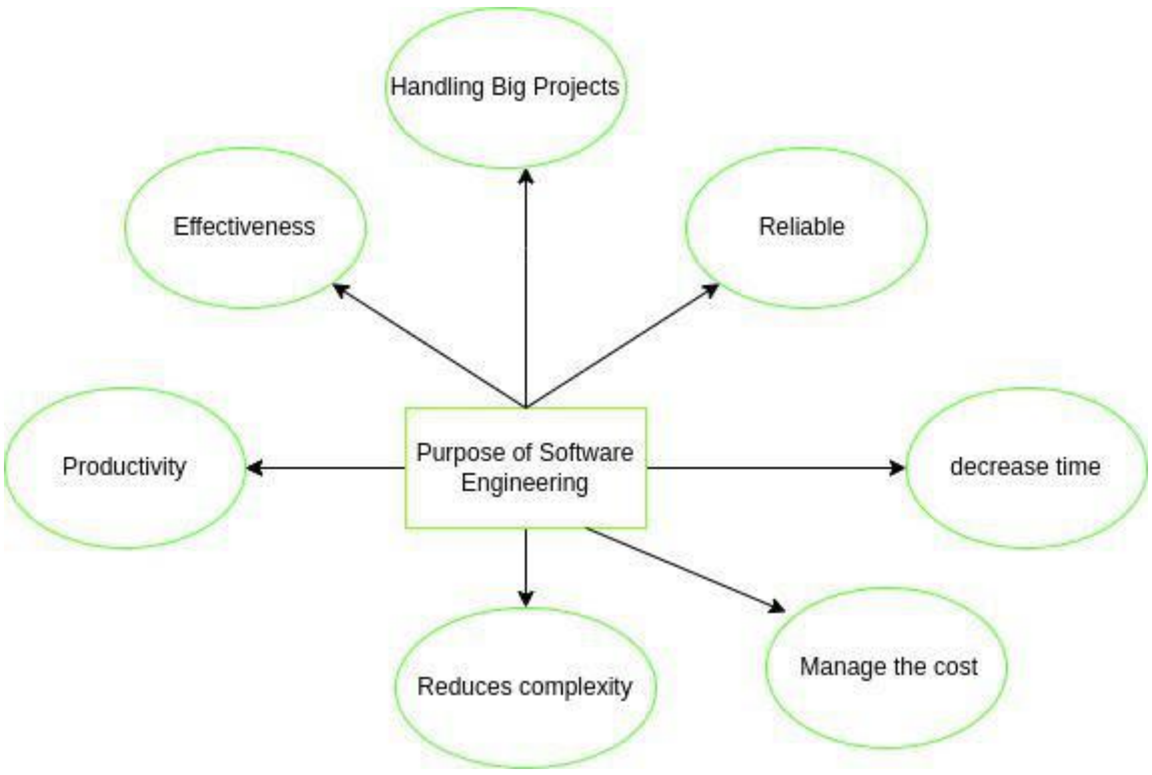
Unit 1 (Software Engineering and Models)

Software Engineering and it's importance

1: Software engineering is the process of designing, developing, testing, and maintaining software.
2: 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.
3: Additionally, software engineering helps to manage the costs, risks and schedule of the software development process.
4: It also provides a way to improve the software development process over time through testing and feedback.
5: It is important because it ensures the creation of reliable, efficient, and maintainable software.
6: It contributes to the overall quality of software systems, enhances scalability, efficiency, and customer satisfaction, while also addressing security concerns.



Evolution of software engineering

1: The evolution of software engineering from the **1960s** to the present reflects a dynamic response to the challenges of software development.

2: In the **1960s** and **1970s**, the field emerged in response to growing complexity, with the Waterfall model dominating early practices.

3: The **1980s** and **1990s** saw the rise of structured methodologies and the advent of Object-Oriented Programming (OOP).

4: The late **1990s** and **2000s** witnessed the Agile movement, emphasizing flexibility and iterative development, along with the integration of DevOps.

5: In the **2010s** and beyond, software engineering embraced continuous integration, cloud computing, and microservices architecture.

6: Current trends include the integration of AI and ML, the rise of low-code/no-code platforms, and an increased focus on security.

Phases In The Software Development

1: Requirement Phase:

- Identifying and documenting the software's functionalities based on user and business requirements.
- Gather information from stakeholders, define system requirements, and create specifications.

2. Design Phase:

- Create a blueprint for the software's architecture, structure, and user interface based on the requirements.
- Architectural design, detailed design of modules, and user interface design.

3. Implementation Phase:

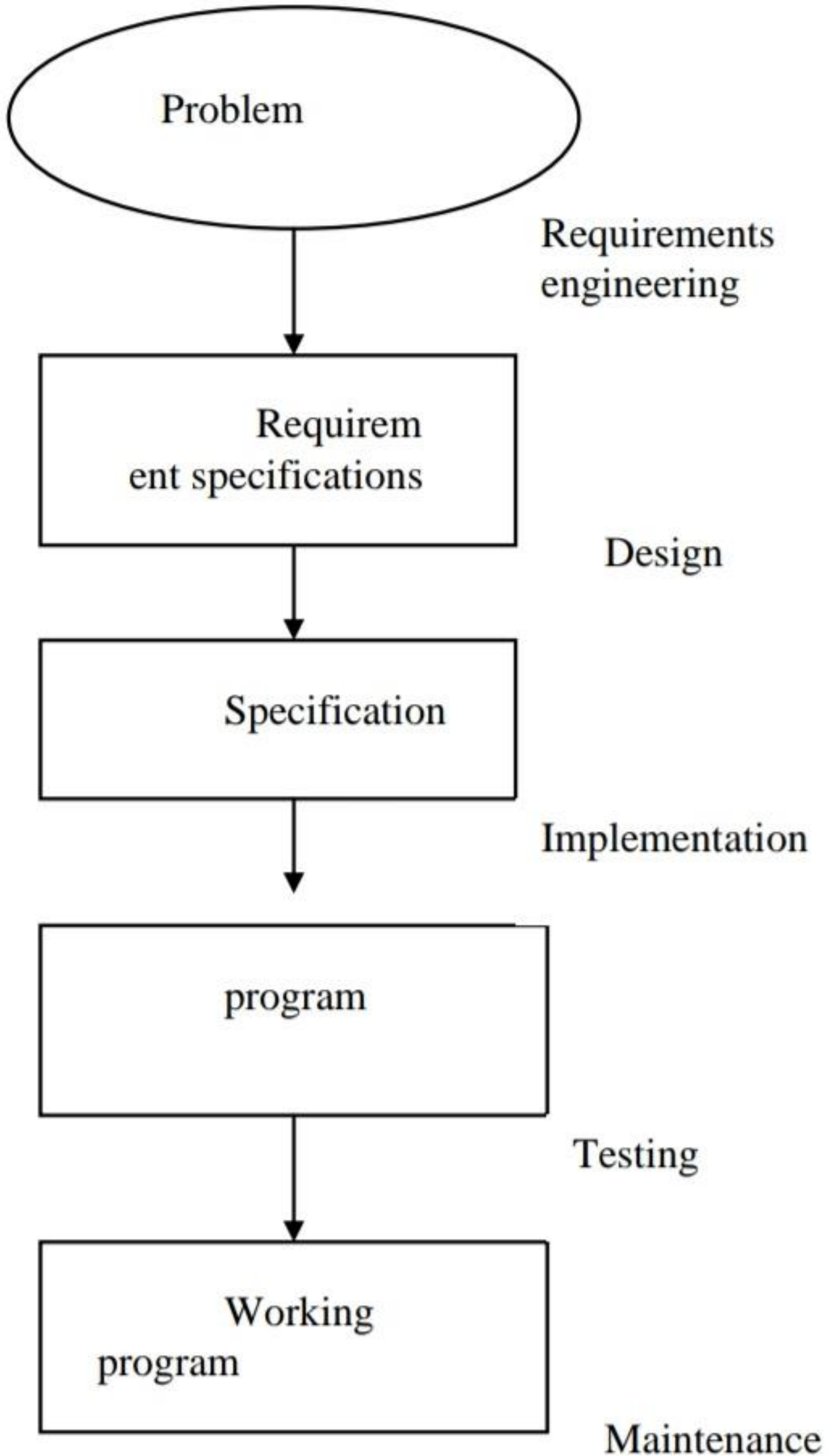
- Translate the design into actual code, building the software system.
- Writing, coding, and integrating the software components as per the design.

4. Testing Phase:

- Verify that the software behaves as intended and identify and fix any defects.
- Systematic testing, including unit testing, integration testing, and system testing.

5. Maintenance Phase:

- Sustain and enhance the software over time to adapt to changing requirements and fix issues.
- Bug fixing, updates, and making modifications as needed.


Problem
Requirements engineering
Requirem ent specifications
Design
Specification
Implementation
program
Testing
Working program
Maintenance

Types of software applications

1: System Software: It is a program designed to run a computer's hardware and applications and manage its resources, such as its memory, processors, and devices.

2: Business Software: software System that access large databases containing business information. These application helps in business operation and management decision-making.

3: Engineering and Scientific software: Computer aided design, system simulation other interactive applications have begun to take on real time and system software characteristics.

4: Embedded software: Embedded software resides in read only memory and used to control products and systems for the consumer and industrial market.

5: Web Based software: The web pages retrieved by a browser are software that incorporates executable instructions.

6: Artificial Intelligence software: Artificial Intelligence software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis.

Code of ethics

1: Software engineers shall act consistently with the public interest.

2: Software engineers shall behave in a manner that is in the best interests of their client (customer) and employer consistent with the public interest.

3: Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4: Software engineers shall maintain integrity and independence in their expert judgment.

5: Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6: Software engineers shall proceed the integrity and reputation of the profession consistent with the civic interest.

7: Software engineers shall be reasonable to and supportive of their colleagues.

8: Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Software Process

1: When you build a product or system it undergoes a series of steps/stages, a road map that helps you to create a timely, high quality product.

2: This road map which you follow to achieve this product or system is called as a software process.

3: The process provides interaction between Users and designers, Users and evolving tools, Designers and evolving tools.

4: Software Process gives stability and control to organization activity. **5:** Software process provides frame work for the development of high quality software.

6: A software process can be defined as a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (for example - project plans, design documents, code, test cases, and user manuals).

Characteristics of Software Process

1: Greater emphasis on systematic development.

2: Computer Aided for Software Engineering (CASE).

3: A concentration on finding out the user‘s requirements.

4: Formal specification of the requirements of a system.

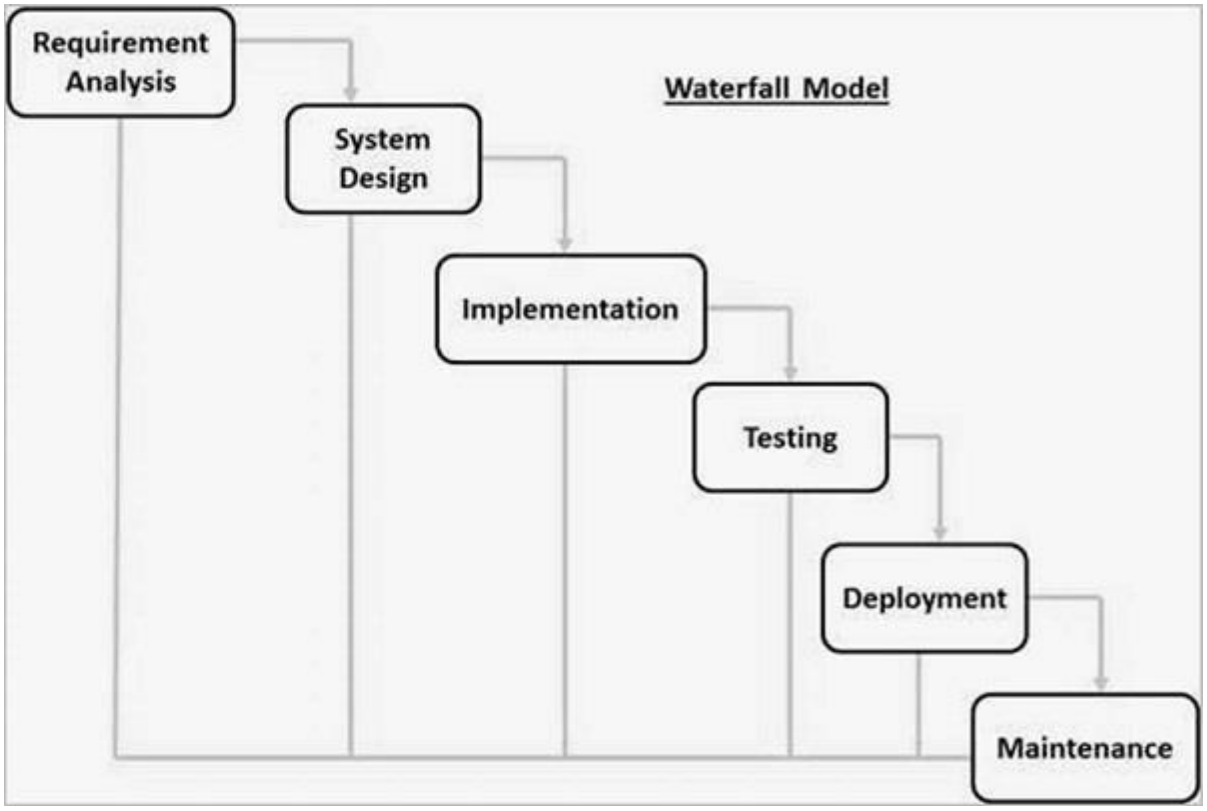
5: Demonstration of early version of a system (prototyping).

6: Greater emphases on trying to ensure error free code.

Software Process comes under Umbrella activity of Software Quality Assurance (SQA) and Software Configuration Management (SCM).

Waterfall model

This model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support.



The sequential phases in Waterfall model are −

1:Requirement Gathering and analysis − All possible requirements of the system to be developed are captured in this phase and documented.

2: System Design − The requirement specifications from first phase are studied in this phase and the system design is prepared.

3: Implementation − With inputs from the system design, the system is first developed in small programs called units.

4: Integration and Testing − All the units developed in the implementation phase are integrated into a system after testing of each unit.

5: Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

6: Maintenance − Patches are released to fix issues in the software. Also to enhance product, uct some better versions are released.

Advantages

1: Simple and easy to use.

2: Easy to manage due to the rigidity of the model.

3: Phases are processed and completed one at a time.

4: Works well for smaller projects where requirements are very well understood.

5: Process and results are well documented.

Disadvantages

1: No working software is produced until late during the life cycle.

2: High amounts of risk and uncertainty.

3: Poor model for complex and object-oriented projects.

4: Poor model for long and ongoing projects.

5: Poor model where requirements are at a moderate to high risk of changing.

Prototyping Model

1: This model is used when the customers do not know the exact project requirements beforehand.

2: In this model, a prototype of the end product is first developed, tested, and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

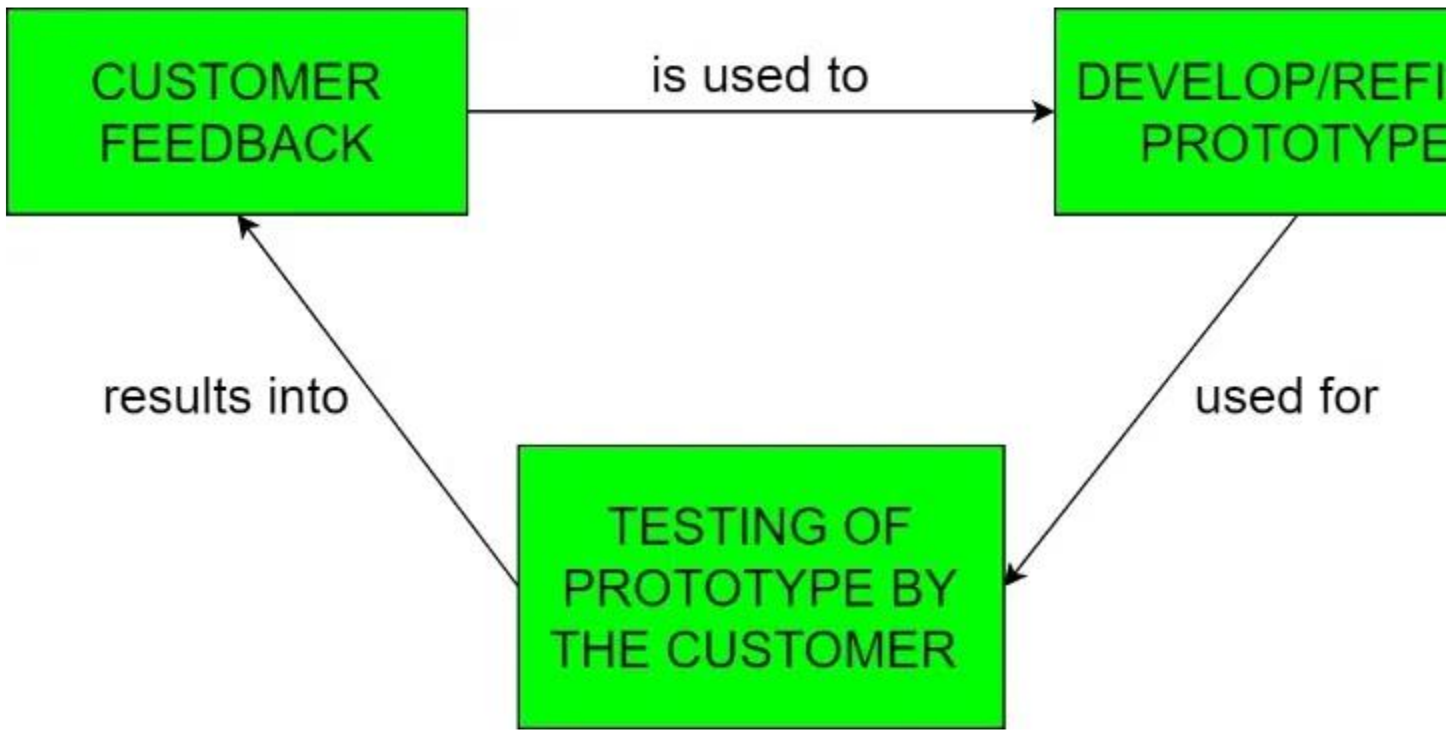
3: In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle.

4: The process starts by interviewing the customers and developing the incomplete high-level paper model.

5: This document is used to build the initial prototype supporting only the basic functionality as desired by the customer.

6: Once the customer figures out the problems, the prototype is further refined to eliminate them.

7: The process continues until the user approves the prototype and finds the working model to be satisfactory.



Advantages

1: The customers get to see the partial product early in the life cycle.
2: New requirements can be easily accommodated as there is scope for refinement.
3: Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
4: The developed prototype can be reused by the developer for more complicated projects in the future.
5: Prototyping can help bridge the gap between technical and non-technical stakeholders by providing a tangible representation of the product.

Disadvantages

1: Costly with respect to time as well as money.
2: Poor Documentation due to continuously changing customer requirements.
3: It is very difficult for developers to accommodate all the changes demanded by the customer.
4: The prototype may not accurately represent the final product due to limited functionality or incomplete features.
5: The prototype may not consider technical feasibility and scalability issues that can arise during the final product development.

Iterative Model

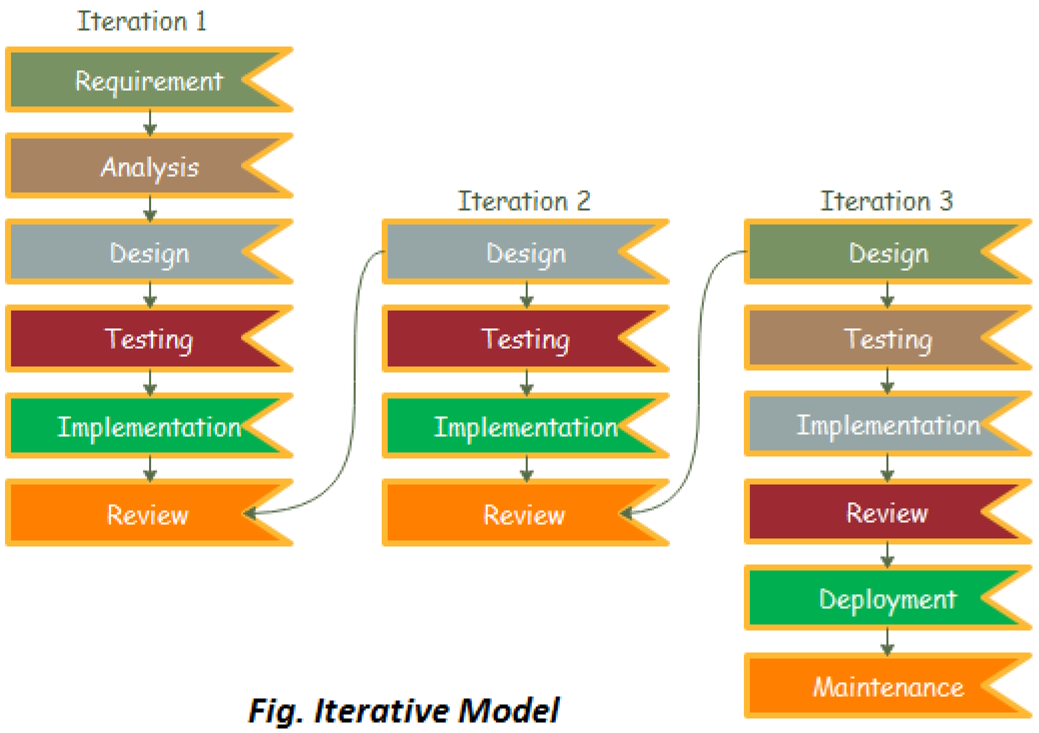


Fig. Iterative Model

The various phases of Iterative model are as follows:

1. Requirement gathering & analysis: requirements are gathered from customers and check by an analyst whether requirements will fulfil or not.

2. Design: The team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

3. Implementation: In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

4. Testing: After completing the coding phase, software testing starts using different test methods like, white box, black box, and grey box test methods.

5. Deployment: After completing all the phases, software is deployed to its work environment.

6. Review: review phase is performed to check the behaviour and validity of the developed product.

7. Maintenance: , after deployment of the software there may be some bugs, errors or new updates that are required. Maintenance involves debugging and new addition options.

Advantage(Pros) of Iterative Model:

1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing 4Cneeds of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

Disadvantage(Cons) of Iterative Model:

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

Spiral Model

The Spiral Model is a **Software Development Life Cycle (SDLC)** model that provides a systematic and iterative approach to software development.

In its diagrammatic representation, looks like a spiral with many loops.

The exact number of loops of the spiral is unknown and can vary from project to project.

Each loop of the spiral is called a **Phase of the** software development process.

1. Planning

The first phase of the Spiral Model is the planning phase, where the scope of the project is determined and a plan is created for the next iteration of the spiral.

2. Risk Analysis

In the risk analysis phase, the risks associated with the project are identified and evaluated.

3. Engineering

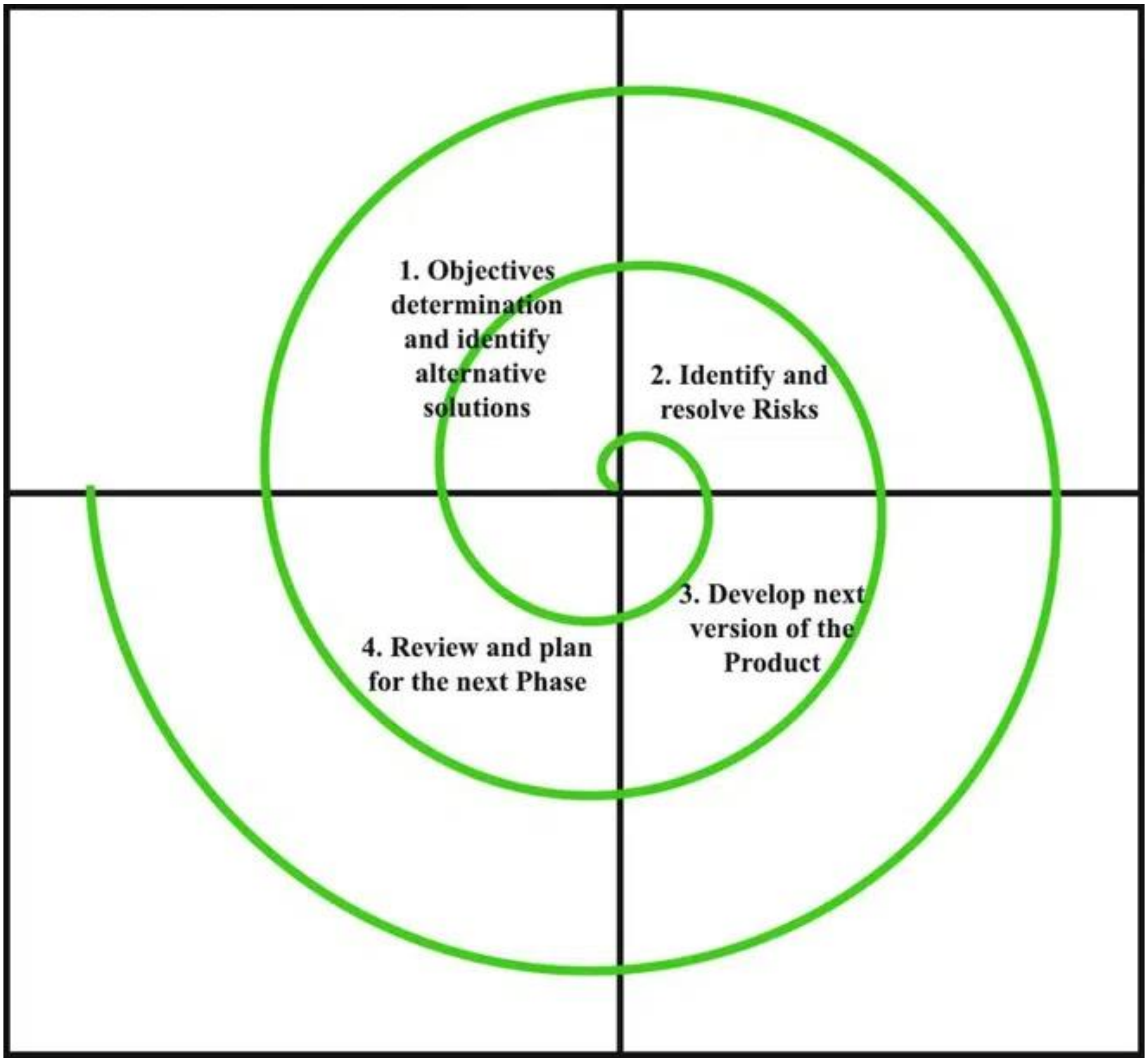
In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.

4. Evaluation

In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.

5. Planning

The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below:

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop the next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so-far developed version of the software. In the end, planning for the next phase is started.

Advantages

1. High amount of risk analysis.
2. Good for large and mission-critical projects.
3. Software is produced early in the software life cycle.
4. Customers can see the development of the product at the early phase of the software development.
5. improve communication between the customer and the development team.

Disadvantages

1. A costly model.
2. Risk analysis requires highly specific expertise.
3. Project‘s success is highly dependent on the risk analysis phase.
4. Doesn‘t work well for smaller projects.
5. The Spiral Model is much more complex than other SDLC models.

RAD model

The Rapid Application Development (RAD) model is a software development methodology that emphasizes prototyping and quick feedback over specific planning. The RAD model distributes the analysis, design, build, and test phases into a series of short, iterative development cycles.

Here's a brief explanation of each phase of Rapid Application Development (RAD):

1. Business Modeling: This phase involves understanding the business needs and requirements that the software application is intended to address.

2. Data Modeling: In this phase, the focus is on designing the data structure and relationships required for the application.

3. Process Modeling: Here, the emphasis is on defining the processes and workflows that the application will facilitate.

4. Application Generation: This phase involves actually building the software application using rapid development tools and techniques.

5. Testing and Turnover: Finally, this phase involves testing the application to ensure that it meets the specified requirements and functions correctly.

Coding standards

1. Coding standards are a set of rules and guidelines that developers follow when writing code.

2. They are also known as coding guidelines or programming style guides.

3. Coding standards help ensure consistency, readability, maintainability, and reliability in software development.

4. They can vary depending on the programming language or technology used and the organization or community involved.

5. Some examples of coding standards include:

- Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs.
- File structure: The file should conform to ISO standards for text files.
- Naming conventions: Small modifications may also be used as a coding standard in Java.
- Scoping conventions: How classes and functions should behave.
- Usage conventions: How code should be laid out.
- Compile errors and warnings: How to handle and compile errors.

Unit 2 (Requirement Analysis)

Need for SRS

1. The requirement specification is the end product of requirement engineering phase. Its purpose is to communicate results of requirement analysis to others.
2. It serves as an anchor point against which subsequent steps can be justified.
3. SRS is starting point for the design phase.
4. It serves as foundation for database engineering ,hardware engineering, software engineering and human engineering.
5. The specification joins each allocated system element.
6. A software requirements specification (SRS) is a complete description of the behavior of the system to be developed.
7. It includes a set of use cases that describe all of the interactions that the users will have with the software.
8. Use cases are also known as functional requirements.

Requirement gathering

Requirement gathering is a critical phase in software engineering where the needs and expectations of stakeholders are identified and documented. Here's an overview of the process:

1. Identify Stakeholders: Determine who the key stakeholders are, including clients, end-users, managers, and subject matter experts.

2. Conduct Interviews and Workshops: Engage with stakeholders through interviews, workshops, or surveys to understand their needs, preferences, and constraints.

3. Document Requirements: Capture requirements in a structured format, such as user stories, use cases, or functional and non-functional requirements.

4. Prioritize Requirements: Prioritize requirements based on their importance and impact on the project's success.

5. Validate Requirements: Verify that the documented requirements accurately reflect the stakeholders' needs and expectations.

Types of Requirements

1. Functional Requirements: These describe the specific behaviors or functions that the software system must perform. They define what the system should do.

2. Non-Functional Requirements: These specify the criteria that the system must meet regarding its quality attributes, such as performance, reliability, usability, and security. Non-functional requirements describe how the system should perform.

3. User Requirements: These represent the needs and expectations of end-users regarding the system's functionality and usability. User requirements focus on what users want to achieve with the software.

4. System Requirements: System requirements define the environment in which the software will operate.

5. Business Requirements: These outline the goals, objectives, and constraints of the organization or business that the software is being developed for.

Problem Analysis

1. Software engineers identify issues, discrepancies, or inefficiencies within a software system or project that need to be addressed.

2. Software engineers gather requirements from stakeholders to understand the desired functionality, performance expectations, and constraints of the software system.

3. Engineers analyze the problem to determine its root causes.

4. Engineers assess the impact of the problem on the software system.

5. Based on the analysis of the problem and its root causes, software engineers brainstorm and propose potential solutions to address the issue.

6.Engineers evaluate each proposed solution based on factors such as feasibility, effectiveness, scalability, maintainability, and cost.

7. Engineers make informed decisions about which solution(s) to pursue based on the analysis and evaluation conducted.

8. Once a solution is selected, software engineers develop a detailed plan for implementing it.

9. Engineers implement the chosen solution(s) and thoroughly test the software to ensure that the problem has been effectively addressed.

Prototyping Model

The prototyping model is a systems development method in which a prototype is built, tested and then reworked as necessary until an acceptable outcome is achieved from which the complete system or product can be developed.

the steps of the prototyping model are as follows:

1. The new system requirements are defined in as much detail as possible.

2. A preliminary, simple design is created for the new system.

3. The users thoroughly evaluate the first prototype and note its strengths and weaknesses, what needs to be added and what should be removed.

4. The first prototype is modified, based on the comments supplied by the users and a second prototype of the new system is constructed.

5. The second prototype is evaluated in the same manner as the first prototype.

6. The preceding steps are iterated as many times as necessary, until the users are satisfied that the prototype represents the final product desired.

Characteristics of SRS

Quality characteristics of a good Software Requirements Specification (SRS) document include:

1. Complete: The SRS should include all the requirements for the software system, including both functional and non-functional requirements.

2. Consistent: The SRS should be consistent in its use of terminology and formatting, and should be free of contradictions.

3.Unambiguous: The SRS should be clear and specific, and should avoid using vague or imprecise language.

4. Modifiable: The SRS should be modifiable, so that it can b.e updated and changed as the software development process progresses.

5. Testable: The SRS should be written in a way that allows the requirements to be tested and validated.

DFD

1. Data flow diagrams (DFDs) are visual representations that show the flow of data within a system.

2. They depict how data moves through processes, stores, and external entities.

3. DFDs use symbols like circles, rectangles, and arrows to represent entities, processes, data stores, and data flows.

4. They're commonly used in software engineering and system analysis to understand and communicate the structure and behavior of a system.

5. DFDs consist of four main components:

- External Entities: Represent sources or destinations of data outside the system.
- Processes: Activities or transformations that manipulate data flows.
- Data Stores: Repositories where data is stored for later use.
- Data Flows: Paths through which data moves between entities, processes, and stores.

ER Diagram

1. Entity-Relationship (ER) diagrams are visual representations used to model the structure of a database.

2. They depict entities as rectangles, attributes as ovals, and relationships as diamonds.

3. ER diagrams help visualize the relationships between entities in a database, such as one-to-one, one-to-many, or many-to-many relationships, aiding in database design and understanding.

Data Dictionary

1. A data dictionary is a collection of descriptions of the data objects or items in a data model to which programmers and others can refer.

2. Often, a data dictionary is a centralized metadata repository.

3. Data dictionaries sometimes play a role in data modeling, which creates a tangible diagram of object relationships that lists each object's name, assigned data values and defined relationships.

4. The type of data, such as text, image or binary value, is described; possible predefined default values are listed; and a brief textual description is provided.

5..This collection of information can be referenced through a data dictionary.

Unit 3 (Software Design)

Design importance in Software

1. Software design is one of the most critical activities in the software development process.
2. A well-designed system leads to software that meets both the functional and nonfunctional requirements.
3. Software design describes how the software system is decomposed and organized into components and modules.
4. It defines the relationship between these modules through interfaces.
5. A good design makes the software understandable, modifiable, reliable and reusable.
6. Software design translates the user's requirements into a 'blueprint' for building the software.
7. It is the link between problem space and solution space.
8. The design represents the software architects' understanding of how to meet the requirements.
9. Software development cannot begin without a preceding design phase.

Design Principles

1. The design process should not suffer from tunnel vision. A good designer should consider alternative approaches, judging each based on requirements of the problem, the resources available to do the job, and the design concepts.

2. The design should be traceable to the analysis model.

3. The design should not reinvent the wheel. Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited.

4. The design should exhibit uniformity and integration.

5. The design should be structured to accommodate change.

6. The design should be structured to degrade tenderly, even when aberrant data ,events or operating conditions are arises. Properly designed software should never explode.

These design principles are correctly applied, then the software engineer creates a design

that depicts both internal and external quality factors. External quality factors are speed, reliability, correctness usability. Internal quality factors are of importance to software engineers.

Coupling and Cohesion

1. Coupling refers to the degree of interdependence between software modules.

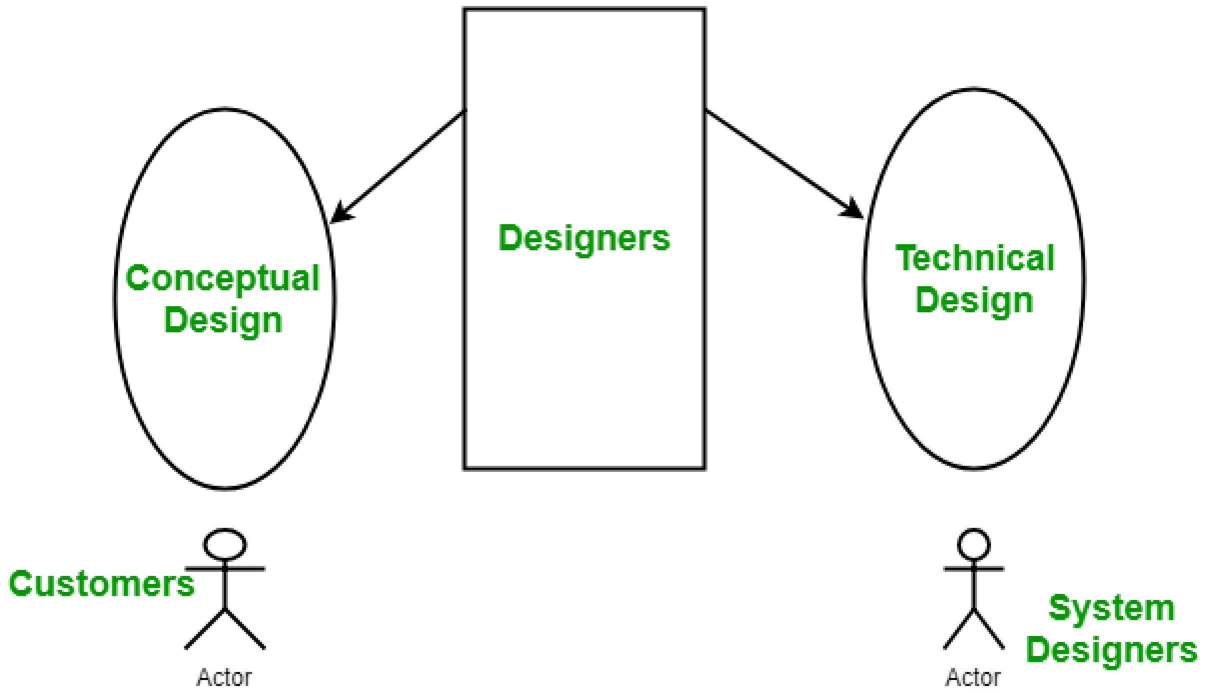
2. High coupling means that modules are closely connected and changes in one module may affect other modules.

3. Low coupling means that modules are independent, and changes in one module have little impact on other modules.

4. Cohesion refers to the degree to which elements within a module work together to fulfill a single, well-defined purpose.

5. High cohesion means that elements are closely related and focused on a single purpose, while low cohesion means that elements are loosely related and serve multiple purposes.

6. High coupling and low cohesion can make a system difficult to change and test, while low coupling and high cohesion make a system easier to maintain and improve.



Design Notation Characteristics

1. Modularity: design notation should support the development of modular software and provide a means of interface specifications.

2. Overall simplicity: design notation should be relatively simple to learn, relatively easy to use and generally easy to read.

3. Ease of editing: the procedural design may need modification as the software process proceeds. The ease with which a design represented can be edited can help facilitate each software engineering task.

4. Machine readability: notation that can be input directly into a computer based development method offers significant benefits.

5. Maintainability: software maintenance is most costly phase of software lifecycle.

6. Maintenance of software configuration nearly always means maintenance of the procedural design representation can be edited can help facilitate each software engineering task.

Important of Abstractions in System Design

1. Abstractions allow developers to focus on the core functionalities and design principles without getting overwhelmed by implementation details.

2. Abstractions enable system designers to think in terms of interfaces and contracts rather than specific implementations.

3. This promotes loose coupling between system components, allowing for easier replacement or upgrade of underlying technologies without impacting the overall system design.

4. Abstractions also aid in managing complexity by providing a simplified mental model of the system.

5. By hiding unnecessary details and exposing only relevant functionalities, abstractions allow developers to reason about the system at a higher level of abstraction.

6. Abstractions facilitate system scalability by providing a modular and extensible framework.

7. Through well-defined abstractions, developers can encapsulate complex functionalities and scale individual components independently.

Unit 4 (Coding, Structured Programming and Programming Practices)

Importance of Implementation Phase

1. In implementation phase software product start to build a absolute, high-quality software system from the "blueprint" provided in the detailed design document.
2. The implementation phase starts after Critical Design Review and proceeds according to the build plan prepared during the detailed design phase.
3. The system is realized through implementation producing the sources (source-code files, header files, make files, and so on) that will result in an executable system.
4. The design model is the basis for implementation.
5. In this stage the software design is realized as a set of working programs or program units.
6. The unit-testing means verifying each unit that meets its specifications and it is verified that the defined input produces the desired results.
7. Implementation includes testing the separate classes and/or packages, but not testing that the packages/classes work together.

---

steps to write good code

1. Before you start writing any code, it's essential to understand the problem you're trying to solve. This includes defining the requirements, constraints, and expected behavior of the software.

2. Select a programming language that is suitable for the task at hand.

3. Install any necessary software tools, such as a text editor or Integrated Development Environment (IDE), compiler, interpreter, or runtime environment.

4. Break down the problem into smaller, more manageable components or modules.

5. Follow the syntax rules and best practices of the chosen programming language.

6. After writing each section of code, test it thoroughly to ensure that it behaves as expected.

7. Once your code is working correctly, review it for clarity, efficiency, and maintainability.

8. Write clear and comprehensive documentation for your code, including comments, function/method descriptions and usage examples.

9. Use a version control system (e.g., Git) to manage changes to your codebase over time.

10.Once your code is complete and thoroughly tested, deploy it to the target environment.

Activities of the Development Team

1. Understanding and clarifying the requirements of the software being developed.

2. Creating the architecture and design of the software based on the requirements gathered.

3. Writing the actual code for the software based on the design specifications.

4. Verifying that the software functions correctly and meets the specified requirements.

5. Collaborating with team members to review code, designs, and requirements. Code reviews help ensure code quality, identify bugs or potential issues, and share knowledge among team members.

6. Providing ongoing maintenance and support for the software after it's deployed.

7. Communicating effectively with team members, stakeholders, and other project stakeholders.

8. Reflecting on the development process and identifying opportunities for improvement.

Design walk through

1. A design walkthrough is a review of a product or system by peers who are at the same level in the organization.

2. The participants meet to systematically discuss and review a piece of software.

3. This quality practice provides designers with early validation of design decisions related to the following: Content development and treatment, Graphical user interface design, and Product functionality elements.

4. In software engineering, a walkthrough is a form of software peer review.

5. During a walkthrough, a designer or programmer leads development team members and other interested parties through a software product.

6. The participants then ask questions and make comments about possible errors.

critical design review

1. A Critical Design Review (CDR) is an independent assessment that evaluates a program's readiness for fabrication, system integration, demonstration, and testing.

2. It also assesses the design's maturity, design build-to or code-to documentation, and remaining risks.

3. The CDR is a formal checkpoint in the product development process that ensures designs are ready for production and fit for purpose.

4. It also helps to identify mistakes and highlight risks and concerns, and highlight where further design improvements could be made.

5. The CDR presents the final designs through completed analyses, simulations, schematics, software code, and test results.

6. It should also present the engineering evaluation of the breadboard model of the project.

External documentation

1. External documentation in software engineering is usually made up of user guides, but can also include a detailed description of the program's design and implementation features.

2. External documentation describes how to use the code, while internal documentation explains how the code works.

3. External documentation is written in a place where people who need to use the software can read about how to use the software.

4. Examples of external documentation include flow charts, UML diagrams, requirements documents, and design documents.

5. Software documentation is commonly written in Markdown and HTML.

6. Some software documentation tools include: Document360, Dropbox Paper and GitBook.

Internal Documentation

1. In software engineering, internal documentation is any comments or instructions written within a program by and for the developer(s).

2. It's used to keep a form of communication open between developers.

3. Internal documentation is also known as technical documentation, and it acts as a one-stop point where teams can access technical information related to software products.

4. This information helps them understand the software development process, keep track of updates, and more.

5. Here are some examples of internal documentation:

- Process documentation

Documents produced during development and maintenance, such as standards, project plans, test schedules, reports, meeting notes, or business correspondence.

- Scheduling documentation

Helps to ensure that the project stays on track and that all tasks are completed on time.

- API documentation

Comprehensive documentation that describes the usage, parameters, return values, and functionality of application programming interfaces (APIs).

Good Programmer practice

1. Adhere to established coding standards and conventions for the programming language you're using.

2. Write code that is self-explanatory and easy to understand. Use meaningful variable and function names, and include comments when necessary to explain complex logic or intentions.

3. Break down your code into smaller, reusable modules or functions. Avoid duplicating code.

4. Write tests for your code before writing the actual implementation. This helps clarify requirements, ensures code correctness, and promotes modular design.

5. Refactor your code continuously to improve its design, readability, and maintainability.

6. Implement robust error handling mechanisms to gracefully handle exceptions, errors, and edge cases.

7. Write code with performance considerations in mind, but prioritize clarity and correctness over premature optimization.

8. Documentation: Maintain comprehensive documentation for your code, including inline comments, README files, and API documentation.

Unit 5 (Software Testing)

Software testing

1. Software testing is the process of evaluating a software product's functionality, quality, and performance before it's released.
2. Testers do this by manually interacting with the software or by running test scripts to find bugs and errors.
3. Software testing is important because it helps identify issues during development so they can be fixed before the product is released.
4. This ensures that only quality products are distributed to customers, which can lead to higher customer satisfaction and trust.
5. Here are some types of software testing:

here are different types of software testing based on the purpose, scope, and methods of testing. Some of the common types of software testing include:

- Unit Testing: Testing individual software components or modules to ensure that they function correctly.
- Integration Testing: Testing the interaction between different modules or components to ensure that they work together correctly.
- System Testing: Testing the entire system or application as a whole to ensure that it meets the specified requirements and works as expected.
- Acceptance Testing: Testing the software with end-users to ensure that it meets their needs and expectations.
- Performance Testing: Testing the performance and scalability of the software under different load conditions.
- Security Testing: Testing the software for vulnerabilities and weaknesses to ensure that it is secure and protects against unauthorized access or attacks.
- Regression Testing: Testing the software after modifications or changes to ensure that it still works correctly and does not introduce new defects or errors.

Testing Activities in each phase

1. Requirements Analysis:

- Determine correctness: Verify that the requirements specified are accurate, complete, and consistent.
- Create functional test data: Develop test cases based on the functional requirements to validate that the software meets the specified functionality.

2, Design:

- Determine correctness and consistency: Review the design documents to ensure that they accurately represent the intended architecture and behavior of the software.
- Create structural and functional test data: Develop additional test cases based on the design specifications.

3.Programming/Construction:

- Determine correctness and consistency: Continuously verify that the implemented code matches the design and requirements. Perform code reviews and inspections.
- Create structural and functional test data: Develop additional test cases or refine existing ones based on the implemented code.
- Apply test data: Execute the test cases against the implemented code to uncover defects, bugs, or regressions.
- Refine test data: Update test cases to reflect changes in the code or requirements and ensure comprehensive coverage of the software.

4. Operation and Maintenance:

- Retest: Perform regression testing to verify that the software still functions correctly after modifications or updates.

White box testing

1. White box testing is a software testing technique that involves testing the internal structure and workings of a software application.

2. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

3. White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure.

4. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.

Advantages

- Thorough Testing: White box testing is thorough as the entire code and structures are tested.
- Code Optimization: It results in the optimization of code removing errors and helps in removing extra lines of code.
- Early Detection of Defects: It can start at an earlier stage as it doesn’t require any interface as in the case of black box testing.

Disadvantages

- Programming Knowledge and Source Code Access: Testers need to have programming knowledge and access to the source code to perform tests.
- Overemphasis on Internal Workings: Testers may focus too much on the internal workings of the software and may miss external issues.
- Bias in Testing: Testers may have a biased view of the software since they are familiar with its internal workings.
- Test Case Overhead: Redesigning code and rewriting code needs test cases to be written again.

Black Box testing

1. Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding.

2. The primary source of black box testing is a specification of requirements that is stated by the customer.

3. In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not.

4. If the function produces correct output, then it is passed in testing, otherwise failed.

5. The test team reports the result to the development team and then tests the next function.

6. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Advantages

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.

Disadvantages

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.

Objectives of Walkthrough

A walkthrough has multiple objectives, including:

1. To provide feedback about the document's content or technical quality.

2. To familiarize the audience with the content.

3. To learn and fully understand software product development.

4. To find defects in developed software products.

5. To verify the validity of the proposed system.

6.To explain or transfer knowledge.

7. To evaluate the document's contents.

8. To achieve a common understanding.

9. To help users understand the product's value proposition faster and encourage them to become regular users.

Software Maintenance

1. Software maintenance is a continuous process that occurs throughout the entire life cycle of the software system.

2. The goal of software maintenance is to keep the software system working correctly, efficiently, and securely, and to ensure that it continues to meet the needs of the users.

3. This can include fixing bugs, adding new features, improving performance, or updating the software to work with new hardware or software systems.

4. It is also important to consider the cost and effort required for software maintenance when planning and developing a software system.

5. It is important to have a well-defined maintenance process in place, which includes testing and validation, version control, and communication with stakeholders.

Types of Software maintenance

1. Corrective Maintenance: This type of maintenance involves fixing defects or faults identified during testing or use. It aims to correct any errors or issues that impact the software's functionality, reliability, or usability.

2. Adaptive Maintenance: Adaptive maintenance involves making modifications to the software to keep it usable in a changing environment, such as updating it to work with a new operating system version or integrating it with new hardware or software components.

3. Perfective Maintenance: Perfective maintenance involves making enhancements or improvements to the software to meet new or changing user requirements. This type of maintenance aims to optimize performance, enhance usability, or add new features to the software.

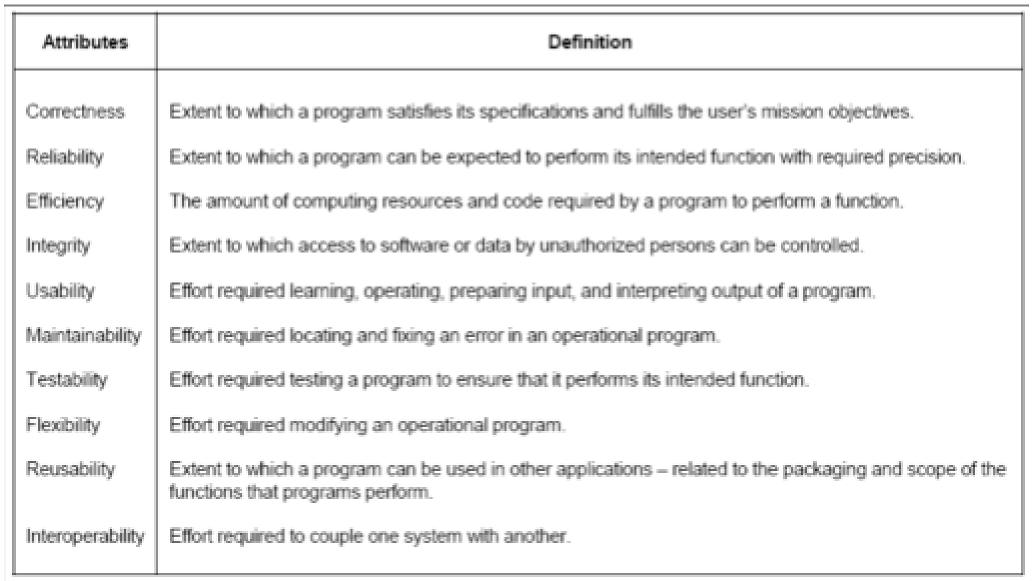
4. Preventive Maintenance: Preventive maintenance involves making changes to the software to prevent future problems or issues. It includes activities such as code refactoring, performance tuning, and updating documentation to ensure that the software remains reliable, maintainable, and efficient over time.

Unit 6 (Quality Assurance)

Software Quality Assurance

1. Software Quality Assurance (SQA) is simply a way to assure quality in the software.
2. It is the set of activities which ensure processes, procedures as well as standards are suitable for the project and implemented correctly.
3. Software Quality Assurance is a process which works parallel to development of software.
4. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue.
5. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.
6. Generally, the quality of the software is verified by the third-party organization like international standard organizations.

Quality Attributes



Attributes	Definition
Correctness	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
Reliability	Extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform a function.
Integrity	Extent to which access to software or data by unauthorized persons can be controlled.
Usability	Effort required learning, operating, preparing input, and interpreting output of a program.
Maintainability	Effort required locating and fixing an error in an operational program.
Testability	Effort required testing a program to ensure that it performs its intended function.
Flexibility	Effort required modifying an operational program.
Reusability	Extent to which a program can be used in other applications – related to the packaging and scope of the functions that programs perform.
Interoperability	Effort required to couple one system with another.

Maturity Model

1. Maturity models are structured as a series of levels of effectiveness.

2. It's assumed that anyone in the field will pass through the levels in sequence as they become more effective.

3. Maturity models can evaluate qualitative data to determine a company's long-term trajectory and performance.

4. Maturity models can be used as a benchmark for comparison and as an aid to understanding.

5. For example, a museum can use a maturity model to be well informed about what programs, exhibits, social media and other initiatives it might take within its mission and vision to address civic issues.

6. Some common maturity models include: coaching, auditing, evaluation, portfolio management, control practices, communication between projects and personal development.

Levels of Maturity Model

1. maturity level can be thought of as a defined evolutionary plateau that the software process is aiming to achieve.

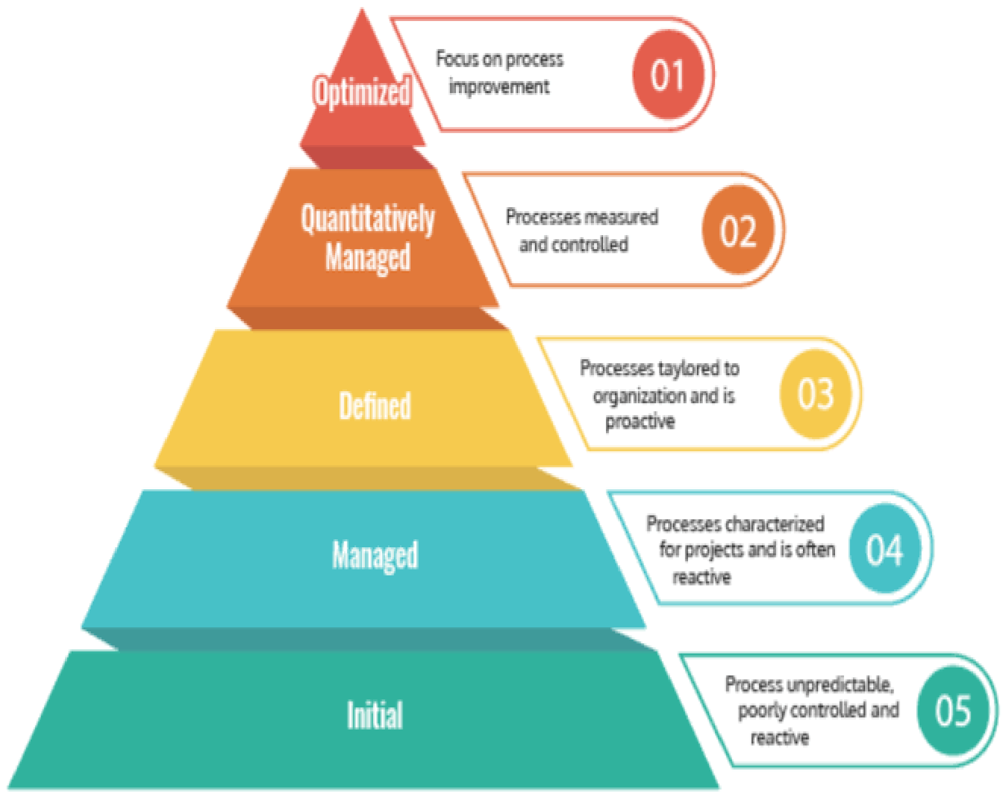
2. The CMM defines five maturity levels which form the top-level structure of the CMM itself.

3. Each level is a foundation that can be built upon to improve the process in order.

4. Starting with basic management practices and succeeding through following established levels.

5. When progressing through the path to improving a software process, the path will without doubt lead through each maturity level.

6. The goal should be to achieve success in each maturity level.


Optimized
Focus on process improvement
01
Quantitatively Managed
Processes measured and controlled
02
Defined
Processes taylored to organization and is proactive
03
Managed
Processes characterized for projects and is often reactive
04
Initial
Process unpredictable, poorly controlled and reactive
05

Unit 7 (Software Configuration Management)

Software configuration control

Software configuration control, also known as version control or configuration management, is the process of managing and controlling changes to software products throughout their lifecycle.
Key aspects of software configuration control include:
1. Version Control: Tracking and managing different versions of software, source code, documents, and other artifacts to ensure that changes can be traced, compared, and reverted if necessary.
2. Change Management: Establishing procedures and workflows for requesting, reviewing, approving, and implementing changes to software and related artifacts.
3. Baseline Management: Defining and maintaining baselines, which are stable versions of software or configurations that serve as reference points for future changes.
4. Configuration Identification: Identifying and documenting the components and configurations of software products, including dependencies, versions, and interfaces.
5. Configuration Auditing: Conducting regular audits and reviews to verify that software configurations are accurate, up-to-date, and compliant with requirements, standards, and policies.

Unit 8 (Latest Trends in Software Engineering)

Web Engineering

1. Web engineering is the use of tools and methods to develop web systems, while also managing their navigation structure, security, and quick change requirements.
2. It also involves the design, development, evolution, and evaluation of web applications.
3. Web engineering is based on disciplined, quantifiable, and systematic approaches to development, operation, and maintenance.
4. It also incorporates principles and guidelines to design, manage, and maintain complex websites, applications, and functionality.
5. Web engineering focuses on usability, performance, security, quality, and reliability.
6. Web engineering borrows many fundamental concepts and principles from software engineering.
7. However, the web engineering process emphasizes similar technical and management activities, though there are subtle differences in the way these activities are conducted.
8. The overriding philosophy is identical, which is a disciplined approach to the development of a computer-based system.

Computer Aided Software Engineering (CASE)

1. Computer-aided software engineering (CASE) is the implementation of computer-facilitated tools and methods in software development.

2. CASE is used to ensure high-quality and defect-free software.

3. CASE ensures a check-pointed and disciplined approach and helps designers, developers, testers, managers, and others to see the project milestones during development.

4. CASE can also help as a warehouse for documents related to projects, like business plans, requirements, and design specifications.

5. One of the major advantages of using CASE is the delivery of the final product, which is more likely to meet real-world requirements as it ensures that customers remain part of the process.

6. CASE illustrates a wide set of labor-saving tools that are used in software development.

7. It generates a framework for organizing projects and to be helpful in enhancing productivity.

CASE Tools

1. Diagramming Tools: It helps in diagrammatic and graphical representations of the data and system processes.

2. Computer Display and Report Generators: These help in understanding the data requirements and the relationships involved.

3. Analysis Tools: It focuses on inconsistent, incorrect specifications involved in the diagram and data flow.

4. Central Repository: It provides a single point of storage for data diagrams, reports, and documents related to project management.

5. Documentation Generators: It helps in generating user and technical documentation as per standards. It creates documents for technical users and end users.

6. Code Generators: It aids in the auto-generation of code, including definitions, with the help of designs, documents, and diagrams.

Agile software development

1. Agile software development is a project management methodology that uses multiple development cycles, called sprints, to create working software quickly.

2. It emphasizes collaboration with customers, adaptation to change, and frequent inspection.

3. Agile is especially useful for complex projects or those with uncertain requirements.

4. Agile software development focuses on the people doing the work and how they work together.

5. Solutions evolve through collaboration between cross-functional teams that are self-organizing.

6. Agile is designed so that all parties can provide feedback as software is developed.

7. The Agile process can be broken down into three main stages:

- Preparation: The product owner creates a backlog of features they want to include in the final product, and the development team estimates how long each feature will take to build.
- Development: The team uses multiple development cycles, called sprints, to create working software.
- Adaptation: The team frequently inspects and adapts to changes.

Extreme Program

1. Extreme Programming (XP) is a framework for agile software development that aims to improve the quality of life for the development team and produce high quality software.

2. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

3. XP uses five guiding values, five rules, and 12 practices for programming.

4. XP has three main groups of practices: software engineering, work environment, and project management.

5. XP attempts to reduce the cost of changes in requirements by having multiple short development cycles, rather than a long one.

5 Rules of XP

1. Refactoring: A practice that keeps designs simple by removing process duplication.

2. Sustainable pace: A key tenet of the Agile Manifesto that encourages sustainable development and work-life balance.

3. Design: Start with the simplest solution and build on it later.

4. Planning: Customers should interact with developers to become the driving force for the development.

5. Standup: Each day starts with a standup, and velocity should always be measured.

12 Principles of XP

1. Feedback: Gather feedback early and often to continuously improve.

2. Assume Simplicity: Start with the simplest solution and evolve as needed.

3. Incremental Change: Make small, incremental changes to the system.

4. Embrace Change: Welcome changing requirements, even late in development.

5. Quality Work: Strive for high-quality work through continuous attention to detail.

6. Balanced Team: Form a balanced team of developers, testers, and customers.

7. Continuous Integration: Integrate and test code frequently to catch issues early.

8. Customer Involvement: Involve customers throughout the development process.

9. Shared Understanding: Ensure everyone has a shared understanding of the project goals and requirements.

10. Simple Design: Focus on simplicity in design and architecture.

11. Pair Programming: Work in pairs to improve code quality and knowledge sharing.

12. Sustainable Pace: Maintain a sustainable pace to avoid burnout and maintain productivity.

Unified Modeling Language

1. Unified Modeling Language (UML) is a general-purpose modeling language.

2. The main aim of UML is to define a standard way to visualize the way a system has been designed.

3. It is quite similar to blueprints used in other fields of engineering.

4. UML is not a programming language, it is rather a visual language.

5. We use UML diagrams to portray the behavior and structure of a system.

6.UML helps software engineers, businessmen, and system architects with modeling, design, and analysis.

7. UML is linked with object-oriented design and analysis.

8. UML makes use of elements and forms associations between them to form diagr0ams.

9. Diagrams in UML can be broadly classified as:

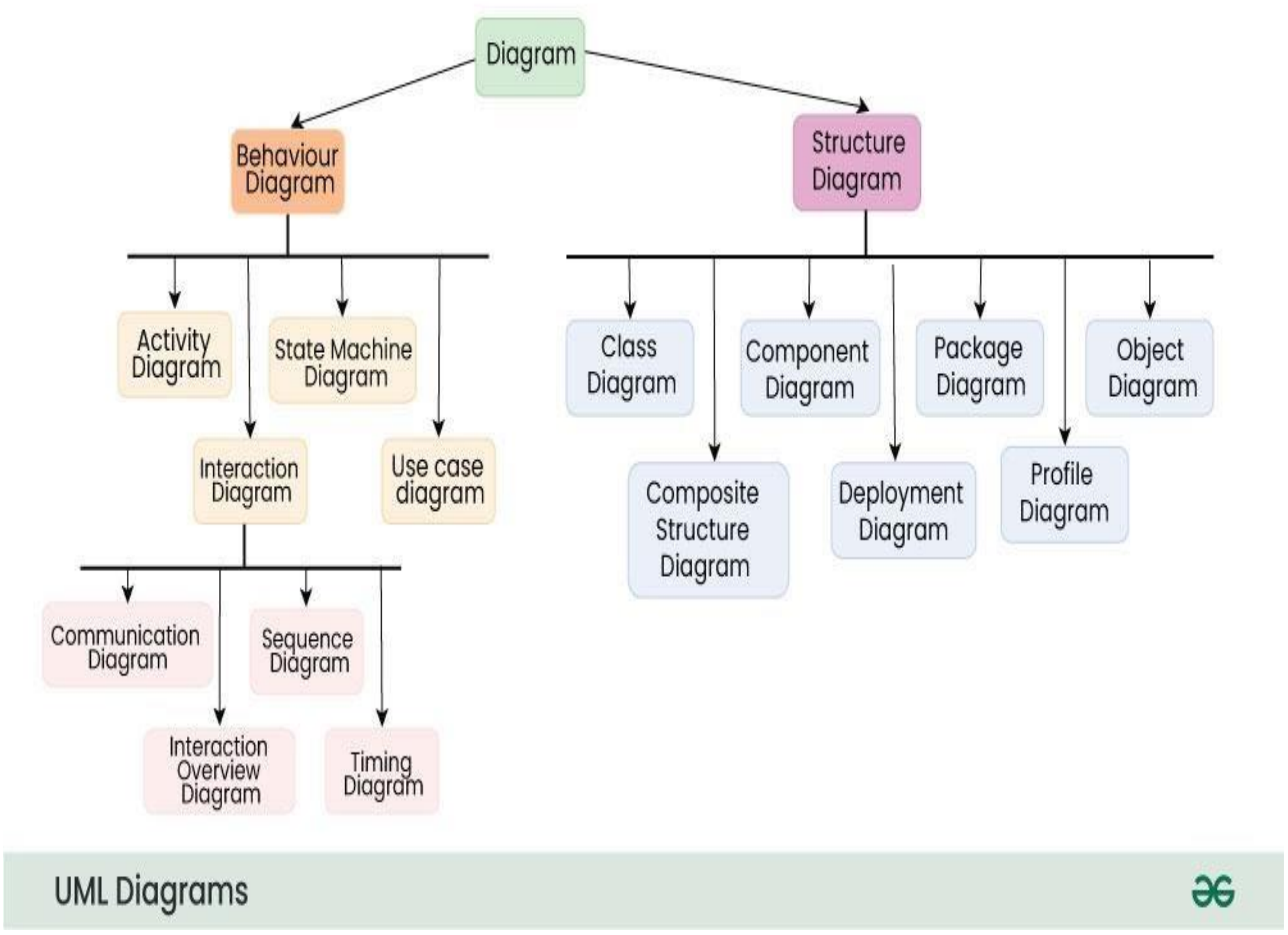


Diagram
Behaviour Diagram
Structure Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Use case diagram
Communication Diagram
Sequence Diagram
Interaction Overview Diagram
Timing Diagram
Class Diagram
Component Diagram
Package Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Profile Diagram
UML Diagrams