SOFTWARE ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE Sheet No.

WHAT IS A SOFTWARE ENGINEERING?

The term software engineering is the product of two words, Software, and engineering.

The Software is a collection of integrated programs.

Engineering is the application of scientific and practical knowledge to invent, design, build, maintain and improve frameworks, processes, etc.

Software Engineering is the application of methods and scientific knowledge to create practical cost-effective solutions for the design, construction, operation and maintenance of software.

EVOLUTION OF SOFTWARES:

- 1945 - 1965 → Origin
- 1965 - 1985 → Crisis
- 1990 - 2000 → Internet
- 2000 - 2010 → Light Weight
- 2010 - Till Now → AI, ML, DL

SOFTWARE COMPONENTS:

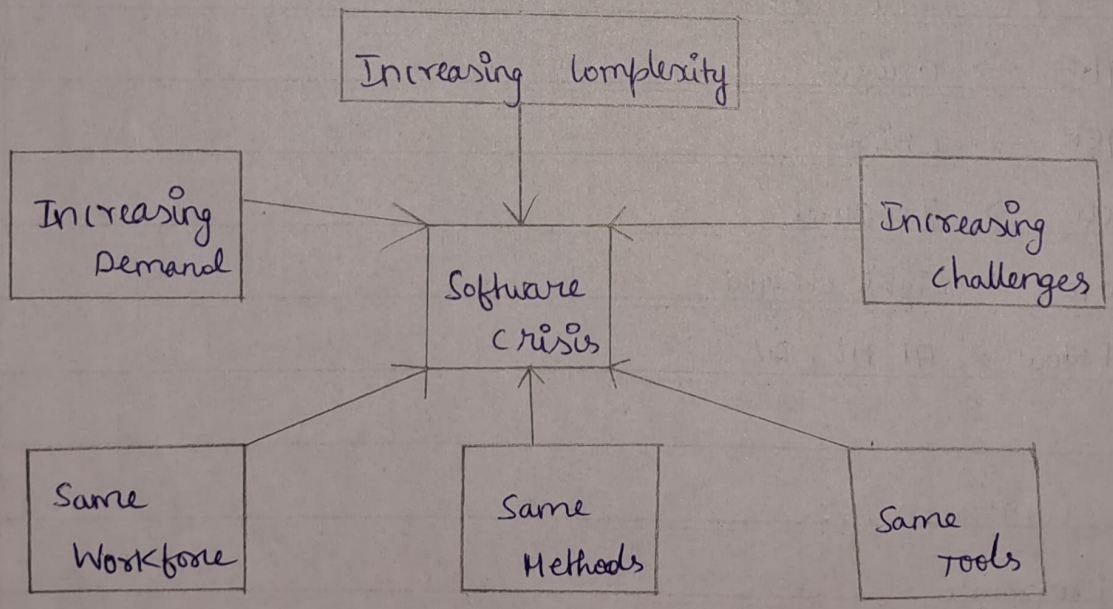
- Set of Programs
- Software Documents
- Operating Procedures

SOFTWARE CHARACTERISTICS:

- Operational
  - Budget
  - Usability
  - Efficiency
  - Correctness
  - Functionality
  - Dependability
  - Security
  - Safety.
- Transitional
  - portability
  - Interoperability
  - Reusability
  - Adaptability
- Maintenance
  - Modularity
  - Maintainability
  - Flexibility
  - Scalability

SOFTWARE CRISIS:

Software crisis is a set of difficulties or problems encountered while developing a software.



CAUSES OF SOFTWARE CRISIS:

- The cost of owning and maintaining software was as expensive as developing the software
- At that time projects were running over-time.

- At that time software was very inefficient
- The quality of the software was low quality
- Software often did not meet user requirements
- The average software project overshoots its schedule by half
- At that time Software was never delivered.
- Non-optimal resource utilization
- Difficult to alter, debug, and enhance.
- The Software complexity is harder to change

SOFTWARE ENGINEERING PROCESSES

Software processes is a coherent set of activities for specifying, designing, implementing and testing software systems.

The Key processes are:

- Specification - defining what the system should do.
- Design and implementation - defining the organization of the system and implementing the system.
- Validation - checking that it does what the customer wants.
- Evolution - changing the system in response to changing customer needs.

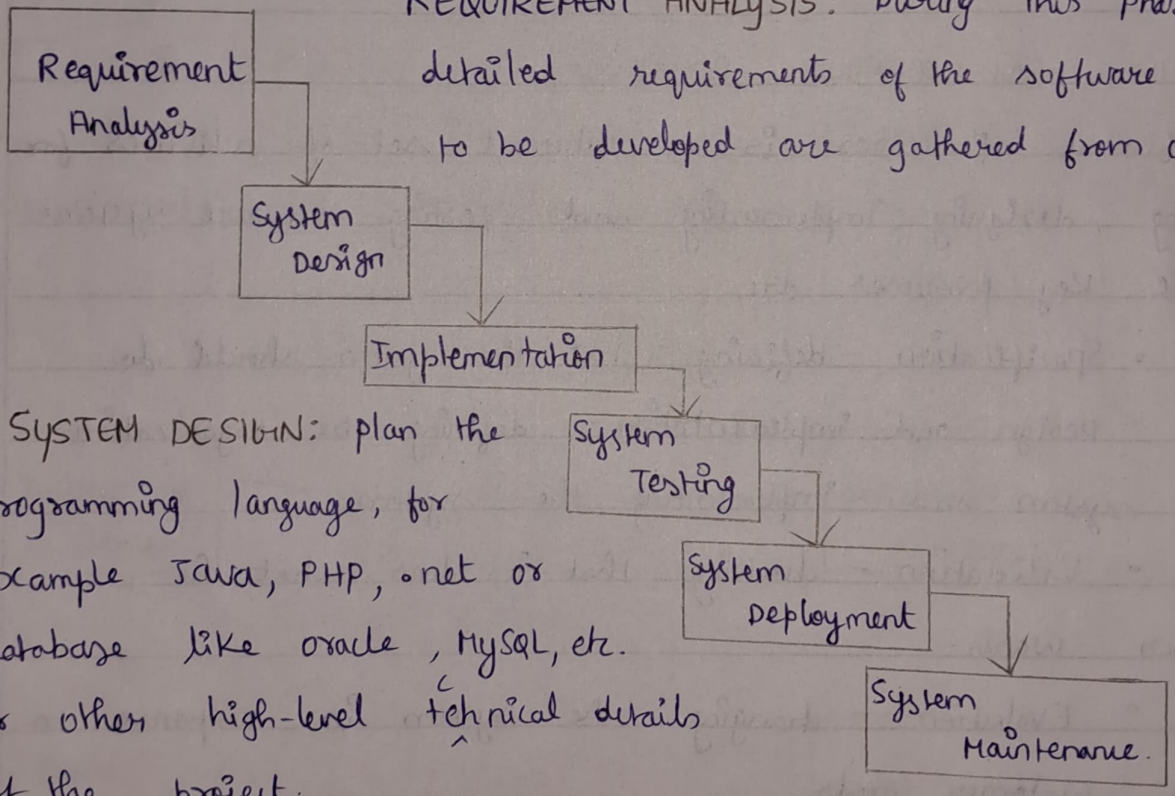
SOFTWARE QUALITY ATTRIBUTES:

- Usability
- Testability
- Maintainability
- Security
- Reliability
- Scalability
- Supportability
- Availability
- Flexibility
- Interoperability
- Portability
- Reusability
- Performance

SOFTWARE DEVELOPMENT LIFE CYCLE [SDLC]

1. Requirement Analysis
2. Feasibility Study
3. Design
4. Coding
5. Testing
6. Deployment
7. Maintenance
8.

WATER FALL MODEL:



REQUIREMENT ANALYSIS: During this phase, detailed requirements of the software system to be developed are gathered from client.

SYSTEM DESIGN: plan the programming language, for Example Java, PHP, .net or database like oracle, MySQL, etc. or other high-level technical details of the project.

IMPLEMENTATION: At this stage the coding of the software takes place.

SYSTEM TESTING: In this phase, you test the software to verify that it is built as per the specifications given by the client.

SYSTEM DEPLOYMENT: Deploy the application in the respective environment.

SYSTEM MAINTENANCE: once your system is ready to use, you may later require change the code as per customer request.

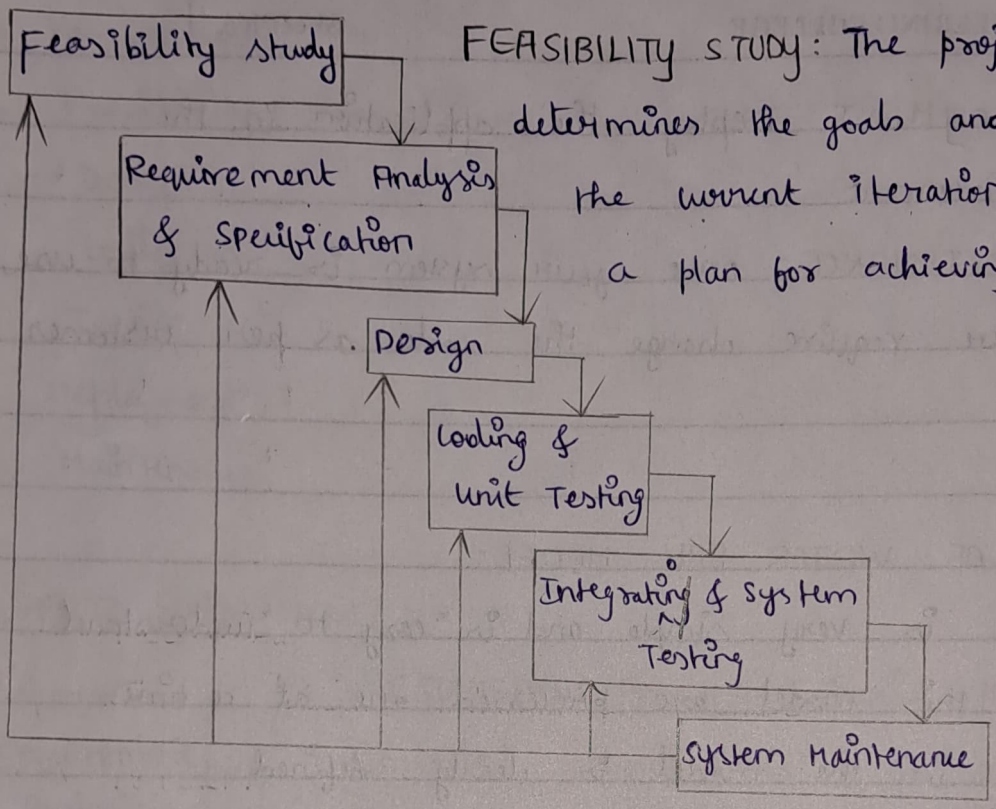
ADVANTAGE OF WATER FALL MODEL:

- This model is very simple and is easy to understand.
- phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well-understood milestones.
- Process, actions and results are very well documented.
- Reinforces good habits: define-before-design, design-before-code
- This model works well for smaller projects and projects where requirements are well understood.

DISADVANTAGE OF WATER FALL MODEL:

- No feedback path.
- Difficult to accomodate change requests.
- No overlapping of phases.

ITERATIVE WATERFALL MODEL:



FEASIBILITY STUDY: The project team determines the goals and objectives for the current iteration, and develops a plan for achieving them.

REQUIREMENT ANALYSIS & SPECIFICATION: The project team gathers and analyses requirements for the current iteration.

DESIGN: The project team designs the solutions for the current iteration.

CODING & UNIT TESTING: The project team builds and tests the solution for the current iteration.

INTEGRATION & SYSTEM TESTING: The project team tests the solution to ensure that it meets the requirements and functions as intended.

SYSTEM MAINTENANCE: The project team provides ongoing support and maintenance for the solution.

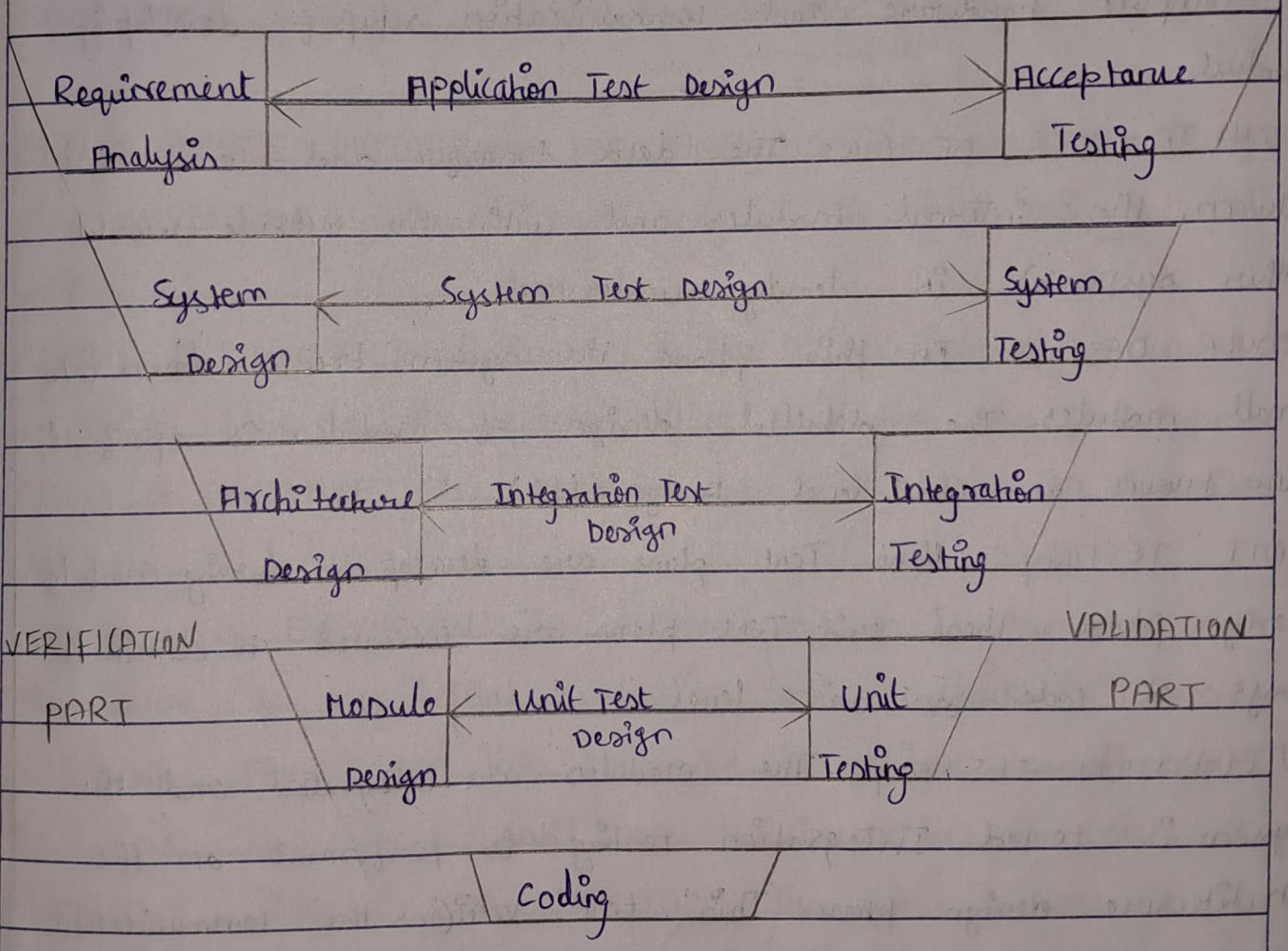
ADVANTAGE OF ITERATIVE WATERFALL MODEL:

- Feedback Path
- Cost - Effective
- Well - organized

DISADVANTAGE OF ITERATIVE WATERFALL MODEL:

- Difficult to incorporate change requests
- Incremental delivery not supported.
- Risk handling not supported
- Limited customer interactions.

V SHAPED MODELS:



VERIFICATION: It is the process of evaluation of the product development process to find whether specified requirements met.

VALIDATION: Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

REQUIREMENT ANALYSIS: This phase contains detailed communication with the customer to understand their requirements and expectations

SYSTEM DESIGN: This phase contains the system design and the complete hardware and communication setup for developing product.

ARCHITECTURAL DESIGN: The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood.

MODULE DESIGN: In this phase the system breaks down into small modules. The detailed design of modules is specified, also known as low-level Design (LLD).

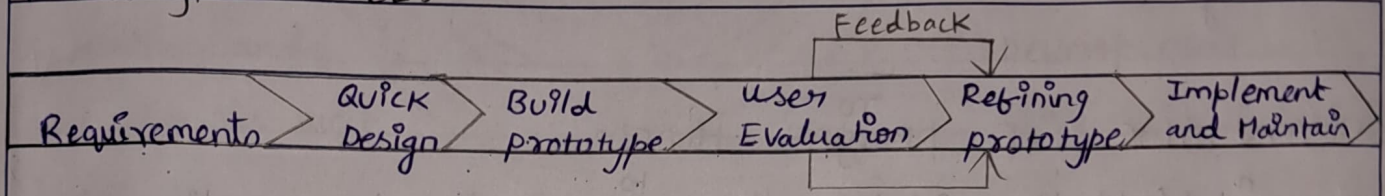
UNIT TESTING: Unit Test plans are developed during module design phase. These unit Test plans are executed to eliminate bugs at code or unit level.

INTEGRATION TESTING: The modules are integrated and the system is tested. Integration testing is performed on the Architecture design phase. This test verifies the communication of modules among themselves.

SYSTEM TESTING: System testing test the complete application with its functionality, inter dependency, and communication. It tests the functional and non-functional requirements of the developed application.

USER ACCEPTANCE TESTING [UAT]: UAT is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets user's requirement and system is ready for use in real world.

PROTOTYPE MODEL:



REQUIREMENTS GATHERING AND ANALYSIS: The requirements of the system are gathered and analyzed through interviews with users.

QUICK DESIGN: A preliminary design is created to give users a brief idea of the system, helping to develop the prototype.

BUILD A PROTOTYPE: The prototype is built based on the quick design.

INITIAL USER EVALUATION: The proposed system is presented to the client for an initial evaluation, and feedback is collected.

REFINING PROTOTYPE: The prototype is refined according to the user's suggestions until all requirements are met.

IMPLEMENT PRODUCT AND MAINTAIN: The final system is developed and implemented, undergoing routine maintenance to minimize downtime and prevent failures.

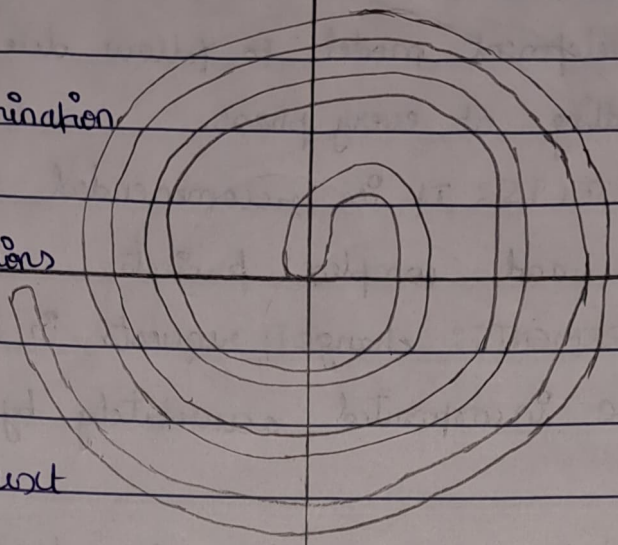
ADVANTAGE OF PROTOTYPE MODEL:

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

DISADVANTAGE OF PROTOTYPE MODEL:

- An unstable / badly implemented prototype often becomes the final product.
- Require extensive customer collaboration
- Difficult to know how long the project will last.
- Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
- Prototyping tools are expensive
- Special tools & techniques are required to build a prototype.
- It is a time-consuming process.

SPIRAL MODEL:



OBJECTIVES DETERMINATION AND IDENTIFY ALTERNATE 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.

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.

DEVELOP 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.

SPIRAL MODEL:

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 OF SPIRAL MODEL:

RISK HANDLING: The projects with many unknown risks that ~~occur~~ occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.

GOOD FOR LARGE PROJECTS: It is recommended to use the spiral model in large and complex projects.

FLEXIBILITY IN REQUIREMENTS: change requests in the requirements at later phase can be incorporated accurately by using this model.

CUSTOMER SATISFACTION: Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

DISADVANTAGES OF SPIRAL MODEL:-

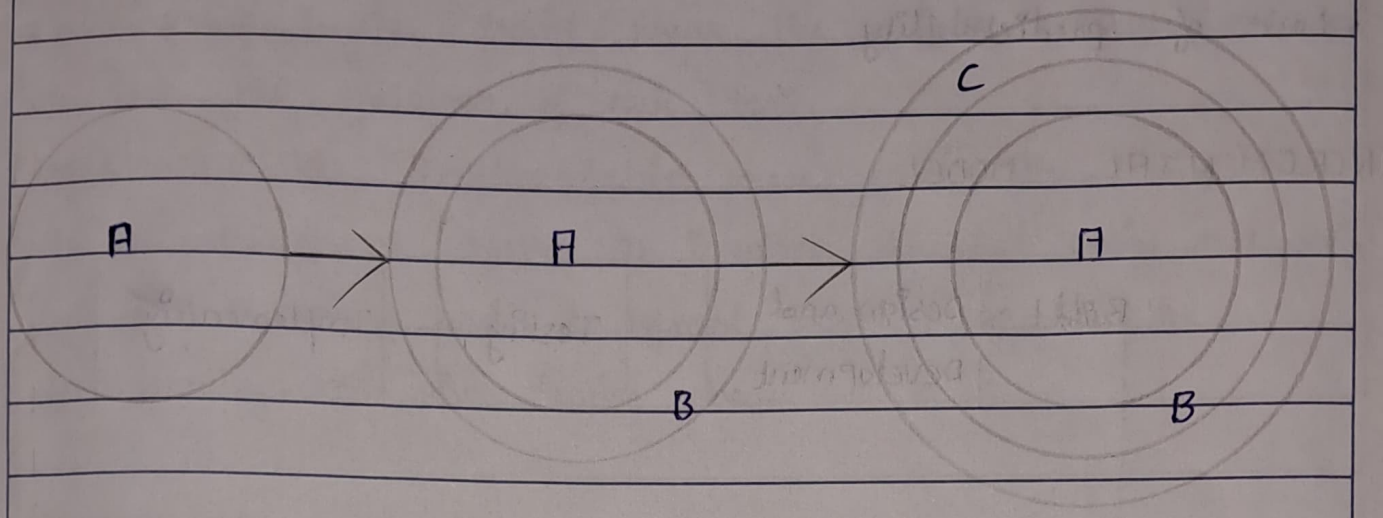
COMPLEX: The spiral model is much more complex than other SDLC models.

EXPENSIVE: Spiral model is not suitable for small projects as it is expensive.

TOO MUCH DEPENDABILITY ON RISK ANALYSIS: The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.

DIFFICULTY IN TIME MANAGEMENT: As the number of phases is unknown at the start of the project, so time estimation is very difficult.

EVOLUTIONARY DEVELOPMENT MODELS:



- It is a combination of Incremental model & Iterative model.
- In this model, the project is divided into smaller parts, and each part is developed, tested and delivered in iterations.
- Each iteration is an evolutionary step towards the final product.
- The requirements for each iteration are defined at the beginning of the iteration and may change based on feedback from the stakeholders.

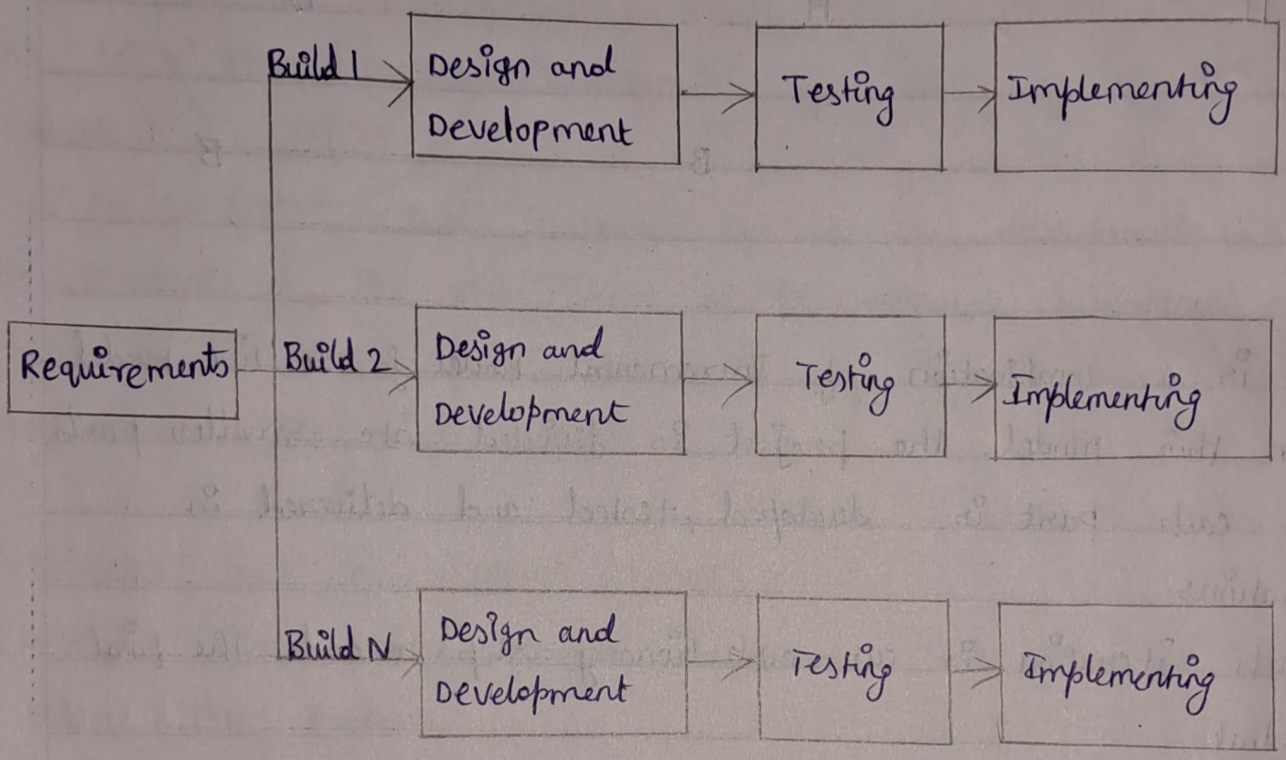
ADVANTAGES OF EVOLUTIONARY DEVELOPMENT MODELS:

- Flexibility
- Continuous feedback
- Risk reduction
- Early working software

DISADVANTAGES OF EVOLUTIONARY DEVELOPMENT MODELS:

- Cost and time overruns
- Difficulty in defining requirements
- Dependency on customer feedback
- Lack of predictability.

INCREMENTAL MODEL



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

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

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

IMPLEMENTATION: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase.

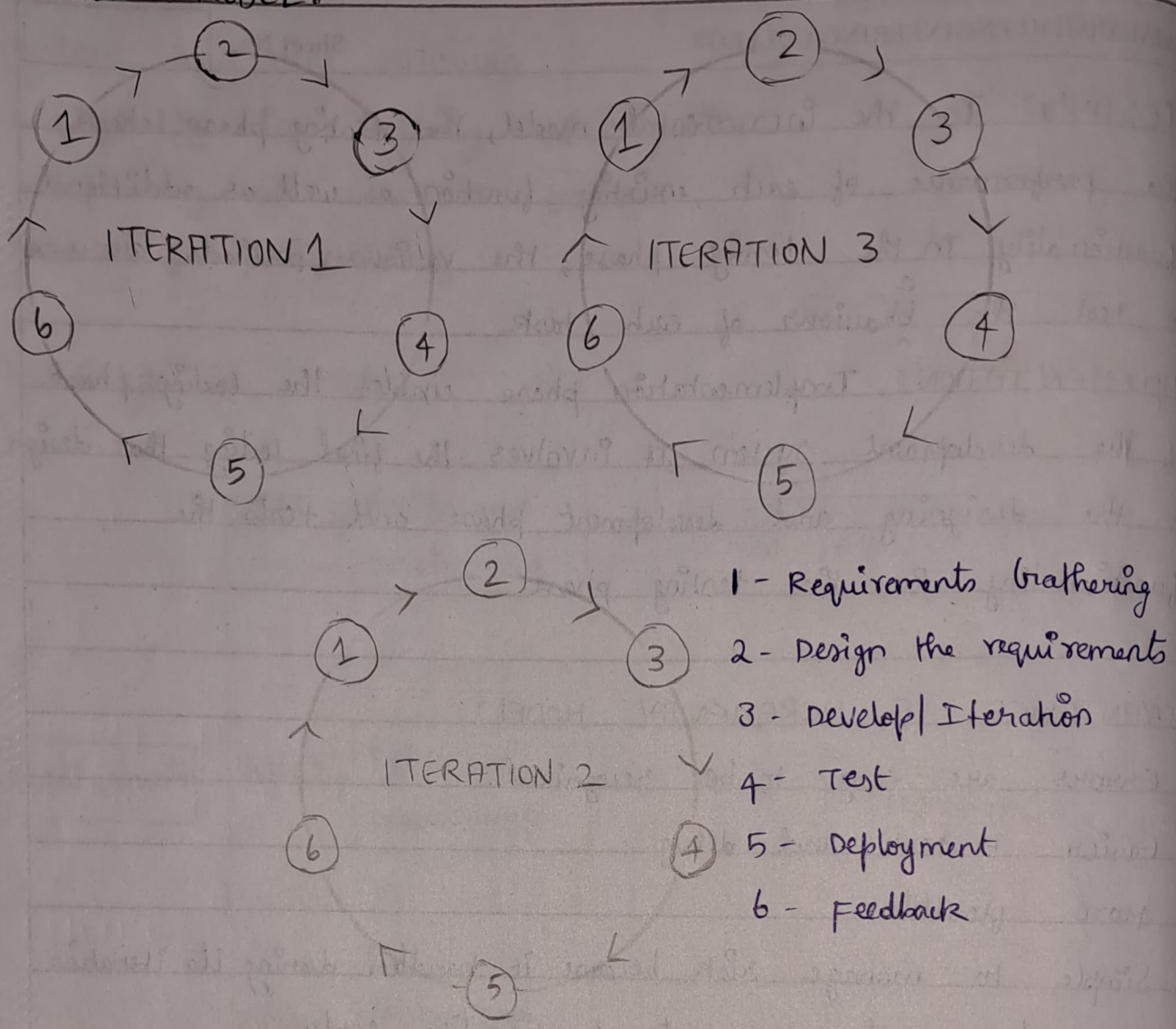
ADVANTAGES OF INCREMENTAL MODEL:

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

DISADVANTAGE OF INCREMENTAL MODEL:

- Need for good planning.
- Total cost is high
- Well defined module interfaces are needed.

AGILE MODEL:



REQUIREMENTS GATHERING: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility

DESIGN THE REQUIREMENTS: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

CONSTRUCTION / ITERATION: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

TESTING: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

DEPLOYMENT: In this phase, the team issues a product for the user's work environment.

FEEDBACK: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feed.

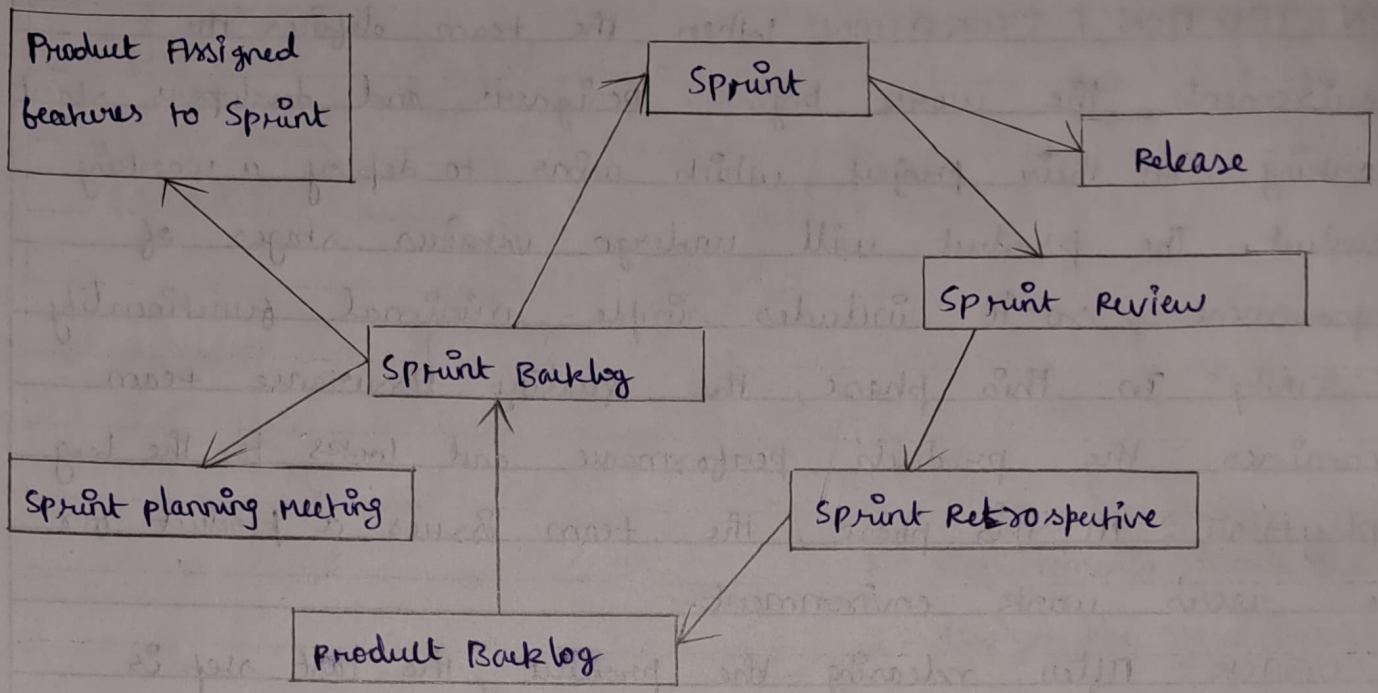
ADVANTAGES OF AGILE MODEL:

- Frequent Delivery.
- Face-to-Face communication with clients.
- Efficient design and fulfils the business requirement.
- Anytime changes are acceptable.
- It reduces total development time.

DISADVANTAGES OF AGILE MODEL:

- Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
- Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

SCRUM MODEL



SPRINT: A Sprint is a time-box of one month or less. A new Sprint starts immediately after the completion of the previous Sprint.

RELEASE: When the product is completed then it goes to the Release stage.

SPRINT REVIEW: If the product still have some non-achievable features then it will be checked in this stage and then the product is passed to the Sprint Retrospective stage.

SPRINT RETROSPECTIVE: In this stage quality or status of the product is checked.

PRODUCT BACKLOG: According to the prioritize features the product is organized.

SPRINT BACKLOG: Sprint Backlog is divided into two parts product assigned features to sprint and Sprint planning meeting.

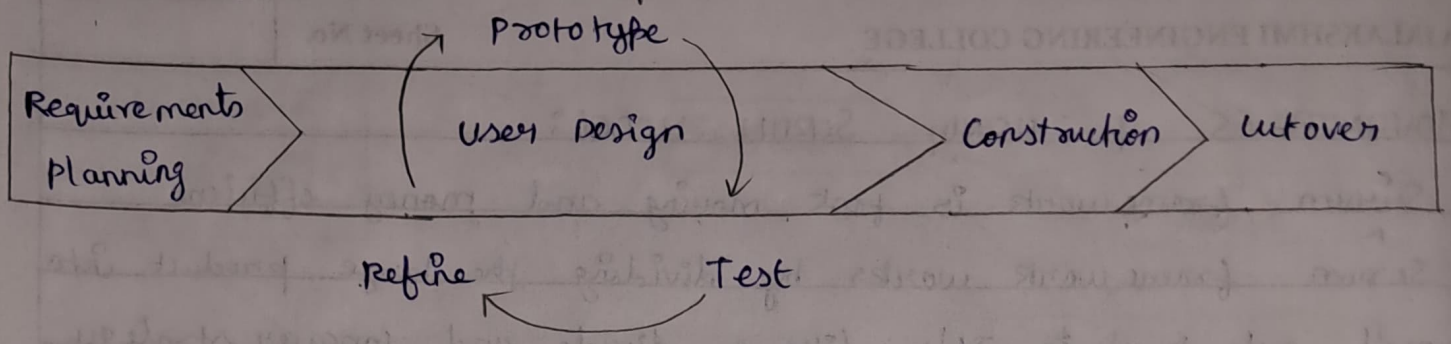
ADVANTAGES OF USING SCRUM MODEL:

- Scrum framework is fast moving and money efficient.
- Scrum framework works by dividing the large product into small sub-products. It's like a divide and conquer strategy
- In Scrum customer satisfaction is very important
- Scrum is adaptive in nature because it have short sprint
- As Scrum framework rely on constant feedback therefore the quality of product increases in less amount of time.

DISADVANTAGES OF USING SCRUM FRAMEWORK:

- Scrum framework do not allow changes into their sprint.
- Scrum framework is not fully described model. If you wanna adopt it you need to fill in the framework with your own details like Extreme programming[XP], Kanban, DSDM.
- It can be difficult for the Scrum to plan, structure and organize a project that lacks a clear definition.
- The daily Scrum meetings and frequent reviews require substantial resources.

RAPID APPLICATION DEVELOPMENT [RAD] MODEL



REQUIREMENTS PLANNING: Identify and prioritize requirements through workshops and user interviews

USER DESIGN: Create a working prototype based on the requirements using rapid prototyping tools.

CONSTRUCTION: Build the software using the prototype as a guide, and iterate on feedback from stakeholders.

CUTOVER: Deliver the software to the customer, train users, and support the system in production.

ADVANTAGES OF RAD MODEL:

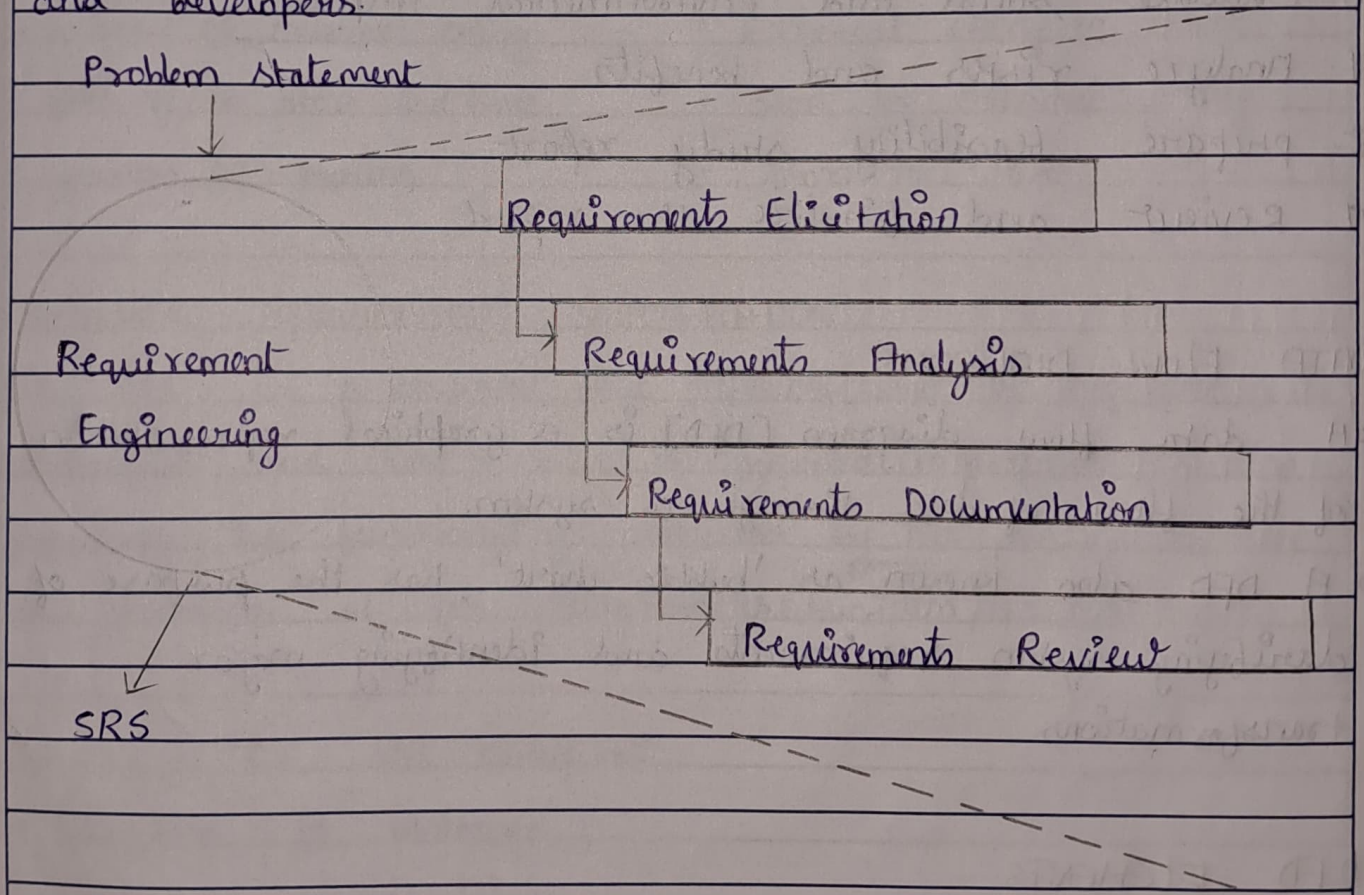
- Quick development
- User involvement
- Flexibility
- Reduced development cost.

DISADVANTAGES OF RAD MODEL:

- High Dependence on user Involvement
- Limited Scalability
- Technical Complexity
- potential for scope creep

REQUIREMENT ENGINEERING PROCESS

- Requirement engineering is the process of determining user expectations for a new or modified product.
- These user expectations, called requirements, must be quantifiable, relevant and detailed.
- In software engineering, such requirements are often called functional specifications.
- Its goal is to create a requirements specification that is complete, correct, and understandable to both customers and developers.



FEASIBILITY STUDY:

A feasibility study is a comprehensive and systematic analysis that helps to determine whether a project or business venture is feasible, sustainable, and viable

STEPS INVOLVED IN FEASIBILITY STUDY:

1. Define project scope and objectives.
2. Conduct market analysis
3. Evaluate technical feasibility.
4. Assess financial feasibility
5. Evaluate legal and regulatory requirements
6. Evaluate social and environmental impact
7. Analyze risks and benefits.
8. prepare feasibility study report.
9. Review and finalize the report.

DATA FLOW DIAGRAMS

- A data flow diagram [DFD] is a graphical representation of the flow of data through a system.
- A DFD also known as 'bubble chart', has the purpose of clarifying system requirements and identifying major transformations.

DFD ELEMENTS

ENTITIES: The sources or destinations of data, such as customers, suppliers, or systems.

PROCESSES: The activities or operations that transform data from one form to another

DATA STORES: The places where data is stored, such as databases, files, or memory.

DATA FLOWS: The paths that data takes as it moves through the system, represented by arrows connecting the elements.

DATA CAN FLOW FROM:

- External entity to process
- process to external entity
- process to store and back
- process to process

DATA CANNOT FLOW FROM:

- External entity to external entity
- External entity to store
- Store to external entity
- store to store

SOFTWARE REQUIREMENT SPECIFICATION:

A SRS is a document that explains what the software will accomplish and how it will be expected to perform. It also specifies the functionality required by the product to meet the demands of all stakeholders [business, users].

GOALS OF SRS DOCUMENT

- Feedback to customer
- Problem decomposition
- Input to design specification
- product validation check

IEEE STANDARDS FOR SRS:

The IEEE Standards aims to ensure that SRS documents are well-structured, clear, and complete, so that they can be used as a basis for design, development, and testing of software systems. It is widely recognized and used in the software industry as a best practice for developing SRS documents.

INTRODUCTION

- Purpose
- Intended Audience
- Intended Use
- Product Scope
- Definitions and Acronyms

OVERALL DESCRIPTION

- User Needs
- Assumptions and Dependencies

SYSTEM FEATURES AND REQUIREMENTS

- Functional Requirements
- External Interface Requirements
- System Features
- Nonfunctional Requirements

VERIFICATION [QUALITY]	VALIDATION [PERFORMANCE]
◦ AM I building the product right?	◦ AM I building the right product?
◦ Verification is a static practice of verifying documents, design, code and program	◦ Validation is a dynamic mechanism of validating and testing the actual product.
◦ It does not involve executing the code.	◦ It always involves executing the code.
◦ It is low level activity	◦ It is higher level activity
◦ Verification is carried out by quality assurance team	◦ Validation is carried out by testing team.
◦ Verification uses methods like inspections, reviews, walkthroughs etc	◦ Validation uses methods like black box testing, white box testing etc.
◦ Verification is to check whether the software conforms to specifications	◦ Validation is to check whether software meets the customer expectation and requirements.

ISO 9000 MODELS

ISO 9000 is a set of international standards developed by ISO for implementing quality management systems [QMS] in organizations to improve their overall quality and performance. It provides a framework for organizations to implement a systematic approach to quality management and achieve their business objectives.

ISO 9001

- Outlines requirements for a QMS
- Widely used ISO 9000 model
- Provides a framework for a systematic approach to quality management
- Includes processes for continuous improvement and customer satisfaction.

ISO 9002

- outlines requirements for a QMS for production, installation, and servicing.
- used by organizations that do not design their own products but instead provide production, installation, and servicing services to customers.

ISO 9003

- outlines requirements for a QMS for final inspection and testing
- used by organizations that do not design or produce their own products but instead provide final inspection and testing services to customers.

SEI - CMM MODEL: [SEI - Software Engineering Institute, CMM - Capability Maturity Model]:

The SEI - CMM is a framework for process improvement in software development. It is a five-level model that describes the maturity of an organization's software development processes, with each level representing a more advanced stage of process maturity. The SEI - CMM is widely used by software development organizations to assess and improve their processes.

THE FIVE LEVELS OF THE SEI - CMM are:

1. INITIAL: processes are ad hoc, unpredictable, and often chaotic.
2. REPEATABLE: Basic project management processes are established, and projects follow a standard process.
3. DEFINED: The organization has developed a standard software development process, and processes are documented and communicated throughout the organization.
4. MANAGED: The organization tracks and measures the software development process and uses these measurements to improve the process.
5. OPTIMIZING: Continuous process improvement is a part of the organization's culture, and organization is constantly striving to improve its processes.

BASIC CONCEPT OF SOFTWARE DESIGN:

Software design is the process of defining the architecture, components, interfaces, and other characteristics of a software system that meet specified requirements. The goal of software design is to produced a system that is maintainable, adaptable, and efficient, while also being easy to use and understand.

ABSTRACTION: The process of breaking down a system into smaller, more manageable components.

MODULARITY: The organization of a system into separate components or modules, each of which performs a specific function.

ENCAPSULATION: The hiding of implementation details of a module from other parts of the system, allowing it to be changed without affecting other modules.

COUPLING: The degree to which modules are dependent on one another, with a lower coupling indicating better modular design.

COHESION: The degree to which the elements within a module are related to each other, with a higher cohesion indicating better module design.

SEPARATION OF CONCERNS: The idea that different aspects of a system should be separated into different components, with each component responsible for a specific concern.

DESIGN PATTERNS: Reusable solutions to common software design problems that can improve the overall quality and maintainability of a system.

TYPES OF SOFTWARE DESIGN LEVELS:

1. ARCHITECTURAL DESIGN: This level of design provides a high-level abstract view of the system, identifying the main components and their interactions.

2. HIGH-LEVEL DESIGN: This level of design focuses on breaking down the system into subsystems and modules and determining how they interact with each other

3. DETAILED DESIGN: This level of design deals with the implementation details of each module and defines their logical structure, including algorithms, data structures and communication interfaces.

COUPLING AND COHESION

COUPLING: Coupling is also the indication of the relationships between modules. It is the concept of the Inter-module. The coupling has also many types but usually, the low coupling is good for software

COHESION: Cohesion is the indication of the relationship within the module. It is the concept of intra-module. Cohesion has many types but usually, high cohesion is good for software.

COUPLING	COHESION
• Measures interdependence between modules.	• Measures relatedness within a module
• High coupling can lead to difficulty in maintenance	• High cohesion can make a system easier to maintain
• Can result in errors and bugs	• Can reduce errors and bugs
• Different types: content, common, control, etc.	• Different types: functional, sequential, communicational, etc.

TYPES OF COUPLING:

CONTENT COUPLING: When one module modifies or accesses the contents of another module.

COMMON COUPLING: When two or more modules share the same global data.

CONTROL COUPLING: When one module controls the flow of execution of another module.

STAMP COUPLING: When data is passed between modules in the form of a large composite data structure.

DATA COUPLING: When two modules communicate by passing simple data elements.

TYPES OF COHESION:

FUNCTIONAL COHESION: Elements in a module work together towards the same objective.

SEQUENTIAL COHESION: Elements in a module form a specific sequence where the output of one becomes the input of the next.

COMMUNICATIONAL COHESION: Elements in a module operate on the same data structure

PROCEDURAL COHESION: Elements in a module perform a series of steps in a specific sequence to achieve a goal.

TEMPORAL COHESION: Elements in a module must be executed in a specific order or at the same time.

LOGICAL COHESION: Elements in a module perform similar operations or manipulate similar data

COINCIDENTAL COHESION: Elements in a module have no meaningful relationship to each other

FUNCTION ORIENTED DESIGN

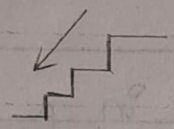
- System is designed from a functional viewpoint.
- Top-down decomposition
- Divide & conquer approach
- DFD is used. [DFD- Data Flow Diagram]

OBJECT ORIENTED DESIGN

- System is viewed as a collection of objects (i.e., entities)
- Bottom-up approach.
- UML is used. [UML - Unified Modelling Languages]

TOP-DOWN DESIGN:

Top-down design is a software design approach that starts with a high-level view of the system and gradually decomposes it into smaller and more detailed parts.



ADVANTAGES:

- provides a clear understanding of the system's overall structure and functionality.
- Helps identify and manage system complexity.
- Makes it easier to maintain and modify the system over time.

DISADVANTAGES:

- Can be difficult to anticipate all system requirements at the outset.
- May result in a rigid architecture that is hard to modify
- Can be time-consuming to design and implement all the required modules.

BOTTOM-UP DESIGN:

Bottom-up design is a software design approach that starts with small, self-contained units of code and gradually combines them to form larger and more complex components.

ADVANTAGES:

- provides a better understanding of the implementation details of the system.
- Enables rapid prototyping and development of small, incremental improvements.
- Can result in a more flexible and adaptable system, as each component can be modified independently.

DISADVANTAGES:

- Can be difficult to ensure that all components work together seamlessly.
- May result in a system with unnecessary complexity or redundancy.
- Can be challenging to manage the testing and validation of each component.

SOFTWARE METRICS

Software metrics is the practice of using quantitative measurements to evaluate and improve software development processes and products, which involves analyzing data related to software attributes and development performance.

TYPES OF SOFTWARE METRICS

1. PRODUCT METRICS: Measures the characteristics of software products, such as size and quality.
2. PROCESS METRICS: Measures the performance of software development processes, such as time and effort.
3. PROJECT METRICS: Measures the progress and success of software development projects, such as budget and schedule.

TESTING OBJECTIVES:

Testing objectives are the goals that need to be achieved through the testing process. The testing objectives can vary depending on the nature of the software being tested, the requirements of the project, and the goals of the stakeholders involved.

- Verification of functionality:
- Validation of usability:
- Identification of defects:
- performance evaluation:
- Security testing:

LEVELS OF TESTING

We have 4 different levels of testing:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing.

1. UNIT TESTING: It is the first level of testing where individual units or modules of the software are tested independently. Unit testing helps to identify defects at an early stage and isolate them.

2. INTEGRATION TESTING: It is the next level of testing where the tested units are combined to test their interaction and integration. Integration testing helps to identify issues that may arise due to the interactions between different units.

3. SYSTEM TESTING: It is a level of testing where the entire software application is tested as a whole. It tests the functionality, performance, and security of the system.

4. ACCEPTANCE TESTING: It is the final level of testing where the software is tested for compliance with business requirements and user expectations. It is done to ensure that the software meets the needs of the end-users.

INTEGRATION TESTING

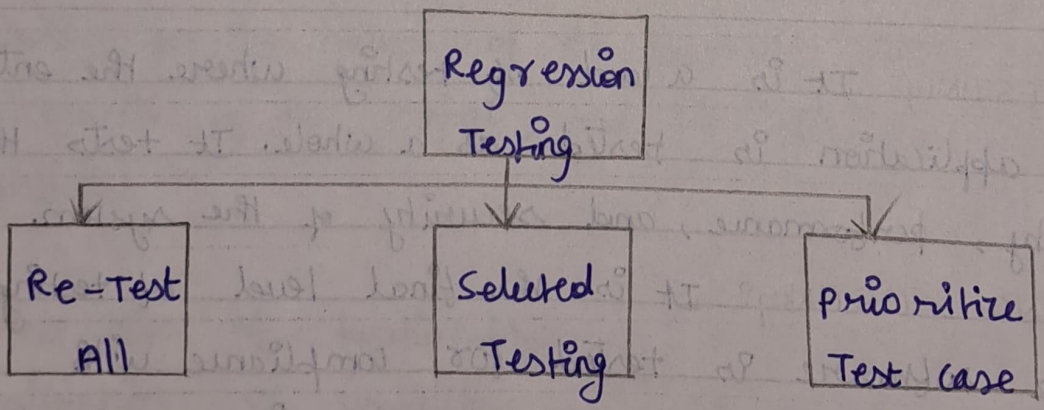
Integration testing is a software testing process that verifies the correct functioning of different modules or components of a software application when integrated. It focuses on detecting defects, errors, and inconsistencies in the interfaces, data flow, and behavior of the application as a whole.

TYPES OF INTEGRATION TESTING

1. Top - down
2. Bottom - up

REGRESSION TESTING

Regression testing is the process of testing existing software applications to make sure that changes or updates have not caused unintended effects or introduced new defects.



Date
Page

FUNCTIONALITY TESTING:

- Focuses on verifying software functions as intended.
- Tests software using different scenarios and cases.
- Identifies defects in software functionality.
- Helps ensure software meets end-user's needs.

PERFORMANCE TESTING

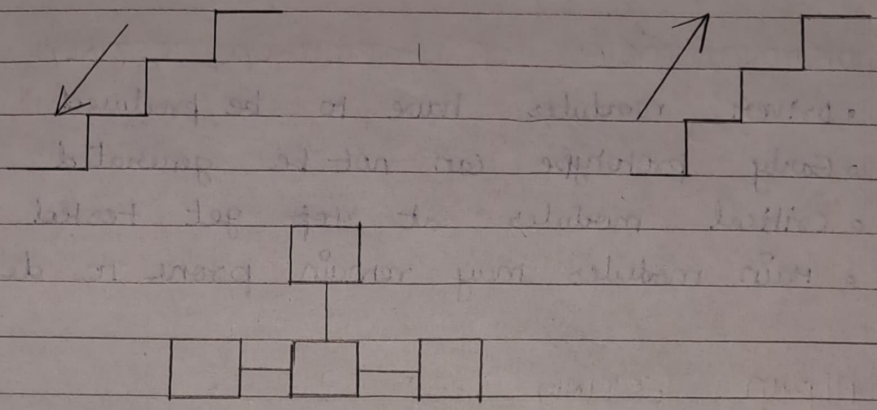
- Tests system performance under specific workloads and user loads.
- Measures speed, scalability, stability and resource utilization under stress conditions.
- Identifies performance bottlenecks, such as slow response times and high resource utilization.
- optimizes system performance and ensures it meets expected user requirements.

BLACK BOX TESTING	WHITE BOX TESTING
• Black box testing is a software testing method done by testers without knowledge of the internal code or implementation.	• White box testing is a method where internal structure is known by the tester.
• programming knowledge is not required.	• It's done by developers with programming and implementation knowledge.
• It's checks the functionality of the system being tested, and is used in higher level testing like acceptance testing.	• The aim is to check how the system performs and it's used for lower level testing like unit and integration testing.

BLACK BOX TESTING	WHITE BOX TESTING
◦ It's also called functional or external testing.	◦ It's also known as structural or interior testing.
ADVANTAGES:	ADVANTAGES:
◦ Can be performed by anyone.	◦ Complete coverage of the code.
◦ Identifies external defects.	◦ Improved code quality.
◦ Cost-effective and requires fewer resources.	◦ More accurate and reliable results.
DISADVANTAGES	DISADVANTAGES
◦ May miss internal defects	◦ Requires deep understanding of internal workings.
◦ Limited to testing external behavior.	◦ Costly and requires advanced testing tools.
◦ May be less accurate and reliable than white box testing.	◦ May not be effective in identifying external defects.

Date
Page

TOP DOWN TESTING AND BOTTOM-UP TESTING



ADVANTAGES OF TOP-DOWN TESTING

- Critical modules are tested on priority
- Early detection and fixing of major design defects.
- Fault localization become easier
- Allows creation of early prototype
- Early prototype makes test case creation easier.

DISADVANTAGES OF TOP-DOWN TESTING

- Many stubs need to be produced.
- Creating stubs for intricate features may become complicated.
- Representing test cases in stubs can be difficult.
- Observation of test output is more difficult.
- Lower-level modules are insufficiently tested.

ADVANTAGES OF BOTTOM-UP TESTING:

- Detects major flaws that occur at the lower level modules.
- Testing can start without waiting for all units to be completed.
- Easy to create test conditions.

- Test result observation is easier.

DISADVANTAGES OF BOTTOM-UP TESTING:

- Driver modules have to be produced
- Early prototype can not be generated
- Critical modules at top get tested inadequately
- Main modules may remain prone to defects.

ALPHA TESTING:

1. Conducted by the development team or a dedicated testing team within the organization.
2. Done in a controlled environment, typically on-premises.
3. Focuses on testing functionality, performance and usability.
4. Aims to identify defects before release to Beta Testing or the public.
5. uses test scenarios and cases to simulate real-world usage scenarios.
6. Feedback is used to improve the software's functionality and overall quality.

BETA TESTING

1. Conducted by a group of end-users who are not part of the development team.
2. Done in a real-world environment outside the organization's premises.
3. Focuses on testing functionality, performance, usability & compatibility.
4. Aims to identify defects and gather feedback from end-users.
5. Uses real-world scenarios and testers have the freedom to use the software in any way they like.
6. Feedback is used to make necessary improvements before releasing the software to the general public.

MAINTENANCE OF SOFTWARE

1. Modifying or updating software products after delivery.
2. Correcting defects, improving performance, adapting to changing requirements, or adding new features.
3. Four types of maintenance: corrective, adaptive, perfective & preventive
4. Ensuring software products continue to meet user needs and expectations.
5. Activities include analyzing requirements, testing, debugging, and documentation.

Date
Page

CATEGORIES OF MAINTENANCE

1. CORRECTIVE MAINTENANCE: This involves correcting defects or errors that are discovered during the use of the software.

2. ADAPTIVE MAINTENANCE: This involves modifying the software to adapt to changes in the operating environment or user requirements.

3. PERFECTIVE MAINTENANCE: This involves improving the functionality or performance of the software without changing its existing features or behavior.

4. PREVENTIVE MAINTENANCE: This involves making changes to the software to prevent future problems, such as improving documentation or enhancing security features.

COST OF MAINTENANCE:

1. Labor costs for maintenance activities, such as analysis, testing, debugging and documentation.
2. Costs of tools and resources needed for maintenance.
3. Downtime and lost productivity during maintenance activities.
4. Potential risks of introducing new defects or errors during maintenance activities.
5. Opportunity costs of not being able to work on new projects or features while performing maintenance

Date
Page

SOFTWARE RE-ENGINEERING

It is the process of updating and improving existing software systems without changing their external behavior or functionality. It involves analyzing the software, identifying weakness, and making improvements to enhance its performance and maintainability.

STEPS:
1. Evaluate and analyze the existing software
2. Define re-engineering goals
3. Develop a plan
4. Implement the plan
5. Test and verify
6. Deploy
7. Maintain.

REVERSE ENGINEERING:

It is the process of analyzing an existing product or system to understand how it works in detail, often using specialized tools and techniques with the aim of creating a replica, improving its functionality, or identifying security vulnerabilities.

STEPS INVOLVED IN REVERSE ENGINEERING:

1. PLANNING: Identify the purpose and scope of the analysis.

2. CODE ANALYSIS: Analyze the software code or application using tools such as disassemblers or decompilers.

3. DOCUMENTATION: Document the findings of the analysis, including the software's structure, data flow, control flow and interfaces.

4. RECONSTRUCTION: Use the information gathered from the analysis to reconstruct the software application or create a new application.

5. TESTING: Test the reconstructed or new software application to ensure that it meets the required specifications.

6. MAINTENANCE: Maintain the reconstructed or new software application by updating it as needed.

ESTIMATION OF VARIOUS PARAMETERS

TIME: Estimated using expert judgment, historical data, and software sizing techniques

EFFORT: Estimated using expert judgement, historical data, and estimation models.

COST: Estimated using expert judgement, historical data, and cost models.

QUALITY: Estimated using static analysis, dynamic analysis, and testing.

RISK: Estimated using risk identification and analysis, expert judgement, or historical data.

Date
Page

CONSTRUCTIVE COST MODELS [COCOMO]

This is a software cost estimation model developed by Dr. Barry Boehm in the late 1970s. It is a widely used method for estimating the effort, time and cost required to develop a software project. COCOMO takes into account various factors such as project size, complexity and development environment to produce a cost estimate.

TYPES OF COCOMO MODELS:

1. BASIC COCOMO: It is a simple and quick model that estimates the effort required for a software project based on the size of the project in lines of code.

2. INTERMEDIATE COCOMO: It is a more sophisticated model that takes into account additional factors such as project complexity, development environment, and team experience.

3. DETAILED COCOMO: It is the most comprehensive model that includes a detailed analysis of all the project parameters, such as the number of modules, the number of interfaces, the database size, and the level of documentation.

FORMULA:

$$\text{Effort} = A * (\text{size})^{B} * EAF \quad [A \times (\text{size})^{B} \times EAF]$$

Where:

- A and B are coefficients that are specific to the type of software being developed.
- Size is the size of the software project in lines of code.
- EAF is the effort adjustment factor that takes into account the project's characteristics.

RISK ANALYSIS

- Identify potential risks and their impact on the software project.
- Access ~~the~~ and prioritize risks based on their likelihood and severity.

RISK MANAGEMENT:

- Develop and implement strategies to mitigate or eliminate risks.
- Use risk management techniques such as avoidance, mitigation, transfer, or acceptance.
- Collaborate with stakeholders involved in software development.

Date
Page

- Integrate risk analysis and management into the entire software development life cycle