T3 SKILLS CENTER
T3 कौशल केंद्र
TWKSAA SOFTWARE ENGINEERING

Er. Rajesh Prasad

- आप एक सागर हो बहते नदी का जल नहीं आप एक बदलाव हो भटकाव की कोई राह नहीं
- उस रास्ते पर चलो जिस रास्ते पर भीड़ कम हो (हर काम हो कुछ अलग)
- देश की मिट्टी से करो आप इतना प्यार जहाँ जाओ वहाँ मिले खूब इज्जत और सम्मान
- छह दिन कीजिए अपना काम एक दिन कीजिए त्वक्सा को दान
- त्वक्सा एक चिंगारी हैं हर जगह जलना हम सब की जिमेवारी हैं
- **Motive: - New (RID PMS & TLR)**

"त्वक्सा सॉफ्टवेयर इंजीनियरिंग के इस पुस्तक में आप सॉफ्टवेयर इंजीनियरिंग के संबंध में सभी बुनियादी अवधारणाएँ सीखेंगे। मुझे आशा है कि इस पुस्तक को पढ़ने के बाद आपके ज्ञान में वृद्धि होगी और आपको कंप्यूटर विज्ञान के बारे में और अधिक जानने में रुचि होगी"

"In this TWKSAA Software Engineering book you will learn all basic concept regarding Software Engineering. I hope after reading this book your knowledge will be improve and you will get more interest to know more thing about computer Science".

"Skill कौशल एक व्यक्ति के पास उनके ज्ञान, अनुभव, तत्वशास्त्रीय योग्यता, और प्रैक्टिकल अभियांत्रिकी के साथ संचित नौकरी, व्यापार, या अन्य चुनौतीपूर्ण परिस्थितियों में सक्रिय रूप से काम करने की क्षमता को कहते हैं। यह व्यक्ति के द्वारा सीखी जाने वाली कौशलों की प्रतिभा, क्षमता और निपुणता को संक्षेप में व्यक्त करता है"।

TWKSAA RID MISSION

(Research)	(Innovation)	(Discovery)
अनुसंधान करने के महत्वपूर्ण कारण:	नवीनीकरण करने के महत्वपूर्ण कारण:	खोज करने के महत्वपूर्ण कारण:
1. नई ज्ञान की प्राप्ति	1. प्रगति के लिए	1. नए ज्ञान की प्राप्ति
2. समस्याओं का समाधान	2. परिवर्तन के लिए	2. ज्ञान के विकास में योगदान
3. तकनीकी और व्यापार में उन्नति	3. उत्पादन में सुधार	3. अविष्कारों की खोज
4. विकास को बढ़ावा देना	4. प्रतिस्पर्धा में अग्रणी होने के लिए	4. समस्याओं का समाधान
5. सामाजिक प्रगति	5. समाज को लाभ	5. समाज के उन्नति का माध्यम
6. विज्ञान और प्रौद्योगिकी का विकास	6. विज्ञान और प्रौद्योगिकी के विकास।	6. विज्ञान और तकनीक के विकास

"T3 Skills Center is a Learning Earning and Development Based Skill Center."

T3 कौशल केंद्र एक सीखने कमाई और विकास आधारित कौशल केंद्र है।

T3 SKILLS CENTER

Index

T3 SKILLS CENTER

❖ **What is Software Engineering?**
- The term software engineering is the product of two words, software, and engineering.
- The software is a collection of integrated(एकीकृत) programs.

➢ **Software=Program+ Documentation + Operating Procedures**
- **Program:** Program is a combination of source code & object code (a compiled file)
- **Documentation:** it is a way for engineers and programmers to describe their product and the process they used in creating it in formal writing.
- **Operating Procedures:** it is a set of written instructions that describes the step-by-step process that must be taken to properly perform a routine activity.

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

✓ **Software Engineering: -** Software engineering is an engineering branch that involves the application of engineering principles, scientific principles, methods, techniques, procedures tools to design, develop, test, deploy, and maintain software systems.

❖ **Why is Software Engineering required?**
- Software engineering is required for several reasons:
  1. Complexity Management
  2. Quality Assurance
  3. User Requirements and Satisfaction
  4. To manage large software
  5. For more Scalability
  6. Cost Management
  7. To manage the dynamic nature of software
  8. Risk Management
  9. Collaboration and Teamwork
  10. Innovation and Adaptability
  11. Compliance and Security

❖ **importance of Software engineering:**
- Reduces complexity
- To minimize software cost
- To decrease time
- Handling big projects
- Reliable software
- Effectiveness

❖ **Characteristics of a good software engineer:**
➢ **There are the following characteristics for a good software engineer.**
- Good technical knowledge of the project range (Domain knowledge).
- Good programming abilities.
- Good communication skills.
- interpersonal skills.
- High motivation.
- fundamentals knowledge of computer science.
- Intelligence.
- Ability to work in a team
- Discipline, etc.

T3 SKILLS CENTER

❖ **What is Software Processes?**

- Software processes, also known as software development processes or software engineering processes, are a set of activities, methodologies, and practices used to design, develop, test, deploy, and maintain software systems.

❖ **Software Process Model:**

- Software process models, also known as software development life cycle models, are systematic approaches that guide the software development process from inception(आरंभ) to deployment and maintenance.

❖ **Software Development Life Cycle (SDLC):**

- A software development life cycle model is a pictorial and diagrammatic representation of the software life cycle.
- a life cycle model maps the various activities performed on a software product from its inception to retirement.

❖ **Why SDLC Need?**

- Software Development Life Cycle (SDLC) is necessary for effective software development due to the following reasons:

1) **Structured Approach:** SDLC provides a structured and systematic approach to software development. It breaks down the entire development process into well-defined phases, tasks, and activities, ensuring that each step is planned and executed in a logical sequence.

2) **Clear Requirements Management:** SDLC emphasizes the importance of gathering and managing requirements effectively. It ensures that user needs and expectations are captured, documented, and validated throughout the development lifecycle.

3) **Risk Management:** SDLC incorporates risk management practices to identify, assess, and mitigate risks throughout the development process. It encourages proactive risk analysis and planning, enabling teams to anticipate potential issues and take preventive measures.

4) **Quality Assurance:** SDLC includes testing and quality assurance activities at different stages of development. It promotes the use of testing methodologies, such as unit testing, integration testing, system testing, and user acceptance testing

5) **Documentation and Traceability:** SDLC emphasizes the importance of documentation at each stage of development. It ensures that the requirements, design, code, and testing activities are documented, making it easier to understand, maintain, and enhance the software over time.

6) **Resource Planning and Management:** SDLC assists in resource planning and management throughout the development process. It helps in estimating project timelines, allocating resources effectively, and managing dependencies.

7) **Project Control and Monitoring:** SDLC provides mechanisms for project control and monitoring. It includes milestones, deliverables, and checkpoints to track the progress of the development process.

8) **Customer Satisfaction:** SDLC focuses on delivering high-quality software that meets customer expectations.

❖ <u>SDLC:</u>

- Software development life Cycle represents the process of developing software.
- ➢ **The stages of SDLC are as follows:**

1.Requirements Gathering and Analysis: In this stage, the project team works closely with stakeholders to gather and document the requirements of the software system.

2.System Design: The system design phase involves translating the requirements into a technical blueprint. This includes defining the system architecture, data models, user interfaces, and software components. The design also considers factors like performance, scalability, security, and maintainability.

3.Implementation or Coding: This is the stage where the actual coding and development of the software take place. Developers write the code based on the design specifications and best coding practices.

4.Testing: The testing stage involves verifying and validating the software to ensure it meets the specified requirements. Different types of testing, such as integration testing, system testing, and user acceptance testing etc.

5.Deployment: Once the software passes the testing phase, it is deployed to the production environment. This involves activities such as software installation, configuration, data migration, and setting up the necessary infrastructure to support the software system.

6.Maintenance: software enters the maintenance phase. During this stage, the software is actively monitored, and any issues or bugs that arise are identified, reported, and fixed.

❖ **SDLC Models:**

- Software Development life cycle (SDLC) is a spiritual model used in project management.
- There are different software development life cycle models specify and design, which are followed during the software development phase.
- These models are also called "Software Development Process Models.

❖ **important phases of SDLC life cycle:**
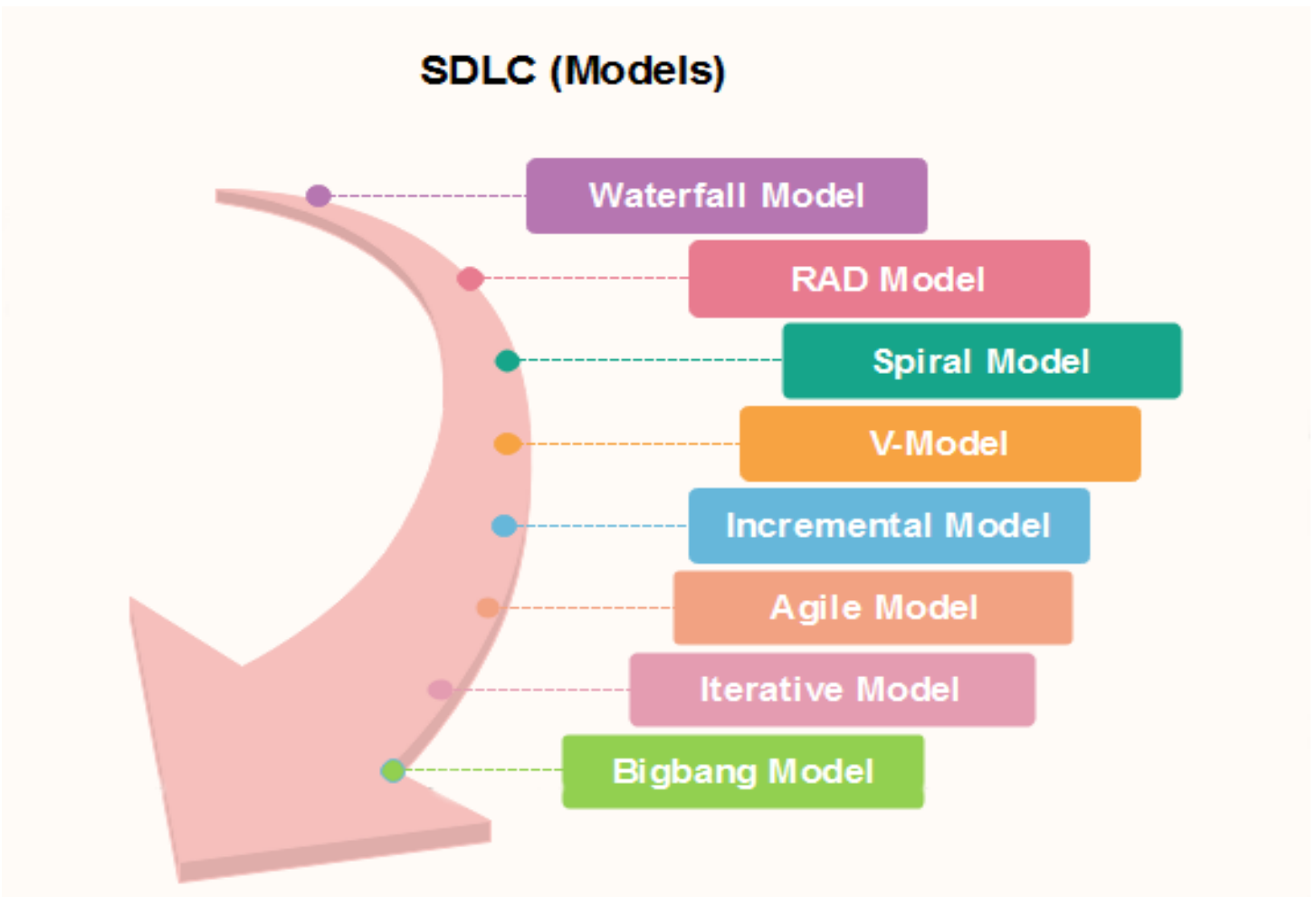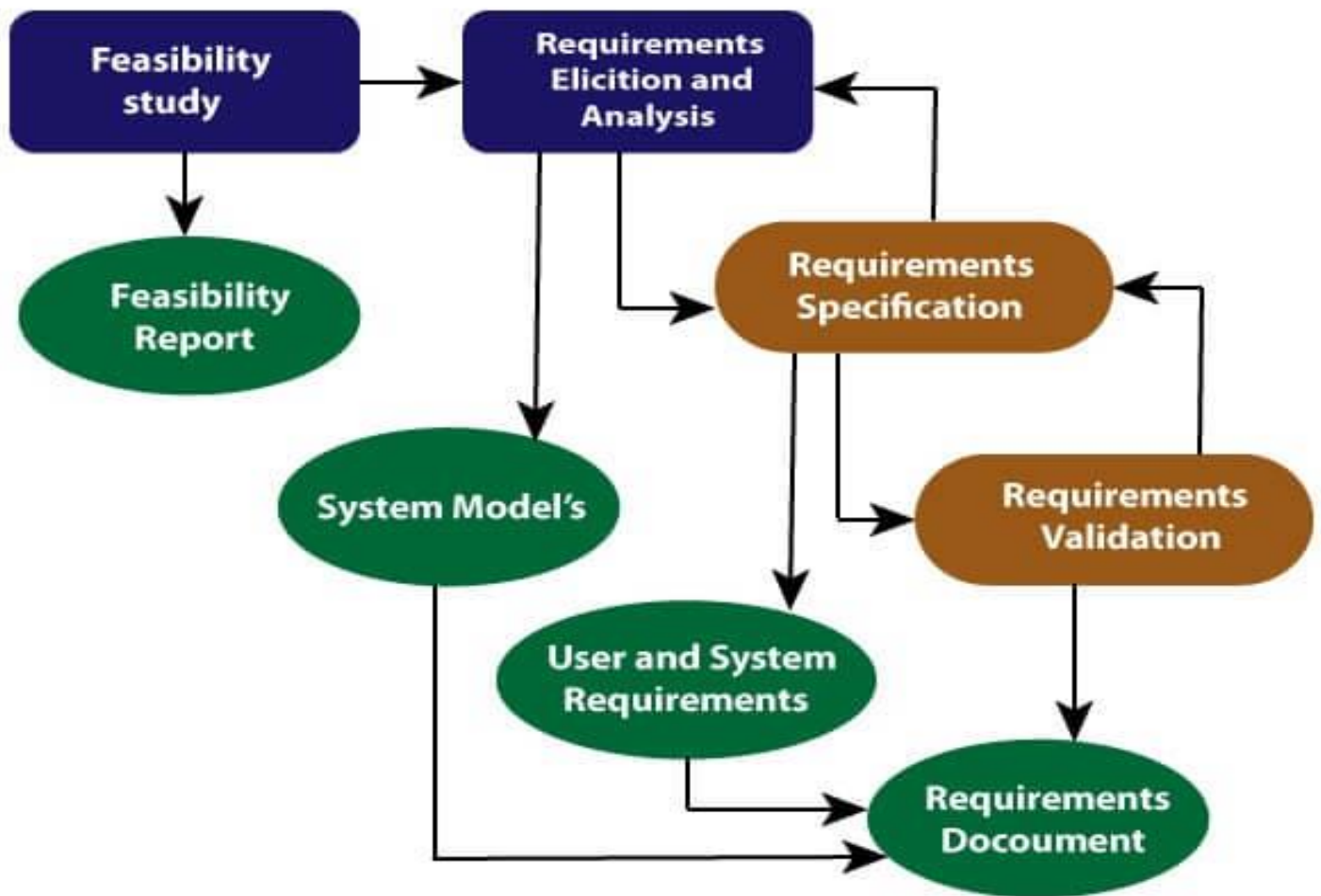

❖ **Requirement Engineering:**

- Requirement's engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process.
- Requirement engineering provides the appropriate mechanism to understand what the customer desires, analysing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.

➢ **Requirement Engineering Process:**

It is a four-step process, which includes:

- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management

Requirement Engineering Process

1. Feasibility Study:

- The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

❖ Types of Feasibility:

1) **Technical Feasibility -** Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2) **Operational Feasibility –** it is assessing the range in which the required software performs a series of levels to solve business problems and customer requirements.
3) **Economic Feasibility -** Economic feasibility decides whether the necessary software can generate financial profits for an organization.

2. Requirement Elicitation and Analysis:

- This is also known as the gathering of requirements.
- requirements are identified with the help of customers and existing systems processes.

❖ **Problems of Elicitation and Analysis:**

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders' express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.

Elicitation and Analysis Process
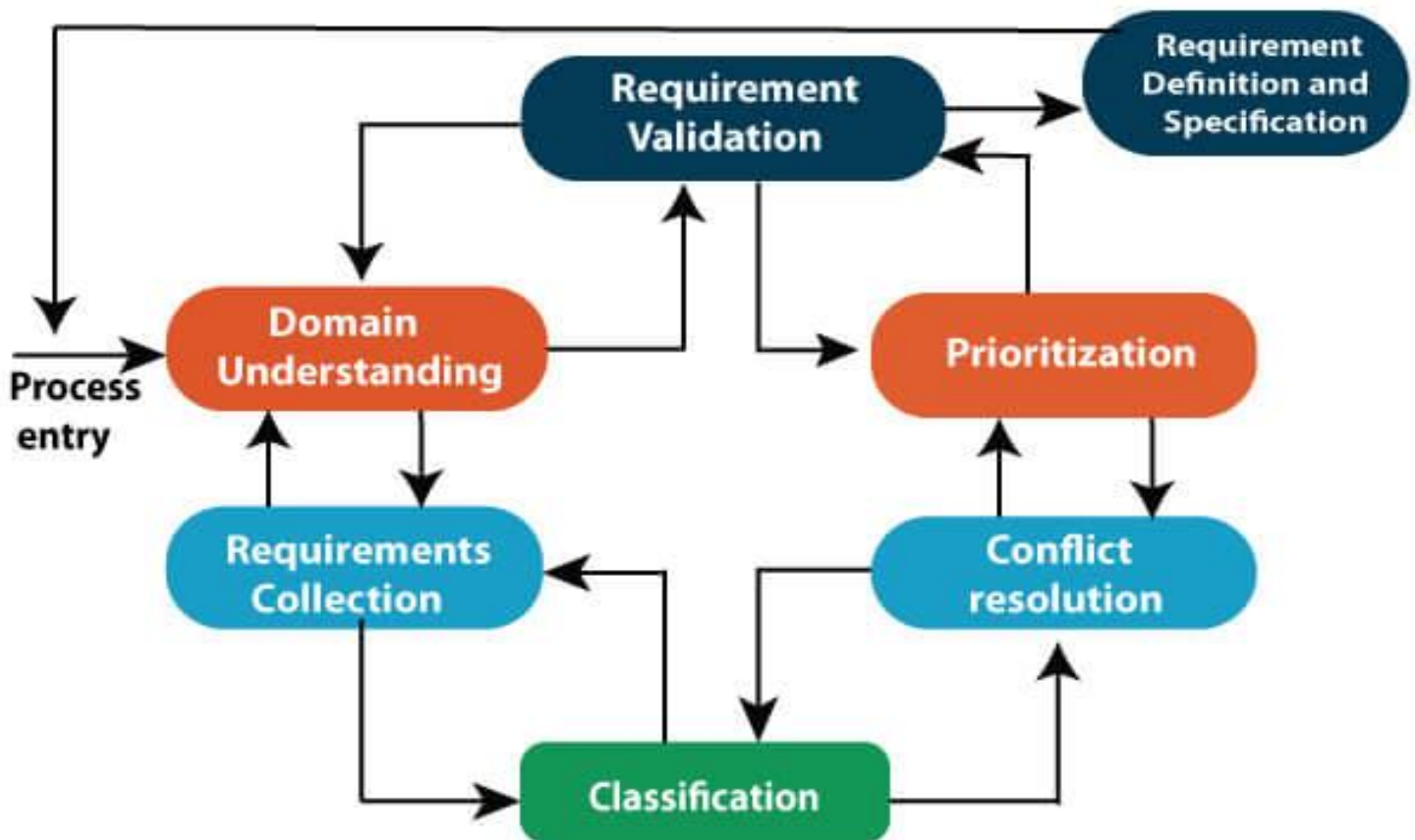
3. Software Requirement Specification:

- Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language.
- The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.
- ➤ **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modelling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.

- ➢ **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- ➢ **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "E-R diagram." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation:

- • After requirement specifications developed, the requirements discussed in this document are validated.
- • Requirements Validation Techniques
- ➢ **Requirement's reviews/inspections**: systematic manual analysis of the requirements.
- ➢ **Prototyping:** Using an executable model of the system to check requirements.
- ➢ **Test-case generation:** Developing tests for requirements to check testability.
- ➢ **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.
- ❖ Software Requirements: Largely software requirements categorized into two categories:
  1. **Functional Requirements:** it is define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements are describing the behaviour of the system as it correlates to the system's functionality.
  2. **Non-functional Requirements:** This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviours of the system.
  - ➢ Non-functional requirements are divided into two main categories:
    1. Execution qualities like security and usability, which are observable at run time.
    2. Evolution qualities like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

❖ SDCL Model:
1. Waterfall Model:

- • The Waterfall model follows a linear sequential approach, where each phase of the software development process flows downward like a waterfall, and progress moves to the next phase only after the previous phase is completed. The phases include:
  - ➢ **Requirements Gathering:** Gathering and documenting user requirements.
  - ➢ **System Design:** Creating the system architecture and detailed design specifications.
  - ➢ **Implementation:** Developing the software components based on the design specifications.
  - ➢ **Testing:** Verifying and validating the software through various testing techniques.
  - ➢ **Deployment:** Deploying the software in the production environment.
  - ➢ **Maintenance:** Maintaining and supporting the software after deployment.
  - ➢ The Waterfall model is well-suited for projects with well-defined requirements and stable environments.

Waterfall Model

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

❖ When to use SDLC Waterfall Model?

➢ Some Circumstances where the use of the Waterfall model is most suited are:

- When the requirements are constant and not changed regularly.
- A project is short
- The situation is calm
- Where the tools and technology used is consistent and is not changing
- When resources are well prepared and are available to use.

❖ Advantages of Waterfall model:

➢ This model is simple to implement also number of resources that are required for it is minimal.

- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.
- It gives easy to control and clarity for the customer due to a strict reporting system.

❖ **Disadvantages of Waterfall model:**

➢ In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.

- This model cannot accept the changes in requirements during development.
- if the application has now shifted to coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

2. Agile Models:

➢ Agile methodologies emphasize iterative and incremental development, collaboration, and flexibility. They focus on delivering working software quickly and adapting to changing requirements. Some popular Agile models include:

- **Scrum:** In Scrum, development occurs in time-boxed iterations called sprints. Each sprint involves planning, development, testing, and review. It promotes close collaboration and regular feedback from stakeholders.
- **Kanban:** Kanban focuses on visualizing and optimizing the flow of work. Tasks are represented on a Kanban board, and work is pulled through the various stages based on available capacity.
- **Lean:** Lean principles aim to eliminate waste and maximize value. It emphasizes continuous improvement, reducing unnecessary steps, and delivering value to customers as efficiently as possible.

➢ Agile models are well-suited for projects with evolving requirements and a need for flexibility and rapid delivery.
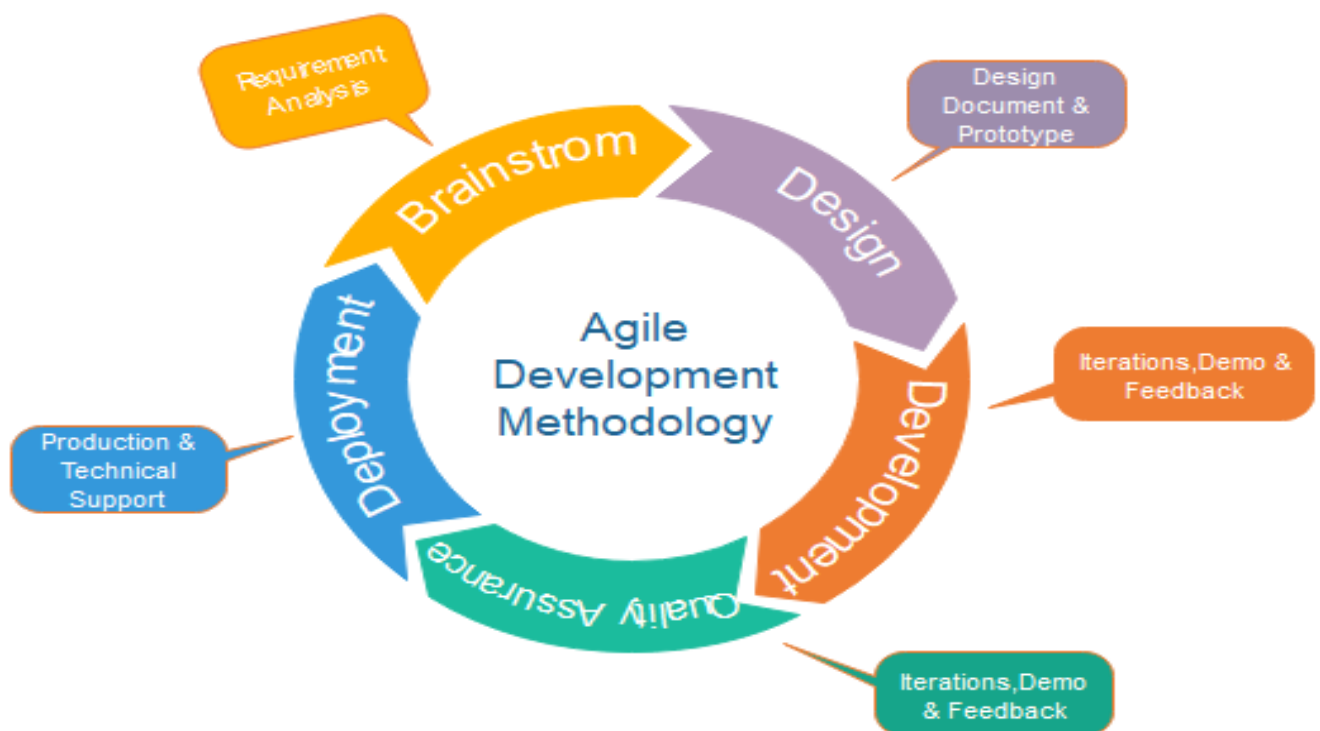
Fig. Agile Model

❖ **Phases of Agile Model:**
- ➢ Following are the phases in the Agile model are as follows:
    1. Requirements gathering
    2. Design the requirements
    3. Construction/ iteration
    4. Testing/ Quality assurance
    5. Deployment
    6. Feedback

1. 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.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements.

3. 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.

4. Testing: the Quality Assurance team examines the product's performance and looks for the bug.

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

6. Feedback: After releasing the product, the last step is feedback.

❖ **Agile Testing Methods:**
1. Scrum
2. Crystal
3. Dynamic Software Development Method(DSDM)
4. Feature Driven Development(FDD)
5. Lean Software Development
6. eXtreme Programming(XP)

❖ **Scrum:**
- ➢ SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.
- • There are three roles in it, and their responsibilities are:
1. **Scrum Master:** The scrum can set up the master team, arrange the meeting and remove obstacles for the process
2. **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
3. **Scrum Team:** team manages its work and organizes the work to complete the sprint or cycle.

❖ **eXtreme Programming(XP):**
- ➢ This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

- • **Crystal:**
- ➢ There are three concepts of this method:
1. **Chartering:** Multi activities are involved in this phase such as making a development team, performing feasibility analysis, developing plans, etc.
2. **Cyclic delivery:** under this, two more cycles consist, these are:
    - • Team updates the release plan.
    - • Integrated product delivers to the users.
3. **Wrap up:** According to user environment, this phase performs deployment, post-deployment.

T3 SKILLS CENTER

❖ **Dynamic Software Development Method (DSDM):**
  ➢ DSDM is a rapid application development strategy for software development and gives an agile project distribution structure. The techniques used in DSDM are:
    1. Time Boxing
    2. MoSCoW Rules
    3. Prototyping

❖ **The DSDM project contains seven stages:**
  - Pre-project
  - Feasibility Study
  - Business Study
  - Functional Model Iteration
  - Design and build Iteration
  - Implementation
  - Post-project

❖ **Feature Driven Development(FDD):**
  - This method focuses on "Designing and Building" features. In contrast to other smart methods, FDD describes the small steps of the work that should be obtained separately per function.

❖ **Lean Software Development:**
  - Lean software development methodology follows the principle "just in time production." The lean method indicates the increasing speed of software development and reducing costs. Lean development can be summarized in seven phases.
    - Eliminating Waste
    - Amplifying learning
    - Defer commitment (deciding as late as possible)
    - Early delivery
    - Empowering the team
    - Building Integrity
    - Optimize the whole
    - When to use the Agile Model?
    - When frequent changes are required.
    - When a highly qualified and experienced team is available.
    - When a customer is ready to have a meeting with a software team all the time.
    - When project size is small.

❖ **Advantage(Pros) of Agile Method:**
  1. Frequent Delivery
  2. Face-to-Face Communication with clients.
  3. Efficient design and fulfils the business requirement.
  4. Anytime changes are acceptable.
  5. It reduces total development time.

❖ **Disadvantages(Cons) of Agile Model:**
  1. 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.
  2. 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.

3. Spiral Model:

- The Spiral model combines elements of both iterative development and risk management. It consists of repeated cycles of risk analysis, requirements gathering, design, development, and testing. Each cycle progresses in a spiral, with increasing levels of functionality and risk mitigation.
- The key steps in the Spiral model are:
  - ➢ **Determine Objectives:** Identify project goals, risks, and alternative solutions.
  - ➢ **Risk Analysis:** Evaluate and mitigate risks through prototyping, simulation, and other techniques.
  - ➢ **Development and Testing:** Develop software components, test them, and obtain feedback.
  - ➢ **Planning:** Review and plan the next iteration based on feedback and risk analysis.
- It is suitable for projects with high complexity, evolving requirements, and significant risks.
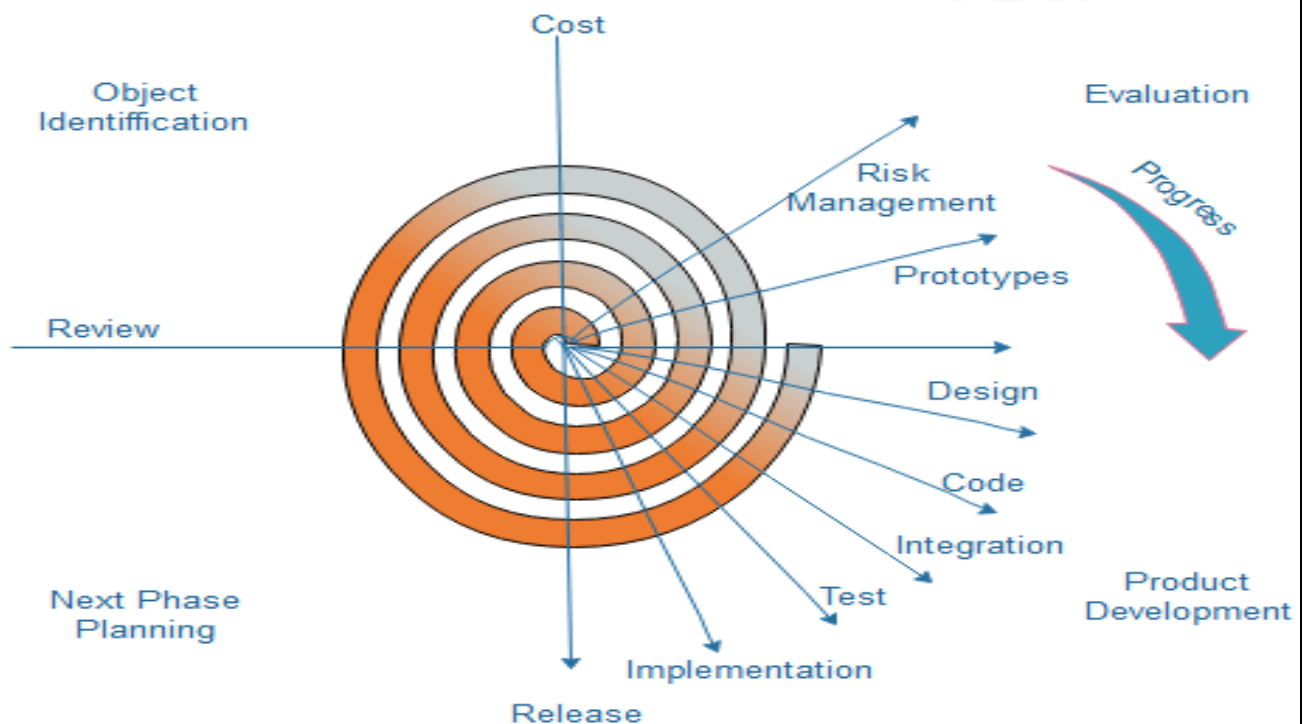
Fig. Spiral Model

- ❖ **When to use Spiral Model?**
  - When deliverance is required to be frequent.
  - When the project is large
  - When requirements are unclear and complex
  - When changes may require at any time
  - Large and high budget projects
- ❖ **Advantages:**
  - High amount of risk analysis
 - Useful for large and mission-critical projects.
- ❖ **Disadvantages:**
  - Can be a costly model to use.
  - Risk analysis needed highly particular expertise
  - Doesn't work well for smaller projects.

4. V-Model:

- The V-Model is an extension of the Waterfall model that emphasizes the relationship between each development phase and its corresponding testing phase. It follows a sequential approach, but testing is planned in parallel with each development phase

1. **Requirements Gathering:** Gather and document user requirements.
2. **System Design:** Create the system architecture and detailed design specifications.
3. **Unit Testing:** Verify individual components through unit testing.
4. **Integration Testing:** Test the integration and interaction of software components.
5. **System Testing:** Conduct end-to-end testing to ensure the system functions as a whole.
6. **User Acceptance Testing:** Validate the software against user requirements.

➢ The V-Model ensures that testing activities are well-aligned with the corresponding development phase, reducing the risk of missed defects.
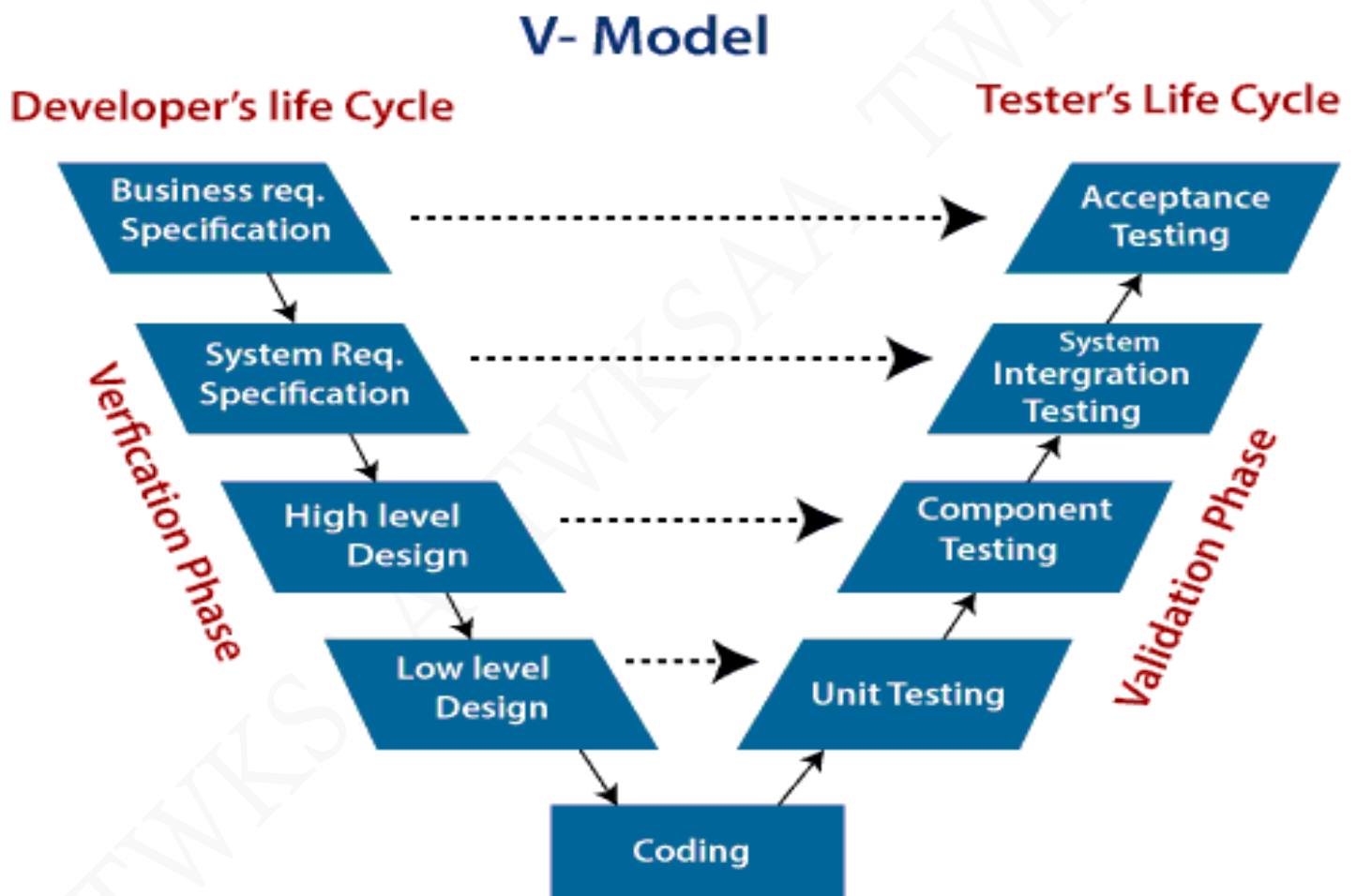
- **Verification:** It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.
- **Validation:** It involves dynamic analysis method (functional, non-functional), testing is done by executing code. 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.

❖ **When to use V-Model?**
- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

❖ **Advantage (Pros) of V-Model:**
- Easy to Understand.
- Testing Methods like planning, test designing happens well before coding.
- This saves a lot of time. Hence a higher chance of success over the waterfall model.
- Avoids the downward flow of the defects.
- Works well for small plans where requirements are easily understood.

❖ **Disadvantage (Cons) of V-Model:**
- Very rigid and least flexible.
- Not a good for a complex project.
- Software is developed during the implementation stage, so no early prototypes of the software are produced.
- If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

5. Iterative and Incremental Models:

- Iterative and Incremental models involve breaking down the software development process into smaller iterations or increments. The software evolves and grows with each iteration, with user feedback incorporated along the way. Examples include the Incremental Model and the Rational Unified Process (RUP).
- The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

❖ **When to use the Iterative Model?**
- When requirements are defined clearly and easy to understand.
- When the software application is large.
- When there is a requirement of changes in future.

❖ **Advantage (Pros) of Iterative Model:**
- Testing and debugging during smaller iteration is easy.
- A Parallel development can plan.
- It is easily acceptable to ever-changing needs of the project.
- Risks are identified and resolved during iteration.
- Limited time spent on documentation and extra time on designing.

❖ **Disadvantage (Cons) of Iterative Model:**
- It is not suitable for smaller projects.
- More Resources may be required.
- Design can be changed again and again because of imperfect requirements.
- Requirement changes can cause over budget.
- Project completion date not confirmed because of changing requirements.
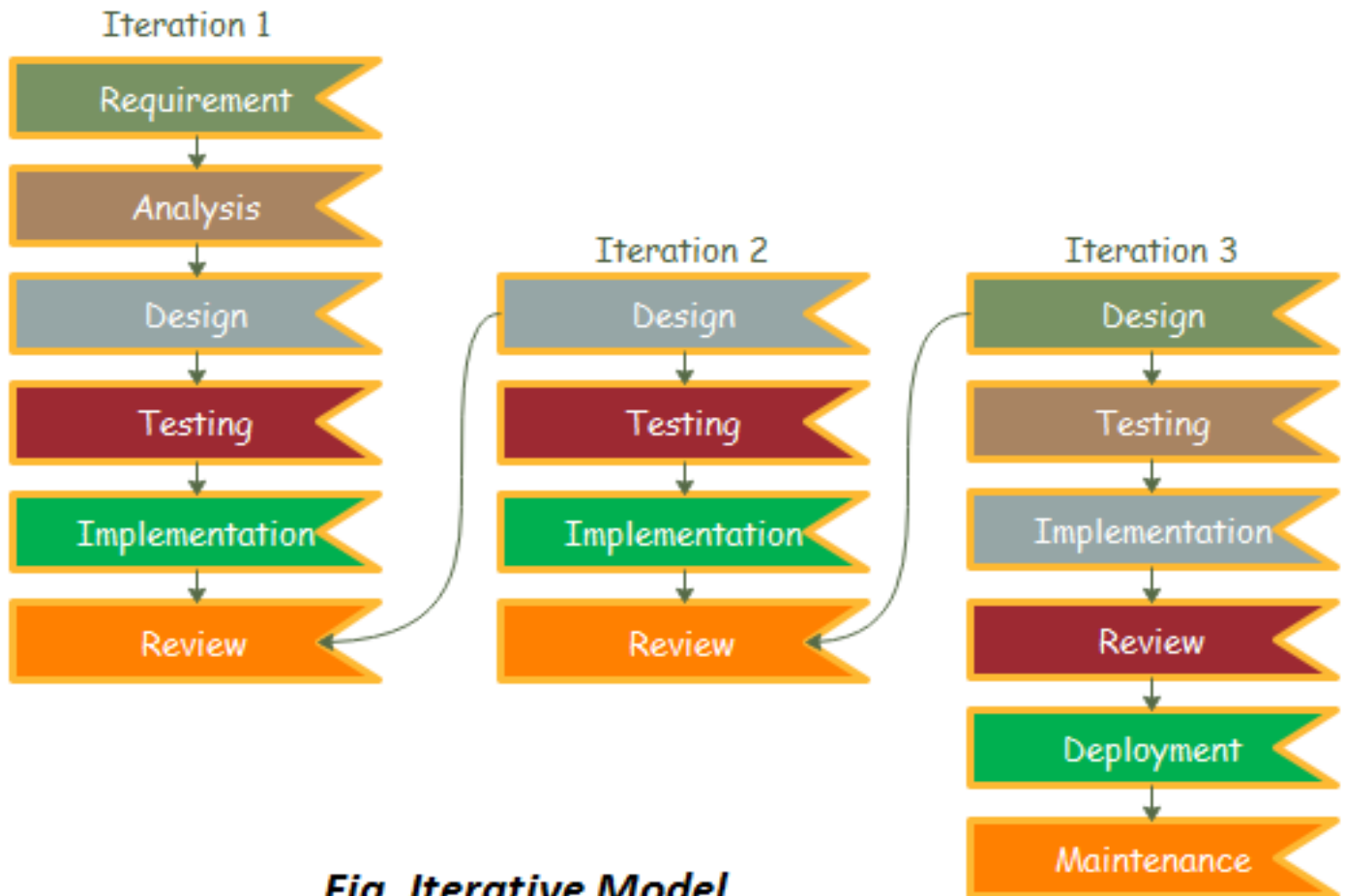
Fig. Iterative Model

6. RAD Model:

- RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period. The RAD model is based on the concept that a better system can be developed in lesser time by using focus groups to gather system requirements.
- Business Modelling
- Data Modelling
- Process Modelling
- Application Generation
- Testing and Turnover

❖ When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

❖ **Advantage of RAD Model:**
- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.
- It reduced development time.
- It increases the reusability of features.

❖ **Disadvantage of RAD Model:**
- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.
- Required user involvement.

7. Big Bang Model:
- developers do not follow any specific process. Development begins with necessary funds and efforts in the form of inputs. And the result may or may not be as per customer's requirement, because in this model, even the customer requirements are not defined.
- This model is ideal for small projects like academic projects or practical projects. One or two developers can work together on this model.

❖ **When to use Big Bang Model?**
- As we discussed above, this model is required when this project is small like an academic project or a practical project. This method is also used when the size of the developer team is small and when requirements are not defined, and the release date is not confirmed

❖ **Advantage (Pros) of Big Bang Model:**
- There is no planning required.
- Simple Model.
- Few resources required.
- Easy to manage.
- Flexible for developers.

❖ **Disadvantage (Cons) of Big Bang Model:**
- There are high risk and uncertainty.
- Not acceptable for a large project.
- If requirements are not clear that can cause very expensive.
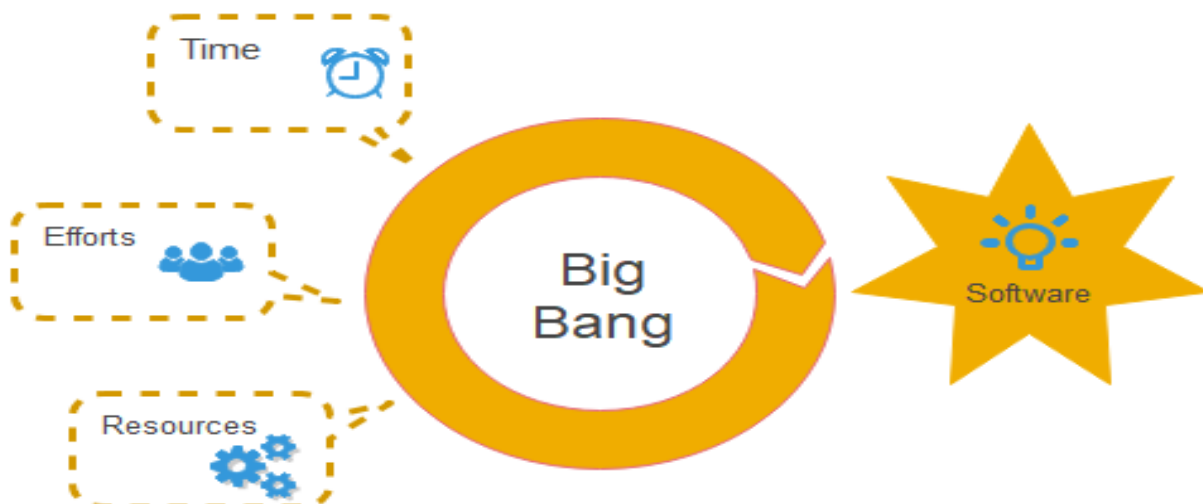
Fig. Big Bang Model

8. Prototype Model:

- The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software.

Fig: Prototype Model
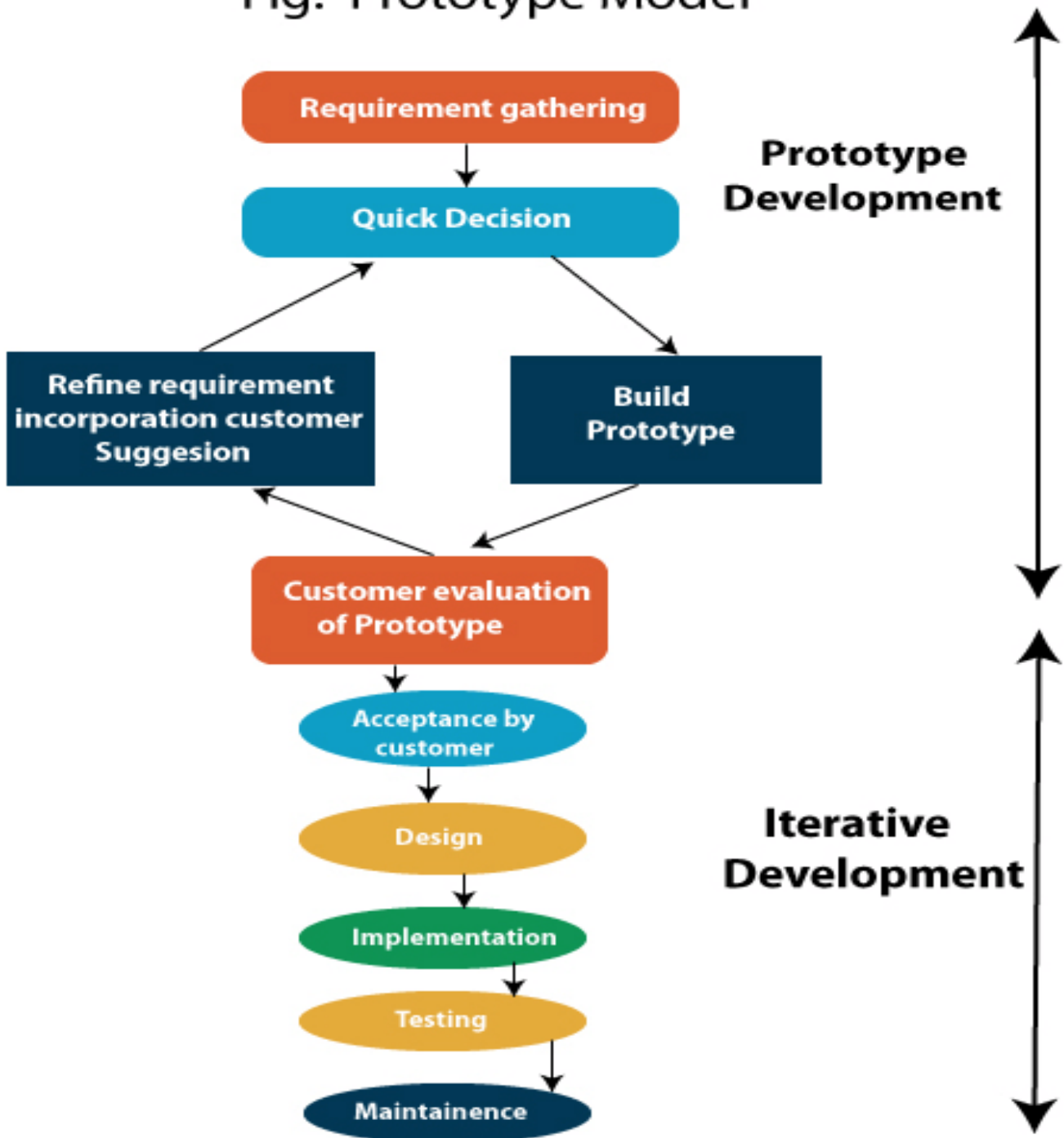
❖ **Steps of Prototype Model:**
- Requirement Gathering and Analyst
- Quick Decision
- Build a Prototype
- Assessment or User Evaluation
- Prototype Refinement
- Engineer Product

❖ **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
- Costs customer money
- Needs committed customer
- Difficult to finish if customer withdraw
- May be too customer specific, no broad market
- 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.
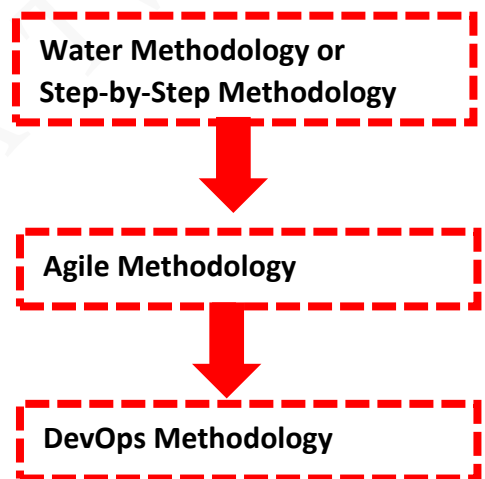- It is a time-consuming process.

9) DevOps:

DevOps: - the term DevOps is a combination of two words i.e. Development and operations.

- DevOps is a Methodology that allows a single team to manage the entire application development life cycle. The development, testing, deployment and operations.
- Object of DevOps is to shorten the system's development life cycle.
- DevOps is a software development approach through which superior quality software can be developed quickly and with more reliability.
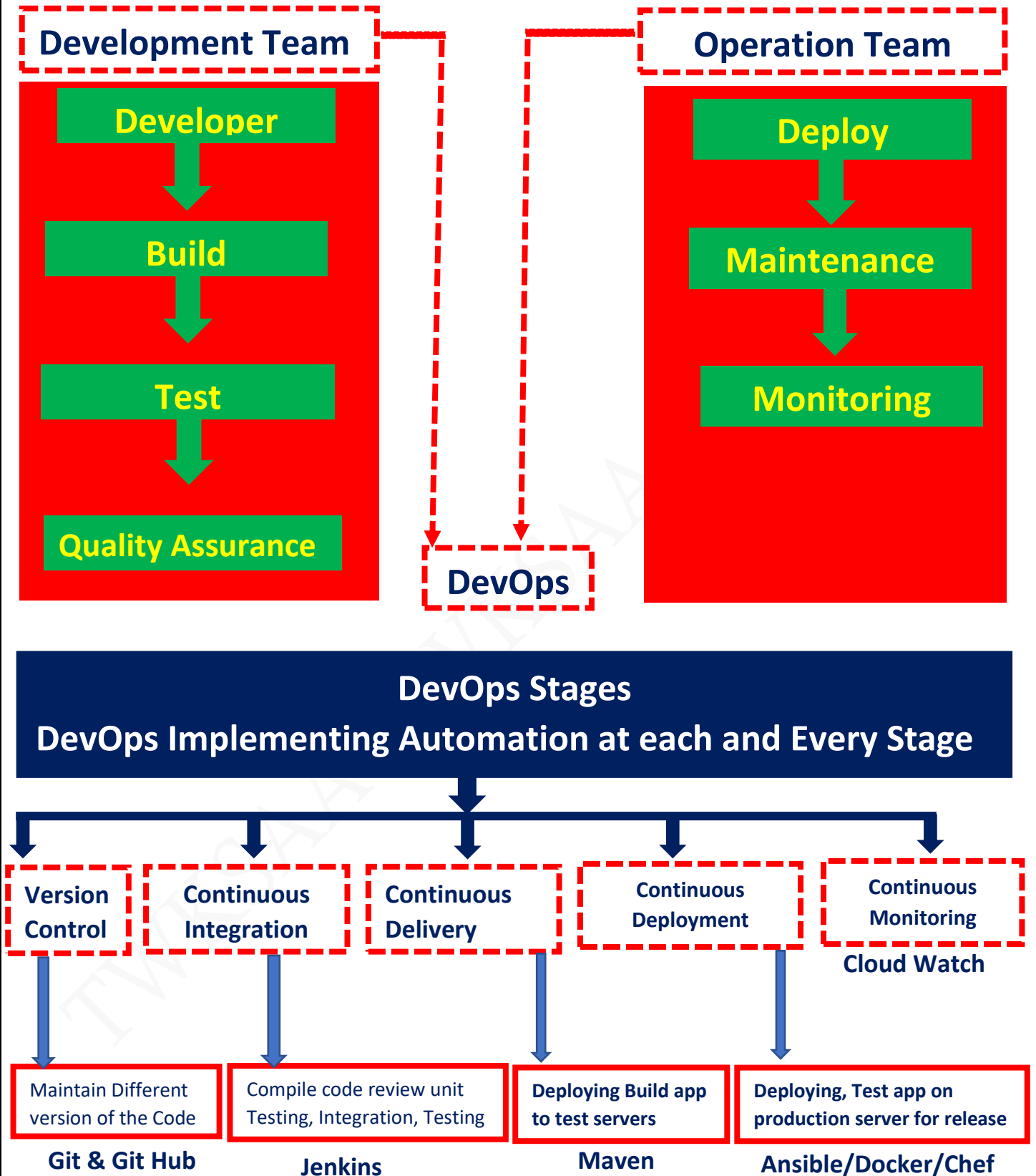
Why Organization Needs DevOps Engineer.

1. Fast Delivery
2. Higher Quality
3. Less Capex+Opex
4. Reduced Outages
5. High Availability
6. Scabble, Flexible & Reliable

SDCL: - Software Development Life Cycle.

Developer

↓

Build

↓

Test

↓

Quality Assurance

↓

Deploy

↓

Maintenance

↓

Monitoring

DevOps

Water Methodology or Step-by-Step Methodology

↓

Agile Methodology

↓

DevOps Methodology

Software Development Evolution

Waterfall | Agile | DevOps

Development Team

Developer → Build → Test → Quality Assurance

Operation Team

Deploy → Maintenance → Monitoring

DevOps

DevOps Stages
DevOps Implementing Automation at each and Every Stage

Version Control	Continuous Integration	Continuous Delivery	Continuous Deployment	Continuous Monitoring

Cloud Watch

Maintain Different version of the Code	Compile code review unit Testing, Integration, Testing	Deploying Build app to test servers	Deploying, Test app on production server for release

Git & Git Hub   **Jenkins**   **Maven**   **Ansible/Docker/Chef**

T3 SKILLS CENTER

DevOps Architecture Features

1. Automation
2. Collaboration
3. Integration
4. Configuration Management

DevOps Life Cycle

1. Continuous Development
2. Continuous Integration
3. Continuous Deployment
4. Continuous Testing
5. Continuous Monitoring
6. Continuous Feedback

DevOps Workflow

1. Sequential Job Execution
2. Parallel Job Execution
3. Branch Level Filtering
4. Fan -in /out in Repo

DevOps Principles

1. End to End responsibility
2. Continuous Improvement
3. Automate Everything
4. Custom Centric Action
5. Monitor And Test Everything
6. Works As One Team

DevOps Tools

1.Git 2. Jenkins 3. Maven 4. Docker 5. Kubernetes 6. Chef 7. Ansible 8. Puppet 9. Sensu 10. Salt Stack 11. Bamboo 12. Jira 13. Selenium 14. Nagios 15. Splunk 16. App Dynamic

DevOps Pipeline CI/CD

1. Source Control
2. Build Tools
3. Containerization
4. Configuration Management
5. Monitoring
6. Feedback

DevOps Methodology

1. Teams
2. Connectivity
3. Automation
4. On- Boarding Process
5. Project Environment
6. Shared Service
7. Naming Conventions
8. Defining Standards Role Across Team
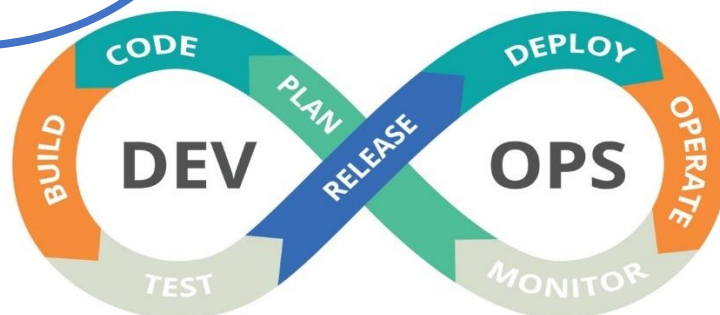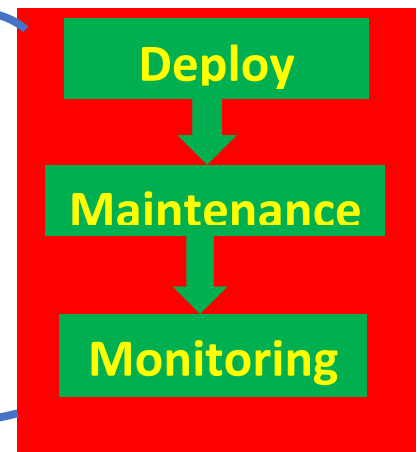
DevOps Automation Tools

1. Infrastructure Automation (AWS)
2. Configuration Management (chef)
3. Deployment Automation (Jenkins)
4. Performance Management (App Dynamic)
5. Log Management (Splunk)
6. Monitoring (Nagios)

Development Team

Operation Team

Developer

Build

Test

Quality

Integration

Deploy

Maintenance

Monitoring

	Waterfall	Agile	DevOps
Basic philosophy	Systems are fully predictable and can be specified in advance. Assumes business needs remain broadly similar throughout project. Adjust schedule to preserve scope	Integrate business, dev and QA for rapid delivery of software. Iterative 'sprint' cycles. Assumes priority of business needs may change. Adjust scope to preserve schedule	Cross-functional teams utilize automation to enable continuous deployment of change. Constant feedback loop. Adjust scope to preserve schedule
Documentation level	Comprehensive	Light	Light
Automation level	Low	Varied	High
Delivery of value	Slow – only at major milestones (3-6 months)	Rapid (daily/weekly)	Continuous
Business ownership of project?	No (typical)	Yes	Yes
Response to new business needs (flexible requirements)	Extremely limited due to detailed specification	Responsive – iterative delivery enables prioritization	Highly responsive – cross-functional teams define business needs more precisely
Collaboration	Low – teams operate in functional silos	Improved – business is highly engaged, short dev cycles	High – all stakeholders involved from project start
Quality	Low – issues not identified until testing phase.	Improved – issues identified after every 'sprint'	High – automated unit testing during development
Risk	Increases as project progresses	Decreases as project progresses	Decreases as project progresses
Customer feedback	Infrequent - at project completion	Frequent – after every sprint	Continuous

T3 SKILLS CENTER

➢ DevOps (Development and Operations) is a software development approach that emphasizes close collaboration and integration between development teams and operations teams. DevOps aims to streamline and automate the software development process, enabling faster and more reliable delivery of software solutions. The DevOps software development life cycle (SDLC) incorporates principles, practices, and tools to facilitate continuous integration, continuous delivery, and continuous deployment. Here are the key stages in the DevOps SDLC:

1. Continuous Planning:

- In this stage, development and operations teams collaborate to define project goals, prioritize features, and plan development sprints or cycles. This involves creating a backlog of tasks, estimating effort, and setting development timelines.

2. Continuous Development:

- The continuous development stage involves writing code, implementing features, and integrating software components. Developers use version control systems to manage code changes, collaborate on shared repositories, and ensure code quality through code reviews. Automated build tools and Continuous Integration (CI) systems are used to build and test the software continuously.

3. Continuous Testing:

- In DevOps, testing is integrated throughout the development process. Automated testing tools and frameworks are utilized to perform unit tests, integration tests, and end-to-end tests. Continuous testing helps in identifying issues and bugs early, allowing for faster feedback and rapid bug fixes.

4. Continuous Integration (CI):

- Continuous Integration involves regularly integrating code changes from multiple developers into a shared repository. CI systems automatically build and test the software with each code commit, ensuring that the changes do not introduce conflicts or regressions. This promotes early detection of integration issues and enables rapid feedback and resolution.

5. Continuous Deployment:

- Continuous Deployment automates the release and deployment of software to various environments, such as development, staging, and production. Automated deployment pipelines are created, which encompass activities like configuration management, environment provisioning, and release orchestration. This ensures consistency, repeatability, and traceability in the deployment process.

6. Continuous Monitoring:

- Continuous Monitoring involves collecting real-time data on the performance, availability, and usage of the software system. Monitoring tools and techniques are employed to track system metrics, log events, and detect anomalies. This helps in identifying performance bottlenecks, system failures, and potential issues that require immediate attention.

7. Continuous Feedback and Improvement:

- DevOps encourages continuous feedback loops to foster collaboration and improvement. Feedback is obtained from stakeholders, users, and operational monitoring, allowing for insights into the software's performance and user experience. This feedback informs further development iterations, bug fixes, and enhancements, enabling continuous improvement of the software system.

T3 SKILLS CENTER

❖ **What is Project?**
- A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal.

❖ **What is software project management?**
- Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.

❖ **Project Manager:**
- A project manager is a character who has the overall responsibility for the planning, design, execution, monitoring, controlling and closure of a project. A project manager represents an essential role in the achievement of the projects.
- A project manager is a character who is responsible for giving decisions, both large and small projects.

❖ **Role of a Project Manager:**

1. Leader
- A project manager must lead his team and should provide them direction to make them understand what is expected from all of them.

2. Medium:
- The Project manager is a medium between his clients and his team. He must coordinate and transfer all the appropriate information from the clients to his team and report to the senior management.

3. Mentor:
- He should be there to guide his team at each step and make sure that the team has an attachment. He provides a recommendation to his team and points them in the right direction.

❖ **Responsibilities of a Project Manager:**
- Managing risks and issues.
- Create the project team and assigns tasks to several team members.
- Activity planning and sequencing.
- Monitoring and reporting progress.
- Modifies the project plan to deal with the situation.

❖ **Activities:**
- Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

➢ **The list of activities are as follows:**
1. Project planning and Tracking
2. Project Resource Management
3. Scope Management
4. Estimation Management
5. Project Risk Management
6. Scheduling Management
7. Project Communication Management
8. Configuration Management

❖ **Software Project Planning:**

- A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.

❖ **Need of Software Project Management:**

- Software development is a sort of all new streams in world business, and there's next to no involvement in structure programming items. Most programming items are customized to accommodate customer's necessities.

❖ **Software Project Manager:**

- Software manager is responsible for planning and scheduling project development. They manage the work to ensure that it is completed to the required standard. They monitor the progress to check that the event is on time and within budget.
- To plan a successful software project, we must understand:
  - ➢ Scope of work to be completed
  - ➢ Risk analysis
  - ➢ The resources mandatory
  - ➢ The project to be accomplished
  - ➢ Record of being followed

❖ **Software Project planning starts before technical work start. The various steps of planning activities are:**
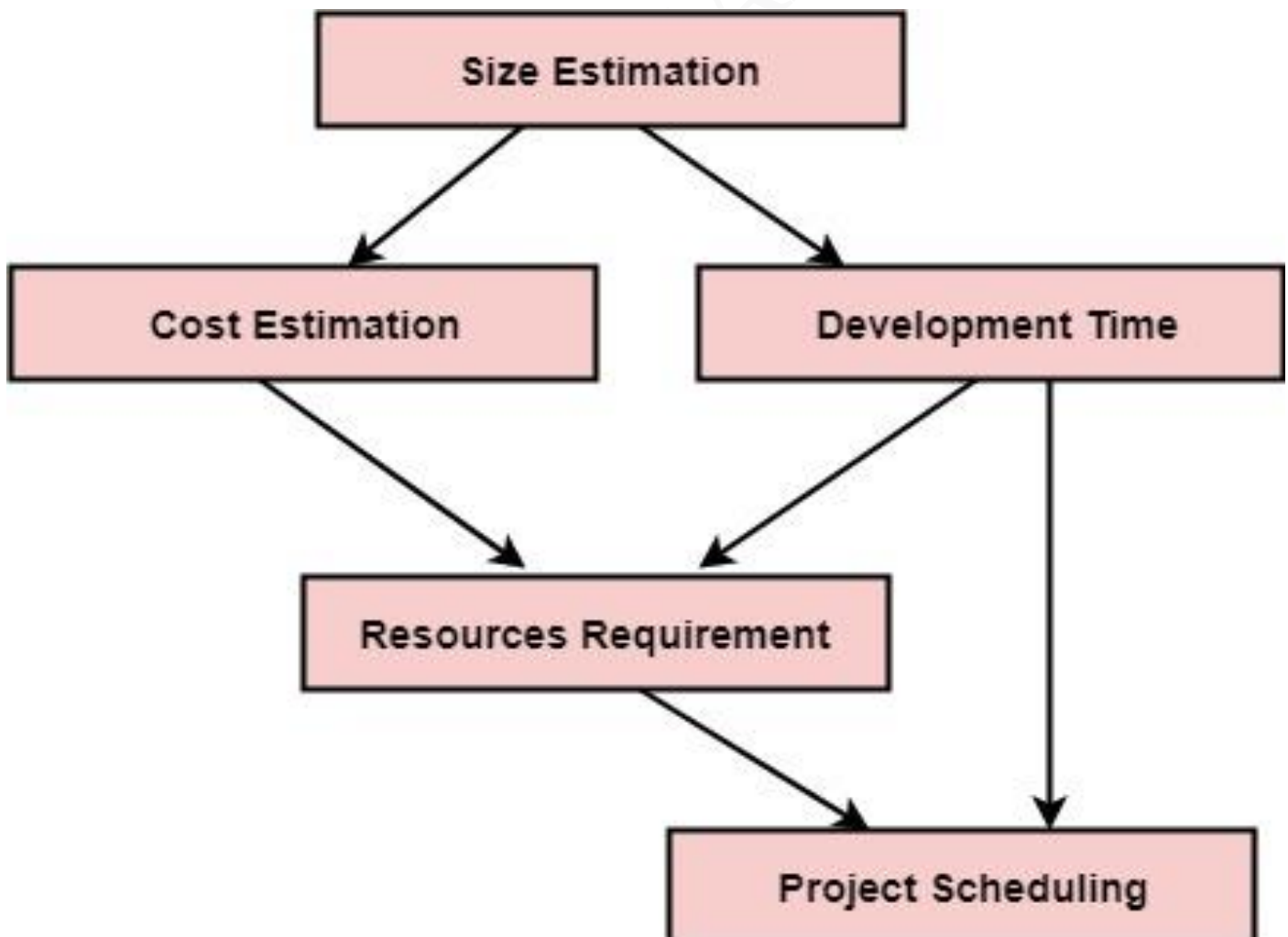
❖ **Software Requirement Specifications:**

- The production of the requirements stage of the software development process is Software Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analysed. SRS is a formal report.

❖ **Characteristics of good SRS:**
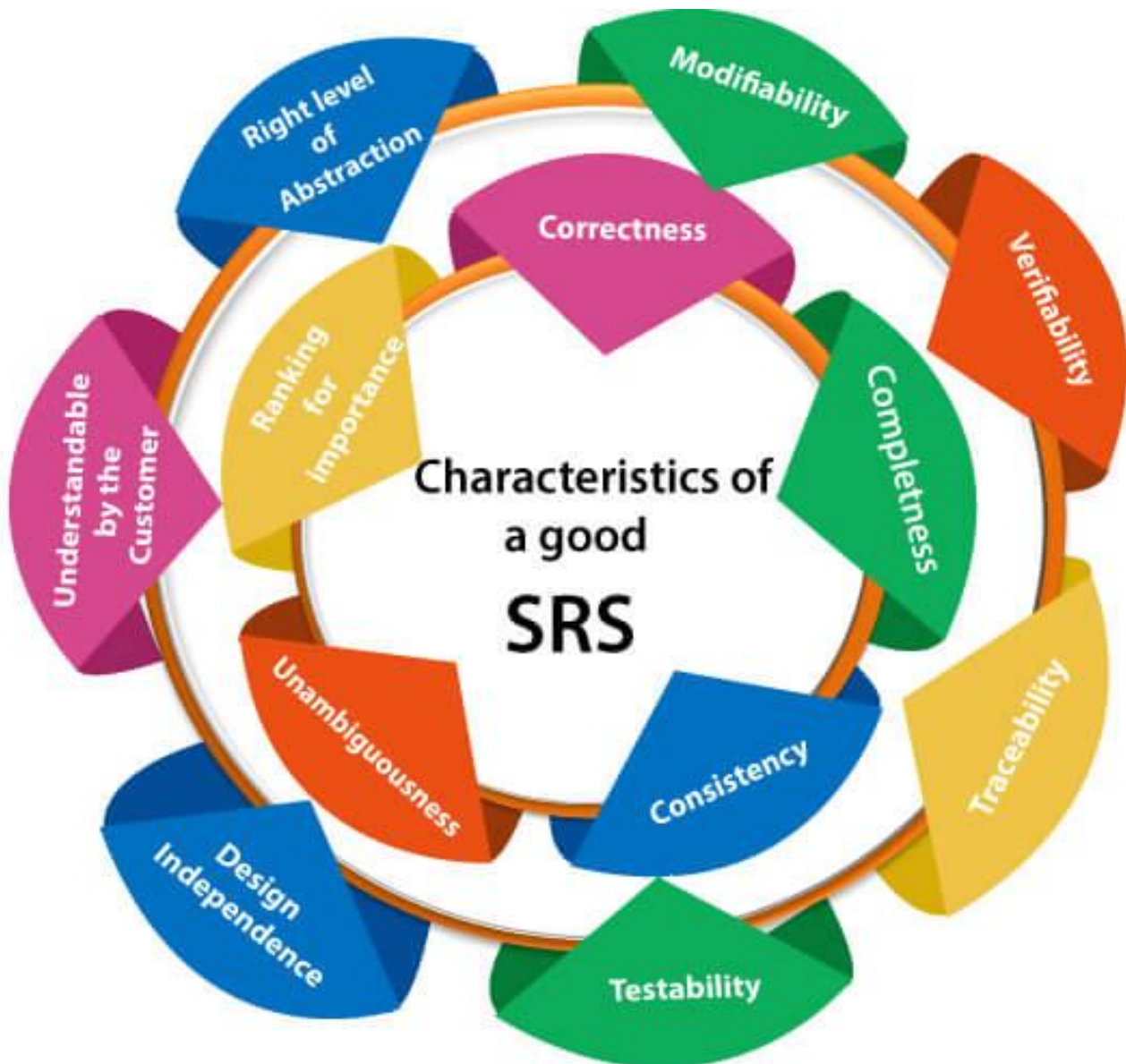
❖ **Software Configuration Management:**

- The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

❖ **Why do we need Configuration Management?**

- Multiple people are working on software which is consistently updating. It may be a method where multiple versions, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently.

❖ **Importance of SCM:**

- It is practical in controlling and managing the access to various SCIs e.g., by preventing the two members of a team for checking out the same component for modification at the same time.

❖ **SCM Process:**

- It uses the tools which keep that the necessary change has been implemented adequately to the appropriate component. The SCM process defines a number of tasks:

➢ **Identification of objects in the software configuration**

- Version Control
- Change Control
- Configuration Audit
- Status Reporting

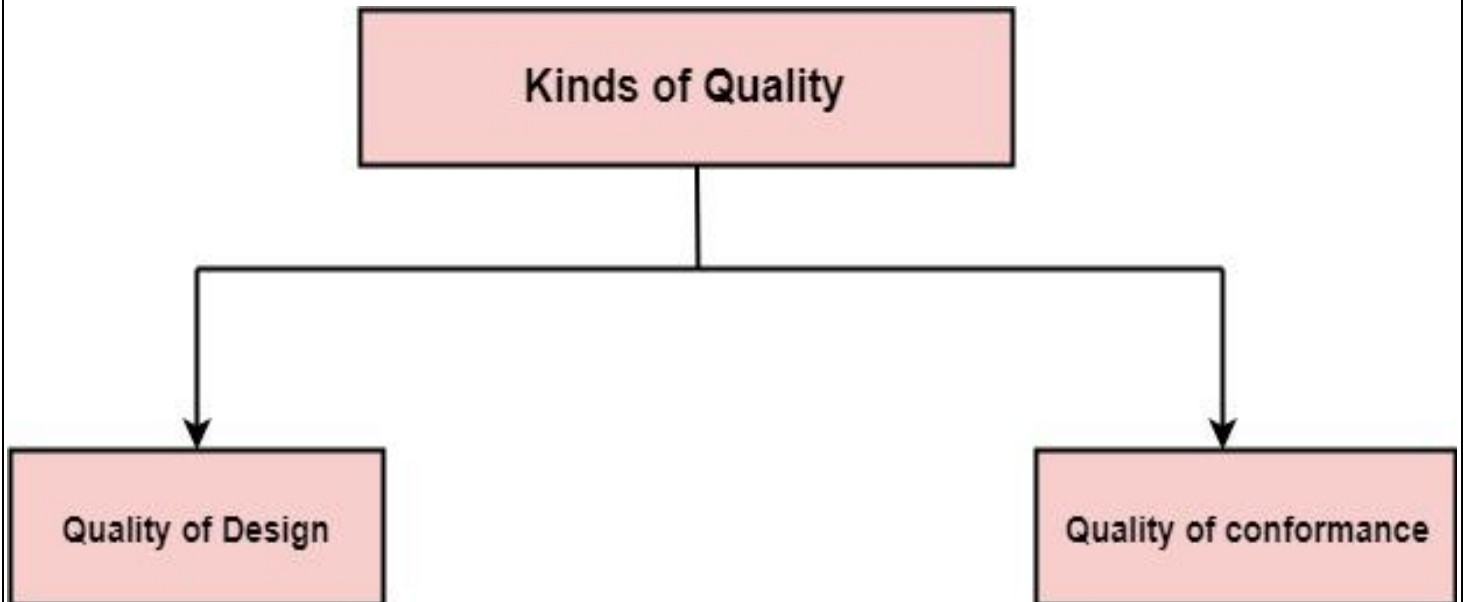## Software Configuration Management Process
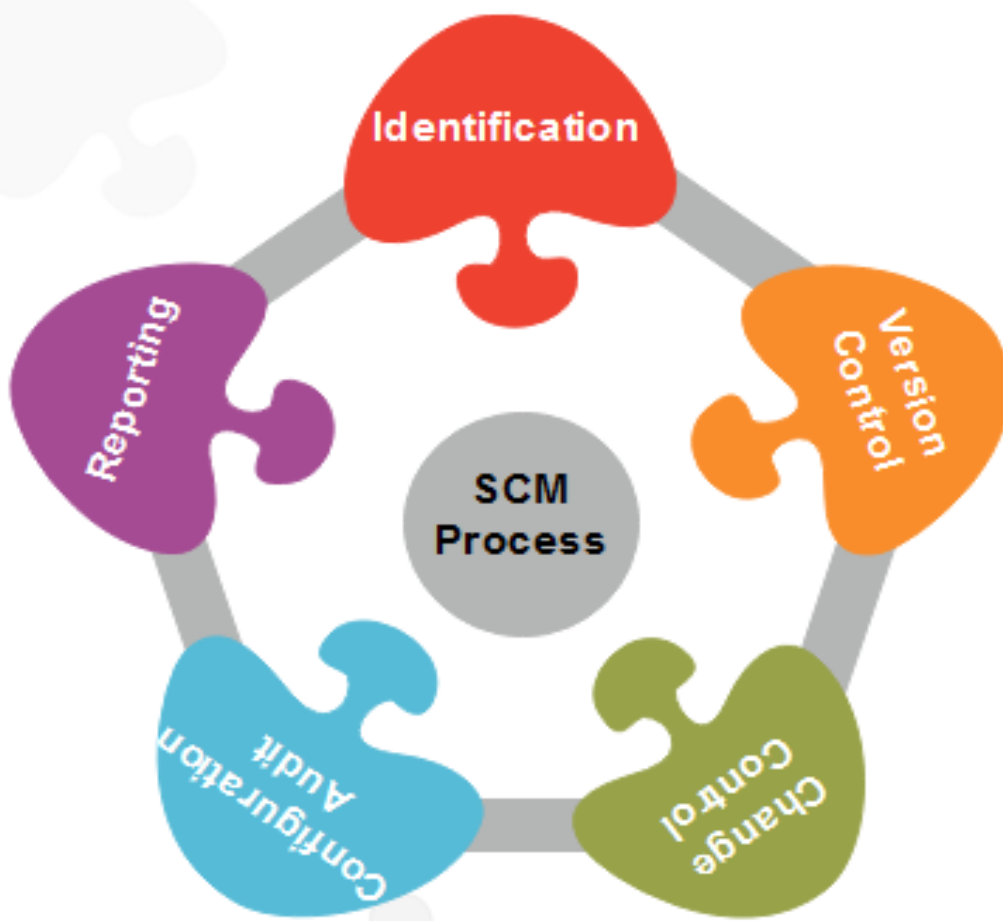
T3 SKILLS CENTER

❖ **Software Quality Assurance:**

➢ **What is Quality?**

- Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.

➢ **There are two kinds of Quality:**

- **Quality of Design:** Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.
- **Quality of conformance:** Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.
- **Software Quality:** Software Quality is defined as the conformance to explicitly state functional and performance requirements, explicitly documented development standards, and inherent characteristics that are expected of all professionally developed software.
- **Quality Control:** Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements place upon it. Quality control includes a feedback loop to the process that created the work product.
- **Quality Assurance:** Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.

❖ **Software Quality Assurance:**

- Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.
- A set of activities designed to calculate the process by which the products are developed or manufactured.

T3 SKILLS CENTER

❖ **Project Monitoring and Control:**

- Monitoring and Controlling are processes needed to track, review, and regulate the progress and performance of the project. It also identifies any areas where changes to the project management method are required and initiates the required changes.
- The Monitoring & Controlling process group includes eleven processes, which are:
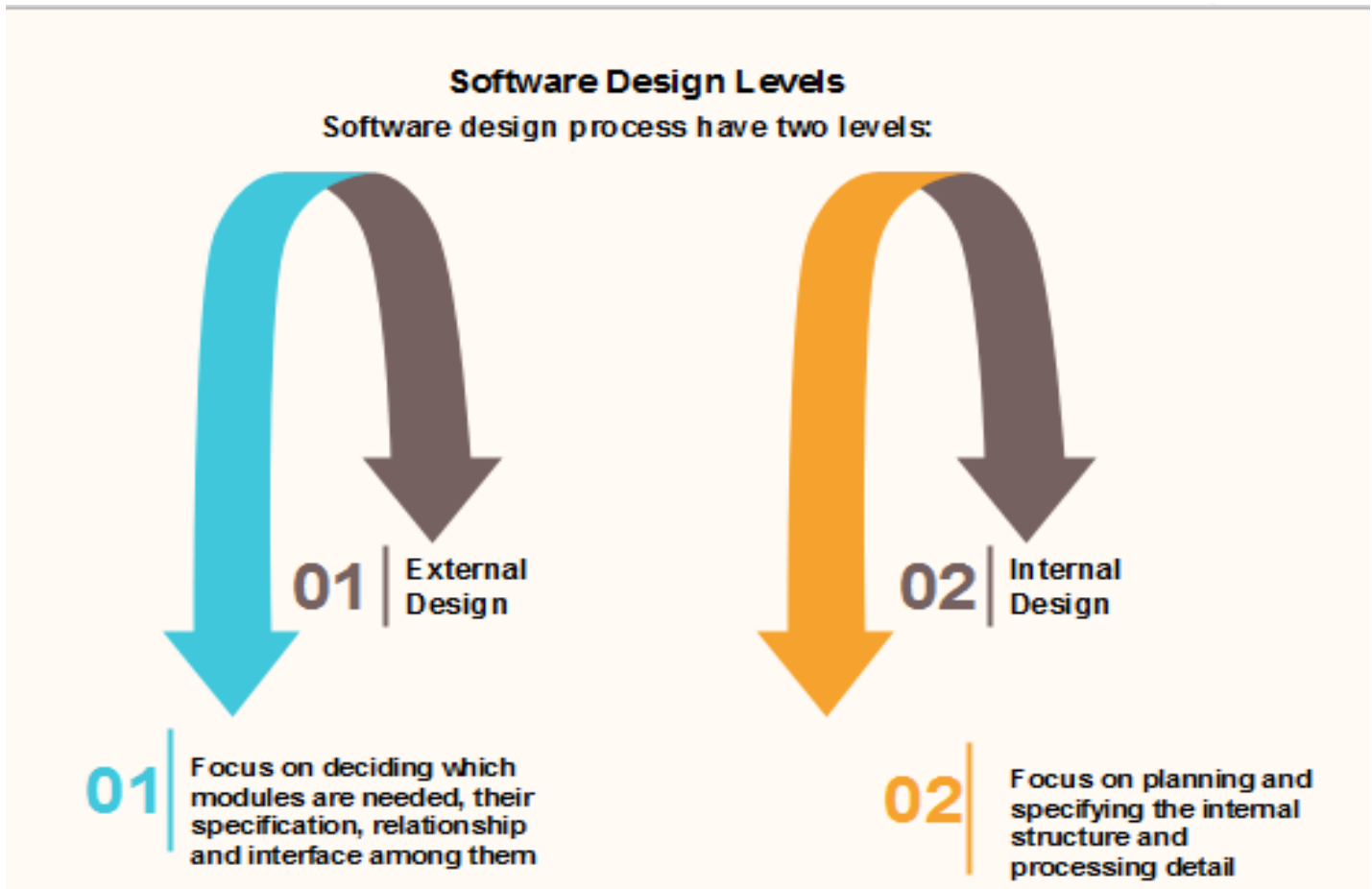

Monitoring and Controlling Process
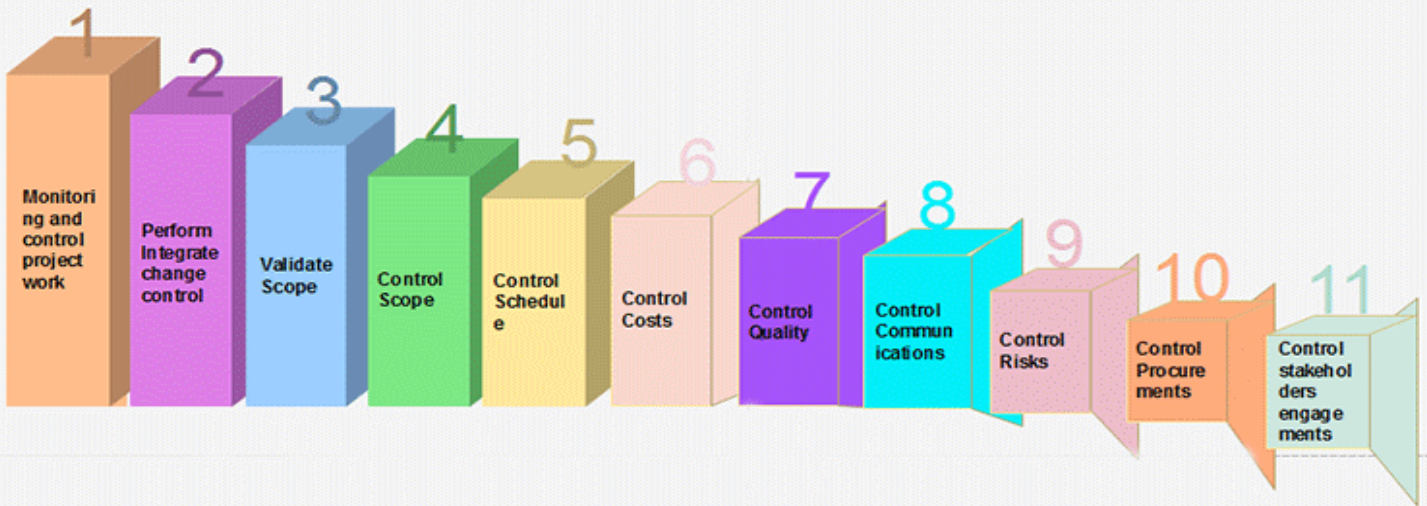
- **Monitor and control project work:** The generic step under which all other monitoring and controlling activities fall under.
- **Perform integrated change control:** The functions involved in making changes to the project plan. When changes to the schedule, cost, or any other area of the project management plan are necessary, the program is changed and re-approved by the project sponsor.
- **Validate scope:** The activities involved with gaining approval of the project's deliverables.
- **Control scope:** Ensuring that the scope of the project does not change and that unauthorized activities are not performed as part of the plan (scope creep).
- **Control schedule:** The functions involved with ensuring the project work is performed according to the schedule, and that project deadlines are met.
- **Control costs:** The tasks involved with ensuring project costs stay within the approved budget.
- **Control quality:** Ensuring that the quality of the projects deliverables is to the standard defined in the project management plan.
- **Control communications:** Providing for the communication needs of each project stakeholder.
- **Control Risks:** Safeguarding the project from unexpected events that negatively impact the project's budget, schedule, stakeholder needs, or any other project success criteria.
- **Control procurements:** Ensuring project's subcontractors and vendors meet the project goals.
- **Control stakeholder engagement:** The tasks involved with ensuring that all of the project's stakeholders are left satisfied with the project work.

❖ Software Design:

- Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.
- The software design phase is the first step in SDLC (Software Design Life Cycle), which moves the concentration from the problem domain to the solution domain.

Software Design Levels
Software design process have two levels:

01 External Design

01 Focus on deciding which modules are needed, their specification, relationship and interface among them

02 Internal Design

02 Focus on planning and specifying the internal structure and processing detail

❖ Objectives of Software Design:

➢ Following are the purposes of Software design:

1. **Correctness:** Software design should be correct as per requirement.
2. **Completeness:** The design should have all components like data structures, modules, and external interfaces, etc.
3. **Efficiency:** Resources should be used efficiently by the program.
4. **Flexibility:** Able to modify on changing needs.
5. **Consistency:** There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

❖ **Software Design Principles:**

- Software design principles are concerned with providing means to handle the complexity of the design process effectively.

➢ **Following are the principles of Software Design:**
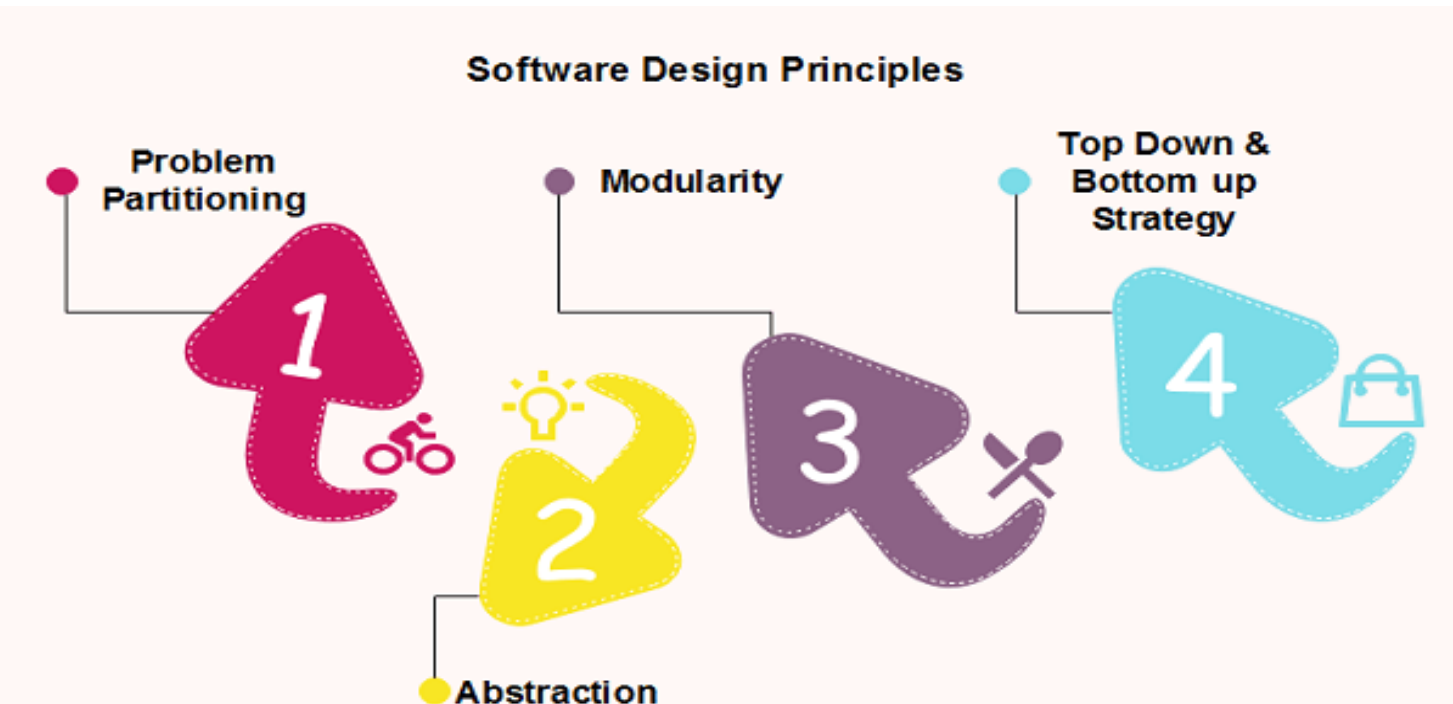
❖ **Problem Partitioning:**

- For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

➢ For software design, the goal is to divide the problem into manageable pieces.

❖ **Benefits of Problem Partitioning:**

- Software is easy to understand
- Software becomes simple
- Software is easy to test
- Software is easy to modify
- Software is easy to maintain
- Software is easy to expand

❖ **Strategy of Design:**

- A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs.

➢ To design a system, there are two possible approaches:

- Top-down Approach
- Bottom-up Approach

1. Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.

2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.
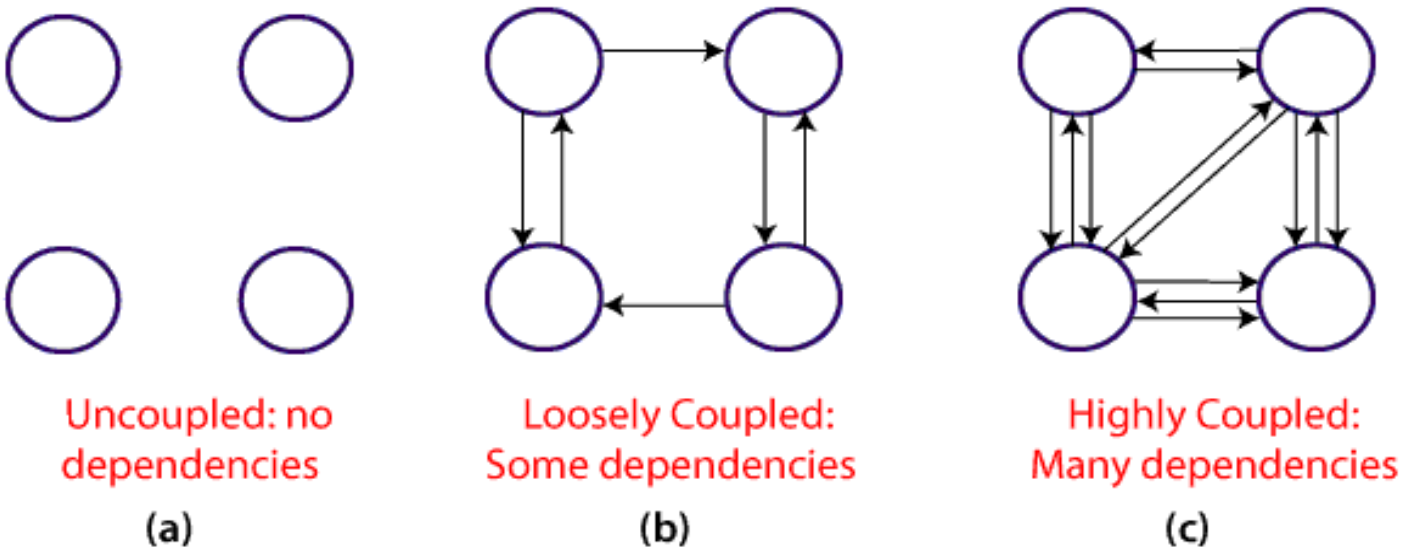
T3 SKILLS CENTER

❖ **Coupling and Cohesion:**

➢ Module Coupling:

- In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. Uncoupled modules have no interdependence at all within them.

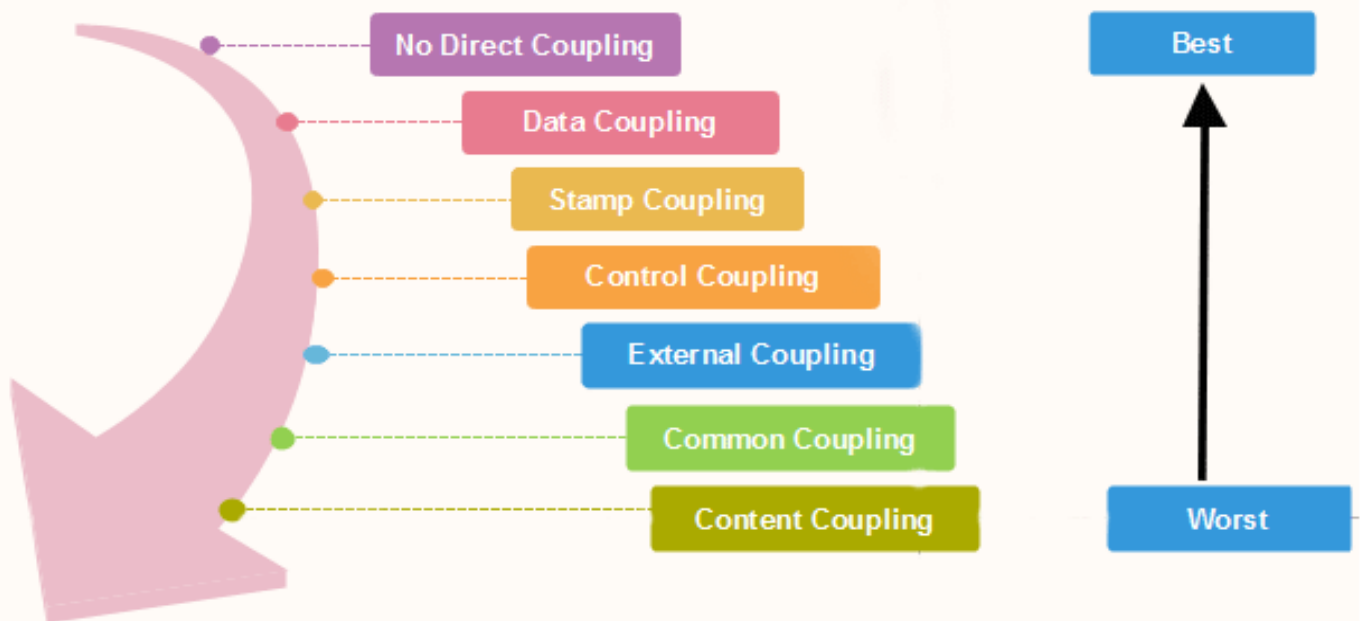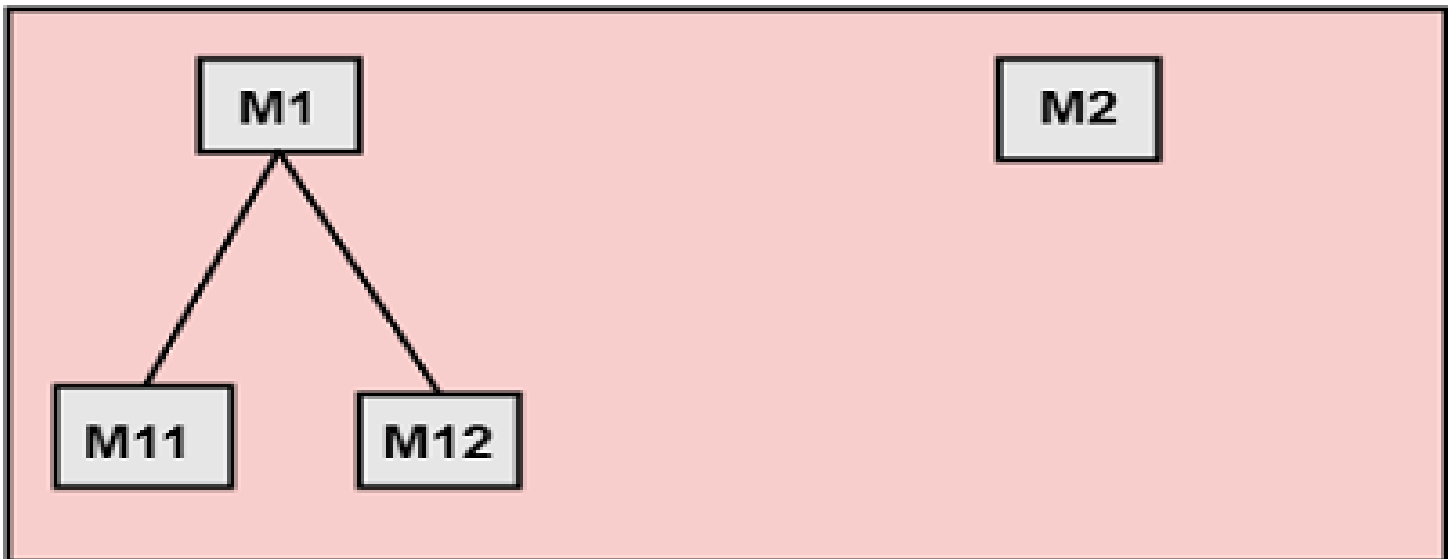- The various types of coupling techniques are shown in fig:

Module Coupling

Uncoupled: no dependencies (a)

Loosely Coupled: Some dependencies (b)

Highly Coupled: Many dependencies (c)

❖ **Types of Module Coupling:**

Types of Modules Coupling

There are various types of module Coupling are as follows:

- No Direct Coupling
- Data Coupling
- Stamp Coupling
- Control Coupling
- External Coupling
- Common Coupling
- Content Coupling

Best

Worst

1. **No Direct Coupling:** There is no direct coupling between M1 and M2.

➤ In this case, modules are subordinates to different modules. Therefore, no direct coupling.
2. **Data Coupling:** When data of one module is passed to another module, this is called data coupling.
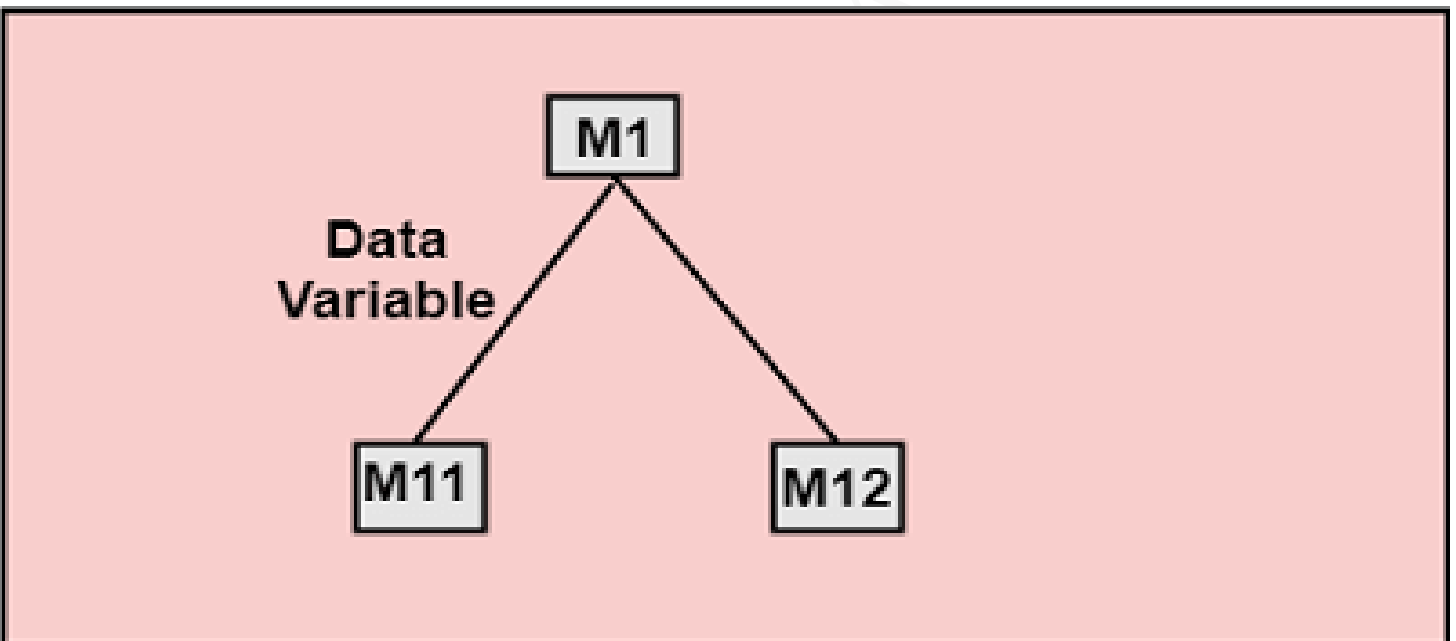
3. **Stamp Coupling:** Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.
4. **Control Coupling:** Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.
5. **External Coupling:** External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

6. **Common Coupling:** Two modules are common coupled if they share information through some global data items.
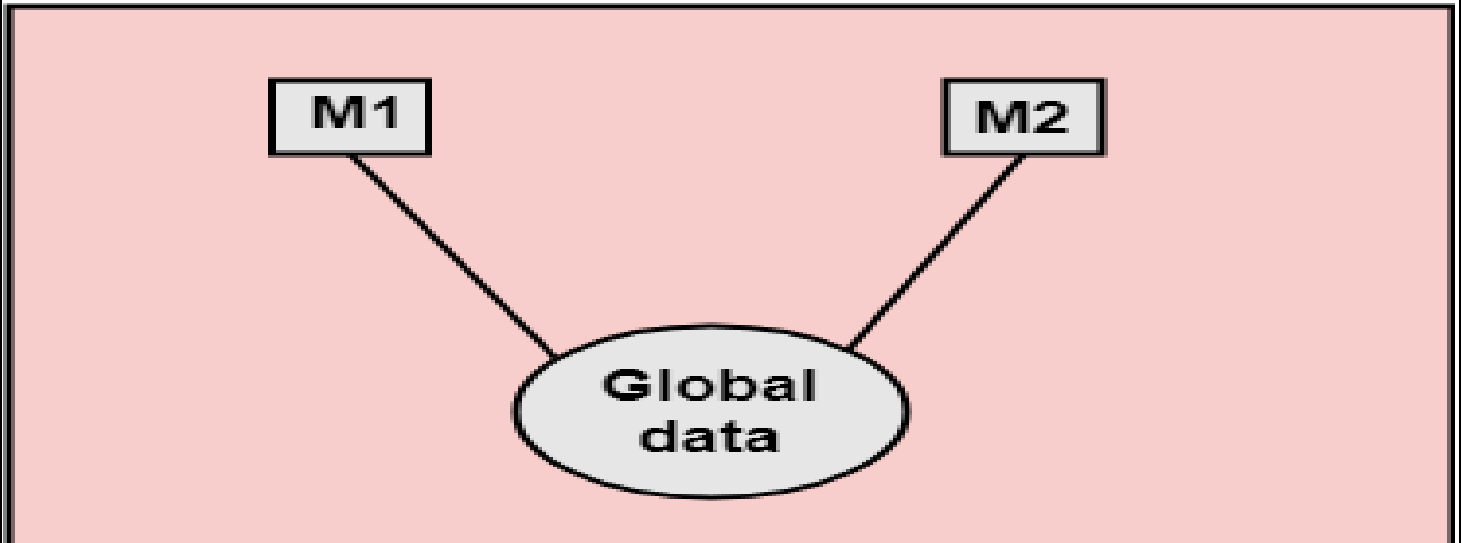
7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

❖ **Module Cohesion:**

* In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module.
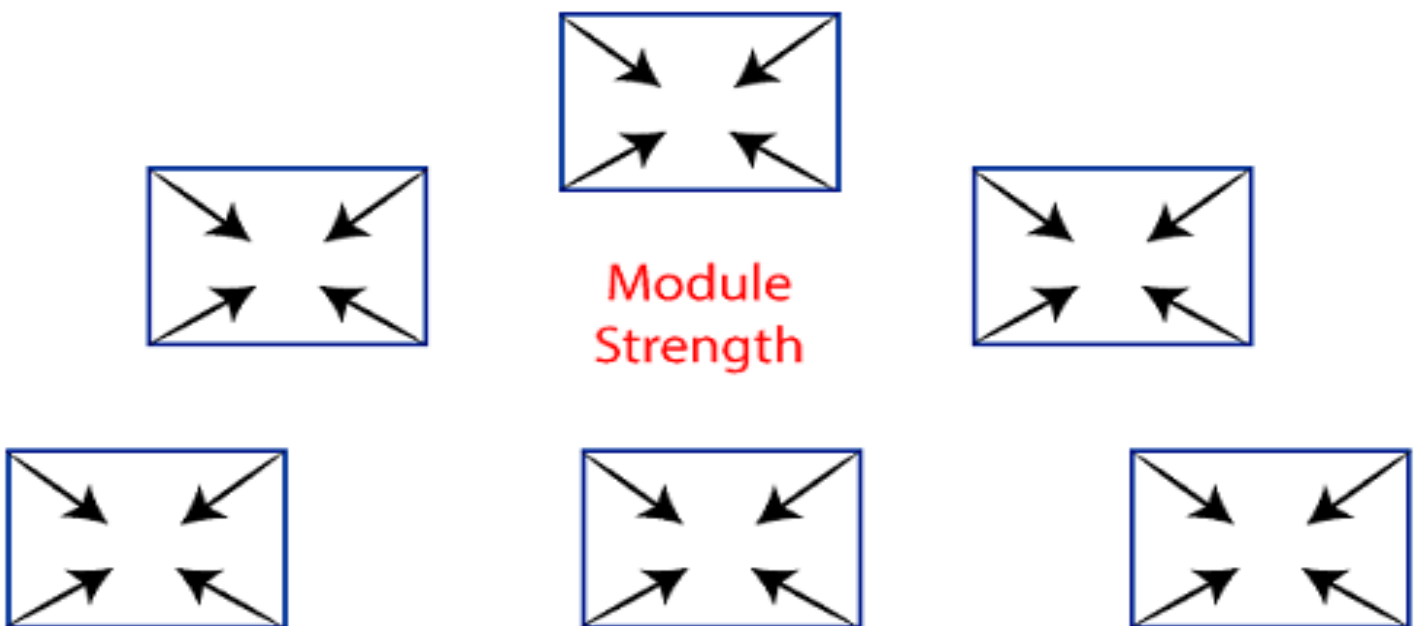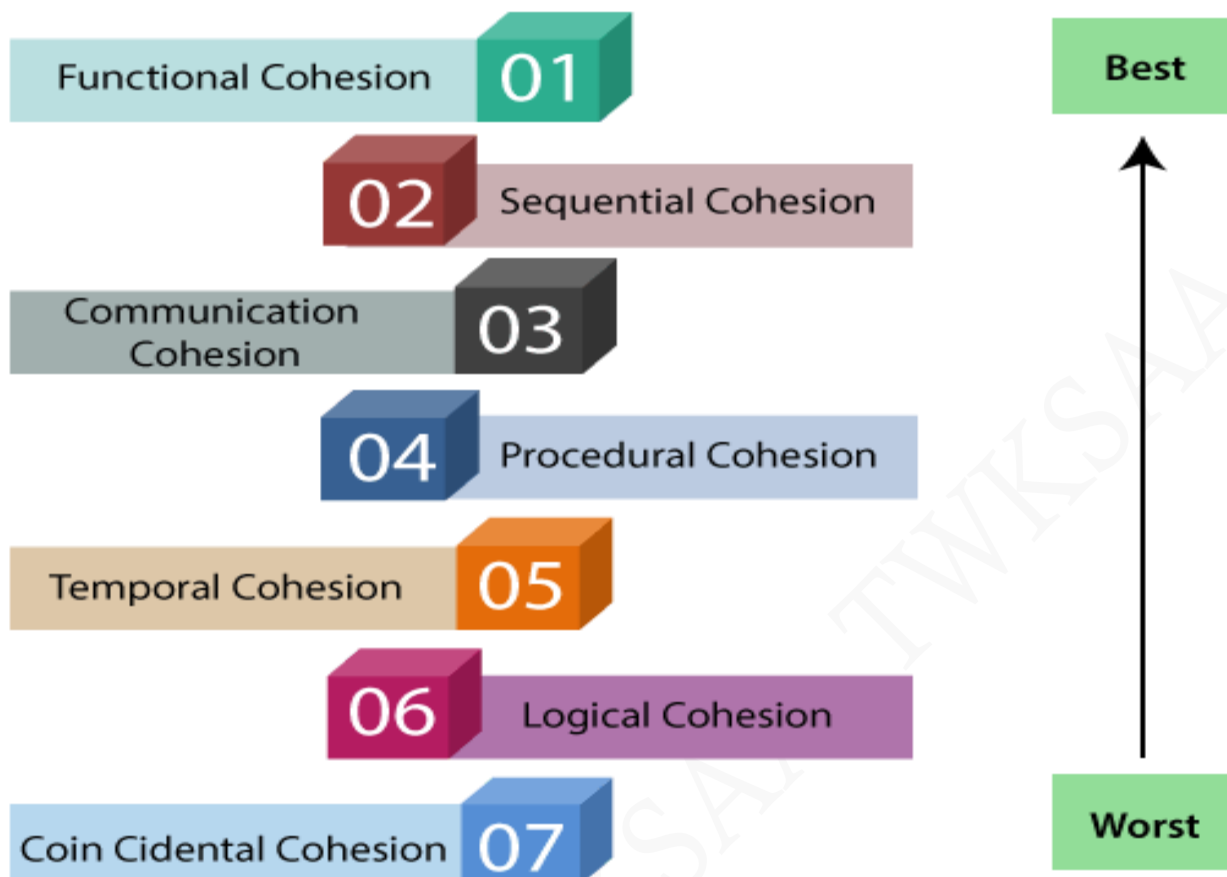

Cohesion= Strength of relations within Modules

➢ Cohesion is an ordinal type of measurement and is generally described as "high cohesion" or "low cohesion."

Types of Modules Cohesion

1) **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.

2) **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.

3) **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

4) **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.

5) **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

6) **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.

7) **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

T3 SKILLS CENTER

❖ **Function Oriented Design:**

- Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function.

  ➢ **Design Notations:**

- Design Notations are primarily meant to be used during the process of design and are used to represent design or design decisions. For a function-oriented design, the design can be represented graphically or mathematically by the following:
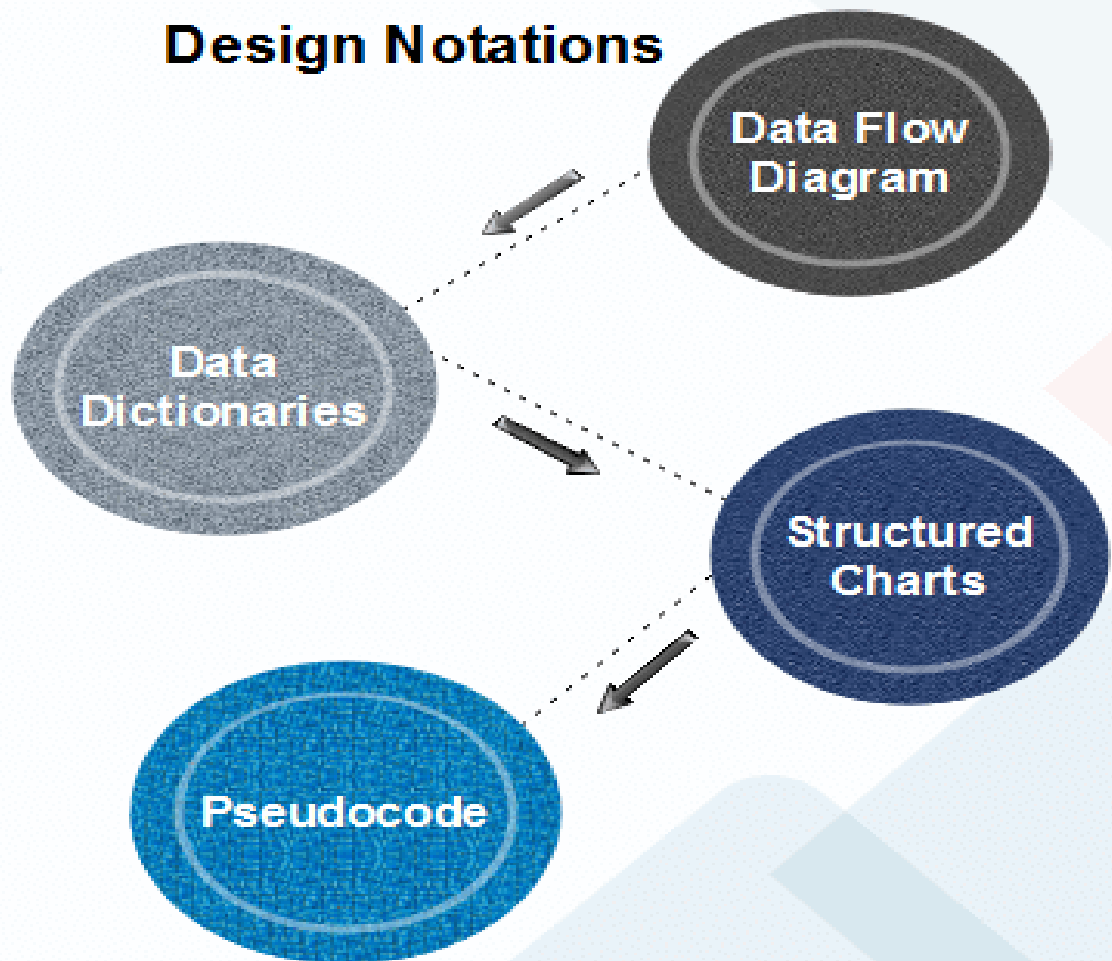
❖ **Data Flow Diagram**:
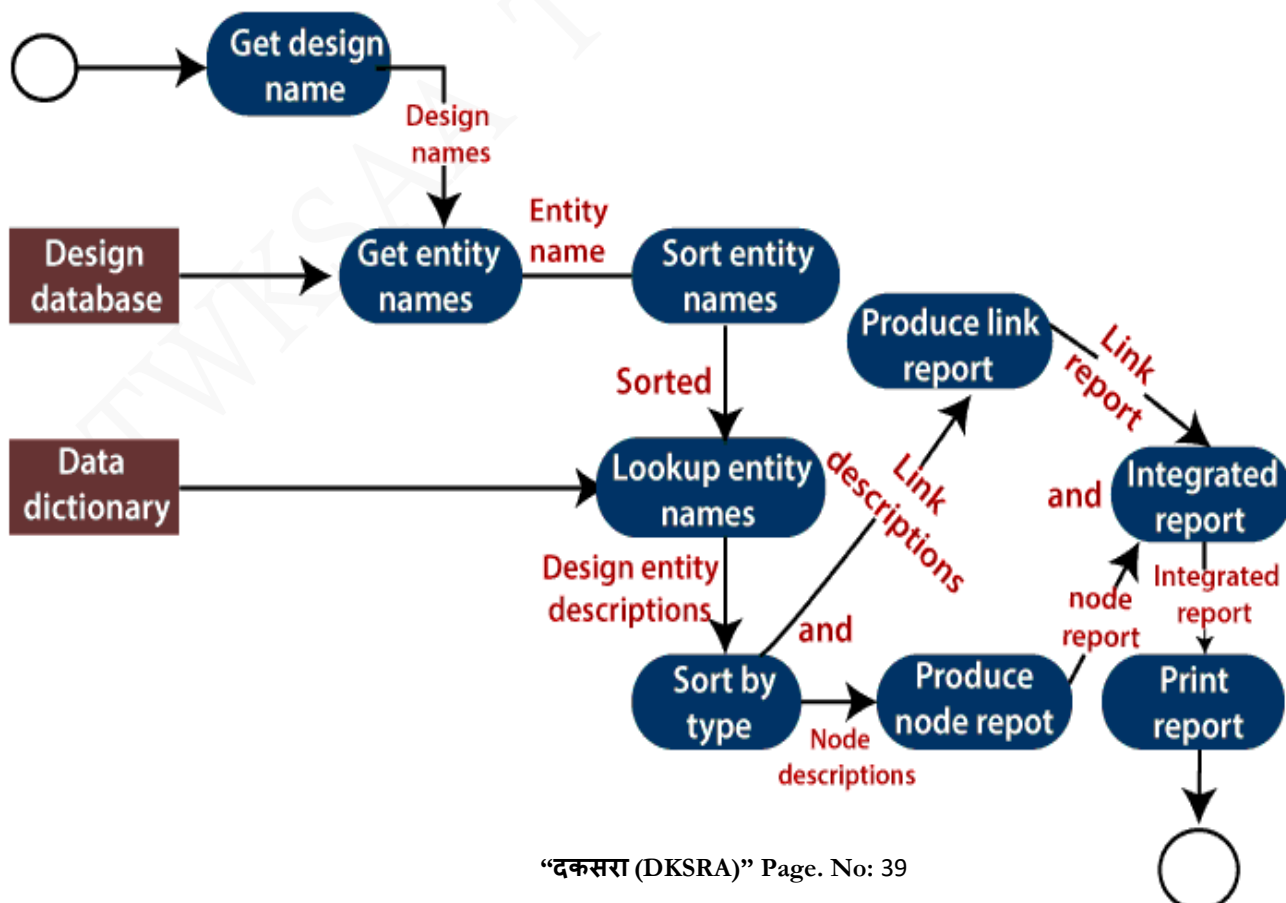
- Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams.
- Data-flow design is an integral part of several design methods, and most CASE tools support data-flow diagram creation. Different ways may use different icons to represent data-flow diagram entities, but their meanings are similar.
- Data-flow diagrams are a useful and intuitive way of describing a system.

➢ **The notation which is used is based on the following symbols:**

Symbol	Name	Meaning
[blue rounded rectangle]	Rounded Rectangle	It represents functions which transforms input to output. The transformation name indicates its function.
[brown rectangle]	Rectangle	It represents data stores. Again, they should give a descriptive name.
[circle]	Circle	It represents user interactions with the system that provides input or receives output.
[arrow]	Arrows	It shows the direction of data flow. Their name describes the data flowing along the path.
"and" and "or"	Keywords	The keywords "and" and "or". These have their usual meanings in boolean expressions. They are used to link data flows when more than one data flow may be input or output from a transformation.

## Data flow diagram of a design report generator

❖ **Data Dictionaries:**

- A data dictionary lists all data elements appearing in the DFD model of a system. The data items listed contain all data flows and the contents of all data stores looking on the DFDs in the DFD model of a system.
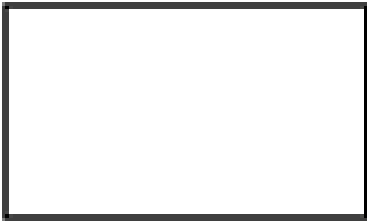
❖ **Structured Charts:**
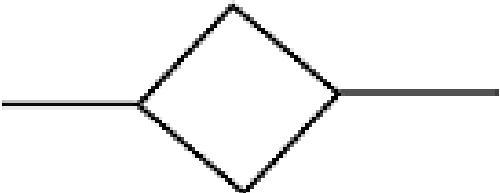
- It partitions a system into block boxes. A Black box system that functionality is known to the user without the knowledge of internal design.
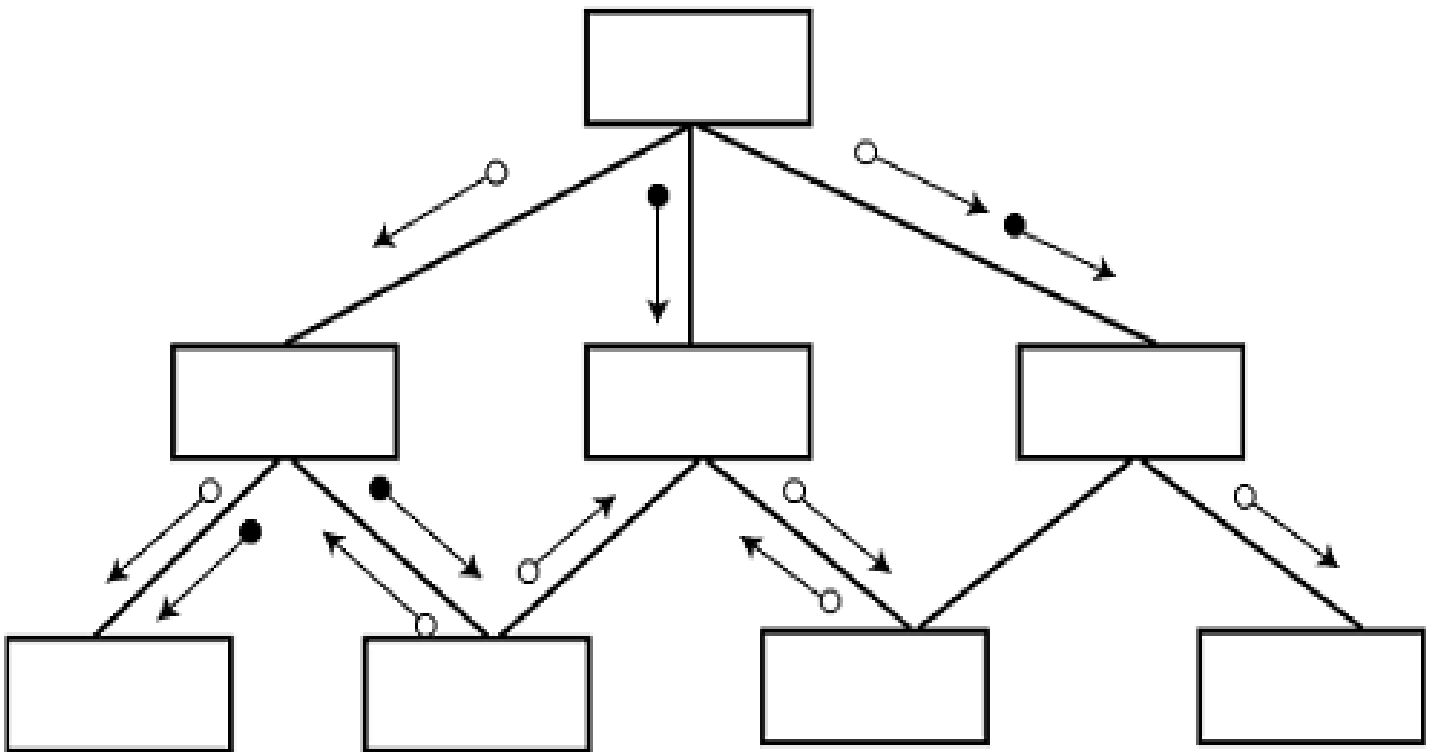
Hierarchical format of a structure chart

❖ **Structured Chart is a graphical representation which shows:**

- System partitions into modules
- Hierarchy of component modules
- The relation between processing modules
- Interaction between modules
- Information passed between modules
- The following notations are used in structured chart:

T3 SKILLS CENTER

SYMBOL	DESCRIPTION
▭	Module
⟶	Arrow
○—⟶	Data couple
●—	Control Flag
(loop symbol)	Loop
◇	Decision

❖ Pseudo-code:

- Pseudo-code notations can be used in both the preliminary and detailed design phases. Using pseudo-code, the designer describes system characteristics using short, concise, English Language phases that are structured by keywords such as If-Then-Else, While-Do, and End.

❖ Object-Oriented Design:

- object-oriented design method, the system is viewed as a collection of objects (i.e., entities). state is distributed among the objects, and each object handles its state data.
  - ➢ **The different terms related to object design are:**

Object Oriented Design
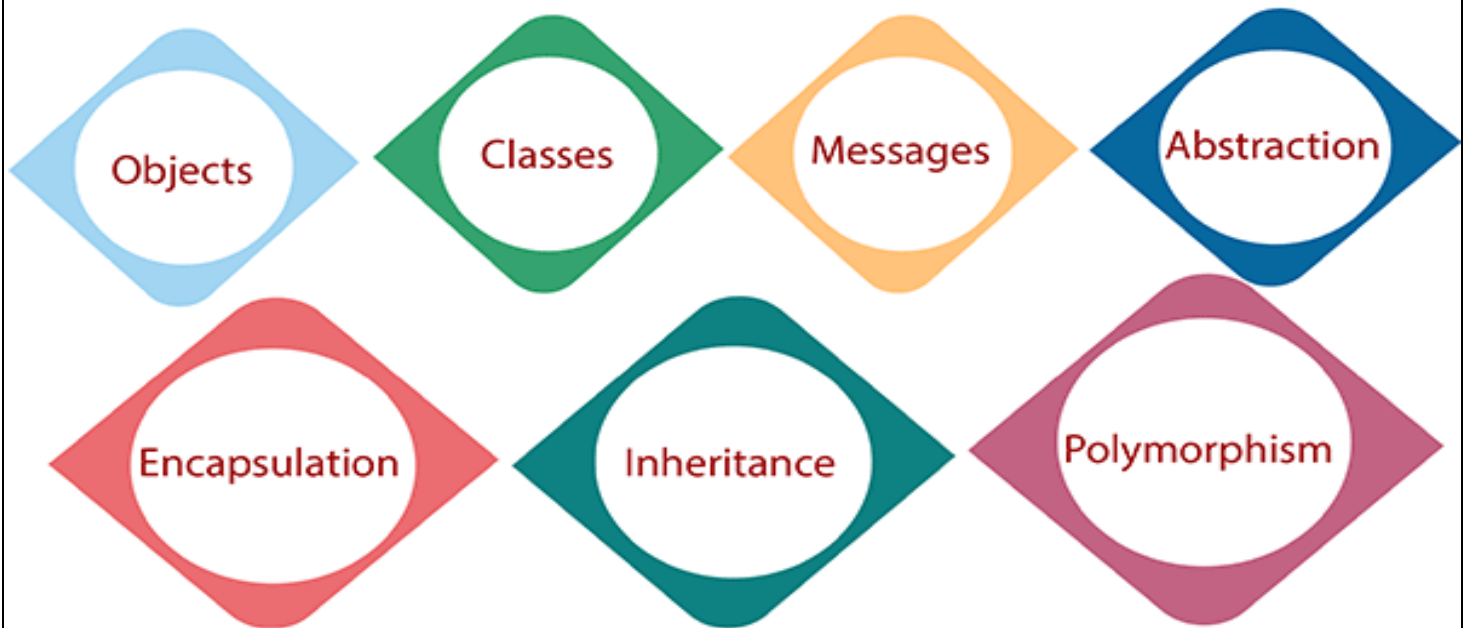

1) **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2) **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
3) **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
4) **Abstraction:** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.
5) **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
6) **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses. This property of OOD is called an inheritance.
7) **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types.

❖ **User Interface Design:**
- The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the program and how data is displayed on the screen.

❖ **Types of User Interface:**
- There are two main types of User Interface:
    1) Text-Based User Interface or Command Line Interface
    2) Graphical User Interface (GUI)

❖ **Text-Based User Interface**: This method relies primarily on the keyboard. A typical example of this is UNIX.

❖ **Advantages**
- Many and easier to customizations options.
- Typically, capable of more important tasks.

❖ **Disadvantages**
- Relies heavily on recall rather than recognition.
- Navigation is often more difficult.

❖ **Graphical User Interface (GUI):** GUI relies much more heavily on the mouse. A typical example of this type of interface is any versions of the Windows operating systems.

❖ **GUI Characteristics:**
- **Windows:** Multiple windows allow different information to be displayed simultaneously on the user's screen.
- **Icons:** Icons different types of information. On some systems, icons represent files. On other icons describes processes.
- **Menus:** Commands are selected from a menu rather than typed in a command language.
- **Pointing:** A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interests in a window.
- **Graphics:** Graphics elements can be mixed with text or the same display.

❖ **Advantages:**
- Less expert knowledge is required to use it.
- Easier to Navigate and can look through folders quickly in a guess and check manner.
- The user may switch quickly from one task to another and can interact with several different applications.

❖ **Disadvantages:**
- Typically decreased options.
- Usually less customizable. Not easy to use one button for tons of different variations.

❖ **UI Design Principles:**

T3 SKILLS CENTER

UI Design Principles

Structure   **Simplicity**   **Visibility**   **Feedback**   **Tolerance**

- ❖ **Structure:** Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another.
- ❖ **Simplicity:** The design should make the simple, common task easy, communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.
- ❖ **Visibility:** The design should make all required options and materials for a given function visible without distracting the user with extraneous or redundant data.
- ❖ **Feedback:** The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
- ❖ **Tolerance:** The design should be flexible and tolerant, decreasing the cost of errors and misuse by allowing undoing and redoing while also preventing bugs wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

❖ Coding:

- The coding is the process of transforming the design of a system into a computer language format. This coding phase of software development is concerned with software translating design specification into the source code.
- Coding is done by the coder or programmers who are independent people than the designer.

❖ Goals of Coding:

- **To translate the design of system into a computer language format:** The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.
- **To reduce the cost of later phases:** The cost of testing and maintenance can be significantly reduced with efficient coding.
- **Making the program more readable:** Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

❖ Characteristics of Programming Language:

➢ Following are the characteristics of Programming Language:

1) **Readability:** A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.
2) **Portability:** High-level languages, being virtually machine-independent, should be easy to develop portable software.
3) **Generality:** Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.
4) **Brevity:** Language should have the ability to implement the algorithm with less amount of code. Programs mean in high-level languages are often significantly shorter than their low-level equivalents.
5) **Error checking:** A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile-time and run-time.
6) **Cost:** The ultimate cost of a programming language is a task of many of its characteristics.
7) **Quick translation:** It should permit quick translation.
8) **Efficiency:** It should authorize the creation of an efficient object code.
9) **Modularity:** It is desirable that programs can be developed in the language as several separately compiled modules, with the appropriate structure for ensuring self-consistency among these modules.
10) **Widely available:** Language should be widely available, and it should be feasible to provide translators for all the major machines and all the primary operating systems.

T3 SKILLS CENTER

❖ **Coding Standards:**
  - General coding standards refers to how the developer writes code, so here we will discuss some essential standards regardless of the programming language being used.
  ➢ The following are some representative coding standards:
1. **Indentation:** Proper and consistent indentation is essential in producing easy to read and maintainable programs.
   - Indentation should be used to:
   - Emphasize the body of a control structure such as a loop or a select statement.
   - Emphasize the body of a conditional statement
   - Emphasize a new scope block
2. **Inline comments**: Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.
3. **Rules for limiting the use of global:** These rules file what types of data can be declared global and what cannot.
4. **Structured Programming:** Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.
5. **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
6. **Error return conventions and exception handling system:** Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

❖ **Coding Guidelines:**
  - General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain.
1. **Line Length**: It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.
2. **Spacing:** The appropriate use of spaces within a line of code can improve readability.

Example:

Bad: cost=price+(price*sales_tax)
              fprintf(stdout ,"The total cost is %5.2f\n",cost);

Better: cost = price + (price * sales_tax)
              fprintf (stdout,"The total cost is %5.2f\n",cost);

3. **The code should be well-documented:** As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. **The length of any function should not exceed 10 source lines:** A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.
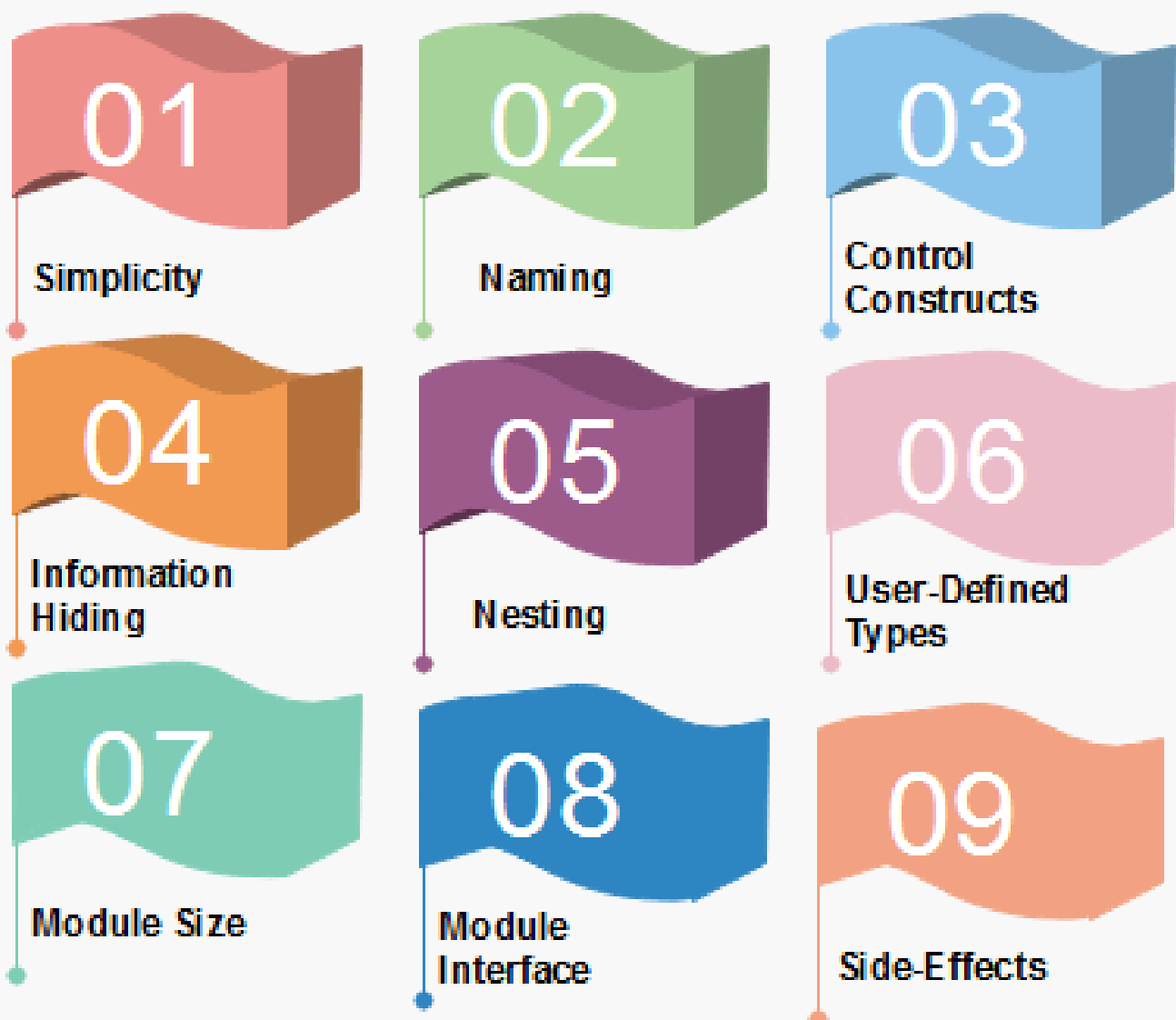
5. **Do not use goto statements:** Use of goto statements makes a program unstructured and very tough to understand.

6. **Inline Comments:** Inline comments promote readability.

7. **Error Messages**: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

❖ **Programming Style:**

- Programming style refers to the technique used in writing the source code for a computer program. Most programming styles are designed to help programmers quickly read and understands the program as well as avoid making errors.

Programming Style

01 Simplicity	02 Naming	03 Control Constructs
04 Information Hiding	05 Nesting	06 User-Defined Types
07 Module Size	08 Module Interface	09 Side-Effects

❖ **Structured Programming:**

- In structured programming, we sub-divide the whole program into small modules so that the program becomes easy to understand. The purpose of structured programming is to linearize control flow through a computer program so that the execution sequence follows the sequence in which the code is written. The dynamic structure of the program than resemble the static structure of the program. This enhances the readability, testability, and modifiability of the program.

❖ **Why we use Structured Programming?**

- We use structured programming because it allows the programmer to understand the program easily.

❖ **Software Reliability:**

- Software Reliability means Operational reliability. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.
- Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

❖ **Software Failure Mechanisms:**

- ➢ **The software failure can be classified as:**
- **Transient failure:** These failures only occur with specific inputs.
- **Permanent failure:** This failure appears on all inputs.
- **Recoverable failure:** System can recover without operator help.
- **Unrecoverable failure:** System can recover with operator help only.
- **Non-corruption failure:** Failure does not corrupt system state or data.
- **Corrupting failure:** It damages the system state or data.

❖ **Software Fault Tolerance:**

- Software fault tolerance is the ability for software to detect and recover from a fault that is happening or has already happened in either the software or hardware in the system in which the software is running to provide service by the specification.
- Software fault tolerance is a necessary component to construct the next generation of highly available and reliable computing systems from embedded systems to data warehouse systems.

❖ **Software Reliability Models:**

- A software reliability model indicates the form of a random process that defines the behaviour of software failures to time.
- Software reliability models have appeared as people try to understand the features of how and why software fails, and attempt to quantify software reliability.
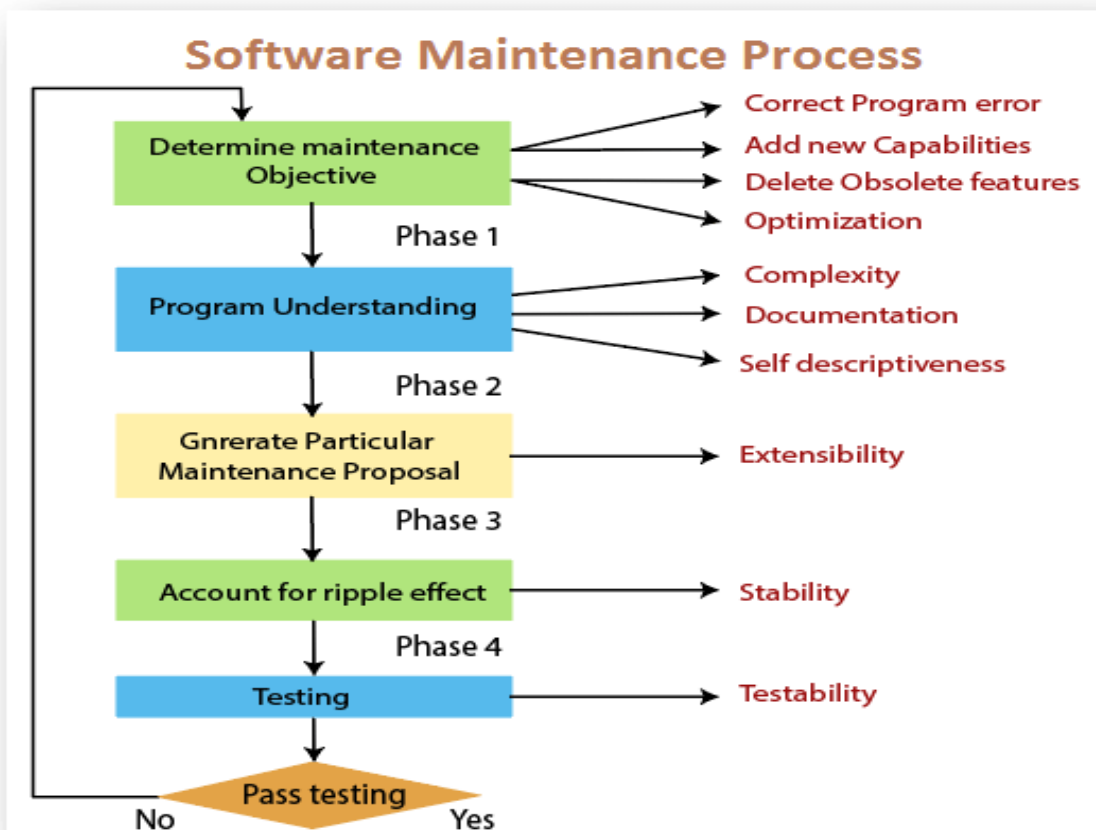
T3 SKILLS CENTER

❖ **Software Maintenance:**

- Software maintenance is a part of the Software Development Life Cycle. Its primary goal is to modify and update software application after delivery to correct errors and to improve performance. Software is a model of the real world. When the real-world changes, the software require alteration wherever possible.

❖ **Need for Maintenance:**

- Software Maintenance is needed for: -
1. Correct errors
2. Change in user requirement with time
3. Changing hardware/software requirements
4. To improve system efficiency
5. To optimize the code to run faster
6. To modify the components
7. To reduce any unwanted side effects.

❖ **Types of Software Maintenance:**

1) **Corrective Maintenance:** Corrective maintenance aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation

2) **Adaptive Maintenance:** It contains modifying the software to match changes in the ever-changing environment.

3) **Preventive Maintenance:** It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering & reverse engineering in which an old system with old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

4) **Perfective Maintenance**: It defines improving processing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.

T3 SKILLS CENTER

❖ **Research(अनुसंधान):**

- अनुसंधान एक प्रणालीकरण कार्य होता है जिसमें विशेष विषय या विषय की नई ज्ञान एवं समझ को प्राप्त करने के लिए सिद्धांतिक जांच और अध्ययन किया जाता है। इसकी प्रक्रिया में डेटा का संग्रह और विश्लेषण, निष्कर्ष निकालना और विशेष क्षेत्र में मौजूदा ज्ञान में योगदान किया जाता है। अनुसंधान के माध्यम से विज्ञान, प्रोद्योगिकी, चिकित्सा, सामाजिक विज्ञान, मानविकी, और अन्य क्षेत्रों में विकास किया जाता है। अनुसंधान की प्रक्रिया में अनुसंधान प्रश्न या कल्पनाएँ तैयार की जाती हैं, एक अनुसंधान योजना डिज़ाइन की जाती है, डेटा का संग्रह किया जाता है, विश्लेषण किया जाता है, निष्कर्ष निकाला जाता है और परिणामों को उचित दर्शाने के लिए समाप्ति तक पहुंचाया जाता है।

❖ **Innovation(नवीनीकरण): -**

- Innovation (इनोवेशन) एक विशेषता या नई विचारधारा की उत्पत्ति या नवीनीकरण है। यह नए और आधुनिक विचारों, तकनीकों, उत्पादों, प्रक्रियाओं, सेवाओं या संगठनात्मक ढंगों का सृजन करने की प्रक्रिया है जिससे समस्याओं का समाधान, प्रतिस्पर्धा में अग्रणी होने, और उपयोगकर्ताओं के अनुकूलता में सुधार किया जा सकता है।

❖ **Discovery (आविष्कार):**

- Discovery का अर्थ होता है "खोज" या "आविष्कार"। यह एक विशेषता है जो किसी नए ज्ञान, अविष्कार, या तत्व की खोज करने की प्रक्रिया को संदर्भित करता है। खोज विज्ञान, इतिहास, भूगोल, तकनीक, या किसी अन्य क्षेत्र में हो सकती है। इस प्रक्रिया में, व्यक्ति या समूह नए और अज्ञात ज्ञान को खोजकर समझने का प्रयास करते हैं और इससे मानव सभ्यता और विज्ञान-तकनीकी के विकास में योगदान देते हैं।

Note: अनुसंधान विशेषता या विषय पर नई ज्ञान के प्राप्ति के लिए सिस्टमैटिक अध्ययन है, जबकि आविष्कार नए और अज्ञात ज्ञान की खोज है।

TWKSAA RID MISSION

(Research)	(Innovation)	(Discovery)
अनुसंधान करने के महत्वपूर्ण कारण:	नवीनीकरण करने के महत्वपूर्ण कारण:	खोज करने के महत्वपूर्ण कारण:
1. नई ज्ञान की प्राप्ति	1. प्रगति के लिए	1. नए ज्ञान की प्राप्ति
2. समस्याओं का समाधान	2. परिवर्तन के लिए	2. ज्ञान के विकास में योगदान
3. तकनीकी और व्यापार में उन्नति	3. उत्पादन में सुधार	3. अविष्कारों की खोज
4. विकास को बढ़ावा देना	4. प्रतिस्पर्धा में अग्रणी होने के लिए	4. समस्याओं का समाधान
5. सामाजिक प्रगति	5. समाज को लाभ	5. समाज के उन्नति का माध्यम
6. देश विज्ञान और प्रौद्योगिकी का विकास	6. देश विज्ञान और प्रौद्योगिकी के विकास।	6. देश विज्ञान और तकनीक के विकास

➢ जो लोग रिसर्च, इनोवेशन और डिस्कवरी करते हैं उन लोगों को ही हमें अपना नायक, प्रतीक एवं आदर्श मानना चाहिए क्योंकि ये लोग हमारे समाज, देश एवं विज्ञान के क्षेत्र में प्रगति, विकास और समस्याओं के समाधान में महत्वपूर्ण भूमिका निभाते हैं।

मैं राजेश प्रसाद एक वीणा उठाया हूँ Research, Innovation and Discovery का जिसका मुख्य उद्देश्य हैं आने वाले समय में सबसे पहले New(RID, PMS & TLR) की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से ही हो |

"अगर आप भी Research, Innovation and Discovery के क्षेत्र में रूचि रखतें हैं एवं अपनी प्रतिभा से दुनियां को कुछ नया देना चाहतें तो हमारे इस त्वक्सा रीड मिशन (TWKSAA RID MISSION) से जरुर जुड़ें" |

- **राजेश प्रसाद**