycle. Many studies have confirmed that development and maintenance are in 40:60 ratio, thus maintenance involves 60 percent of the overall activities. There are different kind of maintenance activities involved such as software enhancement, perfection or adaptation. Software maintenance reapply each of the preceding phases to an existing program rather than a new one.

“The Waterfall model is oldest paradigm for software engineering. However, over the past two decades, criticism of this process model has caused even ardent supporters question its efficiency”.

The waterfall model is most widely used and oldest model for software engineering. There are some flaws in this sequential model mentioned below:

1. The linear format of the project may create confusion as the project team proceeds. The model does not directly support iteration which presents another difficulty in the process.
2. The major flaw of this model is all the requirement specification are taken at the beginning of the project where customer is forced to explicitly mention all requirements altogether. There is no scope of modifying or adding new requirements till the last phase of the project.
3. Customers are required to wait till the last phase of project to get the working project in hand. There is no scope of having a sample project in between the lifecycle, the working is available only at the end of all phases.

**3. 5.3 The Prototyping Model:**

Prototyping model offers solution to some of the major problems faced in waterfall model. The prototype model solves the major problem faced in requirement specifications. In prototype model, first the requirements of the projects are gathered by a personal interaction between the customer and developer. The objective of the software are identified, general requirements are defined in this phase. The outcome of the requirement phase is a ‘quick design’. The design phase is lead by Prototype phase. In the prototype phase a prototype of
the project is developed. The prototype is evaluated by the customer and is used to refine the requirements for the software to be developed. Often, a customer defines a set of general objectives for software but does not identify detailed input, processing or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human machine interaction should take. In these and many other situations, a prototyping model may offer the best approach. The Prototype model begins with requirement gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known and outline the areas where further definition is required. The output is ‘quick design’. A quick design focuses on a representation of those aspects of the software that will be visible to the customer. The quick design leads to the construction of prototype. The prototype is evaluated by the customer and is used to refine the requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, at the same time enabling the developer to better understand the needs of the customer. The prototype model serves as a mechanism for identifying software requirements. Both the developer and the customer like this model:- Customers get a feel of the actual system and developer gets some idea about the requirements of the customer.

![Figure: 3.5 Prototyping Model](image_url)
Prototype model also have some drawbacks such as:

a. The customer sometimes may compromise with the quality and maintainability of the software.

b. The developer may also compromise in implementation process in order to get the prototype working.

Although problems can occur, prototyping can be an effective model for software engineering.

3.5.4. The Incremental Model:

The increment Model combines elements of the waterfall model applied in an iterative fashion. The increment model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable ‘increments’ of the software.


The increment model is based on the idea of developing an initial implementation at the delivering it to the customer, reviewing changes if any and then upgrading it through several versions until final version of the product is developed. Requirement Specification, Development and Validation activities are concurrent activities rather than separate activities. There is a rapid feedback taken from each of these activities. Each Increment or Version of
the software is delivered to the customer and then additional functionalities are added to the increment, as needed by the customer. The process of creating, delivering and updating increments is continued till the final version of the software is developed. Increment model has many advantages over classical water fall model. Here, customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required. If not then the current increment has to be changed and, new functionalities are added as per the requirements. Increment model are cheaper and easier to make changes in the software. The cost of accommodating changing requirements of the customer is reduced. It is easier to get feedback. The incremental model provide a platform for the user to evaluate the requirements. It is useful when staffing is unavailable for the complete implementation by the deadline. Fewer people can implement early increments. Additional staff can be added to implement the next core product, if needed.

Although, this model has many advantages, it also has certain disadvantages such as, the overall process is not visible at the beginning, system structure tend to degrade and it becomes more complex for large long time projects involving multiple teams.

3.5.5. The Spiral Model :

The Spiral Model is an evolutionary software process model that combines the iterative nature of prototyping and controlled and systematic approach of linear sequential model. In Spiral Model, software is developed in a series of incremental releases. The incremental release might be paper model or prototype during early release. The product becomes increasingly more completed versions during late iterations. Boehm described spiral model as “The spiral development model is risk driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems. It has two main distinguishing features. One is a cyclic approach for incrementally growing a system’s degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions”.

The spiral model is divided into a number of framework activities, also called task regions. There may be between three and six task regions. There may be following task regions :

**Customer Communication** – In this task region proper communication between developer and customer is established.

**Planning** – planning tasks is required to define resources, timelines, and other project related information.
**Risk analysis** – these are the tasks required to assess both technical and management risks.

![The Spiral Model](image)

**Figure 3.7: The Spiral Model**

(Source: Boehm, B., “A spiral Model for Software Development and Enhancement”)

**Engineering** – in these tasks one or more representations of the application is built.

**Construction & Release** – these are the tasks required to construct, test, install and provide user support such as documentation and training.

**Customer Evaluation** – it involves tasks required to obtain customer feedback based on evaluation of the software representations created during engineering stage and implemented during the installation stage.

Each region consists of series of work tasks related to the project to be undertaken. For small projects, the number of work tasks and their formalities are low. For larger, more critical projects, each task region contains more work tasks that are defined to achieve a higher level of formality. As the evolutionary process of project, the software engineering team begins at the core and moves around the spiral in a clockwise direction. The first round of the spiral results in the development of the product specification, later spiral might be used to develop prototype of the project and slowly other rounds of the spiral may develop more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on the feedback derived from customer evaluation. The project manager adjusts the planned number of iterations required to
complete the software. The spiral model is a realistic approach to the development of large scale system and software. Because software evolves as the process progresses, the developer and the customer better understand and react to risks at each evolutionary level. The spiral model maintains the systematic stepwise approach suggested by the classic life cycle, but incorporates it into an iterative framework. But, the spiral model is not a universally accepted model. It may be difficult to convince the customer that the evolutionary approach is controllable. It requires considerable risk assessment expertise, and completely relies on this expertise for success.

### 3.5.6. The Cocomo Model:

The COnstructiveCOstModel is model is specially designed by considering the cost of performing various activities in a software project. The total cost of developing a software project is the sum of costs gained in conducting each phase of the project. The cost of each phase includes the cost of performing processes and preparing the products for that phase along with the cost of verifying the product for its completeness and consistency with respect to all previous phases. Modifications and correctness to the products of previous phases are necessary because the processes of the current phase will expose the inaccuracies, inconsistencies, and incompleteness in those products. The cost of producing the System Definition and the Project Plan is the cost of performing the planning function and preparing the documents, plus the cost of verifying that the System Definition states accurately the customer’s needs and the cost of verifying that the Project Plan is feasible. The cost of preparing the Software Requirement Specification includes the cost of performing requirements definition and preparing the specification document plus the cost of modifying and correcting the System Definition and the Project Plan, plus the cost of verifying that the Software Requirement Specification is complete and consistent with respect to the System Definition and the customer’s need.

### 3.6.1 The Agile Methodology of Software Engineering:

The traditional method of software development focuses on a systematic approach of planning, designing the project before writing the actual code. This method has its strength and weakness, strength lies in the logic of documenting before coding and its weakness is that it is not based on human grounds. The traditional models emphasize on putting all ideas in the form of planning and designing at the beginning and then implement those ideas during the coding but being a human, good idea can click any moment of the development phase at the beginning, middle or end. If a good idea appears after coding it cannot be implemented or
all good ideas cannot appear at the beginning. Thus, there is a need of development methodology which is based on human grounds.

In 2001, group of noted software developers, writers and consultants signed the “Manifesto for Agile Software Development” It stated:

“ We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

*Individuals and Interactions* over processes and tools.

*Working software* over comprehensive documentation.

*Customer Collaboration* over contract negotiation.

*Responding to change* over following a plan.”

The Agile family of development methodologies was born out of a belief that an approach more grounded in human reality would yield better results. "Agile emphasizes building working software that people can get hands on with quickly, versus spending a lot of time writing specifications up front. Agile focuses on small, cross-functional teams empowered to make decisions, versus big hierarchies and compartmentalization by function, and Agile focuses on rapid iteration, with as much customer input along the way as possible.”

---

![Agile Methodology Process Diagram](image-url)
Agile methodology is development process framework based on the iterative approach, process adaptability throughout the life cycle of the project and open collaboration. This approach is short more flexible and have a short time-span iterations which seek improvement for the project in small release, with minimal planning rather long planning. This helps to minimize the overall risk, and allows the project to adapt to changes more quickly. In the Agile Method, there is also involvement of stakeholder, at the end of each iteration the stakeholder is consulted about the product and comments are noted. Each iteration of Agile method spans from one to four weeks, or at the most up to two months, every incremental release is followed by stakeholders feedback and revaluation of the project.

3.6.2 Important Characteristics Of Agile Methodology

- People oriented: Agile methodology framework believes quality of the people is most important in making sure the project is successful. There is more reliance on individual work than prescribing to a general architectural planning. The software quality depends on the quality of the people and how well they work together. The project is built around motivated individuals and the workload should be sustainable at a constant pace with all stakeholders. Trust the people to get the job done. Lightweight Agile methodology believes that the most efficient and effective communication is face to face conversation. Development team does not build large documents and control points. However, this does not mean there is no modeling or documentation. The attention is not towards the documentation but towards technical excellence and good design with simplicity. It believes the best architecture, requirements and design will emerge from self organizing teams. It uses sketch mode- The design activity is centered on informal and incomplete exploration of complex design. The focus is on communicating the design between team members rather than on the completeness of the design.
- Iteration and incremental: Many short time boxed iterations with incremental releases. The iterative approach produces earlier release and better stakeholders' feedback. the feedback is used to review the project priorities and make changes to the requirements, functionalities, estimations, plans, resources, etc. as required. This improves stakeholders confidence and reduces uncertainties in the projects. It is particularly useful to clarify requirements when there are uncertain scopes or changing requirements. It also helps improve the estimate and planning,
especially when metrics are collected in each iteration. Constant feedback reduces the project risks.

- Test driven development: The working software is the primary measure of progress in the project. Thus, each iteration release is tested to be working software.

### 3.6.3.1 Agile methodology for Software Development:

Each of this agile methods are unique in their own approach and implementation, however, they have the same philosophy and characteristic. Basically, they uses both iteration and the continuous feedback to refine and deliver a software.

![Forms of Agile Methodologies](image)

**Figure 3.9: Forms of Agile Methodologies**

### 3.6.3.2 Extreme Programming (XP):

This agile methodology is one of the most popular as well as controversial methodology. Extreme Programming is a approach of delivering high quality software quickly and continuously. XP involves continuous customer feedbacks, testing, planning and closing teamwork. It delivers the working software after a very frequent intervals of 1 to 3 weeks. XP’s major focus is customer involvement with the development team. The customer work very closely with the development team to define and prioritize units of functionality known as ‘User Stories’. The development team to define and prioritize granular units of functionality referred to as ‘User Stories’.

The original XP recipe is based on four simple values – simplicity, communication, feedback, and courage – and twelve supporting practices:
• Planning Game
• Small Releases
• Customer Acceptance Tests
• Simple Design
• Pair Programming
• Test-Driven Development
• Refactoring
• Continuous Integration
• Collective Code Ownership
• Coding Standards
• Metaphor
• Sustainable Pace

3.6.3.3 Scrum:

Ken Schwaber, Mike Beedle, Jeff Sutherland and others have contributed significantly to the evolution of Scrum. “The scrum framework manage and control the iterative and incremental projects. Scrum is one of the most popular agile method. Scrum is very simple process, highly productive and combines various engineering practices promoted by the agile methodologies due to which it gained acceptance widely”.14

In Scrum, the customer termed as “product owner” works closely with the team to identify and prioritize system functionality in form of a "Product Backlog". The Product Backlog consists of features, bug fixes, non-functional requirements, etc. - whatever needs to be done in order to successfully deliver a working software system. When priorities are defined by the Product Owner, cross-functional teams estimate and sign-up to deliver "potentially shippable increments" of software during successive Sprints, typically lasting 30 days. Once a Sprint's Product Backlog is committed, no additional functionality can be added to the Sprint except by the team. Once a Sprint has been delivered, the Product Backlog is analyzed and reprioritized, if necessary, and the next set of functionality is selected for the next Sprint.

3.6.3.4 Crystal Methods:

Crystal methods were introduced by Alistair Cockburn in his book “Crystal Clear: A Human-Power Methodology for Small Teams”. Cockburn described it as “a resource-limited,
cooperative game of invention and communication, with a primary goal of delivering useful, working software and secondary goal of setting up for next game” 15

Crystal Methodology is a collection of agile methodologies such as Crystal Clear, Yellow, Crystal Orange and Others. Crystal is one of most adaptable and simple approach of software development. The unique factors of Crystal Method are team size, system criticality, and project priorities. This method presents project policies, practices and processes in a slightly tailored manner in order to meet project’s unique characteristics. Key benefits of crystal lies in teamwork, communication and simplicity. Crystal methodologies like other agile methods provide frequent delivery of working software, high involvement of customer, and adaptability.

3.6.3.5 Dynamic Systems Development Method (DSDM) :

DSDM was introduced in 1994 for devising and promoting a common industry framework for rapid software delivery. DSDM replaced the unstructured RAD (Rapid Application Delivery) approach to software delivery. DSDM aims to provide a complete foundation for planning managing, executing and scaling Agile and iterative software development projects. DSDM primarily involves strong empowered teams, frequent delivery, integrated testing and stakeholder collaboration. DSDM emphasizes on ‘fitness for business purpose’ as the major criteria for delivery and acceptance of a system.

In DSDM method, requirements are analysed at early stage of the project. The processes are built by rework and all development changes must be reversible. Requirements are planned and delivered in short time-boxes called as iterations . It is also important that not every requirement in a project or time-box is considered critical. Within each time-box, less critical items are included so that if necessary, they can be removed to keep from impacting higher priority requirements on the schedule. The DSDM can be implemented in combination with other iterative methodologies such as Extreme Programming and the Rational Unified Process

3.6.3.6 Lean Development :

The Lean methodology was developed by Tom Poppendieck. The practices and principles of Lean software development are inspired by Lean Enterprise movement and practices of companies like Toyota. Lean methodology is an iterative agile process. Lean development
focuses the team on delivering Value to the customer, and on the efficiency of the mechanism that deliver Value. The features of Lean development are:

1. Amplifying Learning
2. Building Integrity in team
3. Empowering the Team
4. Eliminating Waste
5. Deciding as late as Possible
6. Delivering as Fast as Possible
7. Seeing the whole.

Lean emphasizes the speed and efficiency of development workflow. It relies on rapid and reliable feedback between programmers and customers. Elimination of waste is done through such practices as selecting only the truly valuable features for a system, prioritizing those selected, and delivering them in small batches. Lean uses the idea of work product being ‘pulled’ via customer request. It focuses decision-making authority and ability on individuals and small teams, since research shows this to be faster and more efficient than hierarchical flow of control. Lean also focuses on efficiency of the use of team resources and concentrates that everyone should be productive as much of the time as possible. Lean ensures concurrent work rather that stepwise work flow. It strongly recommends the use of automated unit tests be written simultaneously with the code.

3.6.3.7. Feature-Driven Development (FDD)

The Feature-Driven Development was originally conceived by Peter Coad and his Colleagues as a practical process model for object-oriented software engineering. Stephan Palmer and John Felsing have extended and enhanced Coad’s work, describing an adaptive agile process that can be applied to moderately sized and larger software projects. The FDD is a short-term iteration and model based process, in which the process begins by developing an overall model shape, followed by a series of two week iterations of ‘design by feature, build by feature’. The features are the small results which are useful in the eyes of the customer. The rest of the development process around feature delivery by using following practices.
• Domain Object Modeling
• Developing by Feature
• Component/Class Ownership
• Feature Teams
• Inspections
• Configuration Management
• Regular Builds
• Visibility of progress and results

FDD recommends specific programmer practices such as ‘Regular Builds’ and "Component/Class Ownership"., FDD describes specific, very short phases of work which are to be accomplished separately per feature. These include Domain Walkthrough, Design, Design Inspection, Code, Code Inspection, and Promote to Build. FDD is more straightforwardly than other approaches, and is better suited to larger teams.

3.6.4 Advantages of Agile Methodology:

1. Early release, better stakeholders' feedback.

2. Review the project priorities and make changes to the requirements, functionalities, estimations, plans, resources, as required.

3. Improves stakeholders' confidence and reduce uncertainties.

4. Useful to clarify requirements when there are uncertain scopes or changing requirements.

5. Constant feedback reduces the project risk.

3.6.5 Disadvantages of Agile Methodology:

Due to the flexibility of Agile, changes to the project may be implemented at various junctures of the development process. This however has the potential to lead to long-drawn projects which will overshoot the expected timeline for completion. There will also be much less predictability at the start and throughout the duration of the project about what the final product will actually deliver. Hence, this can prove to be a problem when trying to define a business case or fixed price quotations, which could be potentially risky for the company designing the software.
Requirements are clarified at the last minute and are documented in much less detail when using Agile so as to eliminate wasted effort on deliverables that do not last and to allow flexibility in the project. This leads to the availability of lesser information to new starters in a developer team and hence, creates the potential for misconceptions if teamwork and communication within the team is not properly fostered.

Testing is integrated at the various stages of development. This would mean that testers would be needed throughout the project, effectively raising the cost of resources needed for the project. Furthermore, testing requires a large amount of time to be spent, and hence can create additional costs incurred by the project.

3.7.1 Software Quality Management System:

The software quality management system is the methodology used by the organization to ensure that the software product they develop meets the desired quality. Organizations have a quality system department whose responsibility is to perform activities concerning the management of quality of software. The quality system department should have full support from the top management.

A quality system must be well documented to ensure that there is proper application of quality control procedures. The quality system department performs following activities:

1. Summary reports for the top management presenting the effectiveness of the quality system in the organization.
2. Project Auditing.
3. Proper review of Quality System.

Many quality control standards have been developed which ensure and authorize the quality system of the organization. Following are the international standards for maintaining quality system of the organization:

3.7.2 ISO 9000:

International Organization for Standardization (ISO) is an association of 63 countries established to ensure implementation of uniform standards for the projects. ISO 9000 is a set of guidelines for the production process of the project. ISO 9000 is a series of three standards
ISO 9001, ISO 9002 and ISO 9003. ISO 9001 applies to organizations involved in design, development, services and production of products. It applies to many software development organization. ISO 9002 applies to companies who are involved in manufacturing and not designing such as industries including steel and car manufacturing. ISO 9003 is applicable to organization involved in installation and testing of products.

3.7.2.1 ISO 9001 features:

ISO 9001 applies to software development organizations and it involves following requirements:

1. Document management: It is required to manage, control and authorize all the important document related to software development and maintenance process.

2. Planning: ISO 9001 guidelines involves that the project must be properly planned and progress according to the plan and should be monitored.

3. Review: All the document prepared in all the phases should be properly reviewed and checked for effectiveness and correctness.

4. Testing: The product must undergo proper testing according to the specification.

5. Organizational Aspect: The organizational aspect is also considered such as management reporting of the quality team.

The International Organization for Standardization (ISO) has developed the ISO 9001:2000 standard to define the requirements for a quality management system that will serve to produce higher quality products and thereby improve customer satisfaction.

3.7.3 SEI Capability Maturity Model (SEI CMM):

CMM stands for Capability Maturity Model developed by Software Institute of Engineering at Carnegie Mellon University in Pittsburgh, USA. CMM is a reference model for apprising the software process maturity into different levels. SEI CMM can be used in two ways: capability evaluation and software process assessment. Capability evaluation is used to assess the software process capability of an organization. On the other hand software process assessment is used by the organization to improve its own capability. There are different
levels of SEI CMM to make it easy for the organizations to slowly build their quality system starting from the beginning level.

Following are the 5 levels of SEI CMM model:

Level 0: Incomplete: The software process area is either not performed or does not achieve all goals and objectives.

Level 1: Performed: Software process goals are defined as per the CMMI guidelines and have been satisfied. Different work tasks are conducted to produce defined work products.

Level 2: Managed: At this level various project management practices are implemented such as tracking cost and schedule are established. Various configuration control tools are used and size and cost estimation techniques such as COCOMO, function point analysis are implemented. There is provision of repetition of earlier success of project.

Level 3: Defined: A well defined process for management and development activities is documented. Organizations builds up capabilities of its employees through training programs. At this level process is “tailored from the organization’s set of standard processes according to the organization’s tailoring guidelines and contributed work products, measures and other process-improvement information to the organizational process assets”.

Level 4 - Managed

At this level process metrics is used so that management can effectively control the AS-IS process (e.g., for software development). Management can identify ways to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. Process Capability is established from this level.

Level 5 – Optimizing

The focus at this level is on continually improving process performance through both incremental and innovative technological changes/improvements.

3.7.4. Personal Software Process (PSP):

Personal Software Process is suitable for individual use, developed by David Humphrey in 1997. PSP is designed for the software developers, it emphasizes on the quality and productivity of the software engineer. PSP helps engineers to measure and improve the way
the software engineers work. It develops the personal skills and methods of the software developers by estimating, planning and tracking performance against plans and provides a defined process which can be tuned by individuals. Wats Humphery pointed out that engineers must plan their work and create their own personal data related to plan by using well defined and measured processes. Engineers must individually feel the responsibility to maintain the quality of the product. Superior products are not produced by mistake; engineers must strive to do quality work. It is more efficient to prevent defects than to find and fix them.

To consistently produce quality products, engineers must plan, measure, and track product quality, and they must focus on quality from the beginning of a job. PSP has four levels beginning from 0 to 3. At each level software engineer’s personal efficiency in terms of maintaining quality in work is assessed. Following are the different levels of PSP:

![Figure 3.10: A schematic representation of PSP](Source: Fundamentals of Software Engineering, Rajib Mall)

- **PSP Level 0**: Personal measurement, basic size measurement and coding standards.
- **PSP Level 1**: Time and Scheduling Planning
- **PSP Level 2**: Personal quality management. Design and code reviews.
- **PSP Level 3**: Personal process evolution.

### 3.8 Software Reuse or Component Based Software Engineering:
Developing a software product is expensive and with the increase in demand of software companies need to spend high cost for developing ample of software. The increase in cost of development became point of concern for the project managers. The project managers began to find way for cutting cost of development. One solution for minimizing the development cost is reusing the components of software into the new projects. The reuse approach slowly gained popularity and became household affair for almost all the software development companies. This approach of developing a software by reusing components of other software is termed as component based software engineering. In component based software engineering software are not developed from scratch but instead they are developed by assembling the similar components of other software.

Tracz has described the software component as “3C model – concept, content, and context”\textsuperscript{21}. The concept component is “a description of what the component does”.\textsuperscript{22} The content describes how the component is realized, it is the information intended for those who will test or modify the component. The context helps software engineer to find suitable component to meet application requirements.

In the reuse approach of development almost all the artefacts associated with software development can be reused. Following are the items that can be efficiently reused:

1. Requirement specification
2. Design
3. Code
4. Test cases
5. Knowledge

All the above items can be documented, stored and organized properly for further reusability.

Knowledge is the most abstract development artefact that can be reused. Knowledge is one such artefact of software development which is reused automatically without any intended effort in this direction. Knowledge reuse can also be made a planned strategy to increase efficiency of development process. When knowledge reuse occurs without planning it may not yield enough benefits as compared to a planned strategy of reusing knowledge. There may be difficulties in unplanned reuse of knowledge such as a developer having knowledge
of one product may be placed in a team developing a product of different type of knowledge. The second difficulty in unplanned knowledge reuse is that it becomes difficult for the developer to remember all potential knowledge which is not documented.

Thus, there should be a planned strategy for reusing the potential knowledge of the developer. It is challenging for the organizations to implement planned reuse of knowledge. Organization need to create an environment where the developers can properly document their knowledge and make it available for further reusability. When knowledge of the developer is systematically extracted, documented and made available for further reuse it may increase the efficiency of reusability and also overall development of software.

Issues related to software reuse:

Following issues are considered during the reuse program

1. Component creation: Component creation involves identifying and selecting the potential components that can be reused.

2. Component indexing and storing: Component indexing involves proper classification and grouping of component so that they can easily be searched for reuse. Components can be stored in database management systems for efficient indexing and searching.

3. Component searching: The developers need to search for the right component required. Developers need to adopt a proper method to define component for searching in the database.

4. Component understanding: To reuse the component the developers must have thorough understanding of it. Component should be well documented so that they can be easily understood by the developers.

5. Component Adaptation/ Modification: Some of the component may not fix the problem at hand so these type of component need to be modified before reuse.

6. Repository Maintenance: Once the component repository is created, it need to be maintained properly. New components are added to the repository, older component which cannot be reused should be removed, and faulty component should be corrected or removed.

3.9.1 Emerging Trends:
Software Engineering concepts and practices have evolved with time. With the continuous change in the environment of information technology the practices of software engineering also changed. The change in the environment involves changes in the software types, hardware, networking, peripherals or the operating systems. These changes in the technologies also change the way the software are developed. Roger Pressman expresses his views as “The challenges facing software engineers will get no easier as we move into the second decade of the 21st century. New process models, methods, languages, and tools will emerge. But …There is no silver bullet!”

Following are the changes in the environment which occurred in last 2 decades are:

1. Internet has become immensely popular connecting billions and trillions of computers and users.

2. Computers becoming more and more powerful with drastic drop in their price and size with laptops, ipads and palmtops becoming popular.

3. Rapid progress in networking techniques, and there is drastic increase in the data transfer speed.

4. Lastly, Mobile phones have dramatically captured the world with unbelievable popularity and technology advancement. Today, mobile phone are transformed from a simple communicating device to a handheld computing device with a high speed fixed line connections, GPRS and wireless LAN have become common place.

With the above drastic changes in the technologies, software engineers are facing challenges. Engineers need to develop software that suit these technology change, fast delivery of software is needed. There is an increase in demand of software along with demand of faster delivery, to cope with this problems, software are being developed by teams working from globally distributed locations. Changes in the software development methods are suggested so as to reduce the delivery time. Due to enormous growth in internet, software need to be developed which can support internet interface. Size of software are increasing with the increase in cost. Many new delivery models are proposed to reduce the cost of software delivery. Thus, overall there is demand of change in software engineering paradigm.

Following are the new software engineering trends:

1. Client Server software
2. Service Oriented Architecture (SOA)

3. Software as a Service (SAAS)

4. Software Reengineering

5. Web Engineering.

3.9.2. **Client Server Software**

The client server software is software that runs on client server network. In the client server both client and server are component of the software. This client server software are very popular due to the popularity in the client server network. Features of client server software include ;

1. Client server software provide concurrent processing of the work among clients and server and thereby results in faster processing.

2. Client and Server component are loosely coupled making it easy to understand.

3. There is a high level of flexibility in these software in terms of attaching or removing clients or servers as and when required.

4. The client server technology is cost effective as clients can be run on cheap desktops and servers on sophisticated systems.

5. These software can work on heterogeneous hardware platform.

6. Client server solutions are usually fault tolerant.

7. Mobile computing is supported by client server solutions.

8. Client server solutions implement component based development which promises reduction in cost, time and effort of development process.

Thus, client server is latest emerging trend in the software development domain.

3.9.3. **Software as a Service**

In today’s age of technology it is very expensive to own a software. Companies buy software worth thousands and lakhs of rupees, and later also pay the annual monthly charge (AMC) for
maintaining the software, tracking and correcting errors, and keeping it free from virus. Thus overall a software costs heavy for the company even when the usage of the software is only few hours a week. In this situation, it is economical to pay for hourly usage of the software rather owning the software and later paying the AMC. This is the idea of SaaS (Software as a Service).

SaaS is basically a software delivery model and involves customers to pay for any software per unit time of usage. SaaS shifts ownership of the software from customer to a service provider. The software owner provides maintenance, technical operation and support of the software. Services are provided to the clients on amount of usage basis. The service provider is the vendor who hosts the software. The cost of providing software service reduces when more and more customers are subscribed. SaaS makes software available to large number of customers who cannot afford to buy it.

“ The term ‘software as a service’ (SaaS) is considered to be part of the nomenclature of cloud computing, along with infrastructure as a service (IaaS), platform as a service (PaaS), desktop as a service (DaaS), backend as a service (BaaS), and information technology management as a service (ITMaaS)”.

3.9.4 Service Oriented Architecture :

Service Oriented Architecture (SOA) is inspired by the object oriented designing. SOA views software as a set of services and in turn a service also composed smaller services. Services can be combined by other software applications that together, provide the complete functionality of a large software application. These services are implemented and provided by a component to a software developer. Component which provides service guarantee that the services are as per the specification. Examples of services are; viewing online bank-statement, filling form or placing online order. Each service is independent of other services. An application that integrates services together work on distributed system architecture. SOA principally leverage the internet. An application is built by using the services available on the internet and writing only the missing part. SOA facilitates developing large services by integrating existing software services. This leads to faster development.
References:


http://www.agilemanifesto.org


http://www.crystalmethodologies.com


