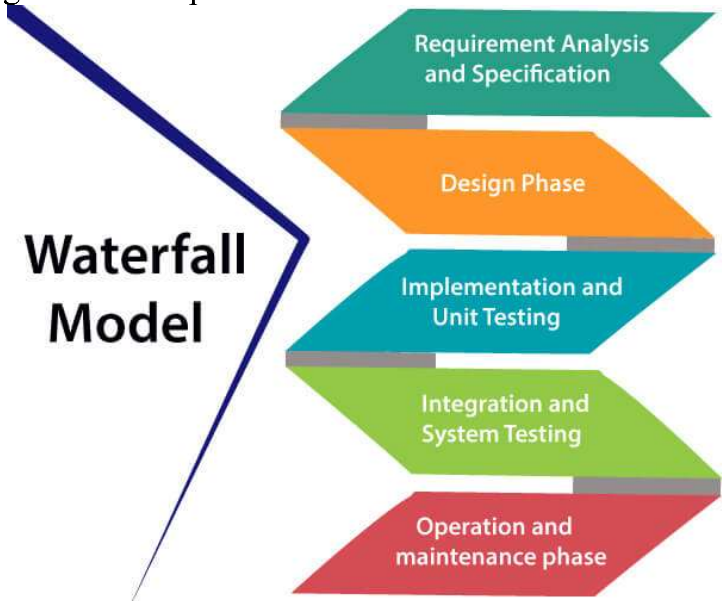
For Free Resources Visit : aajhub.in , HeyTopper.in

UNIT-I:
Conventional Software Management: The waterfall model, conventional software Management performance.
Evolution of Software Economics: Software Economics, pragmatic software cost estimation.
Improving Software Economics: Reducing Software product size, improving software processes, improving team effectiveness, improving automation, Achieving required quality, peer inspections.
The old way and the new: The principles of conventional software Engineering, principles of modern software management, transitioning to an iterative process.

The 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. The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins. This model is named "Waterfall Model", because its diagrammatic representation resembles a cascade of waterfalls.


Waterfall Model
Requirement Analysis and Specification
Design Phase
Implementation and Unit Testing
Integration and System Testing
Operation and maintenance phase

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. It describes the "what" of the system to be produced and not "how. "In 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.

For Free Resources Visit : aajhub.in , HeyTopper.in

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. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

4. **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. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

5. **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 the Waterfall Model: -

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

For Free Resources Visit : aajhub.in , HeyTopper.in

- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Disadvantages of the Waterfall Model: -

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Conventional Software Management Performance: -

Conventional software management in software engineering refers to traditional practices that are theoretically sound but often tied to outdated technology and techniques. Here are some key principles and performance aspects:

1. **Quality Creation:** The quality of software must be measured or quantified, and mechanisms should be put into place to motivate towards achieving the goal.
2. **Early Delivery:** It's important to give end-products to customers early. This allows users to interact with the product, providing a more effective way to identify their real needs.
3. **Problem Identification:** Before attempting to solve a problem, it's crucial to explore all alternatives and not get blinded by an obvious solution.
4. **Design Alternatives Evaluation:** When requirements are agreed upon, various architectures and algorithms should be examined and understood.
5. **Appropriate Process Model:** Each project should select a process that makes the most sense for that project based on factors like corporate culture, willingness to take risks, application area, volatility of requirements, and the extent to which requirements are understood.
6. **Code Inspection:** Inspecting design details and code is a better way of identifying errors than testing.
7. **Good Management:** Good management motivates people to do their best work. It's more important than good technology.

In terms of performance, conventional software management practices have shown that software development is still highly unpredictable. Only about 10%

For Free Resources Visit : aajhub.in , HeyTopper.in

of software projects are delivered successfully within initial budget and schedule estimates. The level of software scrap and rework is indicative of an immature process. Therefore, there's a need for continuous improvement in these practices to increase the success rate of software projects.

Difference between Agile and Waterfall Methodologies

Agile	**Waterfall**
It separates the project development lifecycle into sprints.	Software development process is divided into distinct phases.
It follows an incremental approach	Waterfall methodology is a sequential design process.
Agile methodology is known for its flexibility.	Waterfall is a structure software development methodology so most times it can be quite rigid.
Agile can be considered as a collection of many different projects.	Software development will be completed as one single project.
Agile is quite a flexible method which allows changes to be made in the project development requirements even if the initial planning has been completed.	There is no scope of changing the requirements once the project development starts.
Agile methodology, follow an iterative development approach because of this planning, development, prototyping and other software development phases may appear more than once.	All the project development phases like designing, development, testing, etc. are completed once in the Waterfall model.
Test plan is reviewed after each sprint.	The test plan is rarely discussed during the test phase.
Agile development is a process in which the requirements are expected to change and evolve.	The method is ideal for projects which have definite requirements and changes not at all expected.
In Agile methodology, testing is performed concurrently with software development.	In this methodology, the "Testing" phase comes after the "Build" phase.
Agile introduces a product mindset where the software product satisfies needs of its end customers and changes itself as per the customer's demands.	This model shows a project mindset and places its focus completely on accomplishing the project.

For Free Resources Visit : aajhub.in , HeyTopper.in

Agile methodology works exceptionally well with Time & Materials or non-fixed funding. It may increase stress in fixed-price scenarios.	Reduces risk in the firm fixed price contracts by getting risk agreement at the beginning of the process.
Prefers small but dedicated teams with a high degree of coordination and synchronization.	Team coordination/synchronization is very limited.
Products owner with team prepares requirements just about every day during a project.	Business analysis prepares requirements before the beginning of the project.
Test team can take part in the requirements change without problems.	It is difficult for the test to initiate any change in requirements.
Description of project details can be altered anytime during the SDLC process.	Detail description needs to implement waterfall software development approach.
The Agile Team members are interchangeable, as a result, they work faster. There is also no need for project managers because the projects are managed by the entire team.	In the waterfall method, the process is always straightforward so, project manager plays an essential role during every stage of SDLC.

Software Economics: -

Software economics in software project management is a mature research area that generally deals with most difficult and challenging problems and issues of valuing software and determining or estimation costs usually involved in its production. Boehm and Sullivan outline these difficulties and challenges and also presented how software economics principles can be applied to improve software design, development, and evolution. Software economics is basically situated at intersection of information economics and even software design and engineering. Most of software cost models are generally abstracted into function of five basic parameters. These parameters are given below:

1. **Size –** Size is generally measured or qualified in terms of number of source instructions or in SLOC (Source line of code) or number of function points required to realize desired capabilities. The size of end product or result is required to develop or create required functionality.

2. **Process –** The process is steps that are used to guide all of activities and produce end products, in particular ability and capability of process to

For Free Resources Visit : aajhub.in , HeyTopper.in

avoid or ignore activities that are not adding any value. It also supports heading towards the target or goal and eliminate activities that are not essential or important.

3. **Personnel –** The capabilities of personnel of software engineering in general, and particularly their experience with issues or problems regarding computer science and issues regarding application domain of project. It emphasizes on team and responsibilities of team.

4. **Environment –** It is simply made of various tools and techniques and automated procedures that are available and used to support software development and effort in an efficient way.

5. **Quality –** The required quality along with its features, performance, reliability, scalability, portability, usability, user interface utility, adaptability, and many more.

Pragmatic Software Cost Estimation: -

Pragmatic software cost estimation is a practical approach to predicting the amount of effort, time, and resources required to develop a software project. It involves using realistic methods and models to provide a reliable estimate of the costs associated with software development.

Here are some commonly used models in pragmatic software cost estimation:

1. **COCOMO (Constructive Cost Model):** This model uses a basic formula to estimate the effort and duration of a project based on the size of the software, which is usually measured in lines of code or function points.
2. **Function Point Analysis:** This method measures the functionality provided by the software, independent of the technology used for implementation. It considers the user’s point of view and the functionalities that are visible to the user.
3. **Feature Points:** This method is similar to function point analysis but also takes into account the complexity of the software, making it suitable for systems software.
4. **Use Case Points:** This method estimates the effort based on use cases, which describe interactions between the software and its users.
5. **Parametric Estimating:** This method uses statistical modelling to develop a cost estimate. It involves identifying the key cost drivers, collecting data from past projects, and using this data to predict future costs.

These models are used because they provide a systematic and objective way to estimate the cost of a software project. They help in planning, monitoring, and

For Free Resources Visit : aajhub.in , HeyTopper.in

controlling the project's progress, ensuring that the project is delivered on time and within budget.

Detailed Concept of Pragmatic Software Cost Estimation: -

Pragmatic software cost estimation is a practical approach to predicting the amount of effort, time, and resources required to develop a software project. It involves using realistic methods and models to provide a reliable estimate of the costs associated with software development.
Here are some commonly used models in pragmatic software cost estimation:

1. COCOMO (Constructive Cost Model)
2. Function Point Analysis
3. Feature Points
4. Use Case Points
5. Parametric Estimating

1. COCOMO (Constructive Cost Model)

The Constructive Cost Model (COCOMO) is a procedural cost estimate model for software projects that was created by Barry Boehm in the 1970s. It's often used to reliably predict various parameters associated with a project, such as size, effort, cost, time, and quality. COCOMO is a regression model based on the number of lines of code (LOC) and is used to predict the effort required for the project, total project cost, and scheduled time for the project.
COCOMO has three versions: Basic, Intermediate, and Detailed:

i. **Basic COCOMO Model:** This is the simplest version of the model and is used for projects that are relatively small and straightforward. The effort is measured in person-months and is dependent on kilo-lines of code.
ii. **Intermediate COCOMO Model:** This model takes into account a set of cost drivers that have an impact on project cost.
iii. **Detailed COCOMO Model:** In this model, the effort is calculated as a function of program size and a set of cost drivers given according to each phase of the software life cycle. The five phases of detailed COCOMO are: plan and requirement, system design, detailed design, module code and test, and integration and test.

Different models of COCOMO have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of the constant to be used in subsequent calculations.

For Free Resources Visit : aajhub.in , HeyTopper.in

In COCOMO, projects are categorized into three types: Organic, Semi-detached, and Embedded:

I. **Organic:** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past, and also the team members have a nominal experience regarding the problem.

II. **Semi-detached:** A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environment lie in between that of organic and Embedded.

III. **Embedded:** A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category.

These models help in planning, monitoring, and controlling the project's progress, ensuring that the project is progressing according to the procedure and taking corrective action, if necessary.

2. **Function Point Analysis Model: -**

Function Point Analysis (FPA) is a method used in pragmatic software cost estimation to measure the functionality provided by a software system. It quantifies the amount of business functionality an information system provides to a user and is independent of the technology used to implement it. FPA is based on the user's point of view and measures the functionalities that are visible to the user. The main goal of FPA is to provide a normalized measure for software that allows comparison across projects and technologies. The process of FPA involves the following steps:

i. **Identify the functional user requirements:** This includes all the functionalities that the user expects from the software system.

ii. **Classify the functions:** The functions are classified into five types: External Inputs (EI), External Outputs (EO), Inquiries (EQ), Internal Logical Files (ILF), and External Interface Files (EIF).

iii. **Calculate the Unadjusted Function Points (UFP):** Each type of function is assigned a complexity weight (low, average, or high). The UFP is the sum of the weights of all functions.

iv. **Calculate the Value Adjustment Factor (VAF):** This is based on 14 general system characteristics (GSCs) that rate the general functionality of the software. The VAF is calculated as 0.65 + 0.01 * ∑(GSCs).

v. **Calculate the Adjusted Function Points (AFP):** The AFP is calculated as AFP = UFP * VAF. This is the final measure of the functional size of the software.

For Free Resources Visit : aajhub.in , HeyTopper.in

FPA is widely used because it provides a technology-independent assessment of the size of a software system, which is necessary for estimating the effort, cost, and duration of software projects. It also helps in benchmarking and measuring productivity, estimating resources, and scope management.

3. **Feature Points Model: -**
   The Feature Points model is an extension of the Function Points model, designed to handle systems software or software that has complex algorithms. It was developed to address the limitations of Function Points in dealing with systems where complex processing is more prevalent than data handling.
   In the Feature Points model, the functionality of the software is divided into two categories:

   i. **Data Function Types:** These are similar to those in the Function Points model and include External Inputs (EI), External Outputs (EO), Logical Internal Files (LIF), and External Interface Files (EIF).
   ii. **Algorithmic Function Types:** These are unique to the Feature Points model and include Control Content (CONT), Control Style (STYL), and Control Structure (STRU).

   Each of these function types is assigned a complexity weight (low, average, or high), and the Unadjusted Function Points (UFP) is calculated as the sum of the weights of all functions. The Value Adjustment Factor (VAF) is then calculated based on 14 general system characteristics (GSCs), and the Adjusted Function Points (AFP) is calculated as **`AFP = UFP * VAF`**.

   The Feature Points model is used in pragmatic software cost estimation because it provides a more accurate measure of the functional size of systems software or software with complex algorithms. It helps in planning, monitoring, and controlling the project's progress, ensuring that the project is delivered on time and within budget.

4. **Use Case Points Model: -**
   The Feature Points model is an extension of the Function Points model, designed to handle systems software or software that has complex algorithms. It was developed to address the limitations of Function Points in dealing with systems where complex processing is more prevalent than data handling.
   In the Feature Points model, the functionality of the software is divided into two categories:
   i. **Data Function Types:** These are similar to those in the Function Points model and include External Inputs (EI), External Outputs (EO), Logical Internal Files (LIF), and External Interface Files (EIF).

For Free Resources Visit : aajhub.in , HeyTopper.in

ii. **Algorithmic Function Types:** These are unique to the Feature Points model and include Control Content (CONT), Control Style (STYL), and Control Structure (STRU).

Each of these function types is assigned a complexity weight (low, average, or high), and the Unadjusted Function Points (UFP) is calculated as the sum of the weights of all functions. The Value Adjustment Factor (VAF) is then calculated based on 14 general system characteristics (GSCs), and the Adjusted Function Points (AFP) is calculated as **`AFP = UFP * VAF`.**

The Feature Points model is used in pragmatic software cost estimation because it provides a more accurate measure of the functional size of systems software or software with complex algorithms. It helps in planning, monitoring, and controlling the project's progress, ensuring that the project is delivered on time and within budget.

5. **Parametric Estimating Model: -**
The Parametric Estimating Model is a method used in software cost estimation. It involves the use of mathematical algorithms or parametric equations to estimate the cost of a product or a project.

This model uses regression analysis of a database of two or more similar systems to develop cost estimating relationships (CERs) which estimate cost based on one or more system performance or design characteristics (e.g., speed, range, weight, thrust).

The basic formula for calculating parametric estimates is:
Cost/Time per Parameter x Parameter Value = Estimated Project Cost/ Time

This model is often used during a project or in the project planning phase. It applies a formula or algorithm for making these calculations, using the specific cost or time needed to implement and finish a project or task.

The Parametric Estimating Model, while useful, does have some limitations:

a. **Data Availability:** Parametric estimating can be time-consuming and costly, especially when used for a complex project. The data required for this model may be unavailable or difficult to obtain. Historical data may be unavailable or of low quality.
b. **Data Accuracy:** External data can be skewed and difficult to verify. Without accurate data, the estimates may not be accurate. Many times

For Free Resources Visit : aajhub.in , HeyTopper.in

an analyst unknowingly incorporates flawed data into the database, producing inaccurate Cost Estimating Relationships (CERs).

c. **Misleading Statistics:** For a variety of reasons, the resulting statistics can be misleading. This is often due to the constraints imposed by the amount and quality of the data.

d. **Complexity:** The model requires a deep understanding of the project and its parameters. It also requires expertise in statistical analysis and the ability to develop and validate the mathematical models used.

These models are used because they provide a systematic and objective way to estimate the cost of a software project. They help in planning, monitoring, and controlling the project's progress, ensuring that the project is delivered on time and within budget.

Reducing Software Product Size: -
Reducing software product size is a significant way to improve the economics of software development. Here are some strategies that can be used:

1. Component-Based Development: This is a general term for reducing the "source" language size to achieve a software solution. It involves using pre-built components to reduce the amount of code that needs to be written.

2. Reuse and Object-Oriented Technology: Reusing existing code and using object-oriented technologies can help achieve a given system with fewer lines of human-specified source directives.

3. Automatic Code Production and Higher Order Programming Languages: Improvements in higher order languages (such as C++, Ada 95, Java, Visual Basic), automatic code generators (CASE tools, visual modeling tools, GUI builders), and reuse of commercial components (operating systems, windowing environments, database management systems, middleware, networks) are all focused on achieving a given system with fewer lines of human-specified source directives[1].

4. Managing Scope: Managing the scope of the project and raising the level of abstraction through component-based technology and service-oriented architectures are high leverage techniques that make a difference.

5. Large-Scale Scrum (LeSS) Framework: This is a framework for scaling scrum to multiple teams who work together on a single product. It helps in delivering value while reducing complexity and waste.

For Free Resources Visit : aajhub.in , HeyTopper.in

Remember, the goal is to produce a product that achieves the design goals with the minimum amount of human-generated source material.

Improving Software Processes: -
Improving software processes in Software Product Management (SPM) can be achieved through various strategies. Here are some tips:

- **Planning and Preparation:** Before starting the development process, it's crucial to have a clear plan and prepare all the necessary resources.
- **Identify the Problem:** Understand the problem that the software product is supposed to solve.
- **Design the Solution:** Create a detailed design of the solution that addresses the identified problem.
- **Implement the Solution:** Develop the software product according to the design.
- **Test and Deploy:** Test the software product thoroughly to ensure it works as expected, and then deploy it.
- **Monitor and Maintain:** Continuously monitor the software product's performance and maintain it to ensure it remains effective.
- **Pick the Right SDLC Model:** Choose a Software Development Life Cycle (SDLC) model that best fits the project's requirements.
- **Optimize Your Workflow:** Streamline the development process to improve efficiency.
- **Manage Code Quality:** Ensure the code is clean, efficient, and maintainable.
- **Have a Clear Definition of Done:** Define what it means for a task to be completed.
- **Build Communication Practices:** Establish effective communication practices among the team members.
- **Establish Clear Development Standards:** Set clear standards for the development process.
- **Work with Small Teams and Small Components:** This can improve focus and reduce complexity.
- **Limit Your Work in Progress (WIP):** This can help to maintain focus and reduce multitasking.
- **Leverage the Lean Approach:** This approach focuses on reducing waste and improving efficiency.
- **Adopt Agile Methodologies:** Agile methodologies, such as Scrum or Kanban, can increase collaboration and flexibility.
- **Implement Continuous Integration and Delivery:** This practice can improve efficiency and reduce time-to-market.

For Free Resources Visit : aajhub.in , HeyTopper.in

Remember, the goal is to continuously improve the software development process to deliver high-quality software products efficiently.

Improving Team Effectiveness: -
Improving team effectiveness in Software Product Management (SPM) is crucial for the successful delivery of software products. Here are some strategies:

- **Break teams into squads or pods with individual responsibilities:** This can help to distribute the workload evenly and ensure that all aspects of the product are being addressed.
- **Create cross-functional collaboration:** Encourage different teams to work together. This can lead to more innovative solutions and can help to ensure that all aspects of the product are considered.
- **Build customer-first processes:** Always keep the customer in mind when making decisions. This can help to ensure that the product meets the needs and expectations of the customer.
- **Foster strong relationships between product, engineering, and marketing teams:** These teams need to work closely together to ensure that the product is developed and marketed effectively.
- **Give your team members ownership:** Letting team members make their own decisions and making them accountable for their work can induce a sense of responsibility.
- **Ensure proper communication:** Effective communication is essential for success. It helps to ensure everyone is on the same page and opportunities are being leveraged to the fullest.
- **Identify your team’s strengths and weaknesses:** This can help to assign tasks more effectively and ensure that each team member is working in an area where they can excel.
- **Team building activities:** These can help to improve communication and trust within the team.
- **Use a project management tool:** This can help to keep track of tasks and deadlines, and ensure that everyone knows what they need to do.
- **Wholesome work environment:** A positive and supportive work environment can help to improve team morale and productivity.

Remember, the goal is to continuously improve the team's effectiveness to deliver high-quality software products efficiently.

Achieving Required Quality: -
Achieving the required quality in Software Product Management (SPM) is a multi-faceted process that involves several key steps:

For Free Resources Visit : aajhub.in , HeyTopper.in

- **Define Quality Characteristics:** Identify the characteristics that define quality for your product.
- **Measure Quality Characteristics:** Decide how to measure each of those quality characteristics.
- **Set Quality Standards:** Establish standards for each quality characteristic.
- **Quality Control:** Perform quality control with respect to the standards.
- **Identify Quality Issues:** Find out the reasons that are hindering quality.
- **Management Plan:** Have a clear idea about how the quality assurance process will be carried out through the project.
- **Quality Engineering Activities:** Quality engineering activities required should also be set at the beginning along with team skill check.
- **Proper Checkpoints:** Checkpoints at required intervals should be set.

Peer Inspections: -

Peer inspections, also known as peer reviews, are considered an industry best-practice for detecting software defects early and learning about software artifacts. They are composed of software walkthroughs and software inspections and are integral to software product engineering activities.

In a peer review, co-workers of a person who created a software work product examine that product to identify defects and correct shortcomings. A review verifies whether the work product correctly satisfies the specifications found in any predecessor work product, such as requirements or design documents.

Peer inspections are frequently overhyped as the key aspect of a quality system. However, they are valuable as secondary mechanisms, but they are rarely significant contributors to quality compared with other primary quality mechanisms and indicators, which should be emphasized in the management process.

The National Software Quality Experiment, evaluating the effectiveness of peer reviews, finds, "a favourable return on investment for software inspections; savings exceeds costs by 4 to 1". To state it another way, it is four times more costly, on average, to identify and fix a software problem later.

Therefore, peer inspections play a crucial role in improving software economics in Software Product Management (SPM) by reducing software product size, improving software processes, improving team effectiveness, improving automation, and achieving required quality.

For Free Resources Visit : aajhub.in , HeyTopper.in

The Principles of Conventional Software Engineering: -
The principles of conventional software engineering in Software Product Management (SPM) are guidelines for developing high-quality software products. Some of these principles are:

1. **Make Quality the Top Priority and Quantify It:** Quality of software must be measured or quantified and mechanisms put into place to motivate towards achieving the goal.

2. **Involve the Customer, Prototype, Simplify Design, Conduct Inspections, and Hire the Best People:** Large or high-quality software is possible. There are several techniques that have been given a practical explanation to raise and increase and quality includes involving customer, prototyping, simplifying design, conducting inspections, and hiring good and best people.

3. **Give Products to Customers Early and Get Feedback:** It doesn't matter how hard we try to learn and know about needs of users during requirements stage, most important and effective to determine and identify their real and realistic needs is to give product to users and let them play with it.

4. **Determine the Problem and the Requirements Before Writing Code:** When engineers usually face with what they believe is a problem, they rush towards offering a solution. But we should keep in mind that before we try to solve any problem, be sure to explore all alternatives and just don't get blinded by obvious solution.

5. **Evaluate All Design Alternatives:** When requirements or needs are agreed upon, we must examine and understand various architecture and algorithms.

6. **Use an Appropriate and Correct Process Model:** Each and every project must select an appropriate process that makes most sense for that project generally on basis of corporate culture, willingness to take risks, application area, volatility of requirements, and extent to which all requirements and needs are understood in a good manner.

7. **Use Different Languages for Different Phases:** Our industry generally gives simple solutions to large complex problems. Due to his, many declare that best development method is only one that makes use of notation throughout life cycle.

For Free Resources Visit : aajhub.in , HeyTopper.in

8. **Minimize or Reduce Intellectual Distance:** The structure of software must be very close enough to a real-world structure to minimize intellectual distance.

9. **Before Tools, Put Techniques:** An undisciplined software engineer with a tool becomes very dangerous and harmful.

10. **Get It Right Just Before We Make It Very Faster:** It is very easy and simple to make a program that's being working run very faster than it is to simply make a program work fast.

11. **Inspect Code:** Assessing or inspecting design with its details and code is most and better way of identifying errors other than testing.

12. **Rather Than Good Technology, Good Management Is More Important**: Good management simply motivates other people also to do their work at best, but there are no universal "right" styles for management.

Principles of Modern Software Management: -
The principles of modern software management in Software Product Management (SPM) are guidelines for effectively managing software development projects and teams. Some of these principles are:

1. **Agile Methodologies:** Agile methodologies like Scrum, Kanban, and Lean allow development teams to embrace flexibility, adaptability, and collaborative practices.

2. **Continuous Integration and Delivery (CI/CD):** By implementing automated processes for code integration, testing, and deployment, development teams can achieve faster and more reliable software releases.

3. **Embracing DevOps:** DevOps promotes collaboration and efficiency in software management by breaking down silos between development and operations teams.

4. **User-Centric Design:** Prioritizing user needs, conducting thorough user research, and incorporating user feedback can create software solutions that deliver exceptional user experiences.

5. **Lean Software Development:** This principle focuses on eliminating waste and maximizing value.

6. **Operations:** Services need management.

For Free Resources Visit : aajhub.in , HeyTopper.in

7. **Monitoring:** Services also need to be monitored.

8. **Eventing and Alerting:** What happens if the monitoring solution detects a problem?

9. **Collaboration:** A first responder is the first person, but probably not the only person, who helps to resolve an incident.

10. **Root Cause Analysis:** To prevent an incident from reappearing, the contributing factors must be assessed.