INDEX

Activity 1

Need of software engineering

Software engineering

Software engineering is a technique through which we can develop or create software for computer systems or any other electronic devices. It is a systematic, scientific and disciplined approach to the development, functioning, and maintenance of software.

Basically, Software engineering was introduced to address the issues of low-quality software projects. Here, the development of the software uses the well-defined scientific principle method and procedure.

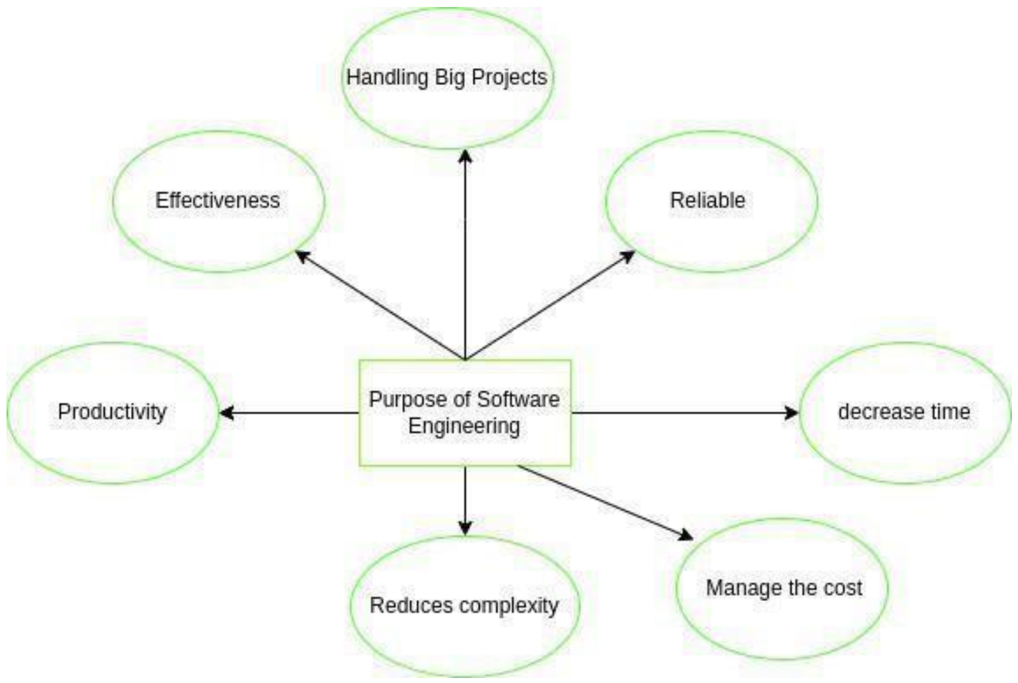
In other words, software engineering is a process in which the need of users are analyzed and then the software is designed as per the requirement of the user. Software engineering builds this software and application by using designing and programming language.

In order to create complex software, we need to use software engineering techniques as well as reduce the complexity we should use abstraction and decomposition, where abstraction describes only the important part of the software and remove the irrelevant things for the later stage of development so the requirement of the software becomes simple. Decomposition breakdown of the software in a number of modules where each module procedure as well defines the independent task

Need of Software Engineering:

- **Handling Big Projects:** A corporation must use a software engineering methodology in order to handle large projects without any issues.
- **To manage the cost:** Software engineering programmers plan everything and reduce all those things that are not required.
- **To decrease time:** It will save a lot of time if you are developing software using a software engineering technique.
- **Reliable software:** It is the company's responsibility to deliver software products on schedule and to address any defects that may exist.
- **Effectiveness:** Effectiveness results from things being created in accordance with the standards.
- **Reduces complexity:** Large challenges are broken down into smaller ones and solved one at a time in software engineering. Individual solutions are found for each of these issues.

- **Productivity:** Because it contains testing systems at every level, proper care is done to maintain software productivity.



Software engineering is the process of designing, developing, testing, and maintaining software. It is needed because software is a complex and constantly evolving field that requires a structured approach to ensure that the end product is of high quality, reliable, and meets the needs of the users. Additionally, software engineering helps to manage the costs, risks and schedule of the software development process. It also provides a way to improve the software development process over time through testing and feedback.

- Software engineering is a discipline that involves applying engineering principles to the development of software. It is a systematic approach to designing, developing, testing, and maintaining software that ensures that the end product is of high quality, reliable, and meets the needs of the users. The goal of software engineering is to produce software that is efficient, easy to use, and easy to maintain.
- One of the main challenges in software development is managing the complexity of large software systems. Software engineering provides a set of techniques and methodologies that help to manage this complexity and improve the software development process. For example, software engineering practices such as Agile methodologies, Scrum, and Waterfall provide a framework for managing the software development process.
- Another important aspect of software engineering is testing and quality assurance. Software engineering provides a variety of testing methods and tools to ensure that the software meets its requirements and is free of bugs. This includes unit testing, integration testing, and acceptance testing.

Activity 2

Agile Methodology

What is the Agile Methodology?

Agile methodologies are iterative and incremental, which means it's known for breaking a project into smaller parts and adjusting to changing requirements.

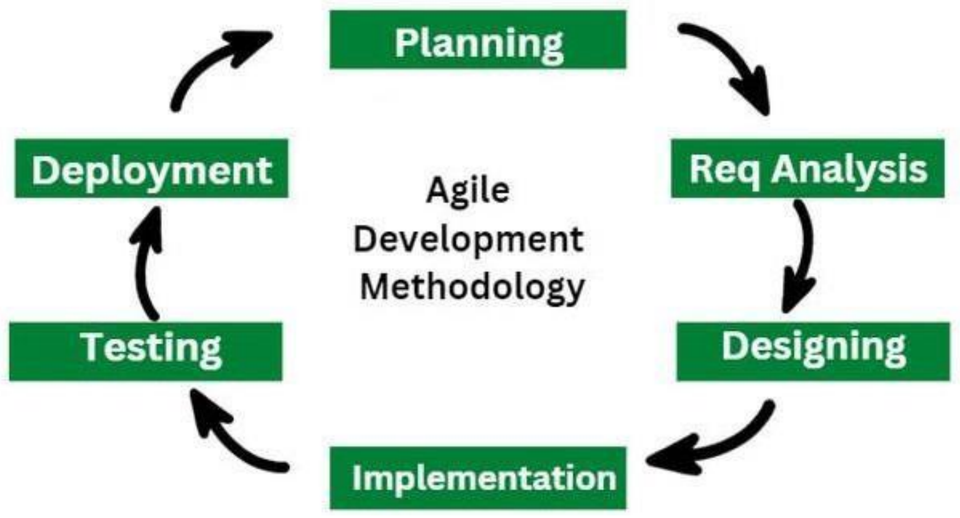
1. They prioritize flexibility, collaboration, and customer satisfaction.
2. Major companies like Facebook, Google, and Amazon use Agile because of its adaptability and customer-focused approach.

History of Agile

- In 1957, people started figuring out new ways to build computer programs. They wanted to make the process better over time, so they came up with iterative and incremental methods.
- In the 1970s, people started using adaptive software development and evolutionary project management. This means they were adjusting and evolving how they built software.
- In 1990s, there was a big change. Some people didn't like the strict and super-planned ways of doing things in software development. They called these old ways "waterfall." So, in response, lighter and more flexible methods showed up. These included:
    - Rapid Application Development (RAD) in 1991.
    - Unified Process (UP), Dynamic Systems Development Method (DSDM) in 1994.
    - Scrum in 1995.
    - Crystal Clear and Extreme Programming (XP) in 1996.
    - Feature-Driven Development (FDD) in 1997.

Life cycle of Agile Methodology

The Agile software development life cycle helps you break down each project you take on into six simple stages:



1. Requirement Gathering

- In this stage, the project team identifies and documents the needs and expectations of various stakeholders, including clients, users, and subject matter experts.
- It involves defining the project's scope, objectives, and requirements.
- Establishing a budget and schedule.
- Creating a project plan and allocating resources.

2. Design

- Developing a high-level system architecture.
- Creating detailed specifications, which include data structures, algorithms, and interfaces.
- Planning for the software's user interface.

3. Development (Coding)

Writing the actual code for the software. Conducting unit testing to verify the functionality of individual components.

4. Testing

This phase involves several types of testing:

- **Integration Testing:** Ensuring that different components work together.
- **System Testing:** Testing the entire system as a whole.
- **User Acceptance Testing:** Confirming that the software meets user requirements.

- **Performance Testing:** Assessing the system's speed, scalability, and stability.

5. Deployment

- Deploying the software to a production environment.
- Put the software into the real world where people can use it.
- Make sure it works smoothly in the real world.
- Providing training and support for end-users.

6. Review (Maintenance)

- Addressing and resolving any issues that may arise after deployment.
- Releasing updates and patches to enhance the software and address problems.

Manifesto for Agile Software Development

The Manifesto for Agile Software Development is a document produced by 17 developers at Snowbird, Utah in 2001. This document consists of 4 Agile Values and 12 Agile Principles. These 12 principals and 4 agile values provide a guide to Software Developers. The Manifesto for Agile Software Development emerged as a transformative guide to **Software Development**.

What are the 12 Agile Principles?

There are 12 agile principles mentioned in the Agile Manifesto. Agile principles are guidelines for flexible and efficient software development. They emphasize frequent delivery, embracing change, collaboration, and continuous improvement. The focus is on delivering value, maintaining a sustainable work pace, and ensuring technical excellence.

Agile Software Development

Agile Software Development is a software development methodology that values flexibility, collaboration, and customer satisfaction. It is based on the Agile Manifesto, a set of principles for software development that prioritize individuals and interactions, working software, customer collaboration, and responding to change.

Agile Software Development is an iterative and incremental approach to software development that emphasizes the importance of delivering a working product quickly and frequently..

What are Agile frameworks?

Agile frameworks are methods of organizing and dealing with software program development initiatives that follow the principles and values of the Agile Manifesto. Agile frameworks intend to supply value to clients faster and extra often, even also allowing groups to conform to converting requirements and remarks.

Activity 3

Extreme Programming (XP)

What is Extreme Programming (XP)?

Extreme Programming (XP) is an Agile software development methodology that focuses on delivering high-quality software through frequent and continuous feedback, collaboration, and adaptation. XP emphasizes a close working relationship between the development team, the customer, and stakeholders, with an emphasis on rapid, iterative development and deployment.

Agile development approaches evolved in the 1990s as a reaction to documentation and bureaucracy-based processes, particularly the waterfall approach. Agile approaches are based on some common principles, some of which are:

1. Working software is the key measure of progress in a project.
2. For progress in a project, therefore software should be developed and delivered rapidly in small increments.
3. Even late changes in the requirements should be entertained.
4. Face-to-face communication is preferred over documentation.
5. Continuous feedback and involvement of customers are necessary for developing good-quality software.
6. A simple design that involves and improves with time is a better approach than doing an elaborate design up front for handling all possible scenarios.
7. The delivery dates are decided by empowered teams of talented individuals.

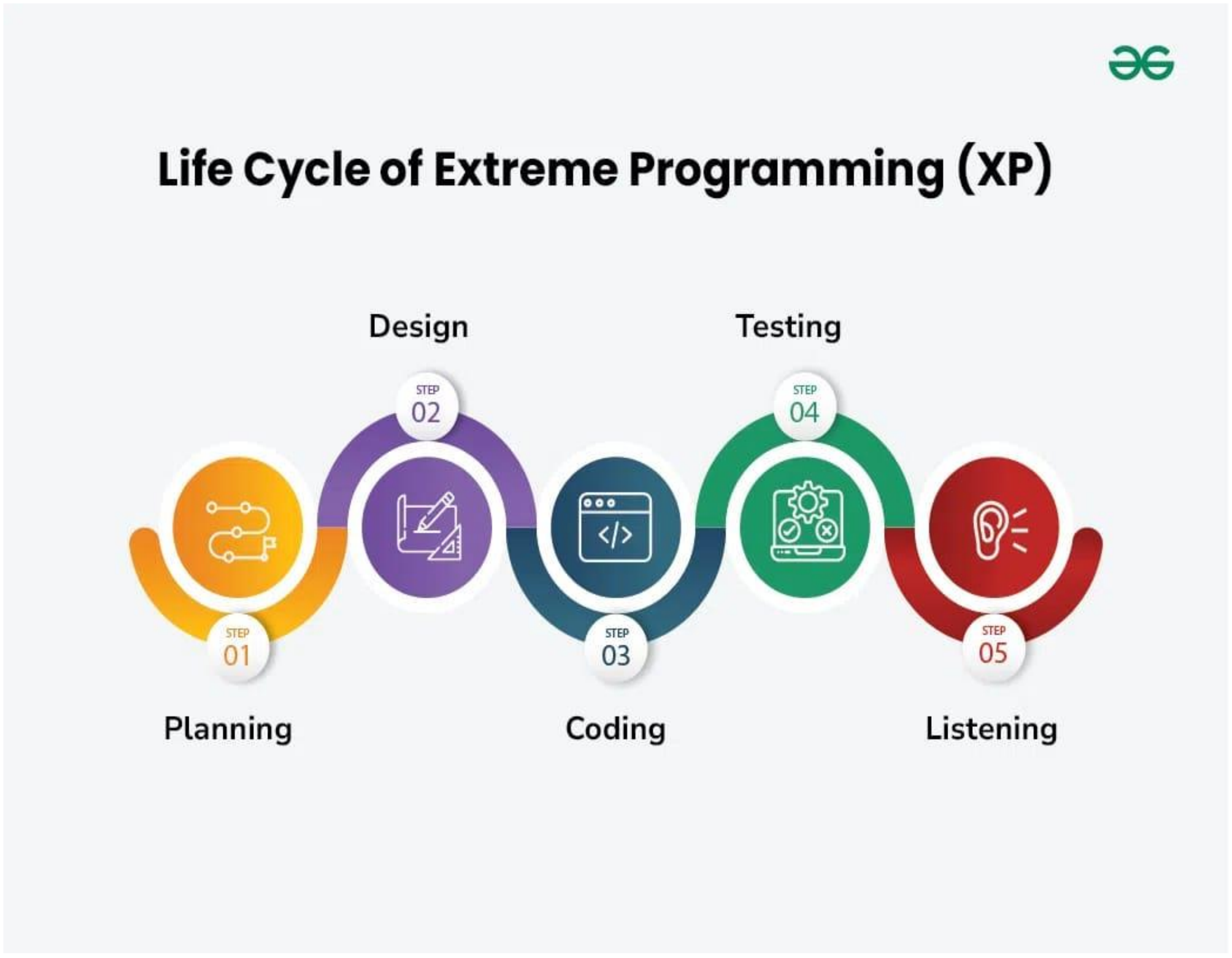
Extreme programming is one of the most popular and well-known approaches in the family of agile methods. an XP project starts with user stories which are short descriptions of what scenarios the customers and users would like the system to support. Each story is written on a separate card, so they can be flexibly grouped.

Good Practices in Extreme Programming

Some of the good practices that have been recognized in the extreme programming model and suggested to maximize their use are given below:

- **Code Review:** Code review detects and corrects errors efficiently. It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their work between them every hour.

- **Testing:** Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach, test cases are written even before any code is written.
- **Incremental development:** Incremental development is very good because customer feedback is gained and based on this development team comes up with new increments every few days after each iteration.
- **Simplicity:** Simplicity makes it easier to develop good-quality code as well as to test and debug it.
- **Design:** Good quality design is important to develop good quality software. So, everybody should design daily.
- **Integration testing:** Integration Testing helps to identify bugs at the interfaces of different functionalities. Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.



Basic Principles of Extreme programming

XP is based on the frequent iteration through which the developers implement User Stories. User stories are simple and informal statements of the customer about the functionalities needed. A User Story is a conventional description by the user of a feature of the required system. It does not mention finer details such as the different scenarios that can occur. Based on User stories, the project team proposes Metaphors. Metaphors are a common vision of how the system would work. The development team may decide to build a Spike for some features. A Spike is a very simple program that is constructed to explore the suitability of a solution being proposed. It can be considered similar to a prototype. Some of the basic activities that are followed during software development by using the XP model are given below:

- **Coding:** The concept of coding which is used in the XP model is slightly different from traditional coding. Here, the coding activity includes drawing diagrams (modeling) that will be transformed into code, scripting a web-based system, and choosing among several alternative solutions.
- **Testing:** The XP model gives high importance to testing and considers it to be the primary factor in developing fault-free software.
- **Listening:** The developers need to carefully listen to the customers if they have to develop good quality software. Sometimes programmers may not have the depth knowledge of the system to be developed. So, the programmers should understand properly the functionality of the system and they have to listen to the customers.
- **Designing:** Without a proper design, a system implementation becomes too complex, and very difficult to understand the solution, thus making maintenance expensive. A good design results elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.
- **Feedback:** One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs. Frequent contact with the customer makes the development effective.
- **Simplicity:** The main principle of the XP model is to develop a simple system that will work efficiently in the present time, rather than trying to build something that would take time and may never be used. It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.
- **Pair Programming:** XP encourages pair programming where two developers work together at the same workstation. This approach helps in knowledge sharing, reduces errors, and improves code quality.

- **Continuous Integration:** In XP, developers integrate their code into a shared repository several times a day. This helps to detect and resolve integration issues early on in the development process.
- **Refactoring:** XP encourages refactoring, which is the process of restructuring existing code to make it more efficient and maintainable. Refactoring helps to keep the codebase clean, organized, and easy to understand.

- **Designing:** Without a proper design, a system implementation becomes too complex, and very difficult to understand the solution, thus making maintenance expensive. A good design results elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.
- **Feedback:** One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs. Frequent contact with the customer makes the development effective.
- **Simplicity:** The main principle of the XP model is to develop a simple system that will work efficiently in the present time, rather than trying to build something that would take time and may never be used. It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.
- **Pair Programming:** XP encourages pair programming where two developers work together at the same workstation. This approach helps in knowledge sharing, reduces errors, and improves code quality.
- **Continuous Integration:** In XP, developers integrate their code into a shared repository several times a day. This helps to detect and resolve integration issues early on in the development process.
- **Refactoring:** XP encourages refactoring, which is the process of restructuring existing code to make it more efficient and maintainable. Refactoring helps to keep the codebase clean, organized, and easy to understand.
- **Collective Code Ownership:** In XP, there is no individual ownership of code. Instead, the entire team is responsible for the codebase. This approach ensures that all team members have a sense of ownership and responsibility towards the code.
- **Planning Game:** XP follows a planning game, where the customer and the development team collaborate to prioritize and plan development tasks. This approach helps to ensure that the team is working on the most important features and delivers value to the customer.
- **On-site Customer:** XP requires an on-site customer who works closely with the development team throughout the project. This approach helps to ensure that the customer's needs are understood and met, and also facilitates communication and feedback.

Activity 4

Risk Management

What is Risk Management?

Risk Management is a systematic process of recognizing, evaluating, and handling threats or risks that have an effect on the finances, capital, and overall operations of an organization. These risks can come from different areas, such as financial instability, legal issues, errors in strategic planning, accidents, and natural disasters.

Why is risk management important?

Risk management is important because it helps organizations to prepare for unexpected circumstances that can vary from small issues to major crises. By actively understanding, evaluating, and planning for potential risks, organizations can protect their financial health, continued operation, and overall survival.

Let’s Understand why risk management important with an example.

Suppose In a software development project, one of the key developers unexpectedly falls ill and is unable to contribute to the product for an extended period.

One of the solution that organization may have , The team uses collaborative tools and procedures, such as shared work boards or project management software, to make sure that each member of the team is aware of all tasks and responsibilities, including those of their teammates.

An organization must focus on providing resources to minimize the negative effects of possible events and maximize positive results in order to reduce risk effectively. Organizations can more effectively identify, assess, and mitigate major risks by implementing a consistent, systematic, and integrated approach to risk management.

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
Risk Identification
Risk Assessment
Risk Monitoring
Risk Planning

Risk Management Process

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.

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.

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.

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.

Understanding Risks in Software Projects

A computer code project may be laid low with an outsized sort of risk. To be ready to consistently establish the necessary risks that could affect a computer code project, it's necessary to group risks into completely different categories. The project manager will then examine the risks from every category square measure relevant to the project.

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.

Classification of Risk in a project

Example: Let us consider a satellite-based mobile communication project. The project manager can identify many risks in this project. Let us classify them appropriately.

- What if the project cost escalates and overshoots what was estimated? – **Project Risk**
- What if the mobile phones that are developed become too bulky to conveniently carry? **Business Risk**
- What if call hand-off between satellites becomes too difficult to implement? **Technical Risk**

Risk management standards and frameworks

Risk management standards and frameworks give organizations guidelines on how to find, evaluate, and handle risks effectively. They provide a structured way to manage risks, making sure that everyone follows consistent and reliable practices. Here are some well-known risk management standards and frameworks:

1. COSO ERM Framework:

COSO ERM Framework was introduce in 2004 and updated in 2017. Its main purpose is to addresses the growing complexity of Enterprise Risk Management (ERM).

- **Key Features**:
  - 20 principles grouped into five components: Governance and culture, Strategy and objective-setting, Performance, Review and revision, Information, communication, and reporting.
  - It promote integrating risk into business strategies and operations.

2. **ISO 31000**:

ISO 31000 was introduce in 2009, revised in 2018. It provides principles and a framework for ERM.

- **Key Features**:
  - It offers guidance on applying risk management to operations.
  - It focuses on identifying, evaluating, and mitigating risks.
  - It promote senior management's role and integrating risk management across the organization.

3. **BS 31100**:

This framework is British Standard for Risk Management and latest version issued in 2001. It offers a structured approach to applying the principles outlined in ISO 31000:2018, covering tasks like identifying, evaluating, and addressing risks, followed by reporting and reviewing risk management efforts.

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.

Activity 5

5 stages of design thinking

What are the 5 Stages of the Design Thinking Process?

The five Stages of Design Thinking are as follows:

- **Empathy:** User's needs are researched.
- **Define:** User's Needs and Problems are defined.
- **Ideate:** Potential solutions are created based on problems.
- **Prototype:** A prototype based on the solutions are created.
- **Test:** Prototypes are tested with real Users.

5 Stages of f Design Thinking

Since design thinking is not a linear process, it is a flexible and adaptive approach. Therefore, these Stages are interconnected, and designers may move back and forth between them as new insights emerge or challenges arise.

Lets understand each Stage in detail.

Stage 1: Empathize

The first stage is Empathize, in this step, designers puts themselves in the user's world to gain a deep understanding of their needs, motivations, and pain points. This Stage involves various research methods, such as ethnographic studies, contextual inquiries, and user interviews.

By empathizing with the target audience, designers can challenge their own assumptions and uncover insights that may not be immediately apparent. This human centric approach ensures that the solutions developed are grounded in real world experiences and address user problems.

Stage 2: Define

Once a comprehensive understanding of the user has been established, the next Stage is Define. Here, designers use the insights that were gathered during the Empathize Stage and they articulate the problem statement or design challenge. This becomes important because it brings everybody in the design team on the same page when it comes it what problem they are solving.

Defining the problem is a crucial step, as it sets the foundation for the rest of the design process. A well defined problem statement helps to align the team, focus their efforts, and ensure that the solutions developed are relevant and impactful.

Stage 3: Ideate

The Ideation Stage is where creativity takes center stage. Now the designers have a deep understanding of the user and a clearly defined problem statement, because of this designers can easily engage in brainstorming sessions and collaborative ideation activities to generate a wide range of potential solutions.

During this Stage, designers are encouraged to think outside the box, suspend judgement, and explore unconventional ideas. Techniques such as mind mapping, storyboarding, and bodystorming can be used to bring up divergent thinking and innovative ideas.

Stage 4: Prototype

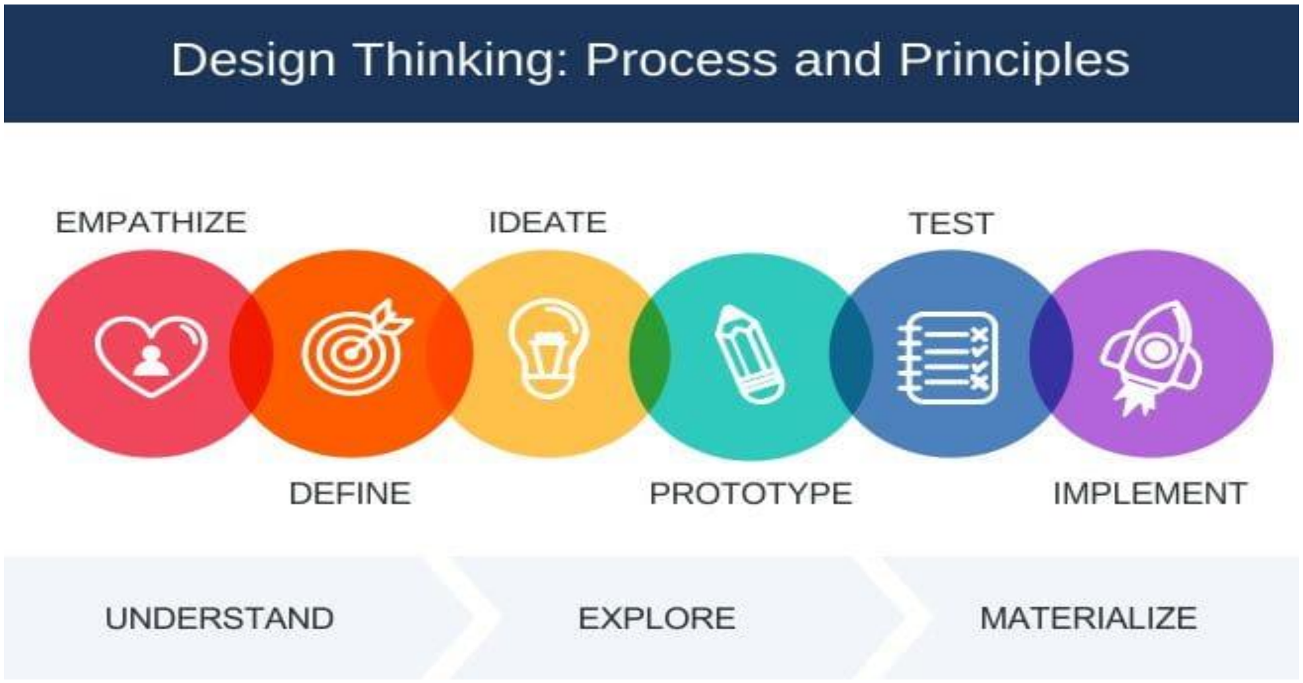
In the Prototyping Stage, the most promising ideas generated during the Ideate Stage are converted into testable prototypes. Prototyping can range from low fidelity sketches and wireframes to high fidelity interactive mockups, depending on the stage of the design process and the specific requirements.

The primary purpose of prototyping is to validate design concepts and gather feedback from users. By iterating on prototypes, designers can refine their solutions, identify potential issues, and ensure that the final product meets the user's needs and expectations.

Stage 5: Test

The final Stage of the Design Thinking Process is Test, where the prototypes are put to the test with real users. User testing sessions provide invaluable feedback, allowing designers to identify usability issues, uncover missed opportunities, and gather insights for further improvement.

Testing can take various forms, from moderated usability studies to remote user testing platforms, depending on the project requirements and resources available. The key is to gather actionable feedback that can be used to refine and optimize the design solution.



Best Practices

Following are some best practices to following while implementing the 5 stages of Design Thinking:

- **Empathy:** Encourage your team to cultivate a deep understanding of the user. This can be done by using research methods, such as contextual inquiries, user interviews, and participatory design sessions.
- **Involving multiple teams:** Involve stakeholders from various parts of the organization, such as engineering, marketing, and business strategy, to ensure a holistic approach to problem solving.
- **Encourage Experimentation:** Create an environment that promotes divergent thinking, calculated risks, and iterative experimentation. Encourage your team to explore unconventional ideas and rapidly prototype and test their solutions.

Activity 6

UML Diagrams

Unified Modeling Language (UML) is a general-purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. UML is not a programming language, it is rather a visual language.

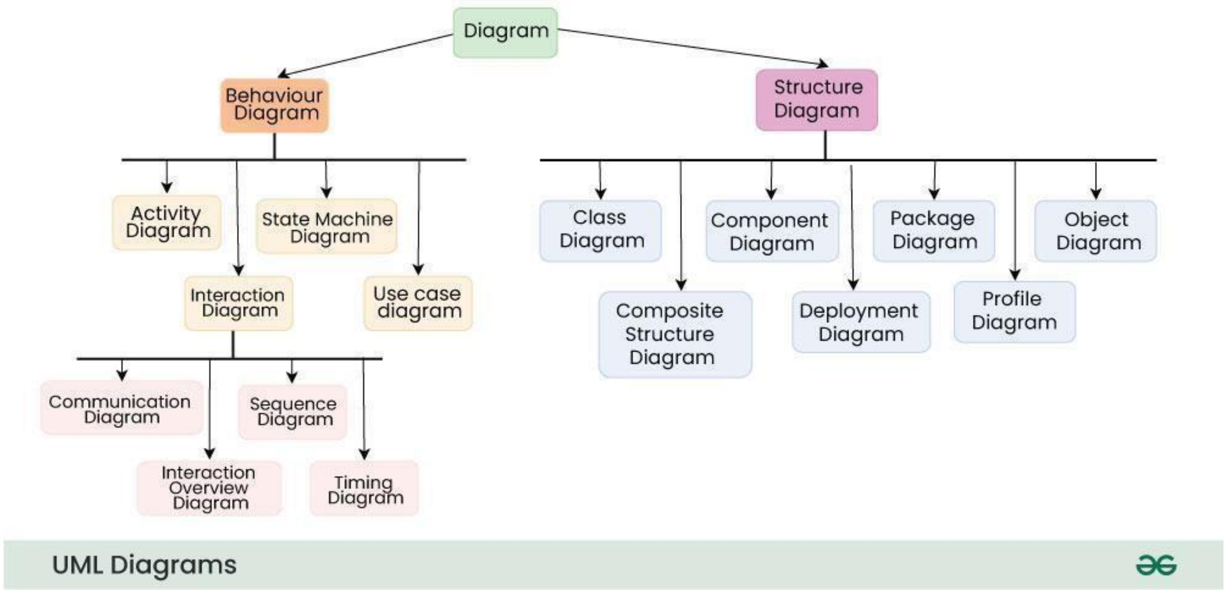
1. Why do we need UML?

We need UML (Unified Modeling Language) to visually represent and communicate complex system designs, facilitating better understanding and collaboration among stakeholders. Below is why we need UML:

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So UML becomes essential to communicate with non-programmers about essential requirements, functionalities, and processes of the system.
- A lot of time is saved down the line when teams can visualize processes, user interactions, and the static structure of the system.

2. Types of UML Diagrams

UML is linked with object-oriented design and analysis. UML makes use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:



4. Structural UML Diagrams

Structural UML diagrams are visual representations that depict the static aspects of a system, including its classes, objects, components, and their relationships, providing a clear view of the system's architecture. Structural UML diagrams include the following types:

4.1. Class Diagram

The most widely use UML diagram is the class diagram. It is the building block of all object oriented software systems. We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.

4.2. Composite Structure Diagram

We use composite structure diagrams to represent the internal structure of a class and its interaction points with other parts of the system.

- A composite structure diagram represents relationship between parts and their configuration which determine how the classifier (class, a component, or a deployment node) behaves.
- They represent internal structure of a structured classifier making the use of parts, ports, and connectors.
- We can also model collaborations using composite structure diagrams.
- They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.

4.3. Object Diagram

An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behaviour when objects have been instantiated, we are able to study the behaviour of the system at a particular instant.

- An object diagram is similar to a class diagram except it shows the instances of classes in the system.
- We depict actual classifiers and their relationships making the use of class diagrams.
- On the other hand, an Object Diagram represents specific instances of classes and relationships between them at a point of time.

4.4. Component Diagram

Component diagrams are used to represent how the physical components in a system have been organized. We use them for modelling implementation details.

- Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development.
- Component Diagrams become essential to use when we design and build complex systems.
- Interfaces are used by components of the system to communicate with each other.

4.5. **Deployment Diagram**

Deployment Diagrams are used to represent system hardware and its software. It tells us what hardware components exist and what software components run on them.

- We illustrate system architecture as distribution of software artifacts over distributed targets.
- An artifact is the information that is generated by system software.
- They are primarily used when a software is being used, distributed or deployed over multiple machines with different configurations.

4.6. **Package Diagram**

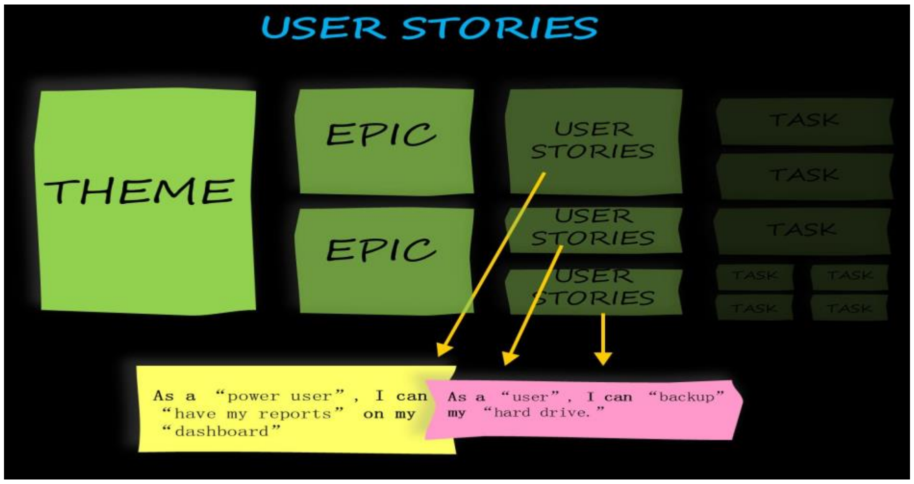
We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages.

- Packages help us to organise UML diagrams into meaningful groups and make the diagram easy to understand.
- They are primarily used to organise class and use case diagrams.

Activity 7

User stories

User stories are a key component of agile software development. They are short, simple descriptions of a feature or functionality from the perspective of a user. User stories are used to capture requirements in an agile project and help the development team understand the needs and expectations of the users.


USER STORIES
THEME
EPIC
EPIC
USER STORIES
USER STORIES
USER STORIES
TASK
TASK
TASK
TASK
TASK
TASK
TASK
As a "power user", I can "have my reports" on my "dashboard"
As a "user", I can "backup" my "hard drive."

Here are some key characteristics of user stories:

1. User-centric: User stories focus on the needs of the user and what they want to achieve with the software.
2. Simple: User stories are short and simple descriptions of a feature or functionality.
3. Independent: User stories can stand on their own and do not rely on other user stories.
4. Negotiable: User stories are open to discussion and can be refined and modified based on feedback from stakeholders.
5. Valuable: User stories provide value to the user and the business.
6. Estimable: User stories can be estimated in terms of time and effort required for implementation.
7. Testable: User stories can be tested to ensure they meet the needs of the user.
8. Prioritized: User stories are prioritized based on their importance to the user and the business goals.
9. Iterative: User stories are developed iteratively, allowing for feedback and changes

throughout the development process.

10. Consistent: User stories follow a consistent format, making them easy to understand and work with.
11. Contextual: User stories are written in a way that provides context to the development team, helping them understand the user's needs and goals.
12. Acceptance criteria: User stories have clear and specific acceptance criteria that define when the story is considered "done" and ready for release.
13. Role-based: User stories are written from the perspective of a specific user role, helping to ensure that the development team is building features that are relevant and useful to that user.
14. Traceable: User stories are tracked and linked to specific features and functionality in the software, making it easy to trace back to the original user need.

In agile software development, user stories are typically written on index cards or in a digital format, and are used to drive the development process. The development team uses user stories to plan and prioritize work, estimate the effort required for implementation, and track progress towards completing the user stories.

By using user stories in agile software development, teams can ensure that they are building software that meets the needs of the users and delivers value to the business.

In Agile software development and product management User Story refers to a short, informal, and simple description of software features that are required by the end-users in the software system. Its main purpose is to provide software features that will add value to the customer requirements. User stories are considered an important tool in Incremental software development. Mainly a user story defines the type of user, their need, and why they need that. So in simple, a user story is a simple description of requirements that needs to be implemented in the software system.

Pattern of User Story:

User stories are completely from the end-user perspective which follows the Role-Feature-Benefit pattern.

As a [type of user], I want [an action], so that [some reason]

For example :
As the project manager of a construction team, I want our team-messaging app to include file sharing and information update so that my team can collaborate and communicate with each other in real-time as a result the construction project development and completion will be fast.

Writing User Stories :

User stories are from a user perspective. So when user stories are written, users are given more importance during the process. Some points outlined which are taken into consideration during writing user stories like

1. Requirements
2. Tasks and their subtasks
3. Actual user
4. Importance to user words/feedback
5. Breaking user stories for larger requirements

With this also some other principles which are given importance during creating user stories are discussed below.

3 C's in User Stories :

1. **Card –**
   Write stories on cards, prioritize, estimate and schedule it accordingly.
2. **Conversation –**
   Conduct conversations, Specify the requirements and bring clarity.
3. **Confirmation –**
   Meet the acceptance criteria of the software.

Working with User Stories :

Below points represents working with user stories

1. Once the user story is fully written then it undergoes review and verification.
2. In project workflow meetings it is reviewed and verified then added to the actual workflow.
3. Actual requirements and functionality are decided based on the stories.
4. User stories are scored based on their complexity and the team starts work based on user stories.

Importance of creating User stories :

1. Stories clear idea about requirements
2. Makes it easy to understand the features
3. Delivers higher customer satisfaction

4. Fasten development process
5. Creates an effective work environment
6. Enables collaboration between teams
7. Delivery of valuable softwa

Activity 8

Monolithic & Microservices Architecture

In software development, how you structure your application can have a big impact on how it works and how easy it is to manage. Two common ways to structure software are called monolithic and microservices architectures. In this article, we'll explore the differences between these two approaches and when you might choose one over the other.

What is a Monolithic Architecture**?**

Software is traditionally designed using a monolithic architecture, in which the entire program is constructed as a single, indivisible unit. Every component of the program, including the data access layer, business logic, and user interface, is deployed and integrated tightly together in this design.

- This means that any changes or updates to the application require modifying and redeploying the entire monolith.
- Monolithic architectures are often characterized by their simplicity and ease of development, especially for small to medium-sized applications.
- However, they can become complex and difficult to maintain as the size and complexity of the application grow

What is a Microservices Architecture?

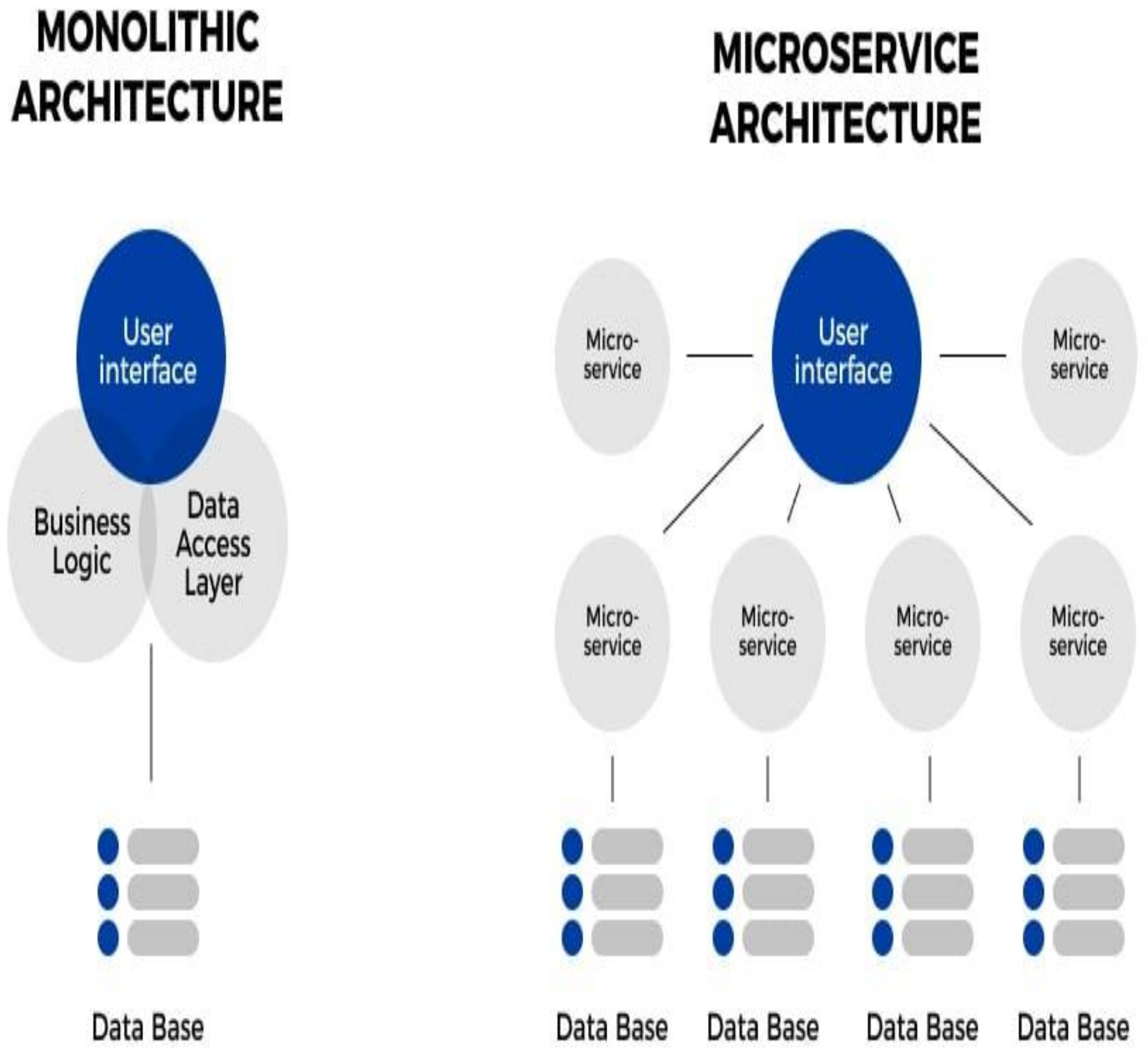
A microservices architecture results in an application designed as a set of small, independent services. Each one represents a business capability in itself. The services loosely couple with one another and communicate over the network, typically making use of lightweight protocols such as HTTP or messaging queues.

- Each service is responsible for a single functionality or feature of the application and can be developed, deployed, and scaled independently.
- The Microservice architecture has a significant impact on the relationship between the application and the database.

Differences between Monolithic and Microservices Architecture

Below are the differences the Monolithic and Microservice architecture:

Aspect	**Monolithic Architecture**	**Microservice Architecture**
Architecture	Single-tier architecture	Multi-tier architecture
Size	Large, all components tightly coupled	Small, loosely coupled components
Deployment	Deployed as a single unit	Individual services can be deployed independently
Scalability	Horizontal scaling can be challenging	Easier to scale horizontally
Development	Development is simpler initially	Complex due to managing multiple services
Technology	Limited technology choices	Freedom to choose the best technology for each service
Fault Tolerance	Entire application may fail if a part fails	Individual services can fail without affecting others
Maintenance	Easier to maintain due to its simplicity	Requires more effort to manage multiple services
Flexibility	Less flexible as all components are tightly coupled	More flexible as components can be developed, deployed, and scaled independently
Communication	Communication between components is faster	Communication may be slower due to network calls


MONOLITHIC ARCHITECTURE
User interface
Business Logic
Data Access Layer
Data Base
MICROSERVICE ARCHITECTURE
User interface
Micro-service
Micro-service
Micro-service
Micro-service
Micro-service
Micro-service
Data Base
Data Base
Data Base
Data Base

Activity 9

DevOps

As the technology landscape continues to shift, the demand for quicker and more reliable software delivery becomes even more imminent. Enter DevOps: an innovation designed to bridge the gap between software development and IT operations. What is DevOps, and why is this term being tossed around so excitedly in IT?

What is DevOps?

DevOps is a transformative culture and practice that unites software development (Dev) and IT operations (Ops) teams. By fostering collaboration and leveraging automation technologies, DevOps enables faster, more reliable code deployment to production in an efficient and repeatable manner.

DevOps Model Defined

DevOps is a software development approach that emphasizes collaboration and communication between development (Dev) and operations (Ops) teams. It aims to shorten the software development lifecycle and improve the quality and reliability of software releases.

Delivery Pipeline

The pipeline represents the different stages that software goes through before it is released to production. These stages might typically include:

- **Build:** The stage where the software code is compiled and packaged into a deployable unit.
- **Test:** The stage where the software is rigorously tested to ensure it functions as expected and identifies any bugs.
- **Release:** The stage where the software is deployed to production for end users.

Feedback Loop

The loop indicates that information and learnings from the production environment are fed back into the earlier stages of the pipeline. This feedback can be used to improve the software development process and future releases.

How DevOps Works?

DevOps will remove the “siloed” conditions between the development team and operations team. In many cases these two teams will work together for the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

Teams in charge of security and quality assurance may also integrate more closely with development and operations over the course of an application's lifecycle under various DevOps models. DevSecOps is the term used when security is a top priority for all members of a DevOps team.

These teams employ procedures to automate labor-intensive, manual processes that were slow in the past. They employ a technological stack and tooling that facilitate the swift and dependable operation and evolution of apps. A team's velocity is further increased by these technologies, which also assist engineers in independently completing activities (such provisioning infrastructure or delivering code) that ordinarily would have needed assistance from other teams.

Why DevOps Matters?

The world has undergone a massive transformation thanks to software and the Internet. It's not just about businesses using software as a tool anymore; it's about software being at the core of everything they do. Whether it's interacting with customers through online platforms or optimizing internal processes like logistics and operations, software is the driving force behind it all. Just as companies in the past revolutionized manufacturing with automation, today's companies need to revolutionize how they create and deliver software to stay competitive.

How to Adopt a DevOps Model?

1. DevOps Cultural Philosophy

Transitioning to DevOps means changing how people work together. Basically, DevOps is about breaking down the walls between two different groups: developers and operations. Sometimes, these groups even become one. In DevOps, they work together to make developers better at their jobs and operations more reliable. They focus on talking a lot, making processes better, and giving customers better service. They take full responsibility for what they do, often doing more than their usual jobs to help customers. This often means working closely with quality assurance and security teams. In companies that embrace DevOps, they see the whole process of making software and keeping it running as their job, no matter what their job titles are.

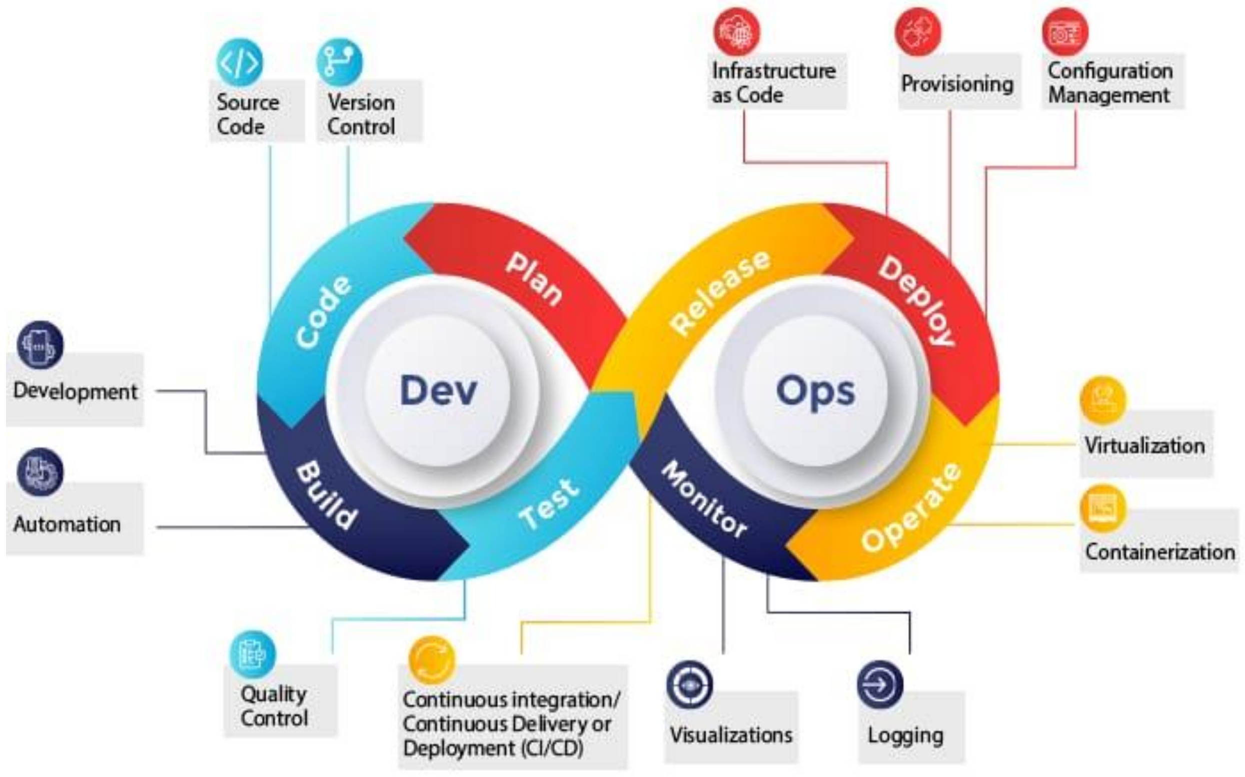
2. DevOps Practices Explained

DevOps enhances software development and IT operations through automation and efficient processes. Key practices include frequent, small updates that reduce deployment risks and allow quick bug fixes, and using a microservices architecture to increase flexibility. Continuous Integration and Continuous Delivery (CI/CD) automate testing and deployment, ensuring reliable updates. Infrastructure automation tools and continuous monitoring keep systems responsive and maintain performance. These practices enable faster, more reliable updates, driving innovation and customer satisfaction.

DevOps Life Cycle

DevOps is a practice that enables a single team to handle the whole application lifecycle, including development, testing, release, deployment, operation, display, and planning. It is a mix of the terms “Dev” (for development) and “Ops” (for operations). We can speed up the delivery of applications and services by a business with the aid of DevOps. Amazon, Netflix, and other businesses have all effectively embraced DevOps to improve their customer experience.

DevOps Lifecycle is the set of phases that includes DevOps for taking part in Development and Operation group duties for quicker software program delivery. DevOps follows positive techniques that consist of **code, building, testing, releasing, deploying, operating, displaying, and planning. DevOps lifecycle** follows a range of phases such as non-stop development, non-stop integration, non-stop testing, non-stop monitoring, and non-stop feedback. Each segment of the DevOps lifecycle is related to some equipment and applied sciences to obtain the process. Some of the frequently used tools are open source and are carried out primarily based on commercial enterprise requirements. DevOps lifecycle is effortless to manipulate and it helps satisfactory delivery.



7 Cs of DevOps

1. Continuous Development
2. Continuous Integration
3. Continuous Testing
4. Continuous Deployment/Continuous Delivery
5. Continuous Monitoring
6. Continuous Feedback
7. Continuous Operations

Activity 10

Tools to check code quality

Code Analysis Tools

Now that we've drilled down into the world of code analysis tools and understand their importance, let's dive into the top 10 Best Code Analysis Tools. These tools will equip you to write secure, efficient, and maintainable software, streamlining your development workflow.

1. SonarQube

SonarQube is one of the **most widely-used and robust code analysis tools** available today. It is an **open-source platform** for continuous code analysis, detection, and eradication of bugs, security issues, and code smells through performing automatic code reviews with static code analysis. SonarQube can support more than **20 languages**, including **Java, C#, JavaScript, Python, and many others.**

Key Features:

- **Multilanguage support:** Supports more than 20 programming languages.
- **Real-time feedback:** The developer gets immediate feedback, ensuring that code quality is maintained throughout the development.
- **Security-focused analysis:** Scan code for potential security vulnerabilities.
- **Reporting**: Provides detailed reports with actionable recommendations.

Advantages:

- **Continuous Integration:** Can be integrated into a CI/CD pipeline via Jenkins, GitHub, GitLab, etc., thus enabling analysis automatically on every build.
- **Customizable rules**: Customize everything in these SonarQube-related rules and coding standards according to the needs and requirements of the projects.
- **Easy-to-understand reports**: In-depth insight into the problems and makes recommendations on how to solve them.

2. Fortify Static Code Analyzer

Fortify by Micro Focus is a **static code analysis tool** focused on security. The tool provides centralized software security management, which also identifies security vulnerabilities in less time and gives the user the authority to fix them.

Key Features:

- **Advanced Static Analysis** - Advanced static analysis, which detects vulnerabilities via clumsy algorithms.
- **Integration Capabilities** - Excellent integration capabilities, as it is able to integrate with CI/CD pipelines and IDEs.
- **Security Rules** - It takes care of specific security requirements with the support of customizable rules.

Advantages:

- **Scalable** - Handles large code bases with good speed and scalability.
- **Actionable Insights** - Helps to eradicate issues with the help of detailed and actionable insights.
- **Security Coverage** - Provides good security coverage along with compliance with various regulatory standards.

3. Checkmarx

Checkmarx is **another powerful static analysis tool** that focuses on security and vulnerability detection. It scans source code for flaws that could lead to security breaches and helps ensure that software is secure by design. A private American company founded in 2006, **checkmarx** provides AST (Application Security Testing) solutions to ensure proper security in every aspect of software development.

Key Features:

- **Deep security scanning**: Detects almost all of the security vulnerabilities with good security rescue identification.
- **Remediation guidance**: Identifies vulnerabilities and also provides detailed guidance on how to fix them.
- **Scalable:** It is highly scalable, along with developer training tools that cater to secure coding practices.

Advantages:

- **Fast Integration** - Development environment with good integration with CI/CD pipelines.
- **Comprehensive security analysis**: Pprovides deep insights into your code’s security posture.
- **Real-time feedback**: Provides real-time feedback on potential security vulnerabilities during development.

4. ESLint

ESLint is a static code analysis tool that helps in **detecting errors in Javascript code**. It provides excellent and seamless features to its users. This **open-source JavaScript error detection tool** is widely used among programmers.

Key Features:

- **Improves code quality**: Ensure there are no problem patterns in **JavaScript code.**
- **Fix suggestions**: Automatically fix certain issues like formatting errors, unused variables, and missing semicolons..
- **ES6+ Support:** ES6+ support to ensure compliance with modern **JavaScript features.**

Advantages:

- **Customizable rules:** Allows customizable rules, which enhances flexibility.
- **Real-time feedback**: Immediate feedback as developers write code, helping catch issues early.
- **Plugin support**: ESLint provides an array of plugins along with shareable configurations.

5. Coverity

Coverity is a **static code analysis tool by Synopsys** that scans code for defects, security vulnerabilities, and compliance issues. It provides precise, actionable feedback and helps maintain high-quality, secure code.

Key Features:

- **Advanced defect detection**: Coverity identifies a wide range of defects, from simple bugs to complex security vulnerabilities.
- **Detailed Analysis:** Provides a comprehensive analysis of the entire code base along with actionable insights.
- **Fast analysis**: Provides quick analysis of large codebases, making it ideal for continuous integration.

Advantages:

- **Integration**: Comprehensive integration with various work environments and development tools.
- **High Accuracy:** High accuracy in detecting any kind of vulnerability.
- **Scalable**: Highly scalable and handles large code bases proficiently.

Activity 11

Need of testing

Software checking out serves as a critical approach for confirming whether or not a software program product aligns with the expected requirements and ensuring it is free from defects. It entails subjecting software program/system additives to rigorous exams and using both Manual or Automation gear to assess diverse hobby elements.

Reasons why Software Testing is really important?
Here are the important thing reasons why software trying out holds the most importance:

1. Saves Money

- Software testing offers numerous advantages, with cost-effectiveness being a key factor that attracts companies to opt for testing services.
- Software testing involves various stages and projects.
- Discovering bugs early on means they can be fixed at a lower cost.

2. Security

- It is every other essential factor why software programs trying out need to not be taken into consideration.
- It is considered to be the most susceptible and touchy component. There are a group of conditions in which the statistics and information of the users are stolen and they are used for the benefits.
- It is considered to be the motive why humans look for nicely tested and reliable products.
- elected product undergoes testing, and the consumer can be ensured that they are going to receive a dependable product.
- The non-public information of the user may be secure. Users can receive merchandise that might be free from vulnerability with the aid of software checking out.

3. Quality of the product

- To make sure that the unique product involves life, it needs to work by the following.
- Following the desires of the product is a prerequisite as it is beneficial in getting the prerequisite consequences.
- Products have to be serving the user in a single way or the opposite. It is a must that it's far going to bring the price, as according to the promise.
- Hence, it should be characteristic in a whole manner for ensuring a powerful purchaser enjoys. It is also important to check the compatibility of the device.
- For instance, in case, you are making plans to launch a utility, it is a need to test the compatibility of the identical in a big selection of operating structures and devices.

4. Satisfaction of the consumer

- The primary goal of the proprietor of the products is to provide the pleasant pride of the customers.
- The reasons why it's far necessary to opt for software testing is because it offers the prerequisite and perfect consumer revel.
- As you choose the excellent venture inside the saturated assignment, you may be able to incomes the reputation of dependable customers.
- Thus, you'll acquire lengthy-term blessings by using opting for software programs to try out. Earning the belief of the consumer is sincerely not a clean undertaking, mostly in case the product is determined to be functioning and glitching on every occasion or the opposite.
- You yourself have used several merchandise and also you without a doubt had numerous terrible studies thanks to that you might have deleted the utility.
- The market is without a doubt saturated within the present days.
- The first impression is sincerely essential and if you fail to give the identical, users are going to locate some other product for you to accomplish all of the necessities.

5. Enhancing the development technique

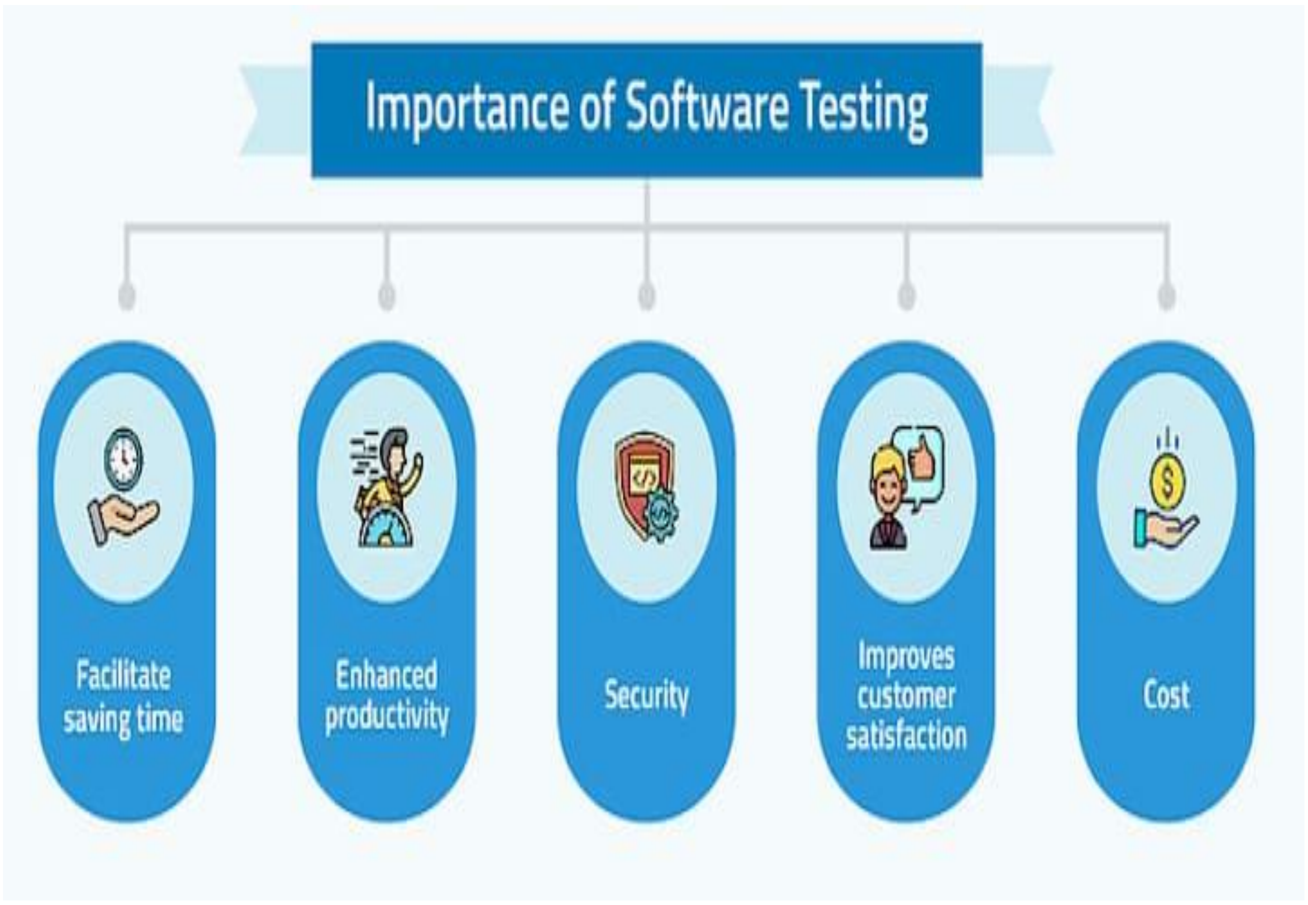
- With the aid of Quality Assurance, you could find a big selection of situations and mistakes, for the reproduction of the error.
- It is really simple and the builders want to repair the same very quickly. In addition to this, software testers need to be operating with the development team parallelly, which is beneficial in the acceleration of the improvement method.

6. Easy even as adding new capabilities

- The extra interconnected and older the code, the tougher it's far to exchange. Tests counteract this calcification tendency through allowing builders to confidently add new capabilities.
- As a new developer, changing older parts of your codebase may be terrifying, however with exams, you’ll as a minimum recognize if you’ve broken some thing essential.
- This enables in making your software program stand beforehand inside the marketplace, and beat the competition.

7. Determining the performance of the software program

- If you discover software program or software that has low or reduced overall performance, you'll find that it brings your recognition down in the marketplace.
- By prioritizing software trying out, businesses can supply dependable, stable, and amazing merchandise that meet person needs and exceed expectancies in brand new competitive market.


Importance of Software Testing
Facilitate saving time
Enhanced productivity
Security
Improves customer satisfaction
Cost

Activity 12
Software Measurement and Metrics

Software Measurement: 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.

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

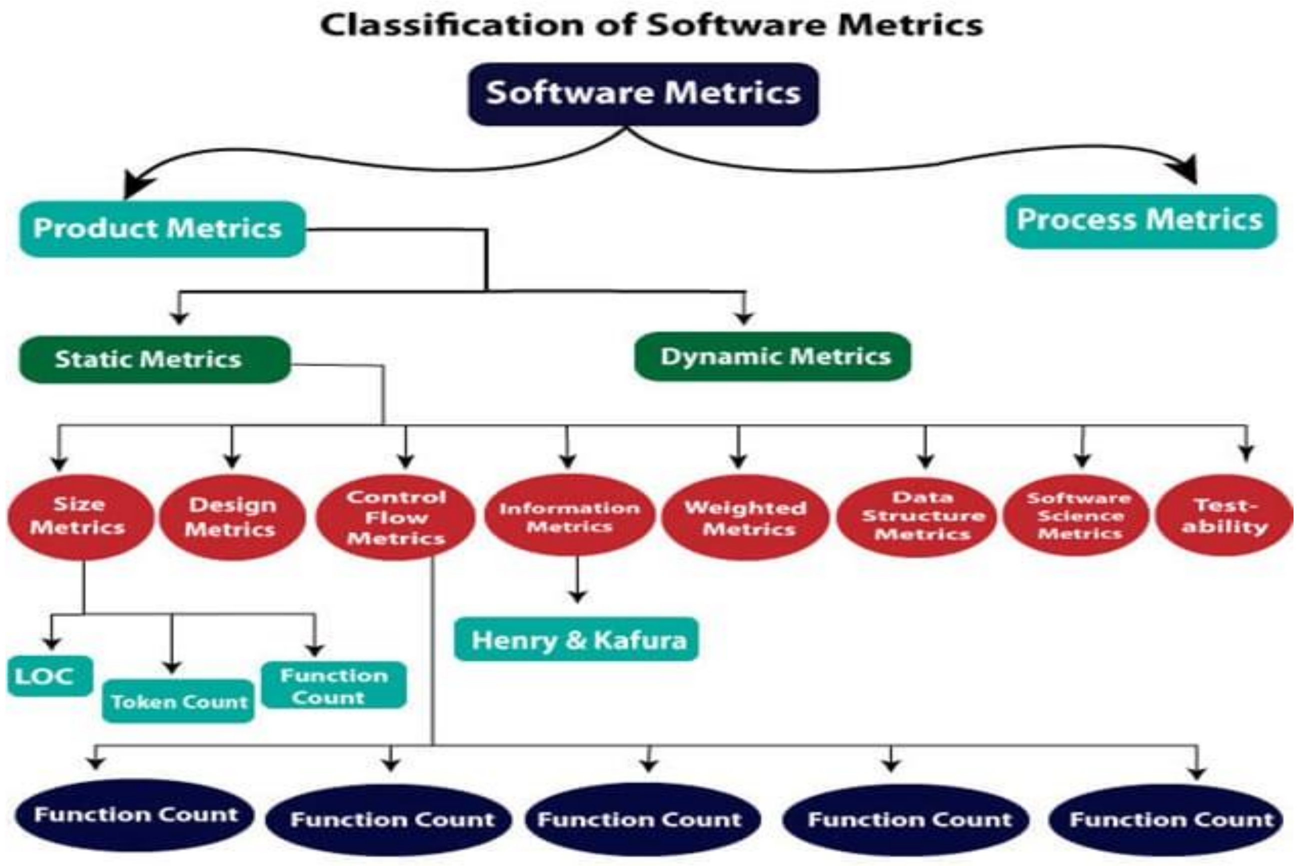
A metric is a measurement of the level at which any impute belongs to a system product or process.

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.

Activity 13
Software Quality

Traditionally, a high-quality product is outlined in terms of its fitness of purpose. That is, a high-quality product will specifically be what the users need to try. For code merchandise, the fitness of purpose is typically taken in terms of satisfaction of the wants arranged down within the SRS document

What is Software Quality?
Software Quality shows how good and reliable a product is. To convey an associate degree example, think about functionally correct software. It performs all functions as laid out in the SRS document. But, it has an associate degree virtually unusable program. even though it should be functionally correct, we tend not to think about it to be a high-quality product. Another example is also that of a product that will have everything that the users need but has an associate degree virtually incomprehensible and not maintainable code. Therefore, the normal construct of quality as “fitness of purpose” for code merchandise isn’t satisfactory.

Factors of Software Quality
The modern read of high-quality associates with software many quality factors like the following:

1. **Portability:** A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code merchandise, etc.
2. **Usability:** A software has smart usability if completely different classes of users (i.e. knowledgeable and novice users) will simply invoke the functions of the merchandise.
3. **Reusability:** A software has smart reusability if completely different modules of the merchandise will simply be reused to develop new merchandise.
4. **Correctness:** Software is correct if completely different needs as laid out in the SRS document are properly enforced.
5. **Maintainability:** A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to the merchandise, and therefore the functionalities of the merchandise may be simply changed, etc
6. **Reliability:** Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make. Designers can improve reliability by ensuring the software is easy to implement and change, by testing it thoroughly, and also by ensuring that if failures occur, the system can handle them or can recover easily.

7. **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth, and other resources. This is important to customers in order to reduce their costs of running the software, although with today's powerful computers, CPU time, memory and disk usage are less of a concern than in years gone by.

Software Quality Management System

Software Quality Management System contains the methods that are used by the authorities to develop products having the desired quality.

Some of the methods are:

- **Managerial Structure:** Quality System is responsible for managing the structure as a whole. Every Organization has a managerial structure.
- **Individual Responsibilities:** Each individual present in the organization must have some responsibilities that should be reviewed by the top management and each individual present in the system must take this seriously.
- **Quality System Activities:** The activities which each quality system must have been
    - Project Auditing.
    - Review of the quality system.
    - It helps in the development of methods and guidelines.

Evolution of Quality Management System

Quality Systems are basically evolved over the past some years. The evolution of a Quality Management System is a four-step process.

1. **Inspection:** Product inspection task provided an instrument for quality control (QC).
2. **Quality Control:** The main task of quality control is to detect defective devices, and it also helps in finding the cause that leads to the defect. It also helps in the correction of bugs.
3. **Quality Assurance:** Quality Assurance helps an organization in making good quality products. It also helps in improving the quality of the product by passing the products through security checks.
4. **Total Quality Management (TQM):** Total Quality Management(TQM) checks and assures that all the procedures must be continuously improved regularly through process measurements.

SOFTWARE
QUALITY