Chapter 1

SOFTWARE ENGINEERING AND PROJECT MANAGEMENT

1.1. Introduction To Software Engineering:

- The term software engineering is the product of two words, **software**, and **engineering**.
- The **software** is a collection of integrated programs.
- **Engineering** is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc.
- **Software Engineering** is the process of designing, developing, testing, and maintaining software.
- It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software.
- Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.

Why is Software Engineering required?

Software Engineering is required due to the following reasons:

- To manage large software
- For more Scalability
- Cost Management
- To manage the dynamic nature of software
- 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.

- **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.
- **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.
- **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.
- **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.
- **Quality Management:** Better procedure of software development provides a better and quality software product.

Importance of Software engineering:

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start

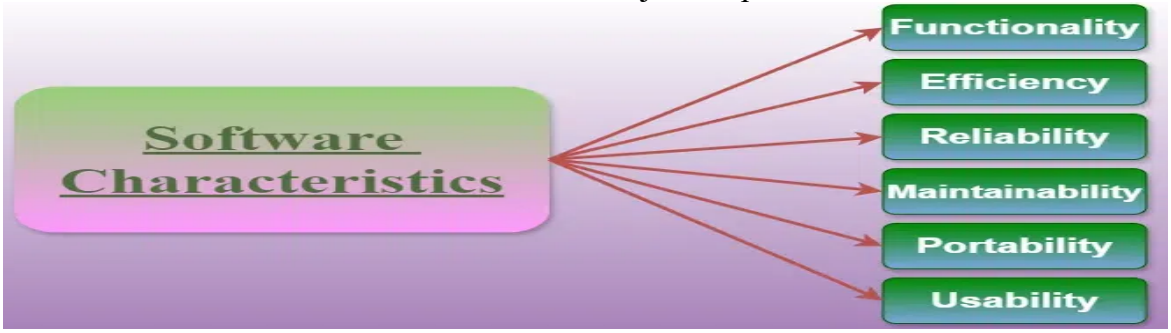
solving each small issue one by one. All these small problems are solved independently to each other.

2. **To minimize software cost:** Software needs a lot of hard work and software engineers are highly paid experts. A lot of manpower is required to develop software with many codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.
3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

1.2. Nature and Characteristics of Software:

- **Software** is defined as a collection of computer programs, procedures, rules, and data.

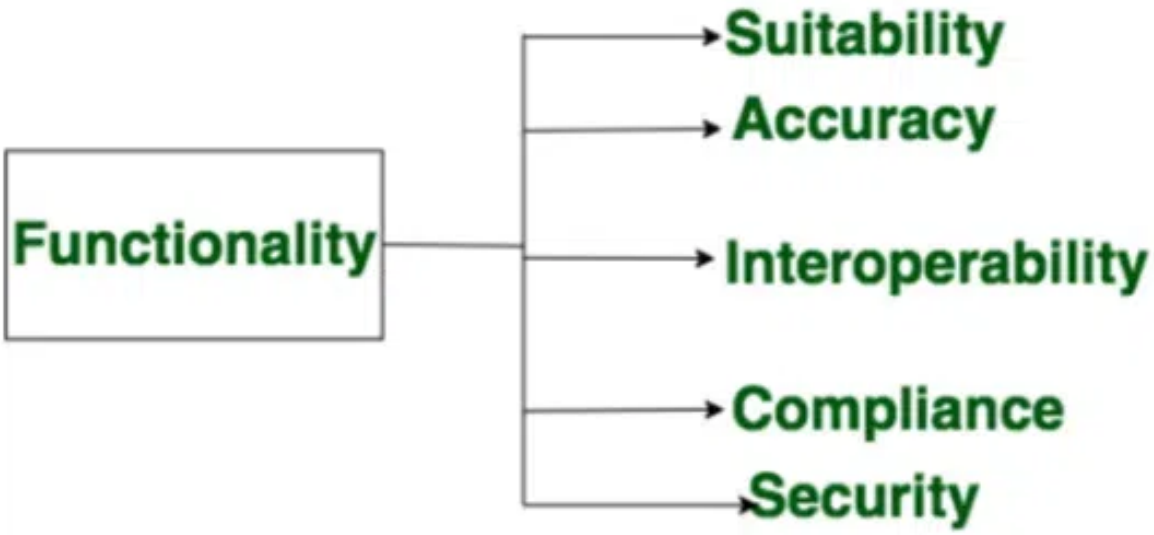
Software Characteristics are classified into six major components.



Functionality:

- It refers to the degree of performance of the software against its intended purpose.
- Functionality refers to the set of features and capabilities that a software program or system provides to its users.
- It is one of the most important characteristics of software, as it determines the usefulness of the software for the intended purpose.

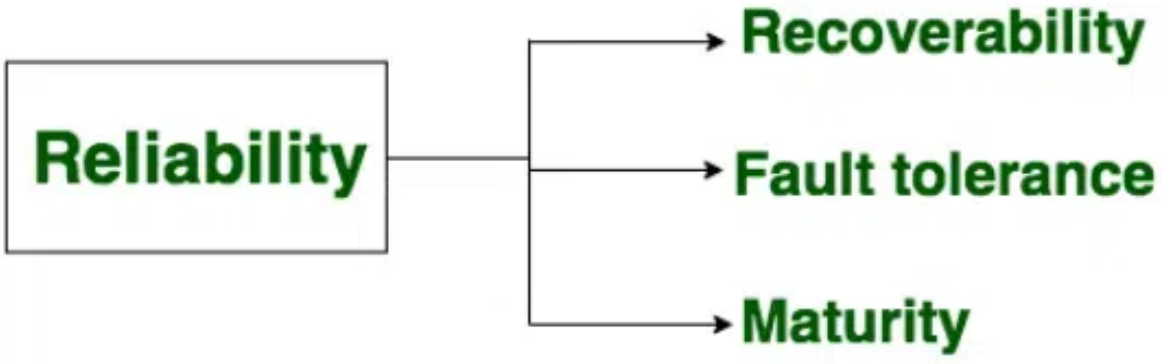
  Required functions are:



Reliability:

- A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time.
- Reliability is a characteristic of software that refers to its ability to perform its intended functions correctly and consistently over time.

Required functions are:

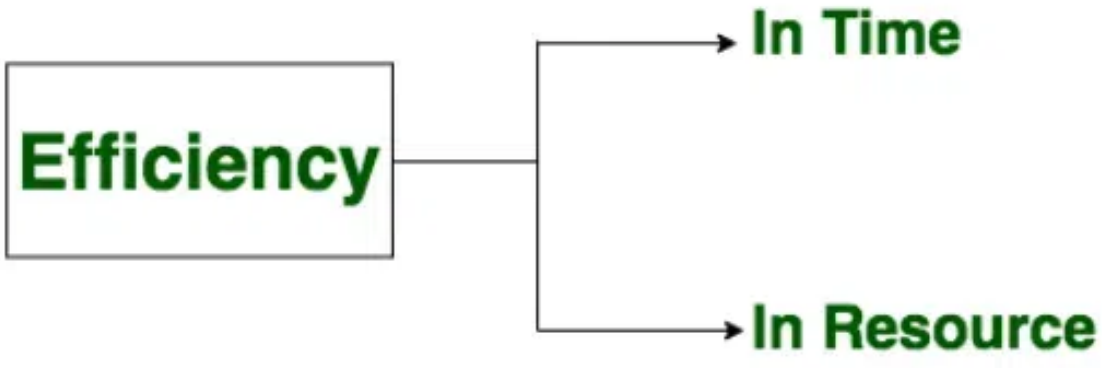


.

Efficiency:

- It refers to the ability of the software to use system resources in the most effective and efficient manner.
- Efficiency is a characteristic of software that refers to its ability to use resources such as memory, processing power, and network bandwidth in an optimal way.

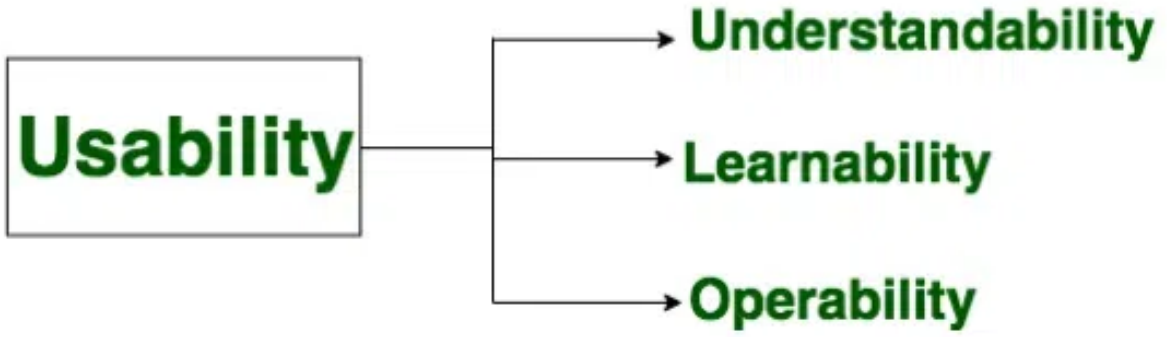
Required functions are:



Usability:

- It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software.

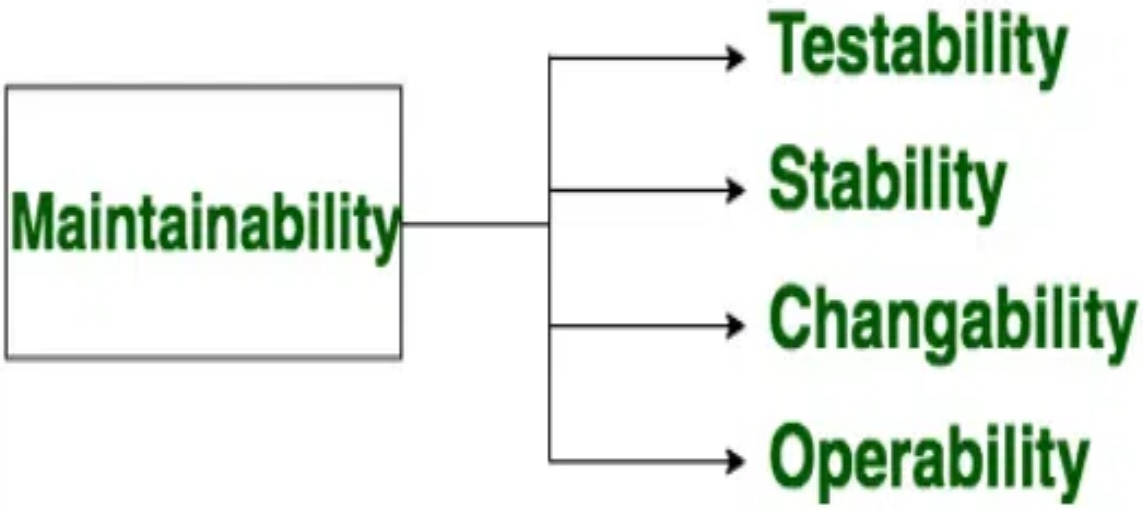
Required functions are:



Maintainability:

- It refers to the ease with which modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

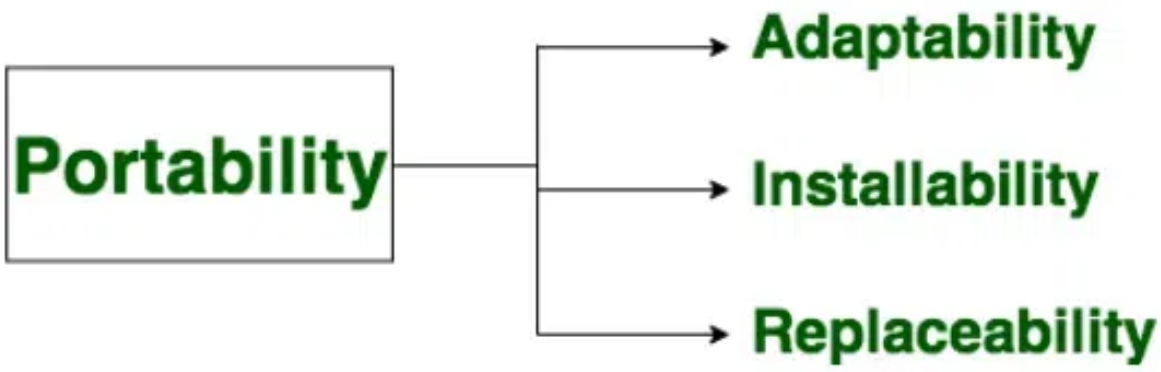
Required functions are:



Portability:

- A set of attributes that bears on the ability of software to be transferred from one environment to another, without minimum changes.

Required functions are:



Characteristics of the Software:

- It is intangible, meaning it cannot be seen or touched.
- It is non-perishable, meaning it does not degrade over time.
- It is easy to replicate, meaning it can be copied and distributed easily.
- It can be complex, meaning it can have many interrelated parts and features.
- It can be difficult to understand and modify, especially for large and complex systems.
- It can be affected by changing requirements, meaning it may need to be updated or modified as the needs of users change.
- It can be impacted by bugs and other issues, meaning it may need to be tested and debugged to ensure it works as intended.

1.3. Software Engineering versus System Engineering:

Aspect	Software Engineering	System Engineering
Focus	Primarily focuses on software development and processes.	Focuses on designing and managing complex systems.

Scope	Narrower scope	Broader scope
Objectives	Develops software applications or systems to meet specific requirements.	Integrates multiple components (hardware, software, processes) to achieve system goals.
Emphasis	Emphasizes software design, coding, testing, and maintenance.	Emphasizes system architecture, integration, and optimization.
Lifecycle	Follows SDLC methodologies.	Involves system lifecycle management from conception to retirement.
Disciplines	Incorporates disciplines like requirements engineering, software design, and quality assurance.	Integrates disciplines such as system architecture, system integration, and system testing.
Skills	Requires strong programming skills, knowledge of algorithms, and software design patterns.	Requires systems thinking, problem-solving skills, and understanding of interdisciplinary concepts.
Examples	Examples: application development, web development, and mobile app development.	Example: aerospace systems, transportation systems, and healthcare systems.

1.4. Software Crisis and Myths:

- The term "software crisis" refers to the numerous challenges and difficulties faced by the software industry during the 1960s and 1970s.
- It became clear that old methods of developing software couldn't keep up with the growing complexity and demands of new projects.
- This led to high costs, delays, and poor-quality software.
- **Software Crisis** is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time.

1.4.1. Reasons for Software Crisis in Software Engineering:

- **Complexity Overload:**

Software systems' increasing scale and functionality led to exponential complexity, making management and maintenance challenging.

- **Lack of Formalization:**

Early software engineering lacked standardized processes, resulting in unstable and error-prone software development.

- **Changing Requirements:**

Evolving requirements due to user needs or market shifts led to scope creep and project delays.

- **Insufficient Quality Assurance:**

Inadequate testing caused software releases with bugs, vulnerabilities, and reliability issues.

- **Limited Reusability:**

Ineffective code and component reuse across projects led to redundancy and wasted resources.

- **Poor Project Management:**

Inefficient practices like communication gaps and unrealistic timelines caused missed deadlines and budget overruns.

- **Rapid Technological Advancements:**

Fast-paced technology made maintaining and updating legacy systems challenging, requiring constant adaptation.

1.4.2. Factors Contributing to Software Crisis:

- **Increasing Complexity:**

The complexity of software systems grew rapidly, making it difficult to design, develop, and maintain them effectively. The lack of tools and methodologies to manage this complexity led to software that was error-prone and hard to understand.

- **Lack of Formal Methods:**

In the absence of formalized methods and practices, software development was often ad hoc and lacked systematic approaches. This resulted in unreliable software that was challenging to modify or extend.

- **Changing Requirements:**

Many software projects faced volatile and changing requirements. The inability to handle these changes efficiently led to project delays, cost overruns, and software not aligning with user needs.

- **Limited Reusability:**

Developers struggled to reuse code and components across projects, leading to duplicated efforts and inefficiencies. This hindered productivity and hindered the evolution of software engineering practices.

- **Poor Quality Assurance:**

Inadequate testing and quality assurance processes resulted in software with numerous bugs, security vulnerabilities, and reliability issues. This undermined user confidence and overall system stability.

- **Lack of Collaboration:**

Limited collaboration among development teams, stakeholders, and end-users led to misunderstandings and misalignments. Communication gaps caused delays and hindered the creation of high-quality software.

- **Rapid Technological Changes:**

The fast-paced evolution of technology rendered some software obsolete quickly. This made maintaining and updating software challenging and forced developers to learn new tools and platforms constantly.

- **Inadequate Project Management:**

Weak project management practices, including poor planning, resource allocation, and progress tracking, contributed to missed deadlines, budget overruns, and unsuccessful projects.

- **Absence of Standards:**

The lack of industry-wide standards and best practices resulted in inconsistent approaches to software development. This hindered interoperability, hindered collaboration, and contributed to the crisis.

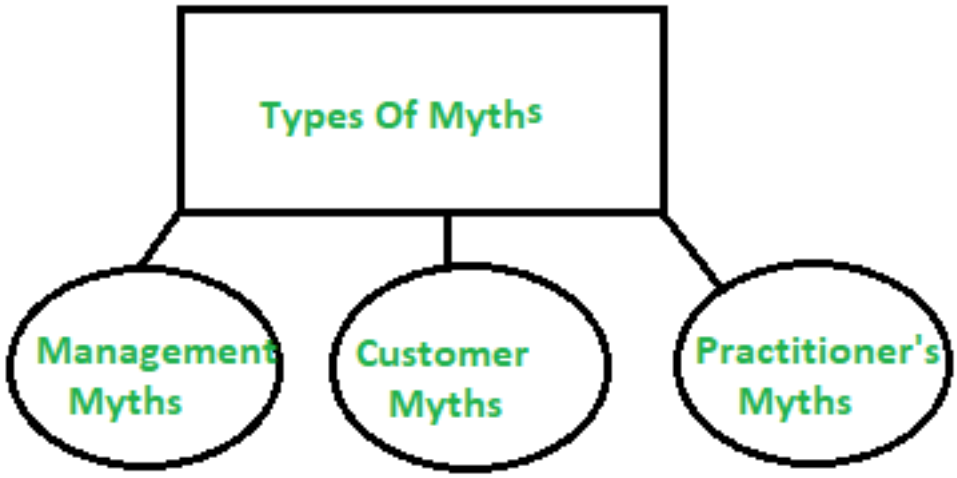
1.4.3. Solutions to the Software Crisis:

- **Structured Methodologies:**
  - The adoption of structured methodologies, like Waterfall and later Agile, brought systematic approaches to software development.
  - These methodologies emphasized planning, iterative development, and continuous user feedback.
- **Formal Software Development:**
  - The introduction of formal methods, including mathematical modeling and specification languages, enhanced software reliability by enabling rigorous verification and validation of software designs.
- **Requirements Engineering:**
  - Emphasis on thorough requirements gathering and analysis helped manage changing requirements effectively, reducing scope creep and ensuring software alignment with user needs.
- **Quality Assurance and Testing:**
  - Robust testing and quality assurance practices, such as unit testing, integration testing, and automated testing, improved software reliability and minimized the occurrence of bugs and vulnerabilities.
- **Code Reusability:**
  - The development of modular programming and component-based development allowed for greater code reusability across projects, boosting productivity and reducing redundancy.
- **Improved Project Management:**
  - The adoption of project management methodologies, such as Scrum and Kanban, facilitated better resource allocation, communication, and tracking, leading to more successful project outcomes.
- **Standardization and Best Practices:**
  - The establishment of industry-wide standards, coding conventions, and best practices enhanced collaboration, interoperability, and the overall quality of software products.

1.5. Software Myths:

- Software Myths are beliefs that do not have any pure evidence. Software myths may lead to many misunderstandings, unrealistic expectations, and poor decision-making in software development projects.

The types of software-related myths are listed below:



(i) Management Myths:

Myth 1:

We have all the standards and procedures available for software development.

Fact:

- Software experts do not know all the requirements for the software development.
- And all existing processes are incomplete as new software development is based on new and different problem.

Myth 2:

The addition of the latest hardware programs will improve the software development.

Fact:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer, they are more important than hardware to produce quality and productivity.
- Hence, the hardware resources are misused.

Myth 3:

- With the addition of more people and program planners to Software development can help meet project deadlines (If lagging).

Fact:

- If software is late, adding more people will merely make the problem worse. This is because the people already working on the project now need to spend time educating the newcomers and are thus taken away from their work. The newcomers are also far less productive than the existing software engineers, and so the work put into training them to work on the software does not immediately meet with an appropriate reduction in work.

(ii)Customer Myths:

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

Myth 1:

A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

Fact:

- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

Myth 2:

Software requirements continually change, but change can be easily accommodated because software is flexible

Fact:

- It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.

(iii)Practitioner's Myths:

Myths 1:

They believe that their work has been completed with the writing of the plan.

Fact:

- It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

Myths 2:

There is no other way to achieve system quality, until it is "running".

Fact:

- Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

Myth 3:

An operating system is the only product that can be successfully exported project.

Fact:

- A working system is not enough, the right document brochures and booklets are also required to provide guidance & software support.

Myth 4:

Engineering software will enable us to build powerful and unnecessary document & always delay us.

Fact:

- Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

1.5.1. Disadvantages of Software Myths:

- **Unrealistic Expectations:** Software myths can create disappointment and frustration among the stakeholders and developers. Sometimes, the fake myth may lead to the no use of the software when it is completely safe.
- **Project Delays:** Software myths will lead to more delays in the completion of the projects and increase the completion time of the projects.
- **Poor Quality Software:** Myths such as "we can fix it later" or "we don't need extensive testing" can lead to poor software quality. Neglecting testing and quality assurance can result in buggy and unreliable software.
- **Scope Creep:** Myths like "fixed requirements" can lead to scope creep as stakeholders may change their requirements or expectations throughout the project. This can result in a never-ending development cycle.
- **Ineffective Communication:** Believing in myths can affect good communication within development teams and between teams and clients. Clear and open communication is crucial for project success, and myths can lead to misunderstandings and misalignment.
- **Wasted Resources:** The Idea of getting "the perfect software" can result in the allocation of unnecessary resources, both in terms of time and money, which could be better spent elsewhere.
- **Customer Dissatisfaction:** Unrealistic promises made based on myths can lead to customer dissatisfaction. When software doesn't meet exaggerated expectations, clients may be disappointed and dissatisfied.
- **Reduced Productivity:** Myths can lead to reduced productivity, as team members may spend time on unnecessary tasks or follow counterproductive processes based on these myths.
- **Increased Risk of Project Failure:** The reliance on myths can significantly increase the risk of project failure. Failure to address these myths can lead to project cancellations, loss of investments, and negative impacts on an organization's reputation.
- **Decreased Competitiveness:** Belief in myths can make an organization less competitive in the market. It can hinder an organization's ability to innovate and adapt.

1.5.2. Steps to Avoid Software Myths:

- **Stay Informed:** Keep up to date with the latest trends, practices, and developments in the field of software engineering. Attend conferences, read industry publications, and participate in online communities to stay informed.
- **Continuous Learning:** Invest in ongoing education and professional development. Software engineering is an evolving field, and staying current is essential.
- **Data-Driven Decision-Making:** Make decisions based on data, evidence, and real-world experiences rather than relying on anecdotal evidence or common misconceptions.
- **Testing and Validation:** Don't rely solely on assumptions. Test your software, gather data, and validate your ideas to confirm their accuracy.
- **Educate Team Members:** Ensure that everyone on your development team is aware of common software myths and is committed to avoiding them. Knowledge sharing and education can help dispel misconceptions.
- **Risk Assessment:** When dealing with software development decisions, conduct risk assessments to identify potential pitfalls and myths that might affect the project.

1.6. Four Ps of Software Project Management:

- For properly building a product, there's a very important concept that we all should know in software project planning while developing a product.
- There are 4 critical components in software project planning which are known as the **4P's** namely:

- Product
- Process
- People
- Project

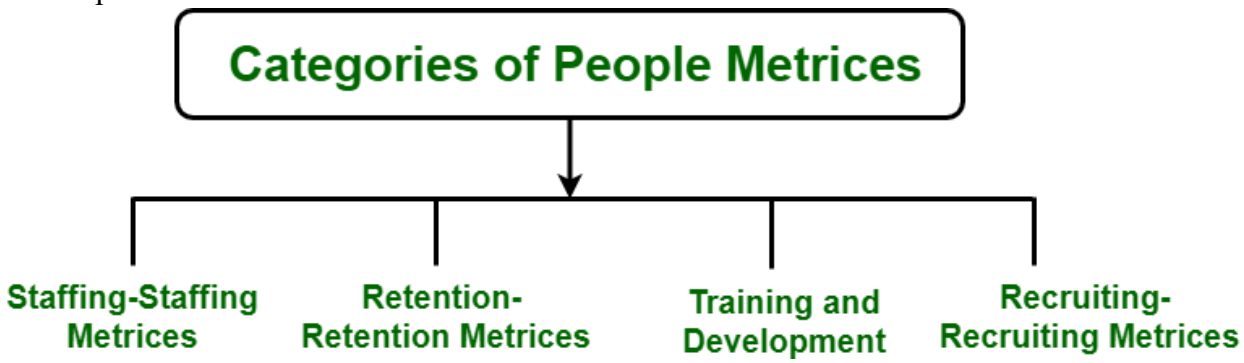
- **People**
  The most important component of a product and its successful implementation is human resources. In building a proper product, a well-managed team with clear-cut roles defined for each person/team will lead to the success of the product. We need to have a good team to save our time, cost, and effort. Some assigned roles in software project planning are **project manager, team leaders, stakeholders, analysts,** and other **IT professionals**. Managing people successfully is a tricky process which a good project manager can do.
- **Product**
  As the name inferred, this is the deliverable or the result of the project. The project manager should clearly define the product scope to ensure a successful result, control the team members, as well technical hurdles that he or she may encounter during the building of a product. The product can consist of both tangible or intangible such as shifting the company to a new place or getting a new software in a company.
- **Process**
  In every planning, a clearly defined process is the key to the success of any product. It regulates how the team will go about its development in the respective time period. The Process has several steps involved like, documentation phase, implementation phase, deployment phase, and interaction phase.
- **Project**
  The last and final P in software project planning is Project. It can also be considered as a blueprint of process. In this phase, the project manager plays a critical role. They are responsible to guide the team members to achieve the project's target and objectives, helping & assisting them with issues, checking on cost and budget, and making sure that the project stays on track with the given deadlines.

1.7. Process and Project Metrics:

- **People Metrics and Process Metrics,** both play important roles in software development. People Metrics helps in quantifying the useful attributes whereas Process Metrics creates the body of the software.
- People metrics focus on how well team members work together and their overall satisfaction, while process metrics measure how smoothly tasks are completed.

1. **People Metrics**
   - It play an important role in software project management. These are also called personnel metrics.

- Some authors view resource metrics to include personnel metrics, software metrics, and hardware metrics but most of the authors mainly view resource metrics as consisting of personnel metrics only.
- In the present context, we also assume resource metrics to include mainly personnel metrics.



People Metrics Categories

Top 7 People Metrices to Track

Following are the People Metrices:

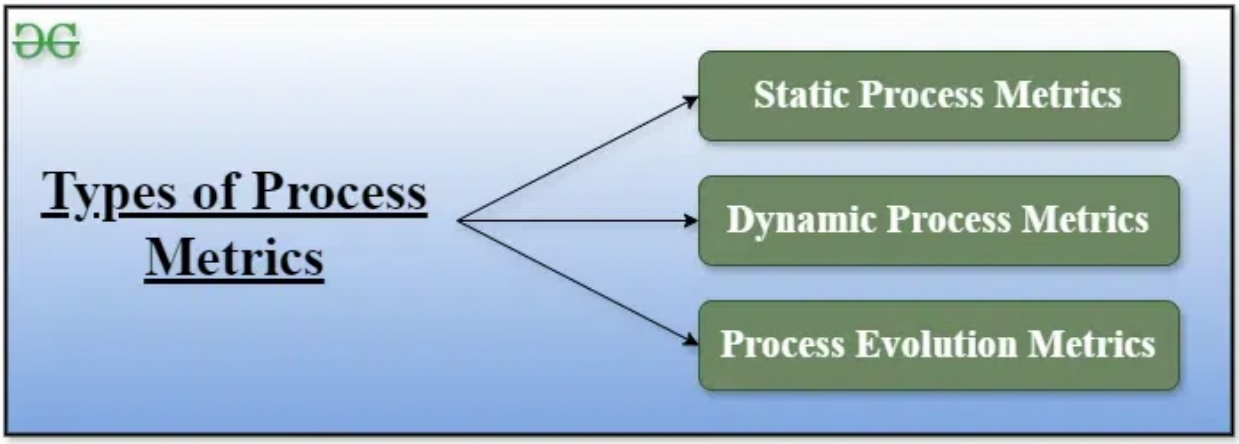
1. **Productivity**: Productivity metrics are simple ways to measure how much work is done in a certain period. They help see how efficient and effective someone or something is at completing tasks.
2. **Employee Net Promoter Score (eNPS) :** Employee Net Promoter Score measures how likely employees are to recommend their workplace to others. It should be track because eNPS evaluate overall employee satisfaction and suggest areas for improvement. It can be calculated by conducting regular surveys asking employees how likely they are to recommend the company to friends or family.
3. **Team Collaboration:** Team Collaboration measures how well team members work together and communicate. It's essential to track because effective teamwork streamlines workflows and enhances project outcomes. To track, monitor communication frequency, participation in team activities, and gather feedback from team members regularly.
4. **Attrition:** Attrition tracks the rate at which employees leave the organization. It's important to track because it helps identify trends and reasons for turnover, allowing proactive measures to retain talent. To track, calculate the percentage of employees leaving within a given period and analyze reasons through exit interviews or surveys.
5. **Absenteeism:** Absenteeism measures the frequency at which employees are absent from work. It's crucial to track because it highlights patterns of absence, enabling the identification and resolution of underlying issues. To track, maintain records of employee attendance, including reasons for absence, and analyze trends over time to minimize disruptions to productivity.
6. **Total cost of workforce:** The Total Cost of Workforce calculates all expenses associated with employing staff. It's important to track because it helps manage budget allocation and optimize resource utilization. To track, compile data on salaries, benefits, training costs, and other expenses related to workforce management to understand the total cost of employing staff.

7. **Quality of Work:** Quality of Work evaluates the standard and effectiveness of tasks completed by employees. It's vital to track because it ensures deliverables meet quality standards, satisfy customer requirements, and uphold organizational reputation.

2. Process Metrics

- Process Metrics are the measures of the development process that create a body of software. A common example of a **process metric** is the length of time that the process of software creation tasks.
- Based on the assumption that the quality of the product is a direct function of the process, process metrics can be used to estimate, monitor, and improve the reliability and quality of software. ISO- 9000 certification, or "**Quality Management Standards**", is the generic reference for a family of standards developed by the **International Standard Organization (ISO).**
- Often, Process Metrics are tools of management in their attempt to gain insight into the creation of a product that is intangible. Since the software is abstract, there is no visible, traceable artifact from software projects. Objectively tracking progress becomes extremely difficult. Management is interested in measuring progress and productivity and being able to make predictions concerning both.

Types of Process Metrics:



- **Static Process Metrics:** Static Process Metrics are directly related to the defined process. For example, the number of types of roles, types of artifacts, etc.
- **Dynamic Process Metrics:** Dynamic Process Metrics are simply related to the properties of process performance. For example, how many activities are performed, how many artifacts are created, etc.)
- **Process Evolution Metrics:** Process Evolution Metrics are related to the process of making changes over a period of time. For example, how many iterations are there within the process)

Top 7 Process Metrices:

- **Lead Time**: Lead Time measures the time taken from initiating a process (such as starting work on a task) to its completion (finishing the task). It indicates how quickly work moves through the development process.

- **Cycle Time**: Cycle Time tracks the duration it takes to complete one full cycle of a process, from beginning to end. It provides insights into the efficiency and effectiveness of the development workflow.
- **Throughput**: Throughput basically Quantifies the rate at which tasks or features are completed within a given timeframe. It reflects the productivity and capacity of the development team.
- **Work in Progress (WIP)**: It indicates the number of tasks or features currently being worked on but not yet completed. It helps in identifying bottlenecks and managing workflow to ensure tasks are completed efficiently.
- **Defect Density**: Defect Density measures the number of defects or bugs found per unit of work or code. It helps in assessing the quality and reliability of the software being developed.
- **Process Efficiency**: Process Efficiency evaluates the ratio of value-added work (tasks that directly contribute to delivering value to the customer) to non-value-added work (tasks that do not directly contribute to value delivery). It identifies opportunities for streamlining processes and reducing waste.
- **Process Compliance**: Process Compliance assesses the extent to which development processes adhere to defined standards, guidelines, or regulations. It ensures consistency and quality in the software development process.

3. Project Metrics:

- **project metrics** help track and measure various aspects of a project's progress, performance, and quality. They provide actionable insights that guide project management, allowing teams to make data-driven decisions, optimize resources, and identify risks.

Benefits of Project Metrics:

1. **Enhanced Project Control**: Project metrics provide objective data on project status, enabling better control over timelines, resources, and quality.
2. **Improved Planning and Scheduling**: Metrics like effort and resource utilization help refine future project estimates, improving accuracy in planning.
3. **Risk Management**: Metrics allow early detection of potential issues, enabling teams to address risks before they impact project delivery.
4. **Resource Optimization**: By tracking resource allocation and performance, project metrics help ensure efficient use of team members, time, and budget.
5. **Informed Decision-Making**: Project metrics support informed, data-backed decisions, from setting realistic goals to making changes during the project lifecycle.
6. **Stakeholder Transparency**: Regularly updated metrics help communicate progress and any issues, ensuring stakeholders remain informed throughout the project.

Types of Project Metrics

1. Effort Metrics

- These metrics measure the time and effort invested in project activities. Examples include *Hours Worked*, *Effort Variance*, and *Resource Utilization*.

- **Benefits**: Effort metrics provide insight into team productivity and can help identify if work is progressing as planned or if adjustments are needed.

2. Cost Metrics

- Focus on tracking the financial aspects of a project, such as *Budget Utilization*, *Cost Variance*, and *Return on Investment (ROI)*.
- **Benefits**: Cost metrics help control spending and ensure that the project remains within budget, which is essential for maintaining profitability and meeting financial goals.

3. Schedule Metrics

- These metrics track project timelines, including *Schedule Variance*, *Cycle Time*, *Lead Time*, and *Milestone Completion Rate*.
- **Benefits**: Schedule metrics are crucial for monitoring progress toward deadlines and detecting any delays early, helping teams stay on track and communicate accurate timelines to stakeholders.

4. Quality Metrics

- Focus on measuring the quality of the product being developed, such as *Defect Density*, *Defect Removal Efficiency*, and *Test Coverage*.
- **Benefits**: Quality metrics help identify issues and areas for improvement, ensuring the final product meets customer and stakeholder expectations.

5. Scope Metrics

- Measure the extent of work completed versus the initial project scope, using metrics like *Requirements Volatility* (changes in requirements) and *Scope Creep* (uncontrolled scope expansion).
- **Benefits**: Scope metrics help control project scope, ensuring that teams stay aligned with the initial requirements and avoid unnecessary feature additions.

6. Risk Metrics

- These metrics track and quantify risks, including *Risk Occurrence Rate* and *Impact Severity*. Some teams use tools like risk assessment matrices to evaluate these.
- **Benefits**: Risk metrics support proactive risk management, helping teams identify, assess, and mitigate risks before they impact project timelines, budget, or quality.

7. Customer Satisfaction Metrics

- Focus on measuring customer satisfaction and the perceived value of the project. Examples include *Customer Satisfaction Score (CSAT)*, *Net Promoter Score (NPS)*, and *Number of Customer-Reported Defects*.
- **Benefits**: These metrics ensure the project aligns with customer expectations and requirements, leading to higher satisfaction and better product reception.

8. Resource Allocation Metrics

- Track how effectively team members and resources are being used in the project. Examples include *Resource Utilization Rate* and *Availability*.
- **Benefits**: These metrics help balance workloads, prevent resource bottlenecks, and ensure that resources are deployed where they are most needed.

1.6 Measurement of Software: Metrics, Measure, Indicator:

- A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process.
- Software measurement is a titrate impute of a characteristic of a software product or the software process.
- It is an authority within software engineering.
- The software measurement process is defined and governed by ISO Standard.

Software Measurement Principles

The software measurement process can be characterized by five activities-

1. **Formulation:** The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
2. **Collection:** The mechanism used to accumulate data required to derive the formulated metrics.
3. **Analysis:** The computation of metrics and the application of mathematical tools.
4. **Interpretation:** The evaluation of metrics results in insight into the quality of the representation.
5. **Feedback:** Recommendation derived from the interpretation of product metrics transmitted to the software team.

Need for Software Measurement

Software is measured to:

- Create the quality of the current product or process.
- Anticipate future qualities of the product or process.
- Enhance the quality of a product or process.
- Regulate the state of the project concerning budget and schedule.
- Enable data-driven decision-making in project planning and control.
- Identify bottlenecks and areas for improvement to drive process improvement activities.
- Ensure that industry standards and regulations are followed.
- Give software products and processes a quantitative basis for evaluation.
- Enable the ongoing improvement of software development practices.

Classification of Software Measurement

There are 2 types of software measurement:

1. **Direct Measurement:** In direct measurement, the product, process, or thing is measured directly using a standard scale.
2. **Indirect Measurement:** In indirect measurement, the quantity or quality to be measured is measured using related parameters i.e. by use of reference.

Software Metrics

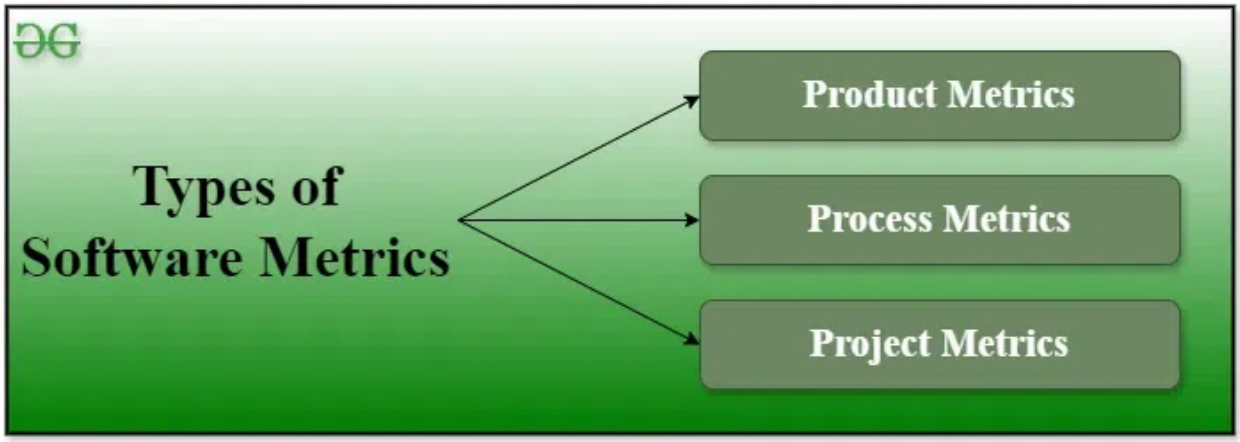
A metric is a measurement of the level at which any impute belongs to a system product or process.
Software metrics are a quantifiable or countable assessment of the attributes of a software product.
There are 4 functions related to software metrics:

1. **Planning**
2. **Organizing**
3. **Controlling**
4. **Improving**

Characteristics of software Metrics

1. **Quantitative:** Metrics must possess a quantitative nature. It means metrics can be expressed in numerical values.
2. **Understandable:** Metric computation should be easily understood, and the method of computing metrics should be clearly defined.
3. **Applicability:** Metrics should be applicable in the initial phases of the development of the software.
4. **Repeatable:** When measured repeatedly, the metric values should be the same and consistent.
5. **Economical:** The computation of metrics should be economical.
6. **Language Independent:** Metrics should not depend on any programming language.

Types of Software Metrics:



1. **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated. Examples include lines of code, cyclomatic complexity, code coverage, defect density, and code maintainability index.
2. **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization. These metrics are used to optimize the development process and maintenance activities of software. Examples include effort variance, schedule variance, defect injection rate, and lead time.
3. **Project Metrics:** The project metrics describes the characteristic and execution of a project. Examples include effort estimation accuracy, schedule deviation, cost variance, and productivity. Usually measures-
    - Number of software developer
    - Staffing patterns over the life cycle of software
    - Cost and schedule
    - Productivity

Advantages of Software Metrics

1. Reduction in cost or budget.
2. It helps to identify the particular area for improvising.
3. It helps to increase the product quality.
4. Managing the workloads and teams.
5. Reduction in overall time to produce the product,.

Disadvantages of Software Metrics

1. It is expensive and difficult to implement the metrics in some cases.
2. Performance of the entire team or an individual from the team can't be determined. Only the performance of the product is determined.
3. Sometimes the quality of the product is not met with the expectation.
4. It leads to measure the unwanted data which is wastage of time.
5. Measuring the incorrect data leads to make wrong decision making.

1.7 Project Estimation, Empirical Estimation Models:

- **project estimation models** are used to predict various aspects of a software project, including cost, time, resources, and effort required to complete the project.
- Effective estimation models are essential for setting realistic goals, planning resources, and managing project risks.
- Here are some commonly used projects estimation models, along with their key characteristics and applications

Benefits of Project Estimation:

1. Resource Allocation:

- Accurately estimating project size helps determine the necessary budget for personnel, hardware, software, and other resources.

2. Project Scheduling:

- By understanding the scope and complexity of the project, realistic timelines can be created for various stages of development, testing, and deployment.

3. Risk Management:

- Project size estimation helps identify potential risks and challenges that may arise during development.

4. Quality Assurance:

- The estimated project size helps determine the necessary testing effort to ensure the software meets quality standards.

5. Client Expectations:

- Accurate project size estimation helps set realistic expectations with clients regarding delivery timelines and project scope.

6. Project Management:

- Project size estimation provides a baseline for tracking progress and identifying potential deviations from the plan.

7. Evaluation and Improvement:

- After project completion, comparing the actual effort and resources used to the estimated size helps identify areas for improvement in future projects.

Types of Project Estimation:

Model	Best For	Description
Expert Judgment	Small projects or when expert insights are available	Estimates based on expert opinions and experience
Analogous Estimation	Early-stage projects with similar historical data	Uses data from similar past projects
Parametric Estimation	Quantifiable projects with historical data	Relies on statistical inputs for cost and time estimates
COCOMO	Medium to large projects with well-defined inputs	Structured cost model with basic, intermediate, detailed levels
Function Point Analysis	Business apps with user-centered requirements	Estimates based on user-oriented functionality
Wideband Delphi	Complex projects with multiple experts involved	Collaborative, consensus-based approach
Three-Point (PERT)	Projects with high uncertainty	Uses optimistic, pessimistic, and most likely estimates
Top-Down Estimation	Early-stage high-level estimation	Starts with overall estimate and breaks it down
Bottom-Up Estimation	Detailed projects with defined tasks	Builds up estimates from individual tasks
ML-Based Models	Data-rich organizations with data science expertise	Predicts estimates using machine learning

- **Empirical Estimation Models:**

- Empirical estimation technique are based on the data taken from the previous project and some based on guesses and assumptions.
- Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step.
- It uses the size of the software to estimate the effort.
- In this technique, an educated guess of project parameters is made.
- Hence, these models are based on common sense

Two most common type of Empirical Project Estimation Techniques are:

1. **Expert Judgment Technique**
2. **Delphi Cost Estimation Technique**

1. <u>Expert Judgement Technique:</u>

- Expert judgment is a technique that relies on the knowledge and experience of domain experts to estimate project costs, schedules, and risks.
- It's a valuable tool when historical data is limited or unavailable.

How it works:

1. **Identify Experts:** Select experts with relevant experience and knowledge in the specific domain.
2. **Gather Estimates**: Collect individual estimates from the experts, either through interviews, surveys, or workshops.
3. **Consolidate Estimates**: Analyze the estimates and identify any significant variations or outliers.
4. **Refine Estimates:** Discuss the estimates with the experts to refine them and reach a consensus.
5. **Adjust for Risk:** Consider potential risks and uncertainties and adjust the estimates accordingly.

Advantages of Expert Judgment:

- Leverages Expertise: It taps into the knowledge and experience of domain experts.
- Quick and Efficient: It can be a relatively quick and efficient technique.
- Flexibility: It can be applied to a wide range of estimation problems.

Disadvantages of Expert Judgment:

- Subjectivity: Estimates can be influenced by individual biases and opinions.
- Lack of Objectivity: It relies on human judgment, which can be prone to errors.
- Limited Accuracy: The accuracy of estimates may vary depending on the expertise and experience of the experts.

When to Use Expert Judgment:

- When there is limited historical data or statistical information.
- When the project is novel or innovative.
- When quick and rough estimates are needed.

2. Delphi Cost Estimation Technique

- The Delphi Method is a structured technique used to elicit expert opinions on a particular topic.
- In the context of software engineering, it's used to estimate project costs, schedules, and risks.

How it works:

1. **Selection of Experts:** A group of experts with relevant knowledge and experience is selected.
2. **Initial Questionnaire:** A questionnaire is sent to the experts, asking them to provide their individual estimates.
3. **Feedback and Iteration:** The responses are collected, analysed, and shared with the group. Experts are encouraged to review their estimates based on the feedback from others.
4. **Consensus Building:** The process of iteration and feedback continues until a consensus is reached or a stable estimate is obtained.

Advantages of Delphi Method:

- **Reduced Bias:** By keeping responses anonymous, it reduces the influence of dominant personalities and groupthink.
- **Structured Approach:** The structured process ensures that all relevant factors are considered.
- **Consensus Building:** It facilitates the convergence of opinions and helps to reach a shared understanding.
- **Flexibility:** It can be adapted to various estimation problems, such as cost, schedule, and risk.

Disadvantages of Delphi Method:

- **Time-Consuming:** The iterative process can be time-consuming, especially if there are many experts involved.
- **Expert Availability:** It requires the availability and cooperation of experts.
- **Subjectivity:** While it reduces bias, it still relies on expert judgment, which can be subjective.
- **Difficulty in Reaching Consensus:** In some cases, it may be difficult to reach a consensus, especially if there are strong disagreements among experts.

When to Use Delphi Method:

- When there is a lack of historical data or reliable statistical models.
- When expert judgment is crucial for making accurate estimates.
- When you want to reduce the influence of individual biases.

1.8 Software Risks, Assumption, Issues, Dependency:

- Software risks are potential events or conditions that could negatively impact the successful development, deployment, or maintenance of a software project.
- These risks can manifest in various forms, leading to delays, cost overruns, quality issues, or even project failure.

Types of Software Risks:

There are mainly 3 classes of risks that may affect a computer code project:

1. **Project Risks**:
   Project risks concern various sorts of monetary funds, schedules, personnel, resources, and customer-related issues. A vital project risk is schedule slippage. Since computer code is intangible, it's tough to observe and manage a computer code project. It's tough to manage one thing that can not be seen. For any producing project, like producing cars, the project manager will see the merchandise taking form.
   For example, see that the engine is fitted, at the moment the area of the door unit is fitted, the automotive is being painted, etc. so he will simply assess the progress of the work and manage it. The physical property of the merchandise being developed is a vital reason why several computer codes come to suffer from the danger of schedule slippage.

2. **Technical Risks:**
   Technical risks concern potential style, implementation, interfacing, testing, and maintenance issues. Technical risks conjointly embody ambiguous specifications, incomplete specifications, dynamic specifications, technical uncertainty, and technical degeneration. Most technical risks occur thanks to the event team's lean information concerning the project.

3. **Business Risks:**
   This type of risk embodies the risks of building a superb product that nobody needs, losing monetary funds or personal commitments, etc.

The risk management process:

- Risk management is a sequence of steps that help a software team to understand, analyze, and manage uncertainty.

Risk management process consists of

- Risks Identification.
- Risk Assessment.
- Risks Planning.
- Risk Monitoring

Risk Management Process

1. **Risk Identification:**
   - Risk identification refers to the systematic process of recognizing and evaluating potential threats or hazards that could negatively impact an organization, its operations, or its workforce. This involves identifying various types of risks, ranging from IT security threats like viruses and phishing attacks to unforeseen events such as equipment failures and extreme weather conditions.

2. **Risk analysis:**
   - Risk analysis is the process of evaluating and understanding the potential impact and likelihood of identified risks on an organization. It helps determine how serious a risk is and how to best manage or mitigate it. Risk Analysis involves evaluating each risk's probability and potential consequences to prioritize and manage them effectively.

3. **Risk Planning:**
   - Risk planning involves developing strategies and actions to manage and mitigate identified risks effectively. It outlines how to respond to potential risks, including prevention, mitigation, and contingency measures, to protect the organization's objectives and assets.

4. **Risk Monitoring:**
   - Risk monitoring involves continuously tracking and overseeing identified risks to assess their status, changes, and effectiveness of mitigation strategies. It ensures that risks are regularly reviewed and managed to maintain alignment with organizational objectives and adapt to new developments or challenges.

Benefits of risk management

Here are some benefits of risk management:

- Helps protect against potential losses.
- Improves decision-making by considering risks.
- Reduces unexpected expenses.
- Ensures adherence to laws and regulations.
- Builds resilience against unexpected challenges.
- Safeguards company reputation.

Limitation of Risk Management

Here are Some Limitation of Risk Management

- Too much focus on risk can lead to missed opportunities.
- Implementing risk management can be expensive.
- Risk models can be overly complex and hard to understand.
- Having risk controls might make people feel too safe.
- Relies on accurate human judgment and can be prone to mistakes.
- Some risks are hard to predict or quantify.
- Managing risks can take a lot of time and resources.

Software Issues:

- Software issues, also known as software bugs or defects, are errors or problems that arise within software applications, leading to unexpected behaviour, incorrect results, or system crashes.
- These issues can range from minor inconveniences to major security vulnerabilities, impacting both individual users and organizations.

Common Types of Software Issues:

1. **Functional Issues:**
   - Errors in mathematical operations or formulas can lead to inaccurate results.
   - Software may not properly apply data validation rules, allowing invalid input.
2. **Security Vulnerabilities:**
   - Weak security measures can expose sensitive data to unauthorized access.
   - Software vulnerabilities can allow malicious software to infiltrate systems.
3. **Performance Issues:**
   - Slow loading times or sluggish performance can frustrate users.
   - Software may struggle to handle increased workload or data volume.
4. **Compatibility Issues:**
   - Software may not function correctly on certain hardware configurations.
   - Software may not be compatible with specific operating systems or versions.
5. **Documentation and Support Issues:**
   - Lack of clear documentation can hinder user understanding and troubleshooting.
   - Ineffective customer support can leave users frustrated and unable to resolve issues.

Causes of Software Issues:

- **Human Error:** Mistakes in coding, testing, or configuration can introduce bugs.
- **Complex Systems:** Large and complex software systems are more prone to errors.
- **Tight Deadlines:** Rushed development can lead to shortcuts and overlooked issues.
- **Poor Testing:** Inadequate testing can fail to identify and fix bugs.
- **Changing Requirements:** Frequent changes to requirements can introduce inconsistencies and errors.

Mitigating Software Issues:

- **Thorough Testing:** Rigorous testing processes, including unit, integration, and system testing, can help identify and fix bugs early in development.
- **Code Reviews:** Peer reviews of code can help identify potential issues and improve code quality.
- **Version Control:** Using version control systems allows developers to track changes, revert to previous versions, and collaborate effectively.
- **Continuous Integration and Continuous Delivery (CI/CD):** Automated build, test, and deployment processes can help identify and fix issues quickly.
- **User Feedback:** Gathering feedback from users can help identify and prioritize issues.
- **Security Best Practices:** Adhering to security best practices can help prevent vulnerabilities and data breaches.
- **Regular Updates and Patches:** Keeping software up-to-date with the latest patches and security fixes is essential.

1.9 Software Engineering Ethics and Professional Practice:

Goal:

- To Establish a code of conduct for professional software engineers to make software engineering a beneficial and respected profession.
- This is a Joint Effort by lEEE-Computer Society and Association of Computing MachineryACM

Software engineers are those who contribute either by direct participation or by

- teaching,
- analysing,
- Specification generating,
- designing,
- developing,
- certifying,
- Maintaining and
- testing

of software system

Roles of Engineers

- ✓ "Professional Software Engineers" include

Practitioners
Educators
Managers
Supervisors
Policy makers

Trainees and Students of the Profession

Software Engineering Ethics:

1. **PUBLIC**

- Software engineers shall act consistently with the public interest
- Accept full responsibility for their own work.
- Moderate the interests of the software engineer, the employer, the client and the users with the public good
- Approve software only if they believe that it is safe, meets specifications, passes appropriate tests
- Be fair and avoid deception in all statements, particularly public ones
- Consider issues of physical disabilities and allocation of resources
- Be encouraged to volunteer professional skills to good cause

2. **CLIENT AND EMPLOYER**

- Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest
- Provide service in their areas of competence
- Not knowingly use software that is obtained or retained either illegally or unethically.
- Use the property of a client or employer only in ways properly authorized
- Identify, document, collect evidence and report to the client or the employer promptly if, a project is likely to fail or to violate intellectual property.

3. **PRODUCT**
- Software engineers shall ensure that their products and related modifications meet the highest professional standards possible
- Strive for high quality and acceptable cost
- Ensure proper and achievable goals and objectives for any project
- Ensure that they are qualified for any project they work on
- Ensure that an appropriate method is used for any project
- Work to follow professional standards
- Strive to fully understand the specifications for software
- Ensure adequate testing, debugging, documentation and review of software
- Treat all forms of software maintenance with the same professionalism as new development

4. **JUDGMENT**
- Software engineers shall maintain integrity and independence in their professional judgment
- Temper all technical judgments by the need to support and maintain human values.
- Only endorse documents if prepared under supervision
- Maintain professional objectivity with respect to any software
- Not engage in deceptive financial practices such as double billing, or other improper financial practices.
- Disclose to all concerned parties those conflicts of interest that cannot reasonably be avoided or escaped

5. **MANAGEMENT**
- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance
- Ensure good management for any project on which they work
- Ensure that software engineers are informed of standards before being held to them.
- Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project
- Provide for due process in hearing charges of violation of an employer's policy or of this Code.
- Not ask a software engineer to do anything inconsistent with this Code.
- Not punish anyone for expressing ethical concerns about a project

6. **PROFESSION**
- Software engineers shall advance the integrity and reputation of the profession consistent with the public interest
- Help develop an organizational environment favourable to acting ethically
- Promote public knowledge of software engineering
- Support, as members of a profession, other software engineers striving to follow this Code.
- Not promote their own interest at the expense of the profession, client or employer.
- Take responsibility for detecting, correcting, and reporting errors in software
- Report significant violations of this Code to appropriate authorities.

7. **COLLEAGUES**
- Software engineers shall be fair to and supportive of their colleagues
- Encourage colleagues to adhere to this Code
- Assist colleagues in professional development
- Credit fully the work of others and refrain from taking undue credit
- Assist colleagues in being fully aware of current standard work practices
- Not unfairly intervene in the career of any colleague

8. **SELF**
- Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession
- Further their knowledge of recent developments
- Improve their ability to create safe, reliable, and useful quality software
- Improve their ability to produce accurate, informative, and well-written documentation
- Improve their knowledge of relevant standards
- Not influence others to undertake any action that involves a breach of this Code.

Professional Practices:

- **Continuous Learning:** Staying up-to-date with the latest technologies and best practices is crucial for maintaining professional competence.
- **Clear Communication:** Effective communication with clients, colleagues, and users is essential for understanding requirements, collaborating effectively, and addressing issues.
- **Accountability:** Engineers should take responsibility for their work, acknowledging mistakes and striving for improvement.
- **Teamwork:** Collaboration with other professionals is vital for successful software development, fostering a positive and productive work environment.
- **Adherence to Standards:** Following established coding standards, guidelines, and regulations ensures consistency and quality in software development.

Ethical Challenges in Software Engineering:

- **Privacy and Security:** Protecting user data and ensuring the security of software systems are major ethical concerns.
- **Bias and Discrimination:** Developers must be mindful of potential biases in algorithms and ensure fairness in software design.
- **Environmental Impact:** The energy consumption and carbon footprint of software systems should be considered.
- **Intellectual Property:** Respecting copyright laws and avoiding plagiarism are essential.
- **Social Responsibility:** Software engineers should consider the broader societal impact of their work and avoid creating harmful or discriminatory software.