UNIT-I

1.1 What is Software Engineering?

1. The term software engineering is the product of two words, software, and engineering.
2. The software is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages. Computer programs and related documentation such as requirements, design models and user manuals.

Engineering is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc.

Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

Why is Software Engineering required?

Software Engineering is required due to the following reasons:

1. To manage Large software
2. For more Scalability
3. Cost Management
4. To manage the dynamic nature of software
5. For better quality Management

Need of Software Engineering

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

1. **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
2. **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
3. **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.
4. **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
5. **Quality Management:** Better procedure of software development provides a better and
6. quality software product.

1.What is software engineering?

I. Your thoughts here

1.Related to the process: a systematic procedure used for the analysis, design, implementation, test and maintenance of software.

2.Related to the product: the software should be efficient, reliable, usable, modifiable, portable, testable, reusable, maintainable, interoperable, and correct.

2. *The definition in IEEE Standard*:

1.Theapplicationofasystematic,disciplined,quantifiableapproachtothedevelopment, operation, and maintenance of software, that is, the application of engineering to software.

2.Thestudyofapproachesasin1993:TheJointIEEEComputerSocietyandACMSteering Committee for the establishment of software engineering as a profession.

1.2 The Evolving Role of Software:

1. Software is a product

1. Delivers computing potential
2. Produces,manages,acquires,modifies,displays,or transmits information.

2. Software is a vehicle for delivering a product

1. Supports or directly provides system functionality
2. Controls other programs(e.g.,an operating system)
3. Effects communications(e.g.,networking software)
4. Helps build other software(e.g.,softwaretools)

3. The Law of Continuing Change (1974):E-type systems must be continually adapted else they become progressively less satisfactory.

4. The Law of Increasing Complexity(1974):As an E-Type system evolves its complexity increases unless work is done to maintain or reduce it.

5. The Law of Self Regulation (1974):The E-type system evolution process is self-regulating with distribution of product and process measures close to normal.

6. The Law of Conservation of Organizational Stability(1980):The average effective global activity rate in an evolving E-type system is invariant over product lifetime.

7.The Law of Conservation of Familiarity (1980): As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behavior to achieve satisfactory evolution.

8.The Law of Continuing Growth (1980):The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.

9.The Law of Declining Quality (1996): The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.

10. The Feedback SystemLaw (1996):E-typeevolutionprocessesconstitutemulti-level,multi-loop,multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

1.3 THE CHANGING NATURE OF SOFTWARE

The 7 broad categories of computer software present continuing challenges for software engineers:

1. System software
2. Application software
3. Engineering/scientific software
4. Embedded software
5. Product-line software
6. Web-applications

7. Artificial intelligence software.

1. System software: System software is a collection of programs written to service other programs. The systems software is characterized by

1. heavy interaction with computer hardware
2. heavy usage by multiple users
3. concurrent operation that requires scheduling, resource sharing, and sophisticated process management
4. complex data structures
5. multiple external interfaces

E.g. compilers, editors and file management utilities.

6. Application software:

1. Application software consists of standalone programs that solve a specific business need.
2. It facilitates business operations or management/technical decision making.
3. It is used to control business functions in real-time
4. *E.g.* point-of-sale transaction processing, real-time manufacturing process control.

5. **Engineering/Scientific software:**

1. Engineering and scientific applications range
2. from astronomy to volcanology
3. from automotive stress analysis to space shuttle orbital dynamics
4. from molecular biology to automated manufacturing

E.g. computer aided design, system simulation and other interactive applications.

5. Embedded software:

1. Embedded software resides within a product or system and is used to implement and control features and functions for the end-user and for the system itself.
2. It can perform limited and esoteric functions or provide significant function and control capability.

E.g. Digital functions in automobile, dashboard displays, braking systems etc.

3. **Product-line software:** Designed to provide a specific capability for use by many different customers, product-line software can focus on a limited and esoteric market place or address mass consumer markets

E.g. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications

4. **Web-applications:** Web Apps are evolving into sophisticated computing environments that not only provide standalone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

5. **Artificial intelligence software:** AI software makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Application within this area includes robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing.

The following are the **new challenges** on the horizon:

1. **Ubiquitous computing**
2. **Netsourcing**
3. **Open source**
4. **The "new economy"**

Ubiquitous computing: The **challenge** for software engineers will be to develop systems and application software that will allow small devices, personal computers and enterprise system to communicate across vast networks.

Net sourcing: The **challenge** for software engineers is to architect simple and sophisticated applications that provide benefit to targeted end-user market worldwide.

Open Source: The **challenge** for software engineers is to build source that is self descriptive but more

importantly to develop techniques that will enable both customers and developers to know what changes have been made and how those changes manifest themselves within the software.

The "new economy": The **challenge** for software engineers is to build applications that will facilitate mass communication and mass product distribution.

1.4 SOFTWARE MYTHS

Beliefs about software and the process used to build it- can be traced to the earliest days of computing myths have a number of attributes that have made them insidious.

Management myths: Manages with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.

Myth: We already have a book that's full of standards and procedures for building software - Wont that provide my people with everything they need to know?

Reality: The book of standards may very well exist but, is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice?

Myth: If we get behind schedule, we can add more programmers and catch up.

Reality: Software development is not a mechanistic process like manufacturing. As new people are added, people who were working must spend time educating the new comers, thereby reducing the amount of time spend on productive development effort. People can be added but only in a planned and well-coordinated manner.

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm built it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths: The customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations and ultimately, dissatisfaction with the developer.

Myth: A general statement of objectives is sufficient to begin with writing programs - we can fill in the details later.

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous statement of objectives is recipe for disaster.

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced and change can cause upheaval that requires additional resources and major design modification.

Practitioner's myths: Myths that are still believed by software practitioners: during the early days of software, programming was viewed as an art from old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our jobs are done.

Reality: Someone once said that the sooner you begin writing code, the longer it'll take you to get done. Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is only one part of a software configuration that includes many elements. Documentation provides guidance for software support.

Myth: software engineering will make us create voluminous and unnecessary documentation and will invariably slows down.

Reality: software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

2.A GENERIC VIEW OF PROCESS

1. SOFTWAREENGINEERING-A LAYERED TECHNOLOGY

Software engineering is a layered technology. Any engineering approach must rest on an organizational commitment to quality. **The bedrock that supports software engineering is a quality focus.**

The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers. **Process defines a framework that must be established for effective delivery of software engineering technology**.

The software forms the basis for management control of software projects and establishes the context in which

1. Technical methods are applied,
2. Work products are produced,
3. Milestones are established,
4. Quality is ensured,
5. And change is properly managed.

Software engineering methods rely on a set of basic principles that govern area of the technology and include modeling activities.

Methods encompass a broad array of tasks that include

1. Communication,
2. Requirements analysis,
3. Design modeling,
4. Program construction,
5. Testing and support.

Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

2.2A PROCESS FRAMEWORK

1. **Software process** must be established for effective delivery of software engineering technology.
2. A **process framework** establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
3. The process framework encompasses a **set of umbrella activities** that are applicable acrossthe entire software process.
4. Each **framework activity** is populated by a set of software engineering actions
5. Each **software engineering action** is represented by a number of different task sets- each a collection of software engineering work tasks, related work products, quality assurance points, and project milestones.

In brief

"A **process** defines who is doing what, when, and how to reach a certain goal." A **Process Framework**

1. Establishes the foundation for a complete software process
2. Identifies a small number of **framework activities**
3. Applies to all s/w projects, regardless of size/complexity.
4. Also, set of **umbrella activities**
5. Applicable across entire s/w process.

Each **framework activity** has

1.Set of **s/w engineering actions**.
2.Each **s/w engineering action** (e.g., design) has
3.Collection of related **tasks (**called **task sets):**
4.Work tasks
5.Work products (deliverables)
6.Quality assurance points
7. Project milestones.

Generic Process Framework: It is applicable to the vast majority of software projects

1. Communication activity
2. Planning activity
3. Modelling activity
4. Analysis action
5. Requirements gathering work task
6. Elaboration work task
7. Negotiation work task
8. Specification work task
9. Validation work task
10. Design action
11. Data design work task
12. Architectural design work task
13. Interface design work task
14. Component-level design work task
15. Construction activity
16. Deployment activity

1. Communication: This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.

2.Planning: This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

3.Modeling: This activity encompasses the creation of models that allow the developer and customer to better understand software requirements and the design that will achieve those requirements. The modelling activity is composed of 2 software engineering actions- analysis and design.

i) Analysis encompasses a set of work tasks.

ii)Design encompasses work tasks that create a design model.

4.Construction: This activity combines core generation and the testing that is required to uncover the errors in the code.

5.Deployment: The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evolution.

These 5 generic framework activities can be used during the development of small programs, the creation of large web applications, and for the engineering of large, complex computer-based systems.

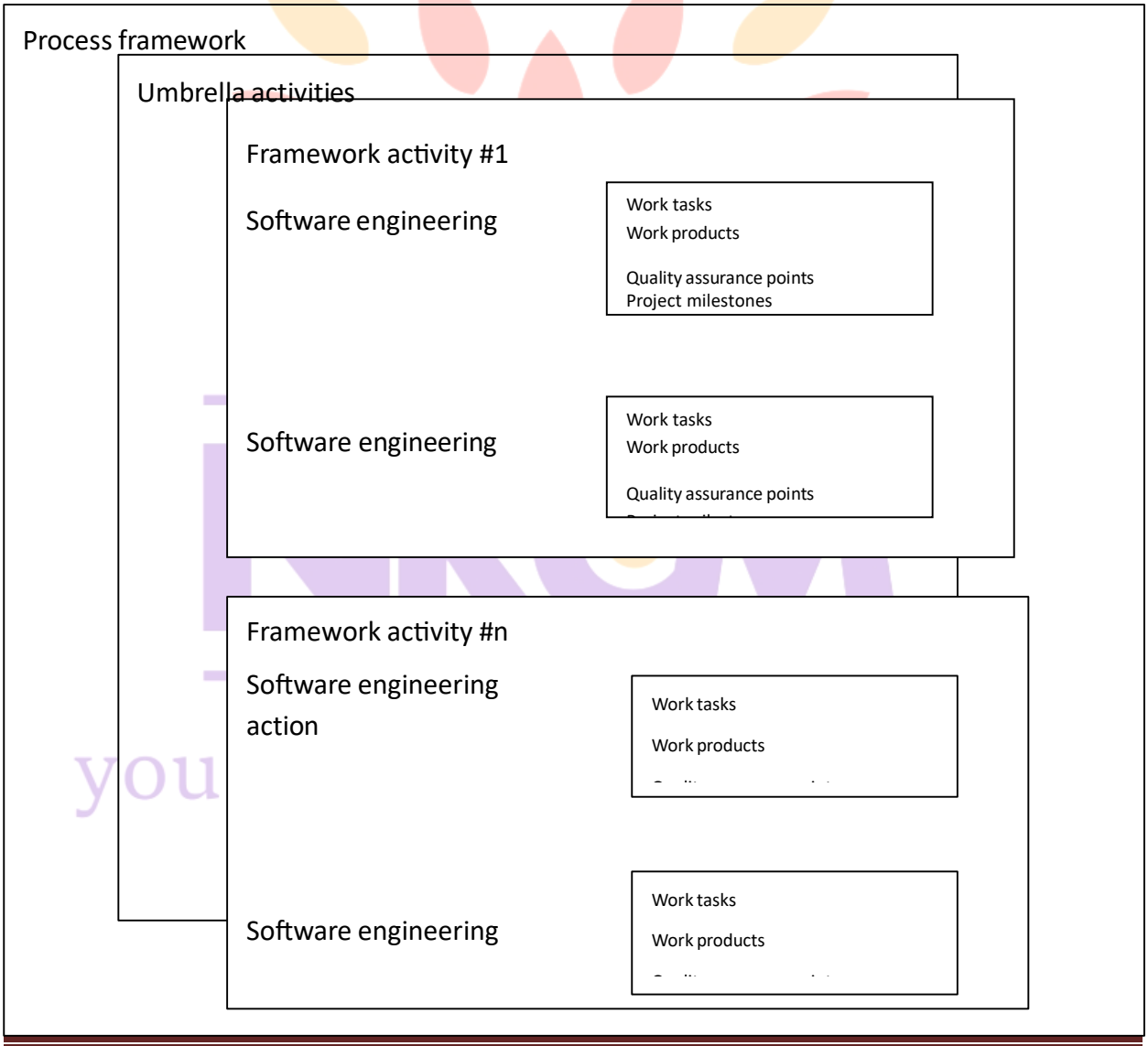
The following are the set of **Umbrella Activities.**

1. **Software project tracking and control –** allows the software team to assess progress against the project plan and take necessary action to maintain schedule.
2. **Risk Management** - assesses risks that may effect the outcome of the project or the quality of the product.
3. **Software Quality Assurance** - defines and conducts the activities required to ensure software quality.
4. **Formal Technical Reviews** - assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next action or activity.
5. **Measurement** - define and collects process, project and product measures that assist the team in delivering software that needs customer's needs, can be used in conjunction with all other framework and umbrella activities.
6. **Software configuration management** - manages the effects of change throughout the software process.
7. **Reusability management** - defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
8. **Work Product preparation and production** - encompasses the activities required to create work products such as models, document, logs, forms and lists.

Intelligent application of any software process model must recognize that adaption is essential for success but process models do differ fundamentally in:

1. The overall flow of activities and tasks and the interdependencies among activities and tasks.
2. The degree through which work tasks are defined within each frame work activity.
3. The degree through which work products are identified and required.
4. The manner which quality assurance activities are applied.
5. The manner in which project tracking and control activities are applied.
6. The overall degree of the detailed and rigor with which the process is described.
7. The degree through which the customer and other stakeholders are involved with the project.
8. The level of autonomy given to the software project team.
9. The degree to which team organization and roles are prescribed.

2.3THE CAPABILITY MATURITY MODEL INTEGRATION(CMMI)



The CMMI represents a process meta-model in two different ways:

1. As a continuous model
2. As a staged model.

Each process area is formally assessed against specific goals and practices and is rated according to the following capability levels.

Level 0: Incomplete. The process area is either not performed or does not achieve all goals and objectives defined by CMMI for level 1 capability.

Level 1: Performed. All of the specific goals of the process area have been satisfied. Work tasks required to produce defined work products are being conducted.

Level 2: Managed. All level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are "monitored, controlled, and reviewed;

Level 3: Defined. All level 2 criteria have been achieved. In addition, the process is "tailored from the organizations set of standard processes according to the organizations tailoring guidelines, and contributes and work products, measures and other process-improvement information to the organizational process assets".

Level 4: Quantitatively managed. All level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment. "Quantitative objectives for quality and process performance are established and used as criteria in managing the process"

Level 5: Optimized. All level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative means to meet changing customer needs and to continually improve the efficacy of the process area under consideration"

The CMMI defines each process area in terms of "specific goals" and the "specific practices" required to achieve these goals. Specific practices refine a goal into a set of process-related activities.

The specific goals (SG) and the associated specific practices(SP) defined for project planning are

SG 1 Establish estimates

SP 1.1 Estimate the scope of the project

SP 1.2 Establish estimates of work product and task attributes

SP 1.3 Define project life cycle

SP 1.4 Determine estimates of effort and cost

SG 2 Develop a Project Plan

SP 2.1 Establish the budget and schedule

SP 2.2 Identify project risks

SP 2.3 Plan for data management

SP 2.4 Plan for needed knowledge and skills

SP 2.5 Plan stakeholder involvement

SP 2.6 Establish the project plan

SG 3 Obtain commitment to the plan

SP 3.1 Review plans that affect the project

SP 3.2 Reconcile work and resource levels

SP 3.3 Obtain plan commitment

In addition to specific goals and practices, the CMMI also defines a set of five generic goals and related practices for each process area. Each of the five generic goals corresponds to one of the five capability levels. Hence to achieve a particular capability level, the generic goal for that level and the generic practices that correspond to that goal must be achieved. To illustrate, **the generic goals (GG) and practices (GP)** for the project planning process area are

GG 1 Achieve specific goals

GP 1.1 Perform base practices

GG 2 Institutionalize a managed process

GP 2.1 Establish and organizational policy

GP 2.2 Plan the process

GP 2.3 Provide resources

GP 2.4 Assign responsibility

GP 2.5 Train people

GP 2.6 Manage configurations

GP 2.7 Identify and involve relevant stakeholders

GP 2.8 Monitor and control the process

GP 2.9 Objectively evaluate adherence

GP 2.10 Review status with higher level management

GG 3 Institutionalize a defined process

GP 3.1 Establish a defined process

GP 3.2 Collect improvement information

GG 4 Institutionalize a quantitatively managed process

GP 4.1 Establish quantitative objectives for the process

GP 4.2 Stabilize sub process performance

GG 5 Institutionalize and optimizing process

GP 5.1 Ensure continuous process improvement

GP 5.2 Correct root causes of problems

3.PROCESS MODELS

Prescriptive process models define a set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software. These process models are not perfect, but they do provide a useful roadmap for software engineering work.

A prescriptive process model populates a process framework with explicit task sets for software engineering actions.

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software.

Communication

1. Involves communication among the customer and other stake holders.
2. It encompasses requirements gathering.

Planning

1. Establishes a plan for software engineering work
2. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry.
3. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.
4. Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage.
5. It addresses technical tasks, resources, work products and work schedule

Modelling (Analyze and Design)

6. Requirement analysis is the most important and fundamental stage
7. A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any).
8. The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS - Design Document Specification.

Construction (Code and Test)

9. In this stage of SDLC the actual development starts and the product is built.

10. Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code.
11. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

12. The testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS - Software Requirement Specification.

Deployment

13. Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).
14. Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

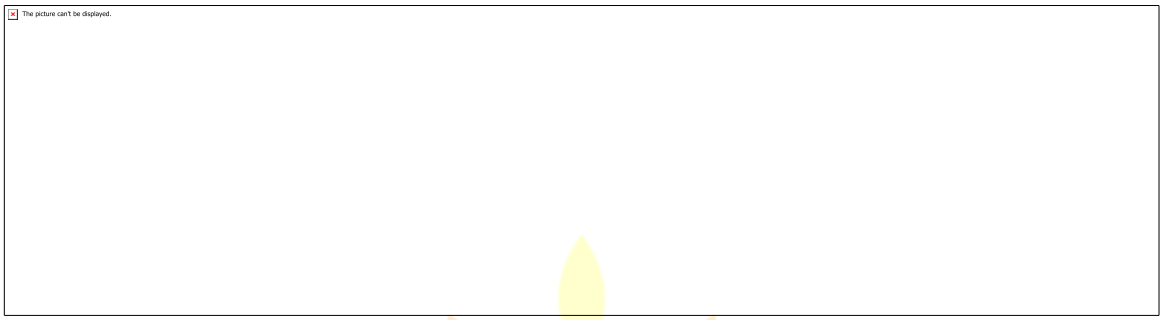
3.1THE WATERFALL MODEL:

The waterfall model, sometimes called the *classic life cycle*, suggests a systematic sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.

Context: Used when requirements are reasonably well understood.

Advantage:

It can serve as a useful process model in situations where requirements are fixed and work is to proceed to complete in a linear manner.

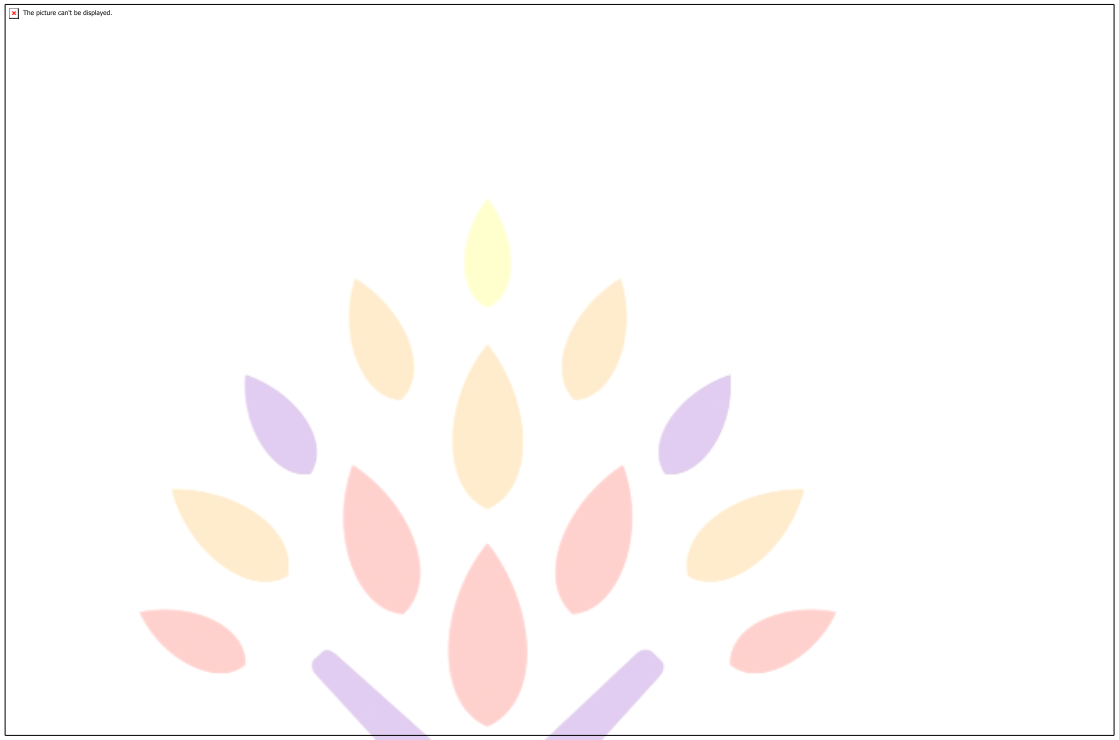


The **problems** that are sometimes encountered when the waterfall model is applied are:

1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
2. It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exist at the beginning of many projects.
3. The customer must have patience. A working version of the programs will not be available until late in the project time-span. If a major blunder is undetected then it can be disastrous until the program is reviewed.

3.2THE SPIRAL MODEL

1. The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
2. The spiral model can be adapted to apply throughout the entire life cycle of an application, from concept development to maintenance.
3. Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.



1. **Anchor point milestones**- a combination of work products and conditions that are attained along the path of the spiral- are noted for each evolutionary pass.

2. The first circuit around the spiral might result in the development of product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.

3. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software.

4. It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world.

1. The first circuit around the spiral might represent a "**concept development project**" which starts at the core of the spiral and continues for multiple iterations until concept development is complete.

2. If the concept is to be developed into an actual product, the process proceeds outward on the spiral and a "**new product development project**" commences.

3. Later, a circuit around the spiral might be used to represent a "**product enhancement project**." In essence, the spiral, when characterized in this way, remains operative until the software is retired.

Context: The spiral model can be adopted to apply throughout the entire life cycle of an application, from concept development to maintenance.

Advantages:

It provides the potential for rapid development of increasingly more complete versions of the software.

The spiral model is a realistic approach to the development of large-scale systems and software. The spiral model uses prototyping as a risk reduction mechanism but, more importantly enables the developer to apply the prototyping approach at any stage in the evolution of the product.

Draw Backs:

The spiral model is not a panacea. It may be difficult to convince customers that the evolutionary approach is controllable. It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.

3.3 Agile Methodology

The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration

4. Testing/ Quality assurance
5. Deployment
6. Feedback

1.Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Agile Testing Methods:

1. Scrum

2. Crystal
3. Dynamic Software Development Method(DSDM)
4. Feature Driven Development(FDD)
5. Lean Software Development
6. eXtreme Programming(XP)

Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

1. Scrum Master: The scrum can set up the master team, arrange the meeting and remove obstacles for the process
2. Product owner: The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
3. Scrum Team: The team manages its work and organizes the work to complete the sprint or cycle.

extreme Programming(XP)

This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

Crystal:

There are three concepts of this method-

1. Chartering: Multi activities are involved in this phase such as making a development team, performing feasibility analysis, developing plans, etc.
2. Cyclic delivery: under this, two more cycles consist, these are:
    1. Team updates the release plan.
    2. Integrated product delivers to the users.
3. Wrap up: According to the user environment, this phase performs deployment, post-deployment.

Dynamic Software Development Method(DSDM):

DSDM is a rapid application development strategy for software development and gives an agile project distribution structure. The essential features of DSDM are that users must be actively connected, and teams have been given the right to make decisions. The techniques used in DSDM are:

1. Time Boxing
2. MoSCoW Rules

3. Prototyping

The DSDM project contains seven stages:

1. Pre-project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and build Iteration
6. Implementation
7. Post-project

Feature Driven Development (FDD):

This method focuses on "Designing and Building" features. In contrast to other smart methods, FDD describes the small steps of the work that should be obtained separately per function.

Lean Software Development:

Lean software development methodology follows the principle "just in time production." The lean method indicates the increasing speed of software development and reducing costs. Lean development can be summarized in seven phase.