# Software Engineering UNIT:-1

**What is Software Engineering?**

**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.

## Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

* **Large software -**It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
* **Scalability-**If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
* **Cost-**As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
* **Dynamic Nature-**The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
* **Quality Management-**Better process of software development provides better and quality software product.

## Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

* Operational
* Transitional
* Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

### Operational

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

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

### Transitional

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

* Portability
* Interoperability
* Reusability
* Adaptability

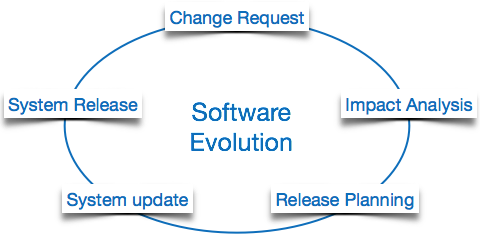
### Maintenance

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

* Modularity
* Maintainability
* Flexibility
* Scalability

**Software Evolution**

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



**Software Evolution Laws**

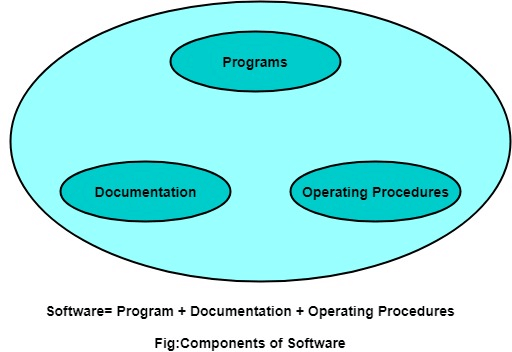
Lehman has given laws for software evolution. He divided the software into three different categories:

* **S-type (static-type) -**This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.
* **P-type (practical-type) -**This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.
* **E-type (embedded-type) -**This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

**Components of software(Program vs. Software)**

Software is more than programs. Any program is a subset of software, and it becomes software only if documentation & operating procedures manuals are prepared.

There are three components of the software as shown in fig:



**1. Program:** Program is a combination of source code & object code.

**2. Documentation:** Documentation consists of different types of manuals. Examples of documentation manuals are: Data Flow Diagram, Flow Charts, ER diagrams, etc.

**3. Operating Procedures:** Operating Procedures consist of instructions to set up and use the software system and instructions on how react to the system failure. Example of operating system procedures manuals is: installation guide, Beginner's guide, reference guide, system administration guide, etc.

**Top 10 Challenges in Software Engineering**

* Requirement volatility.
* Limited budget and resources.
* Lack of communication and collaboration.
* Poor software quality and maintenance.
* Integration and compatibility issues.
* Technical debt management.
* Managing complex codebases.
* Inadequate testing and debugging.

# Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

**Need of SDLC**

Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representative about when and what to do. Otherwise, it would point to chaos and project failure.

**SDLC Cycle**



The stages of SDLC are as follows:

**Stage1: Planning and requirement analysis**

Requirement Analysis is the most important and necessary stage in SDLC. Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

**Stage2: Defining Requirements**

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

**Stage3: Designing the Software**

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

**Stage4: Developing the project**

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code.

**Stage5: Testing**

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

**Stage6: Deployment**

Once the software is certified, and no bugs or errors are stated, then it is deployed. After the software is deployed, then its maintenance begins.

**Stage7: Maintenance**

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

# Software Processes

The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.

A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:

1. **Software specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the requirement must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing client needs.

## The Software Process Model

A software process model is a specified definition of a software process, which is presented from a particular perspective. Models, by their nature, are a simplification, so a software process model is an abstraction of the actual process, which is being described. Process models may contain activities, which are part of the software process, software product, and the roles of people involved in software engineering. Some examples of the types of software process models that may be produced are:

1. **A workflow model:** This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.
2. **2. A dataflow or activity model:** This represents the process as a set of activities, each of which carries out some data transformations. It shows how the input to the process, such as a specification is converted to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers.
3. **3. A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible.

There are several various general models or paradigms of software development:

1. **The waterfall approach:** This takes the above activities and produces them as separate process phases such as requirements specification, software design, implementation, testing, and so on. After each stage is defined, it is "signed off" and development goes onto the following stage.
2. **Evolutionary development:** This method interleaves the activities of specification, development, and validation. An initial system is rapidly developed from a very abstract specification.
3. **Formal transformation:** This method is based on producing a formal mathematical system specification and transforming this specification, using mathematical methods to a program. These transformations are 'correctness preserving.' This means that you can be sure that the developed programs meet its specification.
4. **System assembly from reusable components:** This method assumes the parts of the system already exist. The system development process target on integrating these parts rather than developing them from scratch.

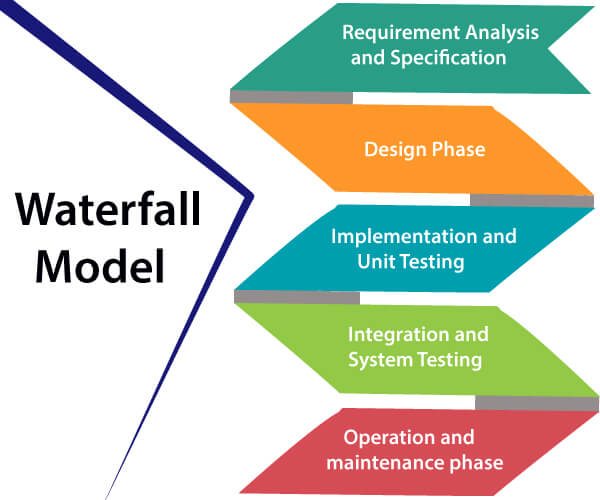
## Software Crisis

1. **Size:** Software is becoming more expensive and more complex with the growing complexity and expectation out of software. For example, the code in the consumer product is doubling every couple of years.
2. **Quality:** Many software products have poor quality, i.e., the software products defects after putting into use due to ineffective testing technique. For example, Software testing typically finds 25 errors per 1000 lines of code.
3. **Cost:** Software development is costly i.e. in terms of time taken to develop and the money involved. For example, Development of the FAA's Advanced Automation System cost over $700 per lines of code.
4. **Delayed Delivery:** Serious schedule overruns are common. Very often the software takes longer than the estimated time to develop, which in turn leads to cost shooting up. For example, one in four large-scale development projects is never completed.

# Waterfall model

Winston Royce introduced the Waterfall Model in 1970.This model has five phases: Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.

**1. Requirements analysis and specification phase:** The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. n this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.



**2. Design Phase:** This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

**3. Implementation and unit testing:** During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially.

1. **Integration and System Testing:** This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results.
2. **Operation and maintenance phase:** Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

**When to use SDLC Waterfall Model?**

Some Circumstances where the use of the Waterfall model is most suited are:

* When the requirements are constant and not changed regularly.
* A project is short
* The situation is calm
* Where the tools and technology used is consistent and is not changing
* When resources are well prepared and are available to use.

**Advantages of Waterfall model**

* This model is simple to implement also the number of resources that are required for it is minimal.
* The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
* The start and end points for each phase is fixed, which makes it easy to cover progress.
* The release date for the complete product, as well as its final cost, can be determined before development.
* It gives easy to control and clarity for the customer due to a strict reporting system.

**Disadvantages of Waterfall model**

* In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
* This model cannot accept the changes in requirements during development.
* It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.

# Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.



## The various phases of incremental model are as follows:

**1. Requirement analysis:** In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

**2. Design & Development:** In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

**3. Testing:** In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

**4. Implementation:** Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

* When the requirements are superior.
* A project has a lengthy development schedule.
* When Software team are not very well skilled or trained.
* When the customer demands a quick release of the product.
* You can develop prioritized requirements first.

**Advantage of Incremental Model**

* Errors are easy to be recognized.
* Easier to test and debug
* More flexible.
* Simple to manage risk because it handled during its iteration.
* The Client gets important functionality early.

**Disadvantage of Incremental Model**

* Need for good planning
* Total Cost is high.
* Well defined module interfaces are needed.

# Prototype Model

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.



## Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

## Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

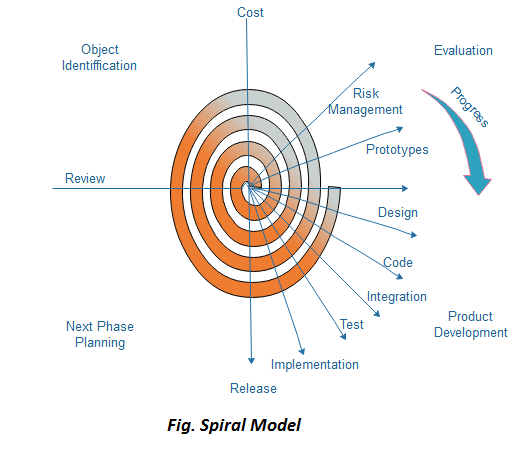
## Disadvantage of Prototype Model

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
   * Costs customer money
   * Needs committed customer
   * Difficult to finish if customer withdraw
   * May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.

# Spiral Model

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

**The Spiral Model is shown in fig:**



**Each cycle in the spiral is divided into four parts:**

**Objective setting:** Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

**Risk Assessment and reduction:** The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

**Development and validation:** The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

**Planning:** Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

**When to use Spiral Model?**

* When deliverance is required to be frequent.
* When the project is large
* When requirements are unclear and complex
* When changes may require at any time
* Large and high budget projects

**Advantages**

* High amount of risk analysis
* Useful for large and mission-critical projects.

**Disadvantages**

* Can be a costly model to use.
* Risk analysis needed highly particular expertise
* Doesn't work well for smaller projects.

# Factors of Management Dependency in Software Development

## Definition of Software Development Management

The process of organizing, coordinating, and directing the creation of software products from conception to completion is referred to as software development management. It entails controlling resources, budgets, timetables, and channels of communication with multiple stakeholders, including clients, project managers, developers, and end users.

## Important Functions of Software Development Management

### 1. Strategic Planning

Setting long-term objectives and a course for the software development team is part of this responsibility. In order to recognize opportunities and risks, one must have a thorough awareness of the business goals, market trends, and technological environment. The strategic planning job includes creating a road map to direct the team's resources and efforts in the direction of the intended results.

### 2. Resource Management

This job includes dealing with individuals, instruments, and frameworks expected to execute the software improvement plan. It requires an exhaustive comprehension of the group's assets and shortcomings and the accessibility of assets like time, spending plan, and innovation. The asset executive's job includes assigning and focusing on assets to guarantee the group can convey great programming within the given limitations.

### 3. Project Management

In order to ensure that projects are finished on time, within budget, and to the appropriate quality, this function involves supervising the daily operations of the software development team. One must have excellent coordination, communication, and problem-solving abilities to manage risks and overcome difficulties. Definition of project requirements, estimation of timetables, creation of project plans, monitoring of progress, and course correction as needed are all part of the project management function.

### 4. Team Management

Leading and encouraging the software development team to accomplish their objectives is a part of this purpose. One needs good leadership, communication, and coaching abilities to create a cohesive team that is in sync with the organization's vision and values. The team management position entails selecting, educating, and growing team members, and it also involves praising and acknowledging, building a cooperative climate, and supporting continuous growth.

### 5. Risk Management

This task includes investigating, assessing, and reducing possible risks influencing the software development project. Analysis and risk prioritization based on likelihood and impact demand a strong eye for detail and a rigorous approach. Risk management requires creating backup plans, keeping an eye on hazards throughout the project's lifespan, and acting proactively to reduce or prevent unfavourable consequences.

### 6. Quality Assurance

As somebody in this position, you would be responsible for ensuring that the software produced by the team meets all requirements of quality. To excel in this role means having expert knowledge of the methods surrounding development and other methodologies tied to quality assurance; using these understandings to devise and implement solid testing plans is essential. A key aspect of carrying out quality assurance work requires both seeking out and eliminating any bugs, safety issues, or problems within the software--just as important is offering thoughtful feedback on how these issues were resolved so that development can continue moving forward. Upkeep matters, too: taking experiences gained from each phase of improvement to maintain a continually evolving testing process will play a huge part in how successful one ultimately is here.

## Benefits of Effective Software Development

### 1. Increased Productivity

Teams' productivity can increase when effective software development management keeps them concentrated and in sync with project objectives. Team members can work more efficiently and spend less time on unnecessary chores by clearly clarifying their roles, responsibilities, and deadlines. It may result in software delivery being completed more quickly and in better control of resources.

### 2. More effective resource allocation

Efficient software development management empowers teams to optimally distribute their time, budget, and workforce by assessing and prioritizing tasks. By cultivating smart resource allocation measures, informed decisions can be made to achieve efficiency and maximum outcomes for a triumphant project outcome.

### 3. Time Management Skills

Effective software development management is paramount to optimizing teams' productivity by ensuring they manage their time efficiently. One way this can be achieved is by setting realistic goals and deadlines, which helps teams channel their resources toward achieving optimal results. By so doing, the team can ensure that software delivery is done promptly while efficiently using existing resources, resulting in higher output per given period. Faster delivery times resulting from proper time management strategies give organizations and businesses a competitive edge that significantly impacts employees.

### 4. Better Communication

In the fast-paced world of software development, communication is crucial. Creating one channel where everyone can be heard and understood will make your team function effectively. An effective management style will encourage open lines of communication among its members to avoid misunderstandings, enhance collaboration, and promptly address any issues or potential problems that may arise during the project. Encouraging a culture of transparency and trust within your team means keeping every member involved in decision-making processes and sharing information so that they can work towards a common goal delivering.

### ****5****. Higher standard of outputs

Teams need effective software development management to maintain high standards for their software deliverables. To accomplish this, the team must develop and keep to best practices. By doing this consistently, the team is able to both minimize errors and guarantee that the product has been adequately tested. Ultimately, this leads to a higher quality product that meets customer expectations, an outcome every software development team strives for! By implementing effective management principles aided by established best practices, teams can optimize their project delivery.

### 6. Satisfaction of customers

Effective software development management ensures the delivery of software that satisfies customer demands and expectations. Teams may provide a product that satisfies consumer expectations by including clients in the development process and soliciting input frequently. Higher client satisfaction may arise from doing business with you again and getting good recommendations.

## Issues in managing software development

### 1. Changing Technologies

Keeping up with technological advances rapidly is one of the biggest issues in developing software management. Changing hardware platforms, development tools, and programming languages can take time and effort. To stay aware of developments and provide cutting-edge software solutions, managers must regularly evaluate and refresh the capabilities of their teams.

### 2. Managing Stakeholders

Another significant problem in managing software development is stakeholder management. Customers, clients, investors, and team members are examples of stakeholders, and each has unique interests and goals. To keep stakeholders informed and involved throughout the project lifecycle, managers must balance between competing demands and expectations and communicate effectively.

### 3. Managing Conflict

Any project will inevitably encounter conflict, and managing conflict is a significant management challenge in the software development industry. Conflicts may develop between teammates, among stakeholders, or even within the team. To reduce the negative effects of conflicts on the project's progress, managers must be adept at spotting and resolving disagreements as well as establishing a climate of cooperation and respect.

### 4. Budgetary Limitations

Budgetary restrictions are a frequent problem in software development management, and the development costs must be carefully weighed against the project's anticipated advantages and results. Making challenging choices about resource allocation, feature prioritization, and project scope may be necessary. Strong financial planning and forecasting abilities and the capacity to adapt and make adjustments during the project lifespan are necessary for effective budget management.

## Best practices for managing software development

### 1. Clear communication

For good software development management, clear communication is a basic best practice. It includes establishing precise objectives, roles, and responsibilities and ensuring that everyone on the team is informed of project updates and changes. Team members should be encouraged to give input and ideas at all stages of a project, and communication must be open and transparent.

### 2. Agile principles

Agile methodology is a well-liked way of developing software that emphasizes adaptability, teamwork, and quick iterations. Agile teams prioritize usable software over thorough documentation and work in brief sprints, frequently lasting two weeks to a month. This method guarantees that the finished product satisfies client needs and enables teams to react swiftly to changing requirements and feedback.

### 3. Integrating and delivering continuously

Automating the testing, integrating, and deploying of code changes is part of the continuous integration along with the delivery (CI/CD) method of software development. Doing this decreases the likelihood of faults or bugs in the finished product, and software always arrives in a release-ready state. Teams that use agile development and regularly release software must use CI/CD.

### 4. Routine team meetings

Frequent team conferences ensure everyone is on the same page and committed to the project's objectives. Regular meetings with reports on progress, team member comments, and conversations about any problems or difficulties should be organized. Meetings may also recognize achievements and milestones, helping create a cohesive team environment.

### 5. Progress observation and tracking

For effective software development management, monitoring and tracking of progress is crucial. Key performance indicators (KPIs) must be established and routinely monitored throughout the project's lifetime to achieve this. Managers can use KPIs to pinpoint areas for development, check that the project is on schedule to complete its objectives, and make data-driven choices on the distribution of resources and priorities.

### 6. Suggestions for Improvement

Successful software development requires frequent feedback and improvement. Regular input from teammates and stakeholders should be encouraged by managers, who should then use the feedback to pinpoint problem areas and enhance project outcomes. It calls for an environment open to communication and teamwork and a readiness to modify and adapt as necessary during a project.