Software Engineering

Dr. Suvendu Chandan Nayak
Sr. Asst. Prof.
CSE, SIT, BBSR

What is Software Engineering?

- Engineering approach of systematic and cost-effective techniques to develop software through Construction Analogy and systematic past experiences.
- Systematic collection of past experience:
  - Techniques,
  - Methodologies,
  - Guidelines.

Engineering Practice

- Heavy use of past experience:
  - Past experience is systematically arranged.
- Theoretical basis and quantitative techniques provided.
- Many are just thumb rules.
- Tradeoff between alternatives.
- Pragmatic approach to cost-effectiveness.

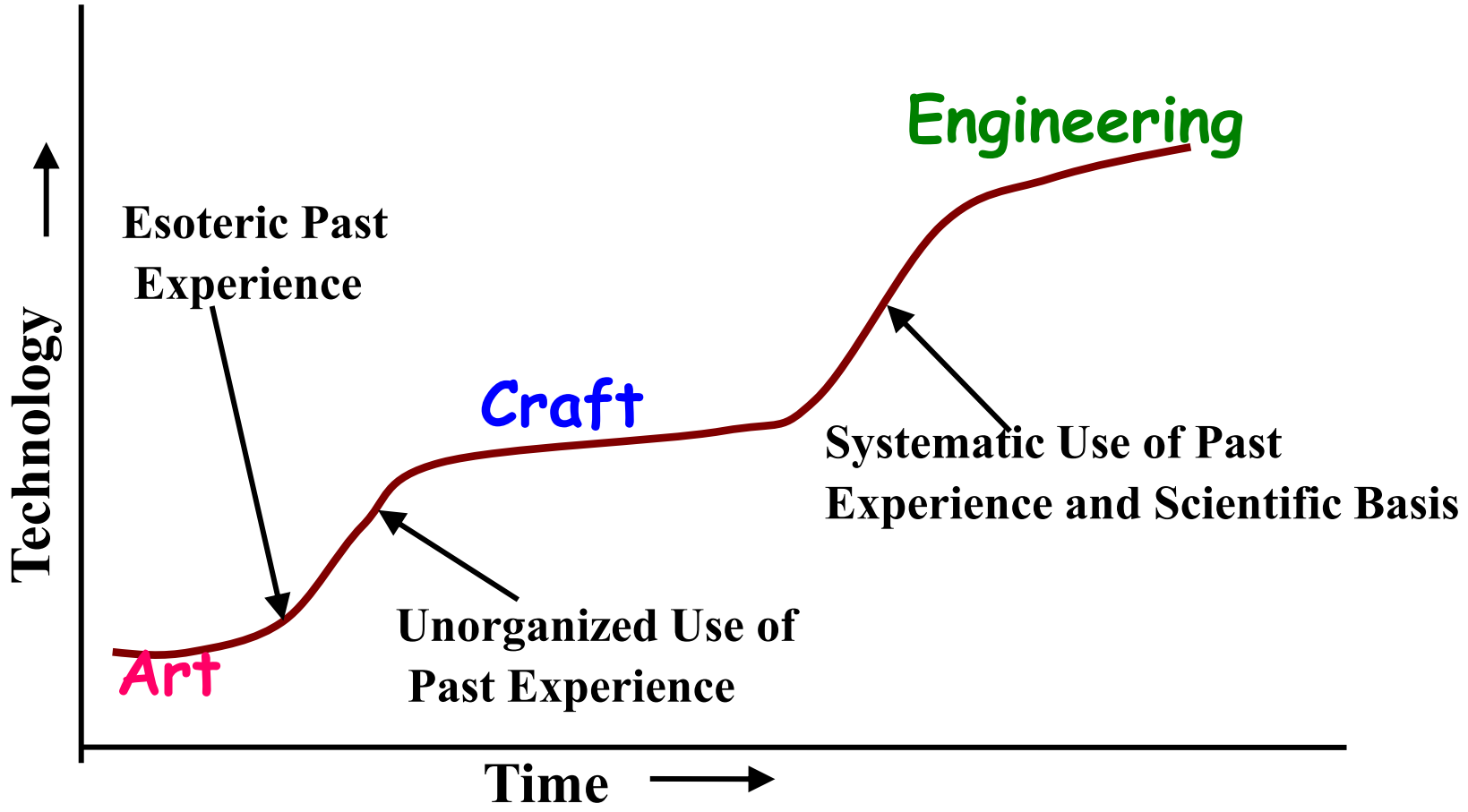
Is SE Science or Art

- **No**
- It uses past experiences, but in science there is exact solution by rigorous proofs.
- In science has unique solution but in engineering several goals, no unique solution, alternative solutions, etc are used.
- Science does not concern with cost maintainability and usability, but engineering concerns with cost- effectiveness.

Is SE Science or Art

- Engineering based on well understood and quantitative approach but science and art based upon subjective judgement.

Technology Development Pattern

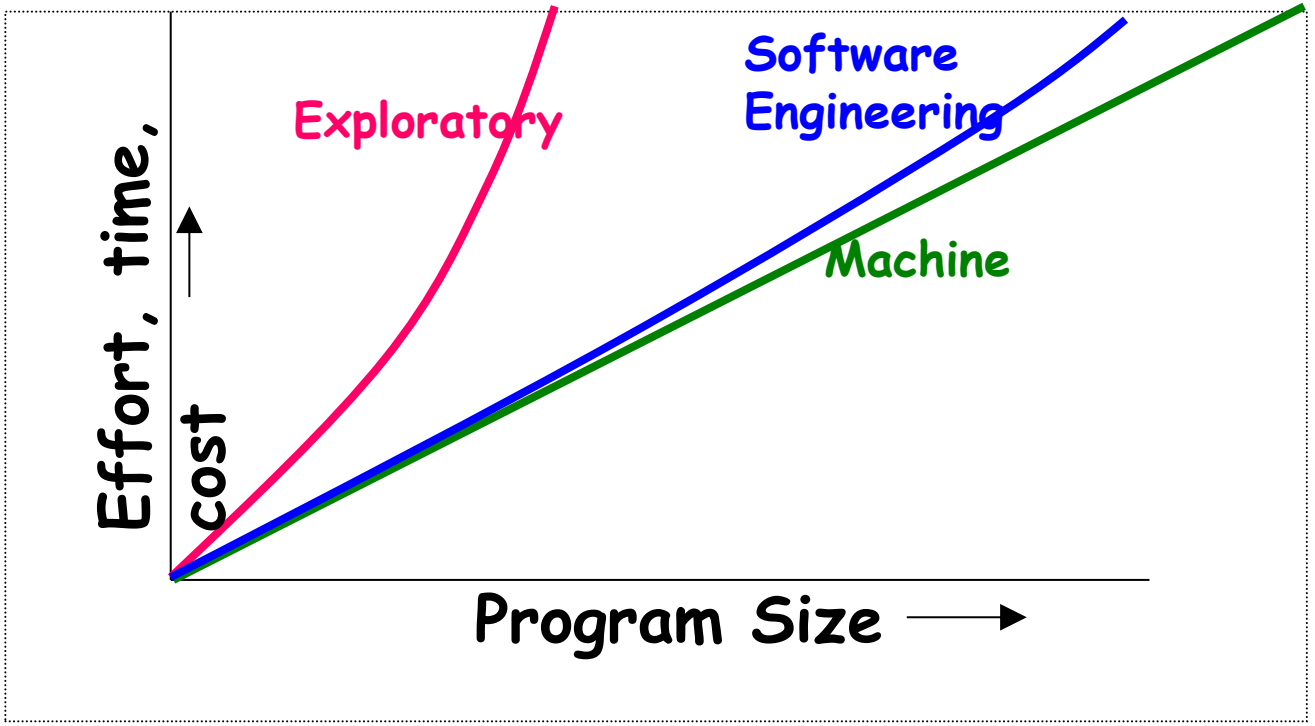


Evolution of an Art into an Engineering Discipline

- The early programmers used an exploratory (also called build and fix) style.
  - In the build and fix (exploratory) style, normally a `dirty' program is quickly developed.
  - The different imperfections that are subsequently noticed are fixed.

What is Wrong with the Exploratory Style?

- Can successfully be used for very small programs only.



What is Wrong with the Exploratory Style? Cont…

- Besides the exponential growth of effort, cost, and time with problem size:
  - Exploratory style usually results in unmaintainable code.
  - It becomes very difficult to use the exploratory style in a team development environment.

What is Wrong with the Exploratory Style? Cont…

- Why does the effort required to develop a product grow exponentially with product size?
  - Why does the approach completely breaks down when the product size becomes large?

Emergence of Software Engineering

- Early Computer Programming (1950s):
  - Programs were being written in assembly language.
  - Programs were limited to about a few hundreds of lines of assembly code.
- Every programmer developed his own style of writing programs:
  - According to his intuition (exploratory programming).
  - Difficult to learn & understand

High-Level Language Programming (Early 60s)

- High-level languages such as FORTRAN, ALGOL, and COBOL were introduced.
- This reduced software development efforts greatly.
- Software development style was still exploratory.
- Typical program sizes were limited to a few thousands of lines of source code.

Control Flow-Based Design (late 60s)

- Size and complexity of programs increased further:
- Exploratory programming style proved to be insufficient.
- Programmers found: Very difficult to write cost-effective and correct programs.
- programs written by others very difficult to understand and maintain.
- Experienced programmers advised: **``Pay particular attention to the design of the program's control structure.''**

Control Flow-Based Design (late 60s)

- A program's control structure indicates:
  - The sequence in which the program's instructions are executed.
- To help design programs having good control structure:
  - Flow charting technique was developed.
  - One can represent and design a program's control structure.

  Usually one understands a program:
  - By mentally simulating the program's execution sequence.

Control Flow-Based Design

- Difficult to understand and debug.
- GO TO statements used as control structure.
- The need to restrict use of GO TO statements was recognized.
- Only three programming constructs are sufficient to express any programming logic:
  - sequence (e.g. a=0;b=5;)
  - selection (e.g.if(c=true) k=5 else m=5;)
  - iteration (e.g. while(k>0) k=j-k;)

This formed the basis of Structured Programming methodology.

Structured Programming

- A program is called structured
  - When it uses only the following types of constructs:
    - sequence,
    - selection,
    - iteration

      ✓ Easier to read and understand,

      ✓ Easier to maintain,

      ✓ Require less effort and time for development.

Structured Programming

- Programmers commit less number of errors:
  - While using structured if-then-else and do-while statements.
  - Compared to test-and-branch constructs.

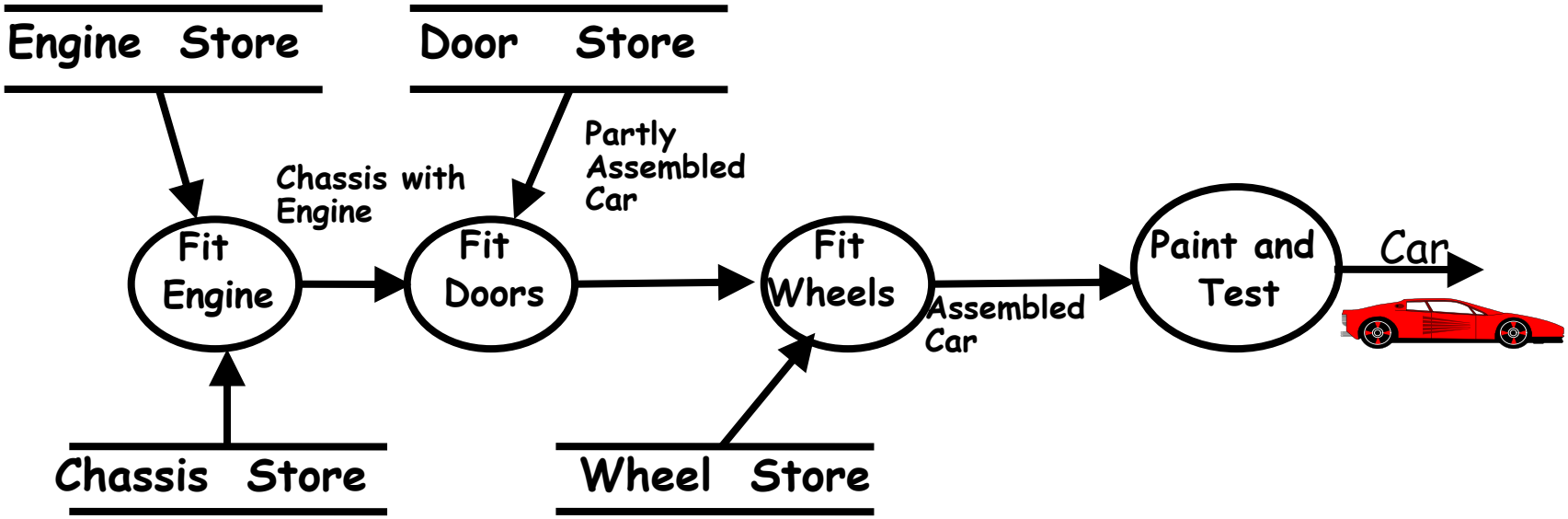
Data Structure-Oriented Design (Early 70s)

- Techniques which emphasize designing the data structure:
  - Derive program structure called data structure-oriented design techniques.
  - Program code structure should correspond to the data structure.
  - A program's data structures are first designed using notations for
    - sequence, selection, and iteration.

Data Flow-Oriented Design (Late 70s)

- The data items input to a system must first be identified,
- Processing required on the data items to produce the required outputs should be determined.
- processing stations (functions) are identified in a system.
- The items (data) that flow between functions.

Data Flow Model of a Car Assembly Unit



Object-Oriented Design (80s)

- Natural objects (such as employees, pay-roll-register, etc.) occurring in a problem are first identified.

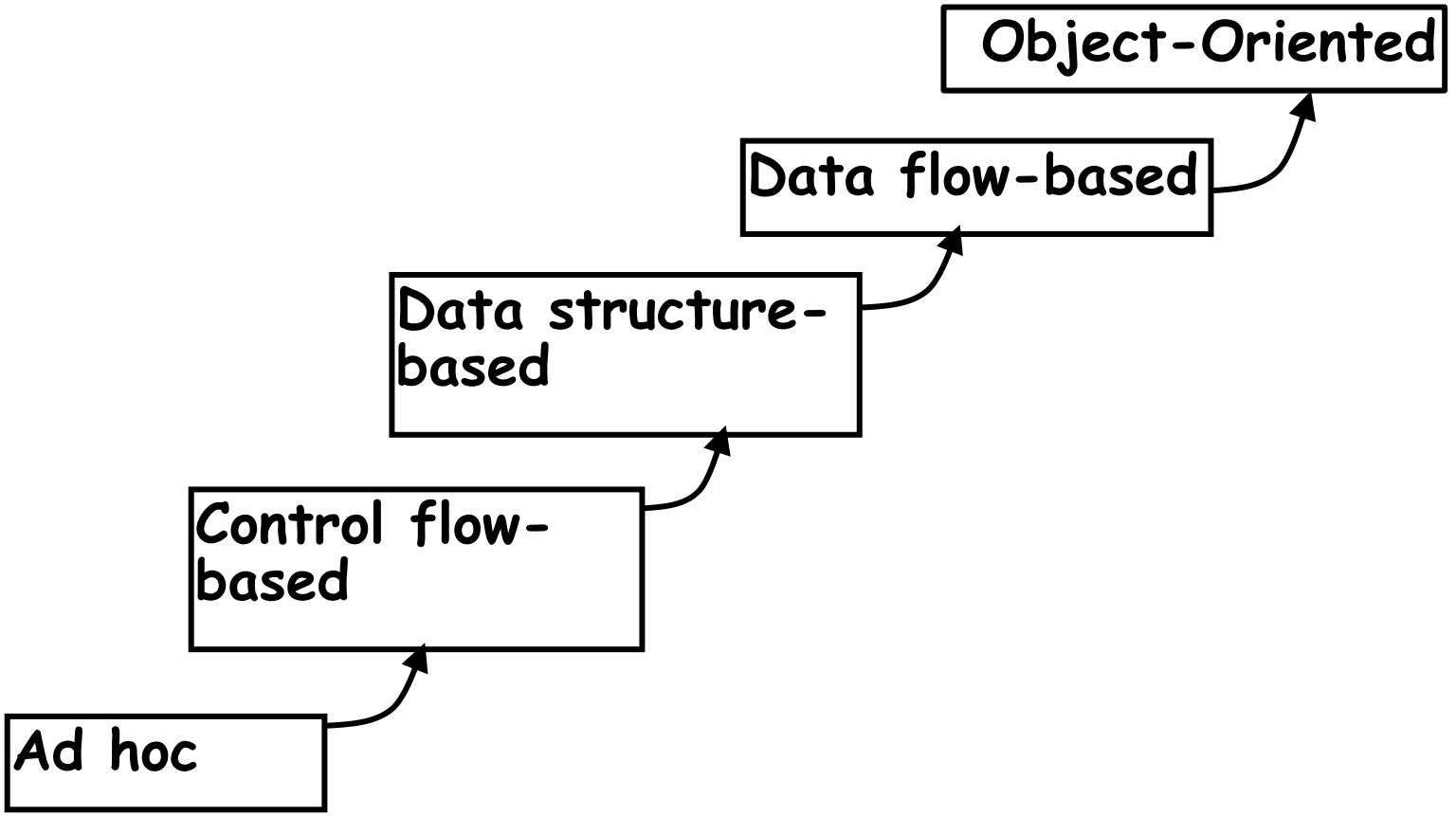
- Relationships among objects:
  - Such as composition, reference, and inheritance are determined.

- Each object essentially acts as : data hiding (or data abstraction) entity.

Object-Oriented Design

- Simplicity
- Reuse possibilities
- Lower development time and cost
- More robust code
- Easy maintenance

Evolution of Design Techniques



Principles Deployed by Software Engineering

- Mainly two important principles are deployed:
  - Abstraction
  - Decomposition

Abstraction

- Simplify a problem by omitting unnecessary details.
  - Focus attention on only one aspect of the problem and ignore irrelevant details.
- Suppose you are asked to develop an overall understanding of some country.
  - No one in his right mind would meet all the citizens of the country, visit every house, and examine every tree of the country, etc.
  - You would possibly refer to various types of maps for that country.
- A map, in fact, is an abstract representation of a country.

Decomposition

- Decompose a problem into many small independent parts.
  - The small parts are then taken up one by one and solved separately.
  - The idea is that each small part would be easy to grasp and can be easily solved.
  - The full problem is solved when all the parts are solved.

Software Crisis

- Software products:
  - Fail to meet user requirements.
  - Frequently crash.
  - Expensive.
  - Difficult to alter, debug, and enhance.
  - Often delivered late.
  - Use resources non-optimally.

Programs versus Software Products

• Usually small in size	• Large
• Author himself is sole user	• Large number of users
• Single developer	• Team of developers
• Lacks proper user interface	• Well-designed interface
• Lacks proper documentation	• Well documented & user-manual prepared
• Ad hoc development.	• Systematic development

Types of Software Projects

- Software products
- Outsourced projects
- Indian companies have focused on outsourced projects.

Evolving role of software

- The role of computer software has undergone significant change over a time span of little more than 50 years.
- Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems.

Differences between the exploratory style and modern software development practices

- Use of Life Cycle Models
- Software is developed through several well-defined stages:
  - requirements analysis and specification,
  - design,
  - coding,
  - testing, etc.

Differences between the exploratory style and modern software development practices

- Emphasis has shifted
  - from error correction to error prevention.
- Modern practices emphasize:
  - detection of errors as close to their point of introduction as possible.

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style,
  - errors are detected only during testing,
- Now,
  - focus is on detecting as many errors as possible in each phase of development.

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style,
  - coding is synonymous with program development.
- Now,
  - coding is considered only a small part of program development effort.

Differences between the exploratory style and modern software development practices (CONT.)

- A lot of effort and attention is now being paid to:
  - Requirements specification.
- Also, now there is a distinct design phase:
  - Standard design techniques are being used.

Differences between the exploratory style and modern software development practices (CONT.)

- During all stages of development process:
  - Periodic reviews are being carried out
- Software testing has become systematic:
  - Standard testing techniques are available.

Differences between the exploratory style and modern software development practices (CONT.)

- There is better visibility of design and code:
  - Visibility means production of good quality, consistent and standard documents.
  - In the past, very little attention was being given to producing good quality and consistent documents.
  - We will see later that increased visibility makes software project management easier.

Differences between the exploratory style and modern software development practices (CONT.)

- Because of good documentation:
  - fault diagnosis and maintenance are smoother now.
- Several metrics are being used:
  - help in software project management, quality assurance, etc.

Differences between the exploratory style and modern software development practices (CONT.)

- Projects are being thoroughly planned:
  - estimation,
  - scheduling,
  - monitoring mechanisms.
- Use of CASE tools.

Legacy software

- **Legacy software** is **software** that has been around a long time and still fulfills a business need.
- It is mission critical and tied to a particular version of an operating system or hardware model (vendor lock-in) that has gone end-of-life.
- Generally the lifespan of the hardware is shorter than that of the **software**.

Software myths

- Many software problems arise due to myths that are formed during the initial stages of software development.
- Software myths propagate false beliefs and confusion in the minds of management, users and developers.

Software myths

- Manager myths: Managers, who own software development responsibility, are often under strain and pressure to maintain a software budget, time constraints, improved quality, and many other considerations.
- User Myths: In most cases, **users** tend to believe myths about the software because software managers and developers do not try to correct the false beliefs. These myths lead to false expectations and ultimately develop dissatisfaction among the users.

Software myths

- Developer Myths:

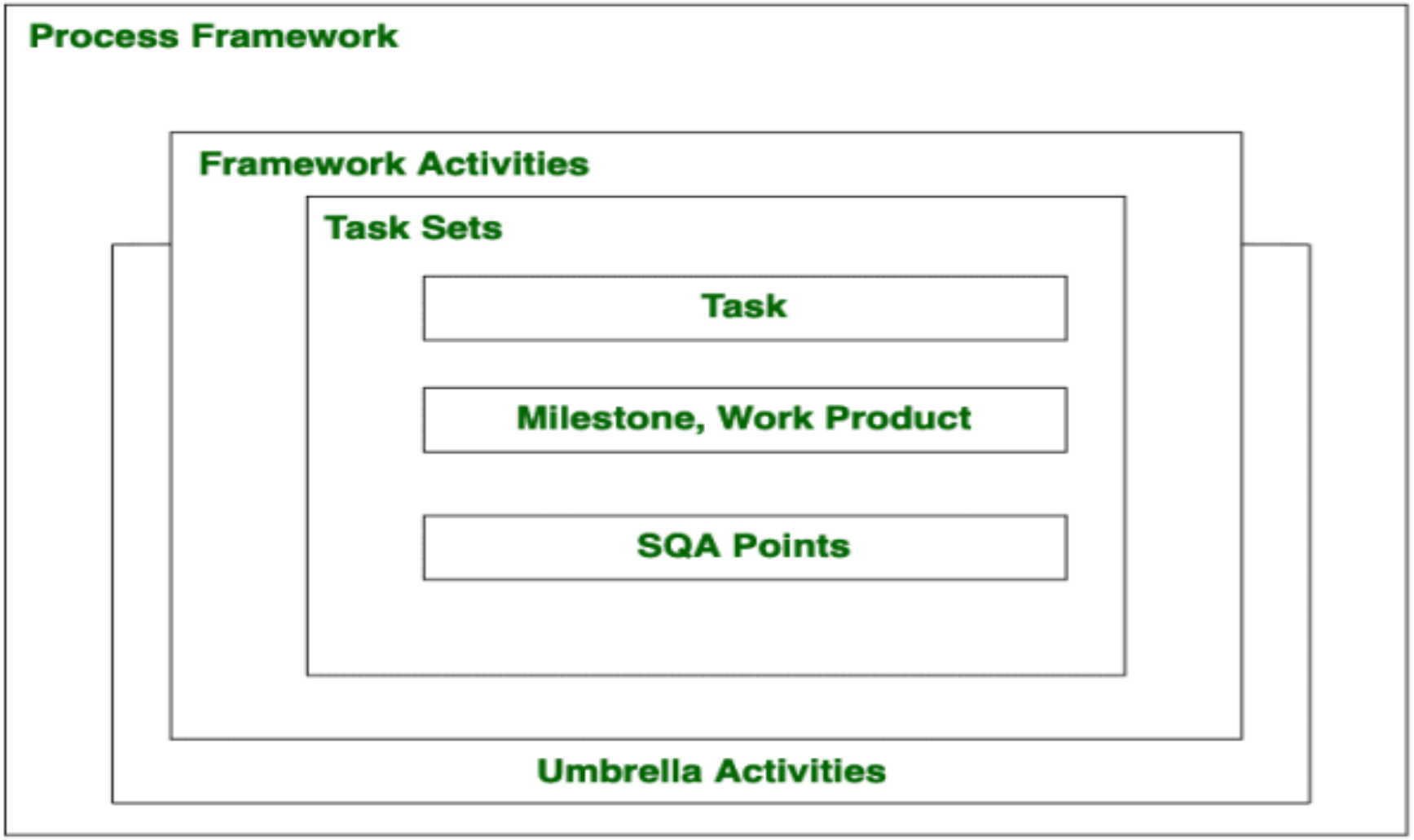
  - Software development is considered complete when the code is delivered.
  - The success of a software project depends on the quality of the product produced.
  - Software engineering requires unnecessary documentation, which slows down the project.
  - The only product that is delivered after the completion of a project is the working program(s).
  - Software quality can be assessed only after the program is executed.

Software Process Framework

It is an abstraction of the software development process.
It details the steps and chronological order of a process.

- ***The process of framework defines a small set of activities that are applicable to all types of projects.***
- ***The software process framework is a collection of task sets.***
- ***Task sets consist of a collection of small work tasks, project milestones, work productivity and software quality assurance points.***

Software Process Framework



Software Process Framework

Process framework encompasses five activities:

- ***Communication:*** In this activity, heavy communication with customers and other stakeholders, requirement gathering is done.
- ***Planning:*** discusses the technical related tasks, work schedule, risks, required resources etc.
- ***Modeling:*** Modelling is about building representations of things in the 'real world'. In modelling activity, a product's model is created in order to better understanding and requirements.
- ***Construction:*** construction is the application of set of procedures that are needed to assemble the product. In this activity, we generate the code and test the product in order to make better product.
- ***Deployment:***software are represented to the customers to evaluate and give feedback. on the basis of their feedback we modify the products for supply better product.

Software Process Framework

Umbrella Activities: are a set of steps or procedure that the software engineering team follows to maintain the progress, quality, change and risks of the overall development tasks.

- **Software Project Tracking and Control**
- **Formal Technical Reviews**
- **Software Quality Assurance**
- **Software Configuration Management**
- **Document Preparation and Production**
- **Re-usability Management**
- **Measurement and Metrics**
- **Risk Management**

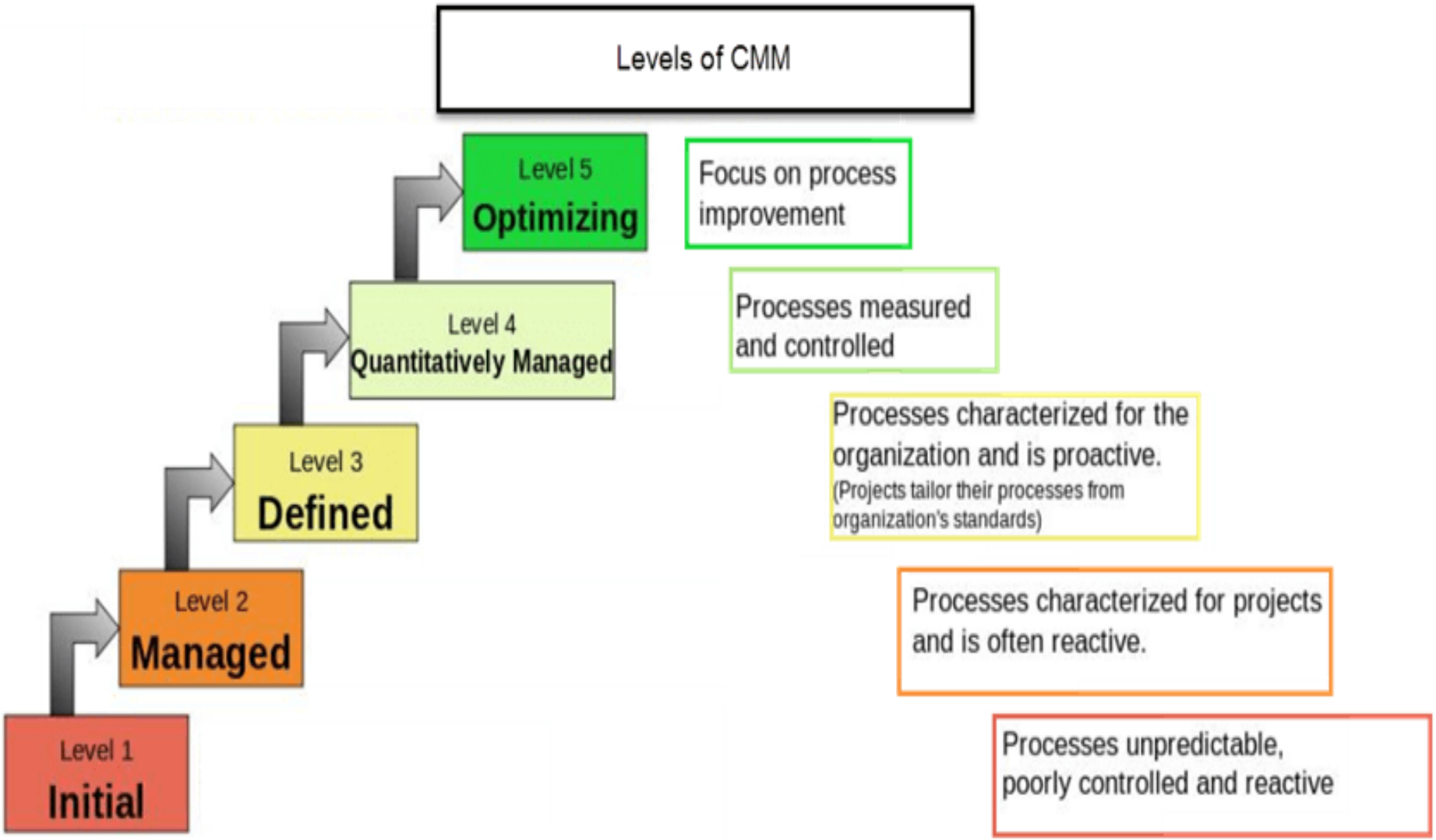
CMM (Capability Maturity Model)

- It is a bench-mark for measuring the maturity of an organization's software process.
- CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promote by the U.S. Department of Defense (DOD).

Different Levels:

- Initial
- Repeatable/Managed
- Defined
- Quantitatively Managed
- Optimizing

CMM (Capability Maturity Model)



CMM (Capability Maturity Model)

Initial

- Company has no standard process for software development.
- Nor does it have a project-tracking system that enables developers to predict costs or finish dates with any accuracy.
- Processes are usually ad hoc and chaotic.
- The organization usually does not provide a stable environment.
- Organizations often produce products and services that work but company has no standard process for software development.

Managed

- Company has installed basic software management processes and controls. But there is no consistency or coordination among different groups.
- Organization has achieved all the specific and generic goals of the maturity that is processes are managed, planned, performed, measured, and controlled but no consistency.

CMM (Capability Maturity Model)

Defined

- Company has a standard set of processes and controls for the entire organization so that developers can move between projects more easily and customers can begin to get consistency from different groups.
- Organization has achieved all the specific and generic goals.
- Processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

Quantitatively Managed

- Implementing standard processes, company has installed systems to measure the quality of those processes across all projects.
- Organization has achieved all the specific goals of the process areas assigned to maturity levels.
- Sub-processes are selected that significantly contribute to overall process performance. These selected sub-processes are controlled using statistical and other quantitative techniques.
- Quantitative objectives for quality and process performance are established and used as criteria in managing processes.

CMM (Capability Maturity Model)

Optimizing

- Processes in order to improve productivity and reduce defects in software development across the entire organization.
- Organization has achieved all the specific goals of the process areas assigned to maturity levels 2, 3, 4, and 5.
- Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes.
- Continually revised to reflect changing business objectives, and used as criteria in managing process improvement.
- Optimizing processes that are agile and innovative depends.
- The organization's ability to rapidly respond to changes and opportunities is enhanced by finding ways to accelerate and share learning.

Life Cycle Model

- *A software life cycle model (or process model):*
  - a descriptive and diagrammatic model of software life cycle:
  - identifies all the activities required for product development,
  - establishes a precedence ordering among the different activities,
  - Divides life cycle into phases.
  - Several different activities may be carried out in each life cycle phase.

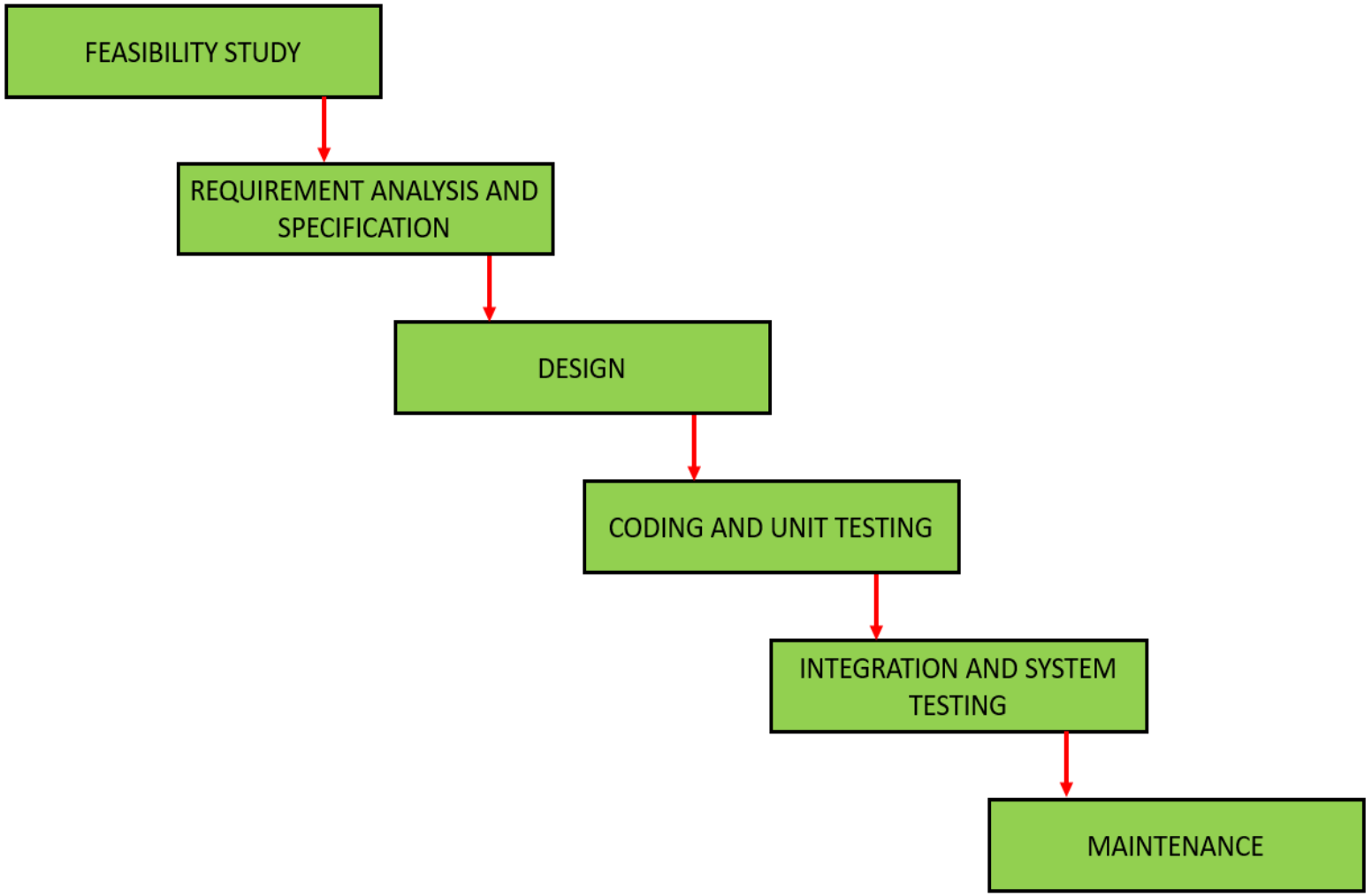
Why Model Life Cycle ?

- A written description:
  - Forms a common understanding of activities among the software developers.
  - Helps in identifying inconsistencies, redundancies, and omissions in the development process.
  - Helps in tailoring a process model for specific projects.
  - Helps development of software in a systematic and disciplined manner.
  - precise understanding among team members as to when to do what.
  - defines entry and exit criteria for every phase.
  - monitor the progress of the project.

Different Life Cycle Model

- Classical waterfall model
- Iterative waterfall
- Evolutionary
- Prototyping
- Spiral model
- RAD model
- Agile models
- V model

Classical waterfall model



Classical waterfall model

1: Feasibility Study:

Main aim of feasibility study is to determine whether developing the software product is :

Financially worthwhile

Technically feasible

Work out an overall understanding of the problem

Data inputs

Processingneeded

Outputdata

Various constraints(timeline, resource, performance..)

Formulate different solution strategies(Alternatives)

Examine each strategy

Based on that Decide whether the project is feasible

Classical waterfall model

2: Requirements Analysis & Specification:

The aim of this phase is:

To understand the requirements of the customer accurately

Then Document them properly

Consists of two distinct activities:

Requirements gathering and analysis

Writing Requirements specification

Classical waterfall model

2.1: Requirements Gathering

Requirements are usually collected from the end-users through

Interviews

Discussions

Survey/ Questionnaire

Workshops

Group or one-to-one meetings etc..

2.2: Requirements Analysis:

The data you initially collected from the users:

May contain several contradictions& ambiguities

These ambiguities & contradictions:

Must be identified

Resolved by discussions with the customers

Then requirements are organized:

Prepare Software Requirements Specification (SRS)document.

Classical waterfall model

3: Design

Design phase **transforms** the "Requirements specification" into **a form** suitable for **implementation** in some programming language.

Two design approaches are there:

Function oriented approach

Object oriented approach

3.1: Function Oriented Design Approach

Consists of two activities:

Structured analysis

Output is **"Data flow Diagram"**

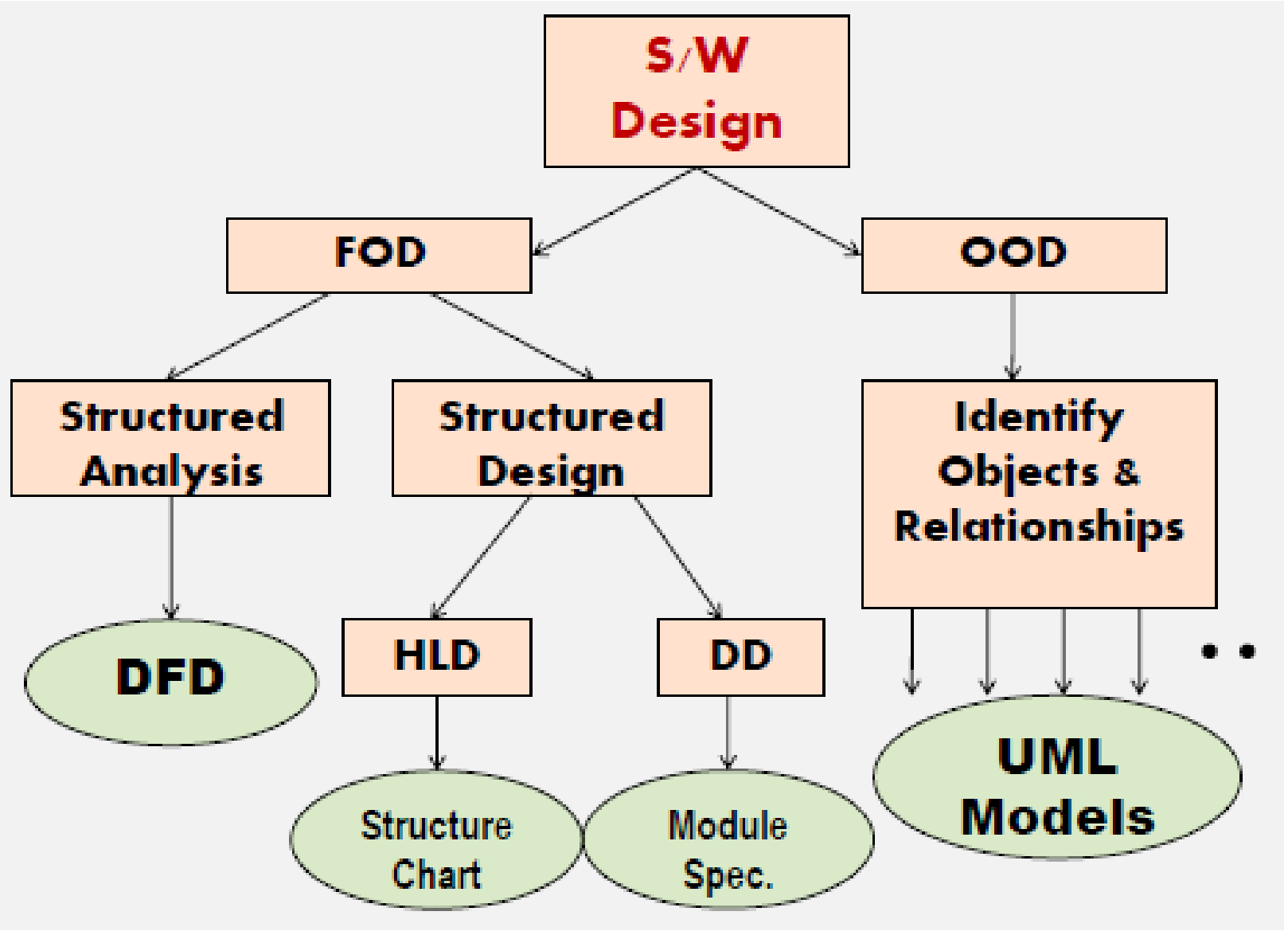
Structured design

Output:

1: High Level Design -**"Structure Chart"**

2: Detail Design –**"Module Specification"**

Classical waterfall model



Classical waterfall model

Object Oriented Design Approach

First **identify various objects**(real world or conceptual entities) in the problem & **relationships** among them.

Ex: The objects in a pay-roll software may be:

Employees

Managers

Payroll register

Departments etc.

Classical waterfall model

4: Implementation:

Purpose of **implementation(coding + unit testing)** phase is to :

Translate software design into source code

During the **implementation** phase**:**

Each designed module is **coded** & **unit tested** independently for correctness

The end product of **implementation phase:**

Program **modules** that have been **tested individually**

Classical waterfall model

5: Integration & System Testing

5.1: Integration Testing:

The modules are integrated in a planned manner:

Integrated in a number of steps

During each integration step:

The partially integrated system is tested

5.2: System Testing

System testing is carried out after Integration testing

Goal of system testing:

Ensure that the "Developed system" works according to "*Requirements"* specified in the SRS document.

Classical waterfall model

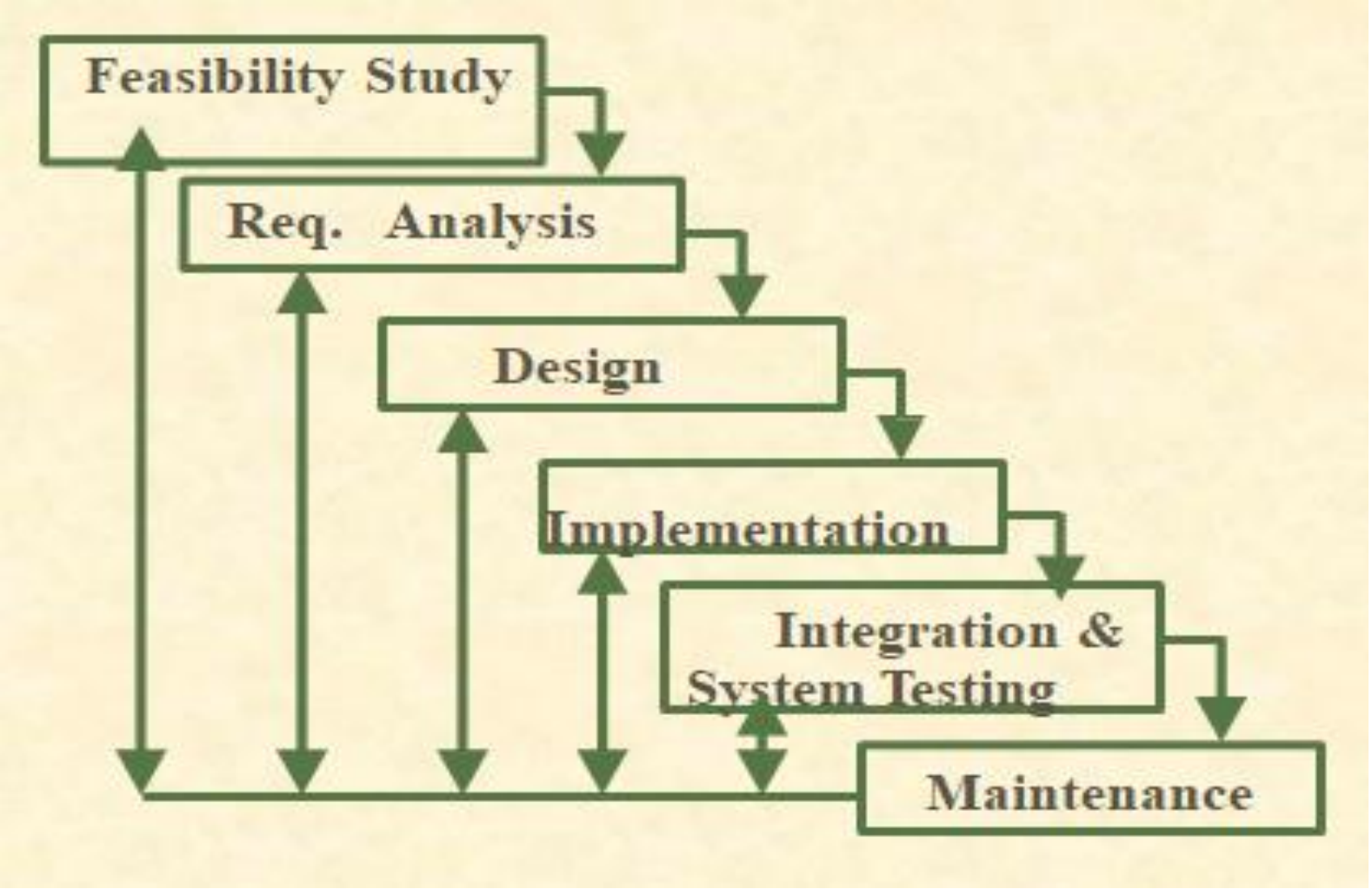
6: Maintenance

Maintenance of any software product:

Involves any change done to the product after it is delivered

Requires more effort than development 40:60

Iterative waterfall model



Iterative waterfall model

Classical waterfall model is **idealistic** & **rigid**:

It assumes, **no defect is found** in earlier phases, after we have moved to the next phase

In practice: **Defects** are discovered in earlier phases

Ex: A design defect might go unnoticed till the coding or testing phase

Iterative waterfall model

Once a defect is detected:

We need to go back to the phase where it was introduced

Redo some work in that phase & subsequent phases

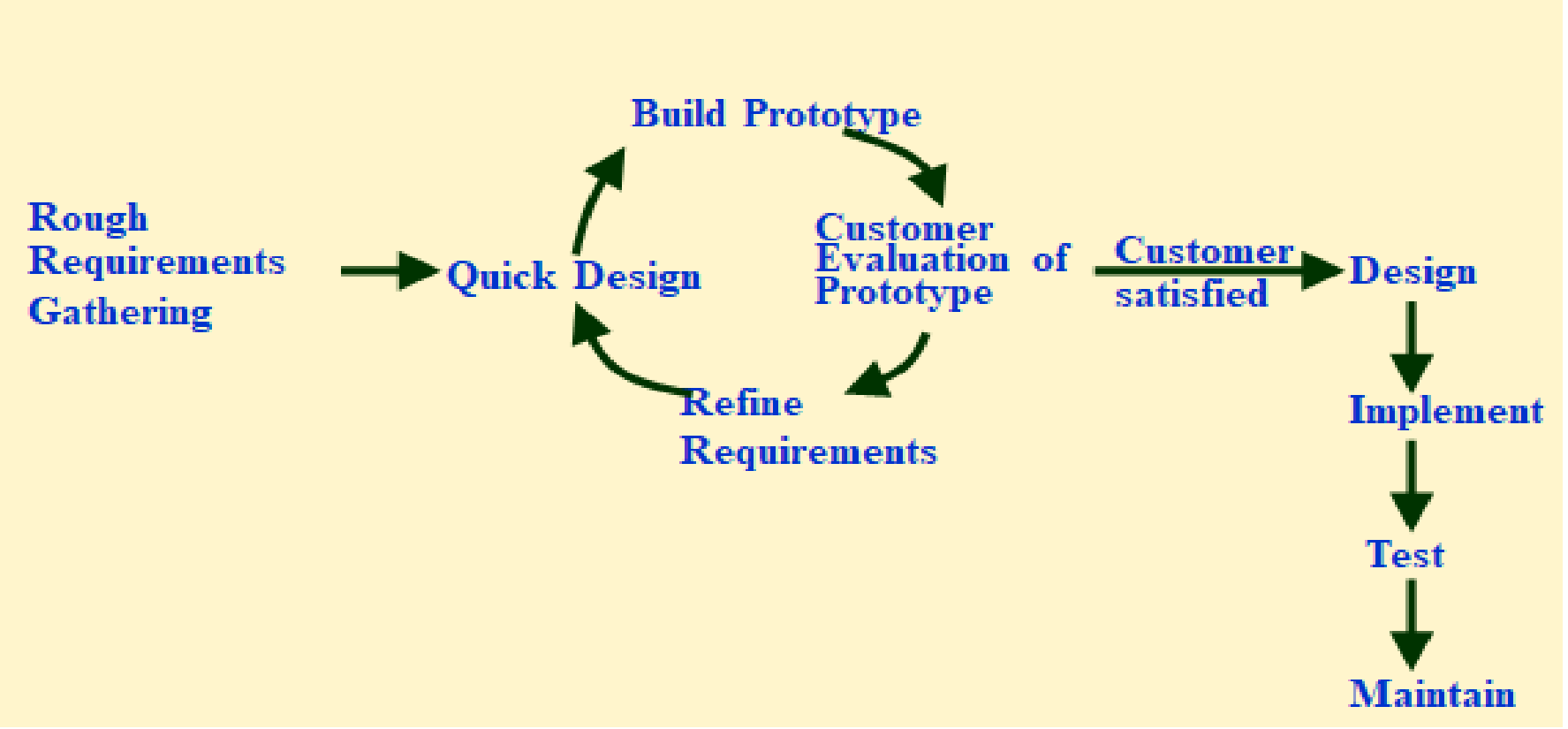
Phase containment of errors

Detection of errors as close to its point of introduction as possible

Prototyping Model

Unclear *user requirements*

Unresolved *technical issues*



Prototyping Model

Water fall model assumes that **requirements are completely known before** development starts

It's not the case always

In such cases, before starting actual development,

A **"working prototype"** of the system should first be built

A prototype is a toy implementation of a system with:

limited functional capabilities (dummy functions) , low reliability & inefficient performance

It **Illustrates** the **system** to the **customer**

For providing **complete requirements**

Prototyping Model

Start with approximate or rough requirements

Carry out a quick design

The **prototype** is submitted to **customer** for **evaluation**

Based on the user feedback, requirements are refined This cycle continues until the user **approves** the prototype

The actual system is developed using the **classical waterfall** approach

Prototyping Model

Final working prototype (with all user feedbacks) serves as an **animated requirements specification**

The **prototype** is usually **thrown away:**

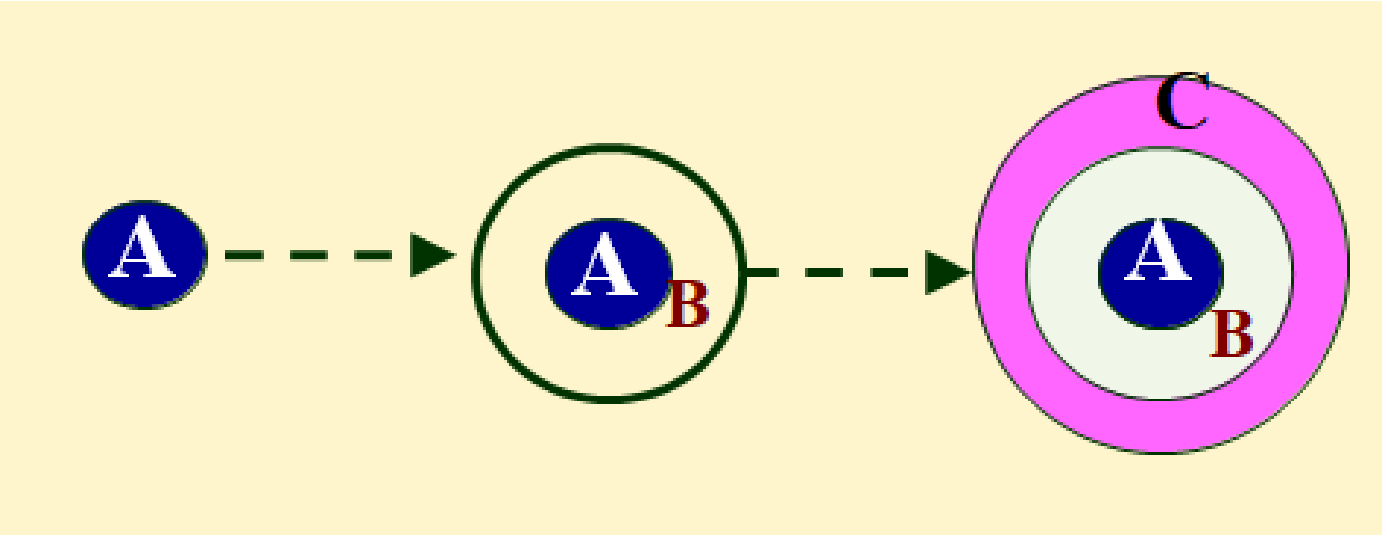
But, the experience gained **helps** with developing the actual product

Evolutionary / Incremental Model

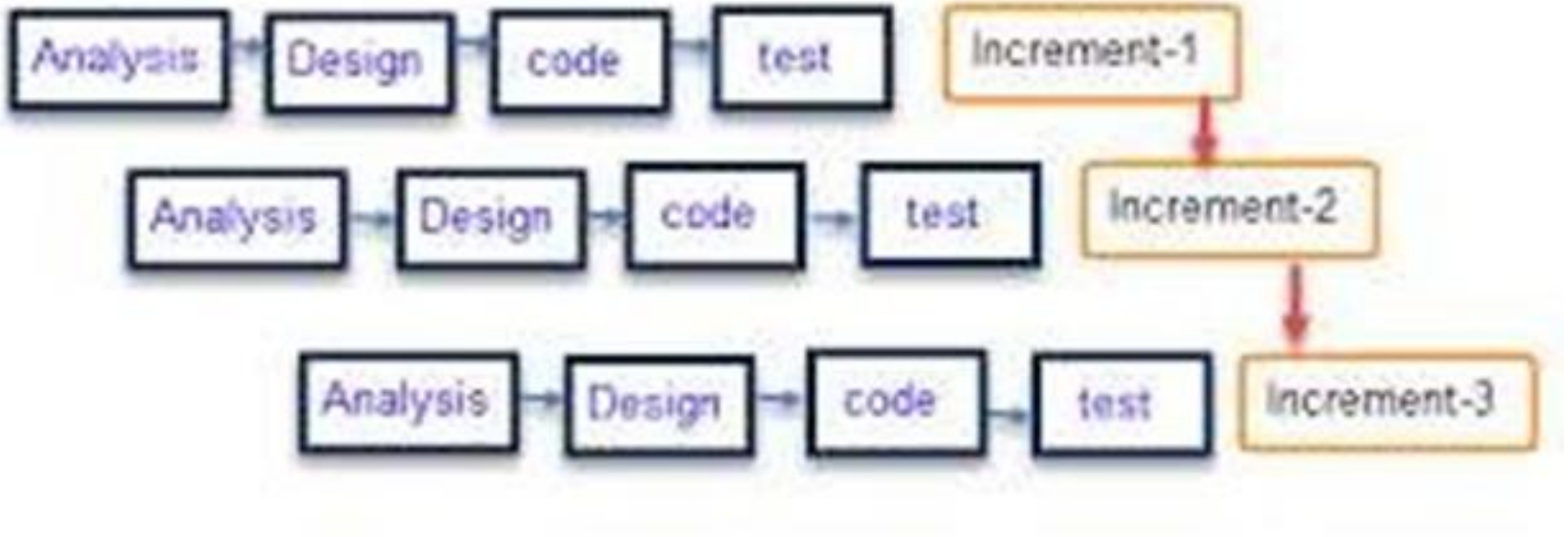
The system is broken into several modules & implemented incrementally & delivered.

Develop the core modules /skeleton of the system first

Add functionalities progressively to make successive versions



Evolutionary / Incremental Model



Incremental Model

Evolutionary / Incremental Model

Advantages:

Users get a chance to **experiment** with a partially developed system : before final product is released

Helps finding **exact user requirements** **Core**

modules get **tested** thoroughly:

Reduces chances of errors in final product

Evolutionary / Incremental Model

Disadvantages:

Often, it is **difficult to subdivide** **problems** into **functional units**:

Which can be incrementally implemented & delivered

Evolutionary model is **suitable for large problems**,

Which can be easily divided into modules

Spiral Model

Proposed by **Barry Boehm** in **1988**

This model appears like a **spiral with many loops**

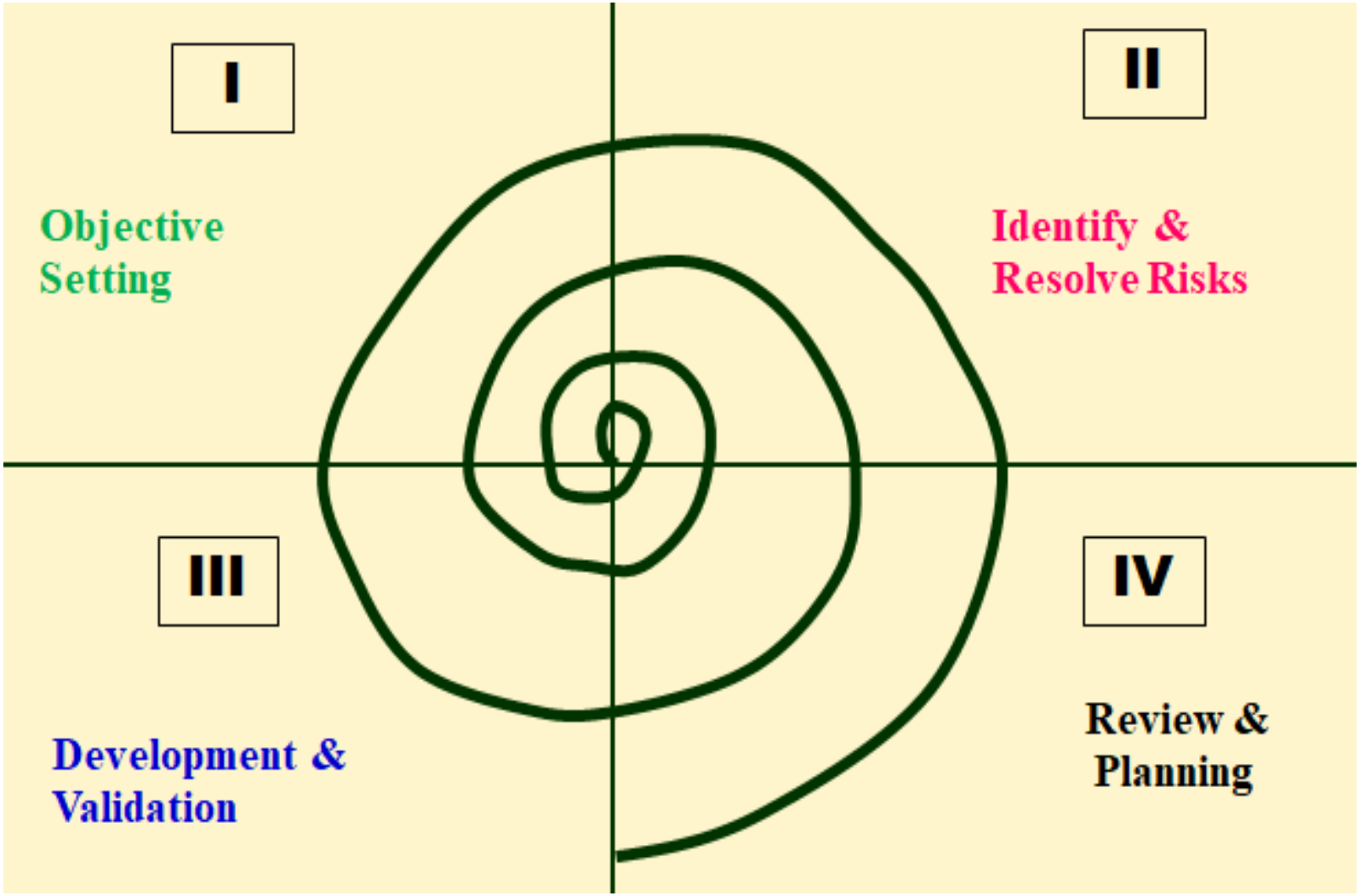
Each **loop** represents a **phase of the s/w dev. process**

The innermost loop may be feasibility study phase
Next loop: requirements Analysis phase
Next one: system design, and so on

There are **no fixed number of loops** or phases

The team decides: How to structure the project into phases

Spiral Model



Spiral Model

Each loop in the spiral is split into 4 quadrants

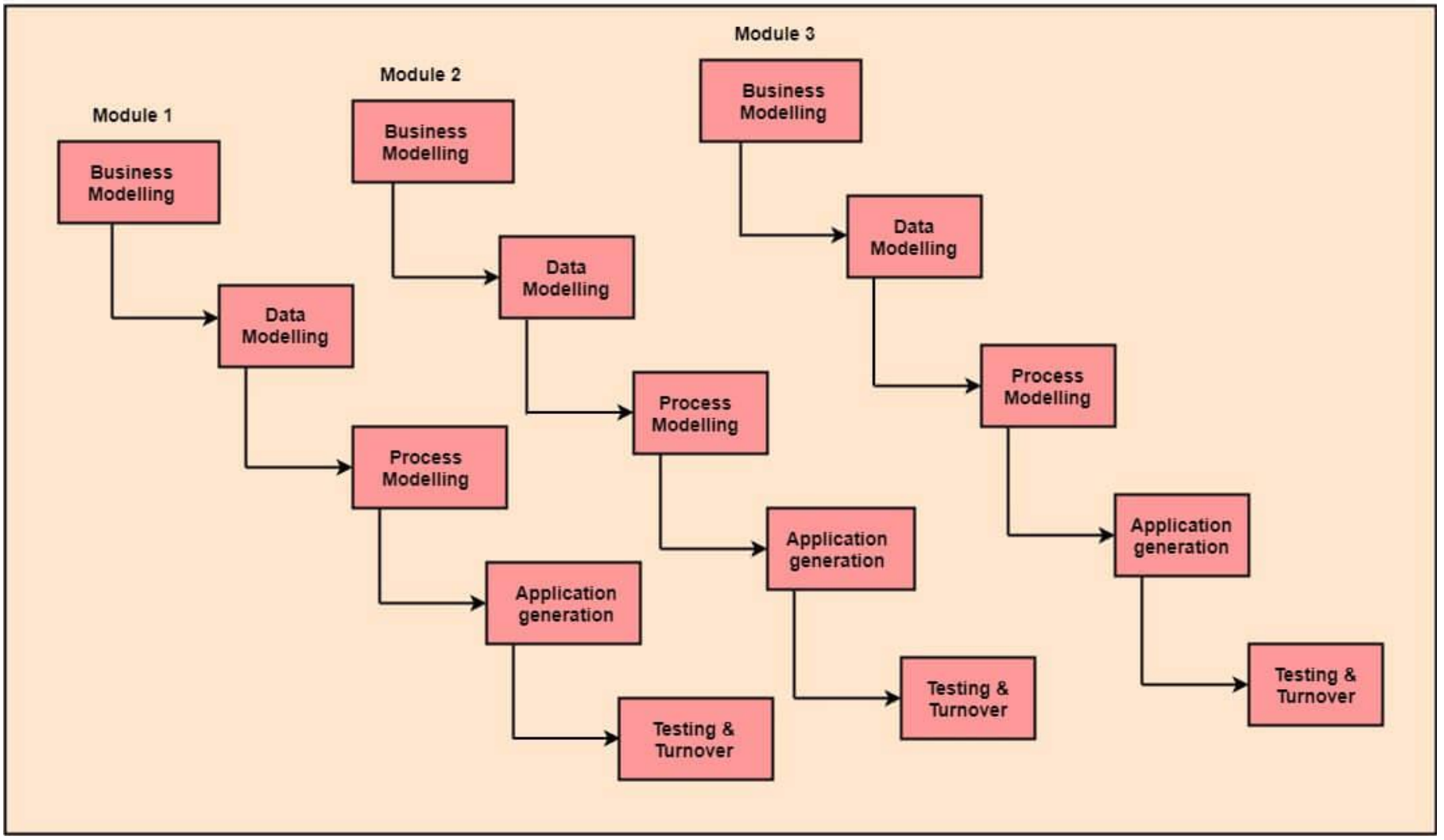
The following activities are carried out in each phase :

Objective Setting (1st Quadrant)

Identify & Resolve Risks (2nd Quadrant) Development & Validation (3rd Quadrant) Review & Planning (4th Quadrant)

RAD Model

Fig: RAD Model



RAD Model Phase

1.Business Modelling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

2. Data Modelling: The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.

3. Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

RAD Model Phase

4. Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

5. Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

When to Use RAD Model

- When a system needs to be produced in a short span of time (2-3 months)
- When the requirements are known
- When the user will be involved all through the life cycle
- When technical risk is less
- When there is a necessity to create a system that can be modularized in 2-3 months of time
- When a budget is high enough to afford designers for modeling along with the cost of automated tools for code generation

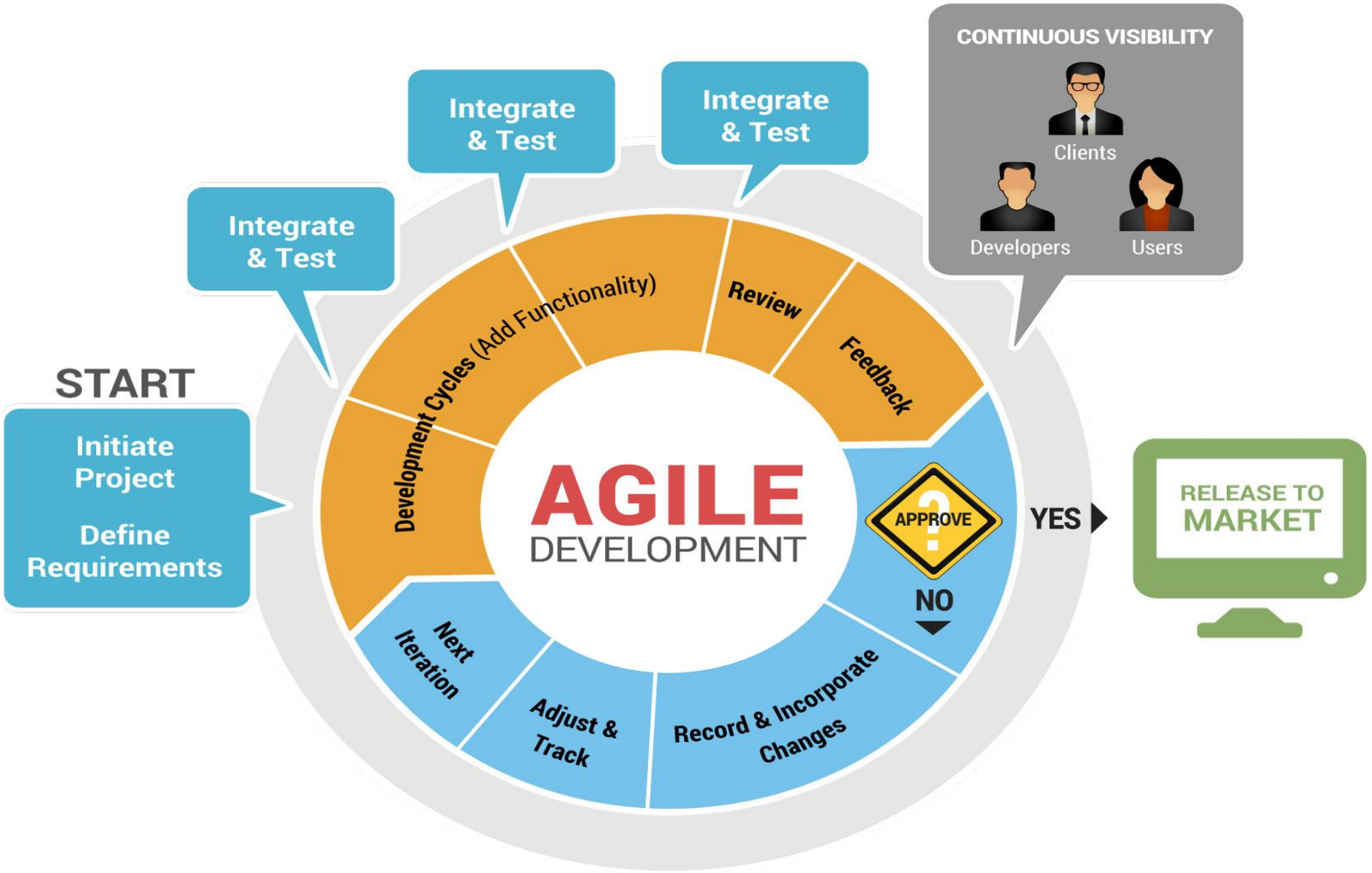
Advantages of RAD Model

- 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.

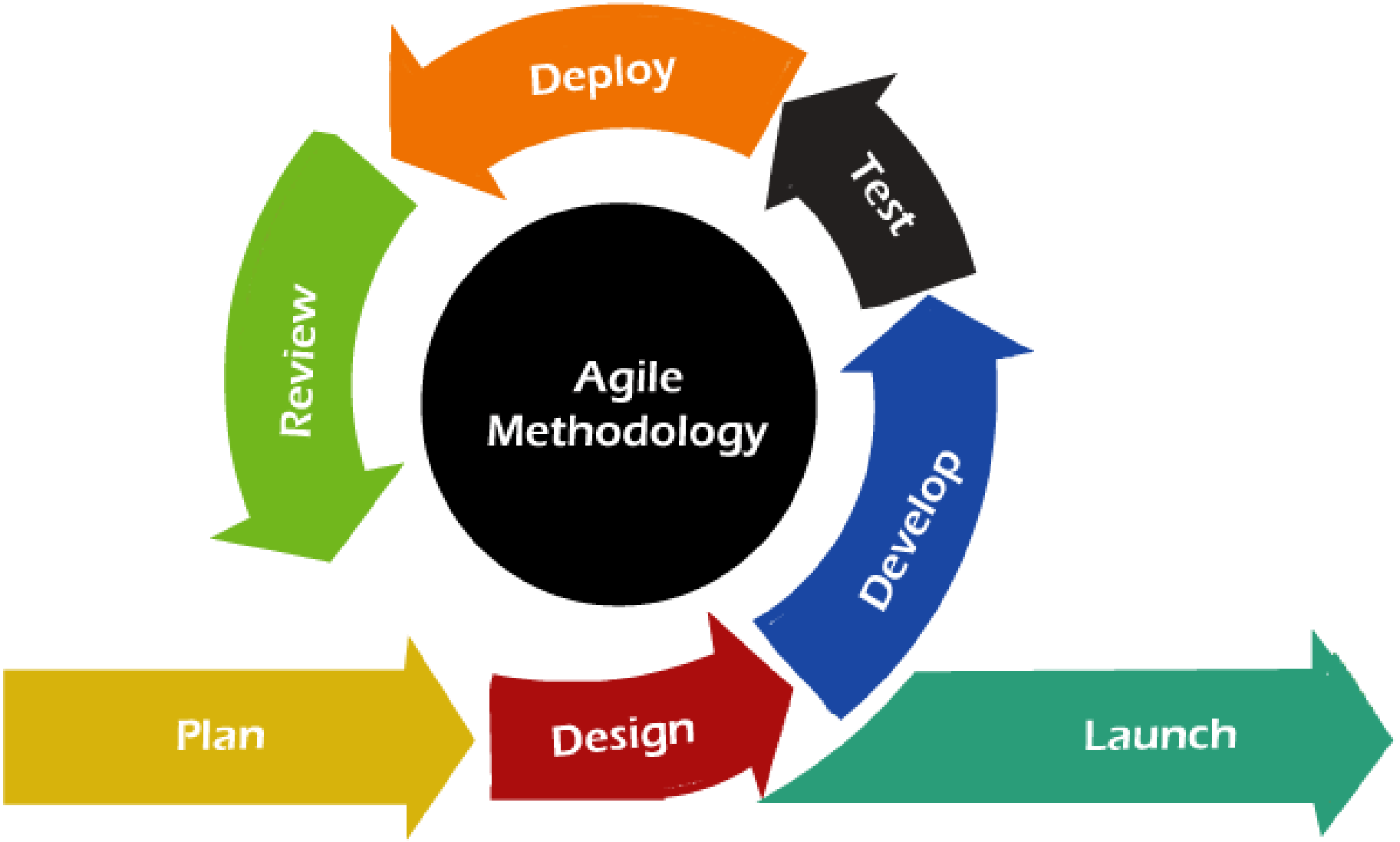
Disadvantages of RAD Model

- 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.

Advantages of RAD Model



Agile Model Phase



Agile Model Phase

Requirements Gathering: In this Agile model phase, you must define the requirements. The business opportunities and the time and effort required for the project should also be discussed. By analyzing this information, you can determine a system's economic and technical feasibility.

Design the Requirements: Following the feasibility study, you can work with stakeholders to define requirements. Using the UFD diagram or high-level UML diagram, you can determine how the new system will be incorporated into your existing software system.

Develop/Iteration: The real work begins at this stage after the software development team defines and designs the requirements. Product, design, and development teams start working, and the product will undergo different stages of improvement using simple and minimal functionality.

Agile Model Phase

Test: This phase of the Agile Model involves the testing team. For example, the Quality Assurance team checks the system's performance and reports bugs during this phase.

Deployment: In this phase, the initial product is released to the user.

Feedback: After releasing the product, the last step of the Agile Model is feedback. In this phase, the team receives feedback about the product and works on correcting bugs based on the received feedback.

When to use the Agile Model?

- It is used when there are frequent changes that need to be implemented.
- Low-regulatory-requirement projects
- Projects with not very strict existing process
- Projects where the product owner is highly accessible
- Projects with flexible timelines and budget

Advantages of the Agile Model

- Communication with clients is on a one-on-one basis.
- Provides a very realistic approach to software development
- Agile Model in software engineering enables you to draft efficient designs and meet the company's needs.
- Updated versions of functioning software are released every week.
- It delivers early partial working solutions.
- Changes are acceptable at any time.
- You can reduce the overall development time by utilizing this Agile Model.
- It allows concurrent development and delivery within an overall planned context.
- The final product is developed and available for use within a few weeks.

Disadvantages of Agile Model

- There is a higher risk of sustainability, maintainability, and extensibility.
- In some corporations, self-organization and intensive collaboration may not be compatible with their corporate culture.
- Documentation and design are not given much attention.
- Without clear information from the customer, the development team can be misled.
- Not a suitable method for handling complex dependencies.

Popular Agile Models

- **Crystal**
- **Atern**
- **Scrum**
- **Extreme Programming (XP)**
- **Lean Development**
- **Unified process**

Scrum Model

- **Scrum** is a framework for project management that emphasizes teamwork, accountability and iterative progress toward a well-defined goal.
- Provides effective collaborations among teams working on complex products.

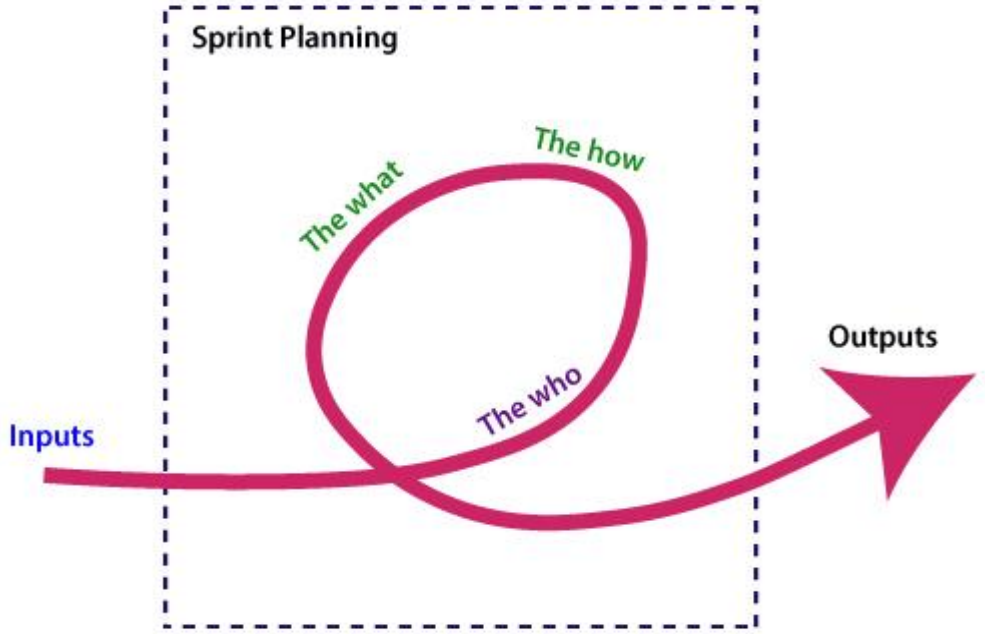
- It is agile technology that consists of meetings, roles, and tools to help teams working on complex projects collaborate and better structure and manage their workload.

- Major focus on responsibility, teamwork, and iterative progress toward a well-defined business goal.

Scrum Model

Sprint:

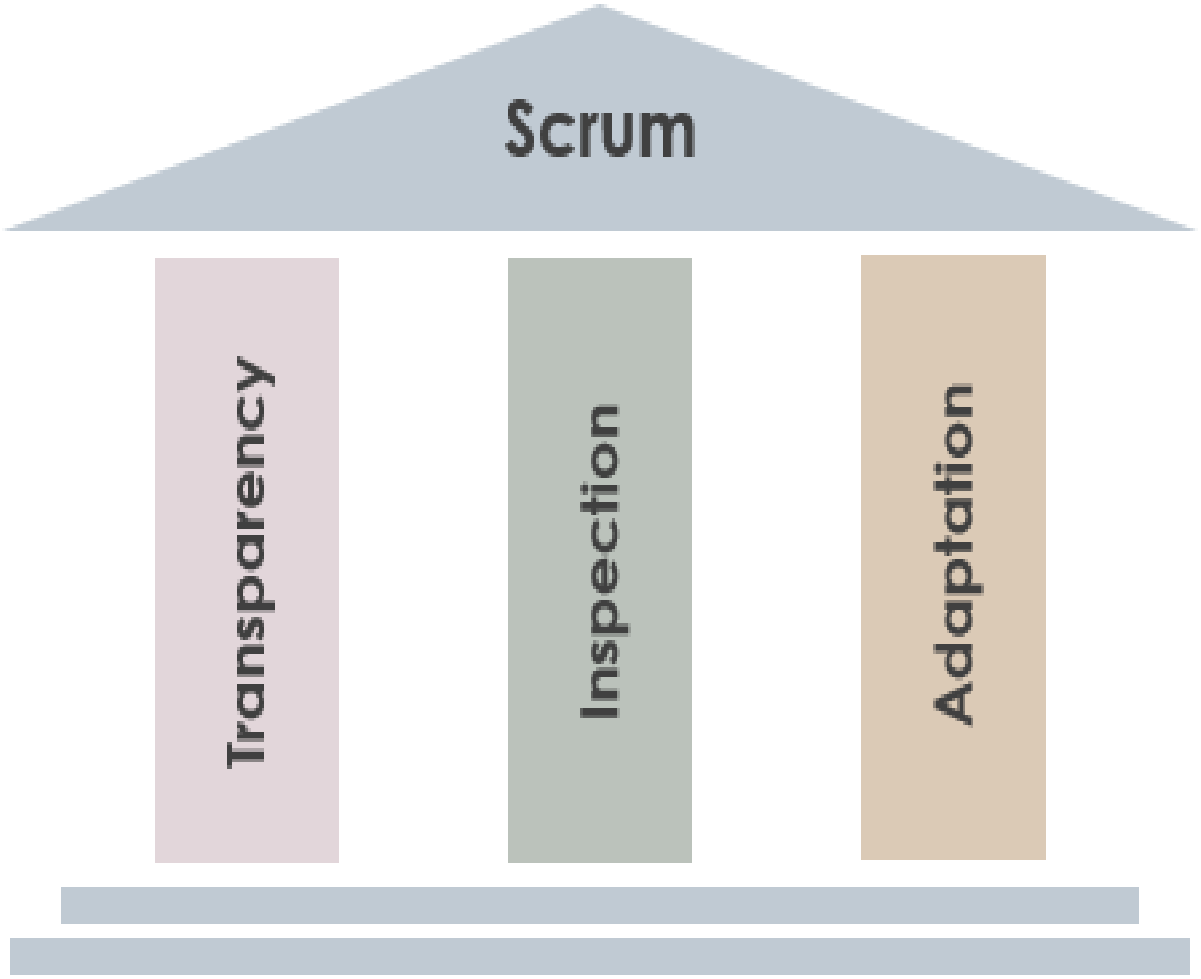
- Sprint is a short, time-boxed period for Scrum team that works to complete a set amount of work.
- Sprints are the core component of Scrum and agile methodology. The right sprints will help our agile team to ship better software.



Scrum Model

- **The What:** The product owner describes the goal of the sprint and the backlog items which contribute to achieve that goal.
- **The How:** Agile development team plans its necessary work on how to achieve and deliver the sprint goal.
- **The Who:** The product owner defines the goal based on the value that the customers seek. And the developer needs to understand how they can or cannot deliver that goal.
- **The Inputs:** The product backlog provides the list of input stuff that could potentially be part of the current sprint. The team looks over the existing work done in incremental ways.
- **The Outputs:** The critical outcome of sprint planning is to meet described team goal. The product set the goal of sprint and how they will start working towards the goal.

Three pillars of Scrum



Transparency

Giving visibility to the significant aspects of the process to those responsible for the outcome.

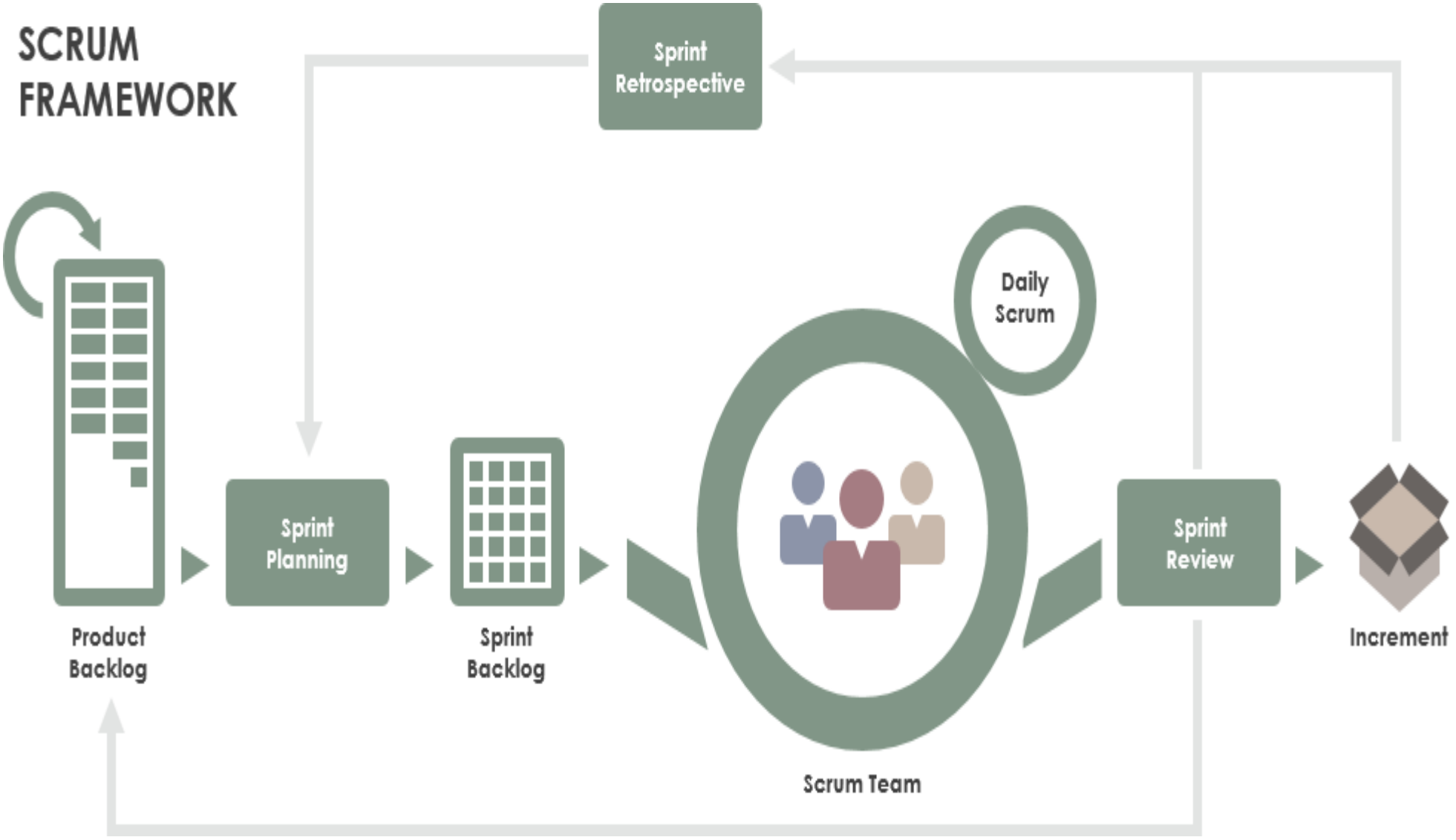
Inspection

Timely checks on the progress toward a sprint goal to detect undesirable variances.

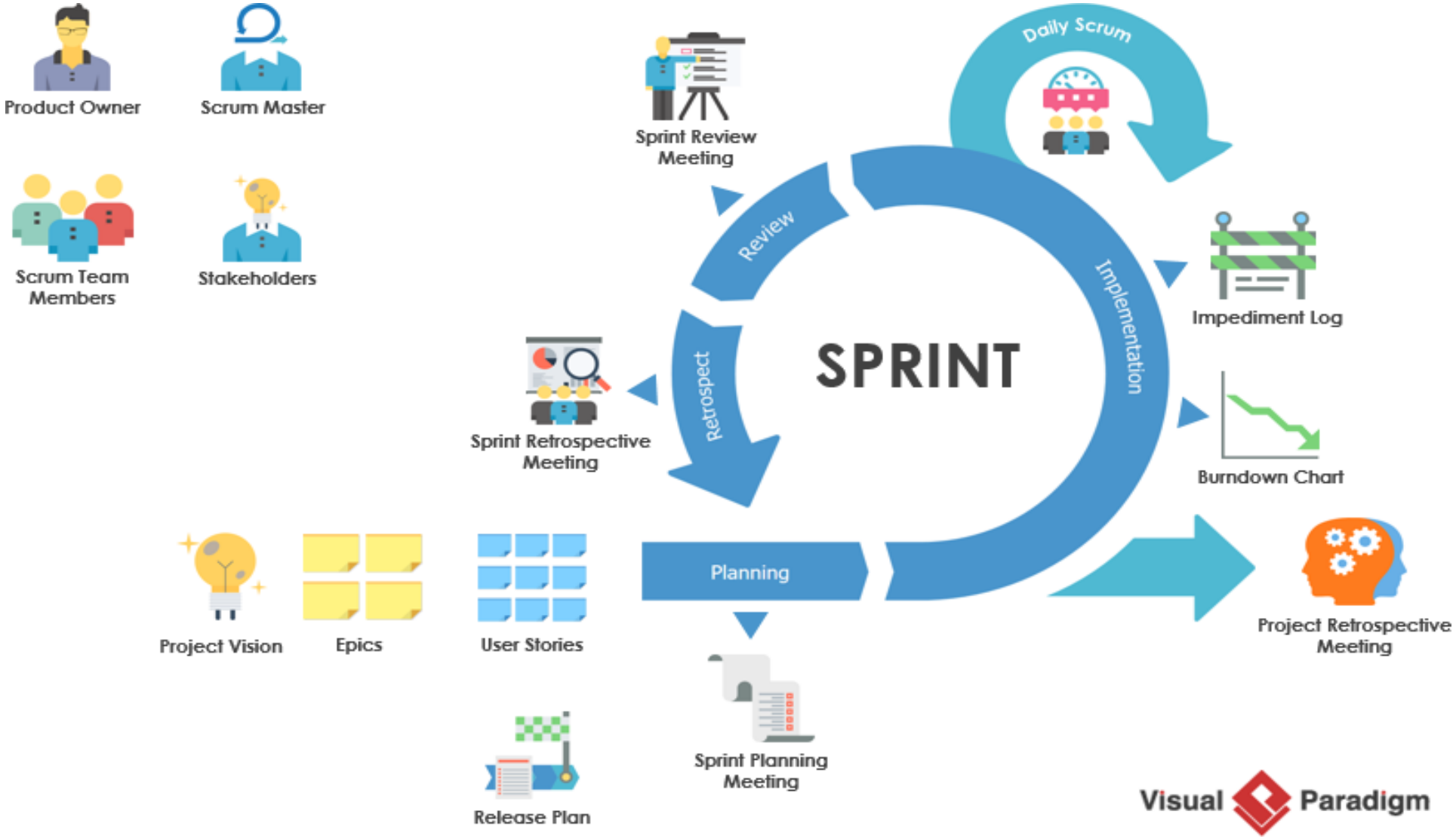
Adaptation

Adjusting a process as soon as possible to minimize any further deviation or issues.

Scrum Framework



Scrum Components



Scrum Components

Product owner:

Is a scrum development role for a person who represents the business or user community and is responsible for working with the user group to determine what features will be in the product release. The main responsibilities of the Product Owner are:

- Develop the direction and strategy for the products and services, including the short and long-time goals;
- Provide or have access to knowledge about the product or the service;
- Understand and explain customer needs for the Development team;
- Gather, prioritize and manage the product or service requirements;
- Take over any responsibilities related to the product or service budget, including its profitability;
- Determine the release date for the product or service features;
- Work together with the development team on a daily basis to answer questions and take decisions;
- Accept or reject completed features related to the Sprints;

Scrum Components

Scrum master:

Is the facilitator for an agile development team. Scrum is a methodology that allows a team to self-organize and make changes quickly, in accordance with agile principles.

The scrum master manages the process for how information is exchanged.

The main responsibilities of the Scrum Master are:

- Act as a coach, helping the team to follow scrum values and practices;
- Help to remove impediments and protect the team from external interferences;
- Promote a good cooperation between the team and stakeholders;
- Facilitate common sense within the team;
- Protect the team from organization distractions;

Scrum Components

Scrum team:

Is formed with from 3 to 9 people who MUST fulfill all technical needs to deliver the product or the service.

They will be guided directly by the Scrum Master, but they will not be directly managed.

They must be self-organized, versatile, and responsible enough to complete all required tasks.

Scrum Components

Scrum artifacts:

Are used to help define the workload coming into the team and currently being worked upon the team.

There are many more artifacts, for example, User stories, Release backlog, Burn-up chart etc.

User stories:

- A Stories is a brief statement of a product requirement or a business case.
- Are expressed in plain language to help the reader understand what the software should accomplish.
- Product owners create stories.
- A scrum user then divides the stories into one or more scrum tasks.
- A user story is typically functionality that will be visible to end users.
- A user story follows the format “I as WHO want to do WHAT, so that WHY.
- A user story delivers value to the customer/user. It’s a product requirement from the customer/user.

Scrum Components

User stories:

Example:

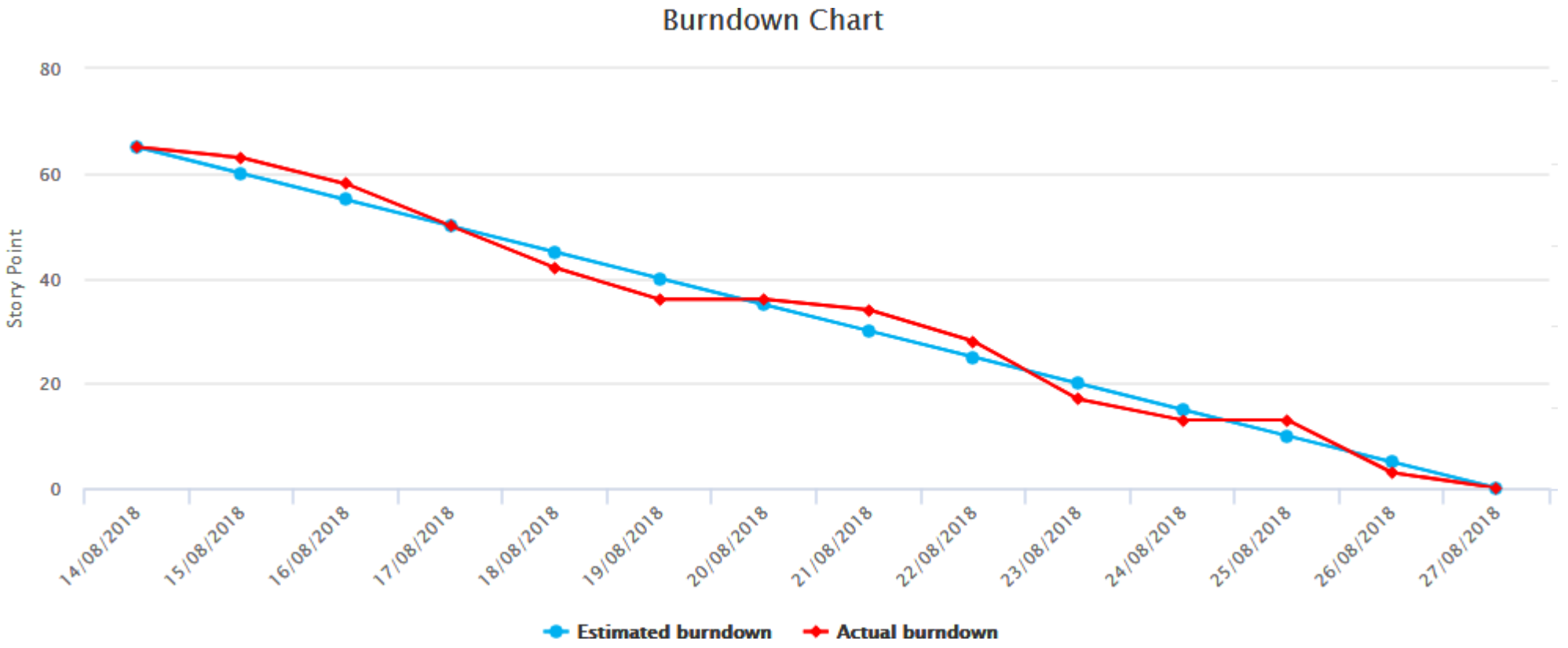
- As a customer, I want to be able to create an account so that I can see the purchases I made in the last year to help me budget for next year."

Burn-down chart

- A burn down chart is a graphical representation of work left to do versus time.
- The outstanding work (or backlog) is often on the vertical axis, with time along the horizontal.
- That is, it is a run chart of outstanding work.
- It is useful for predicting when all of the work will be completed.

Scrum Components

Burn-down chart



Scrum Components

Product backlog:

Is a collection of user stories which present functionality which is required/wanted by the product team.

Usually the product owner takes responsible for this list.

Sprint backlog:

Contain a collection of stories which could be included in the current sprint. Two important points to note

1) The team decides what gets added to the sprint. The team therefore takes ownership and responsibility of the delivery of those items.

2) Before an item can be removed from the sprint backlog and added to the sprint, the team must ensure they have all the information needed within the backlog.

Advantages Scrum Model

- Transparent system pushes developers to comply with their assignments and deliver it on time
- Defined deadline at every step keep developers motivated and empowered at every step
- Feedback at every level of the project ensures that quality project is delivered in the end

Disadvantages Scrum Model

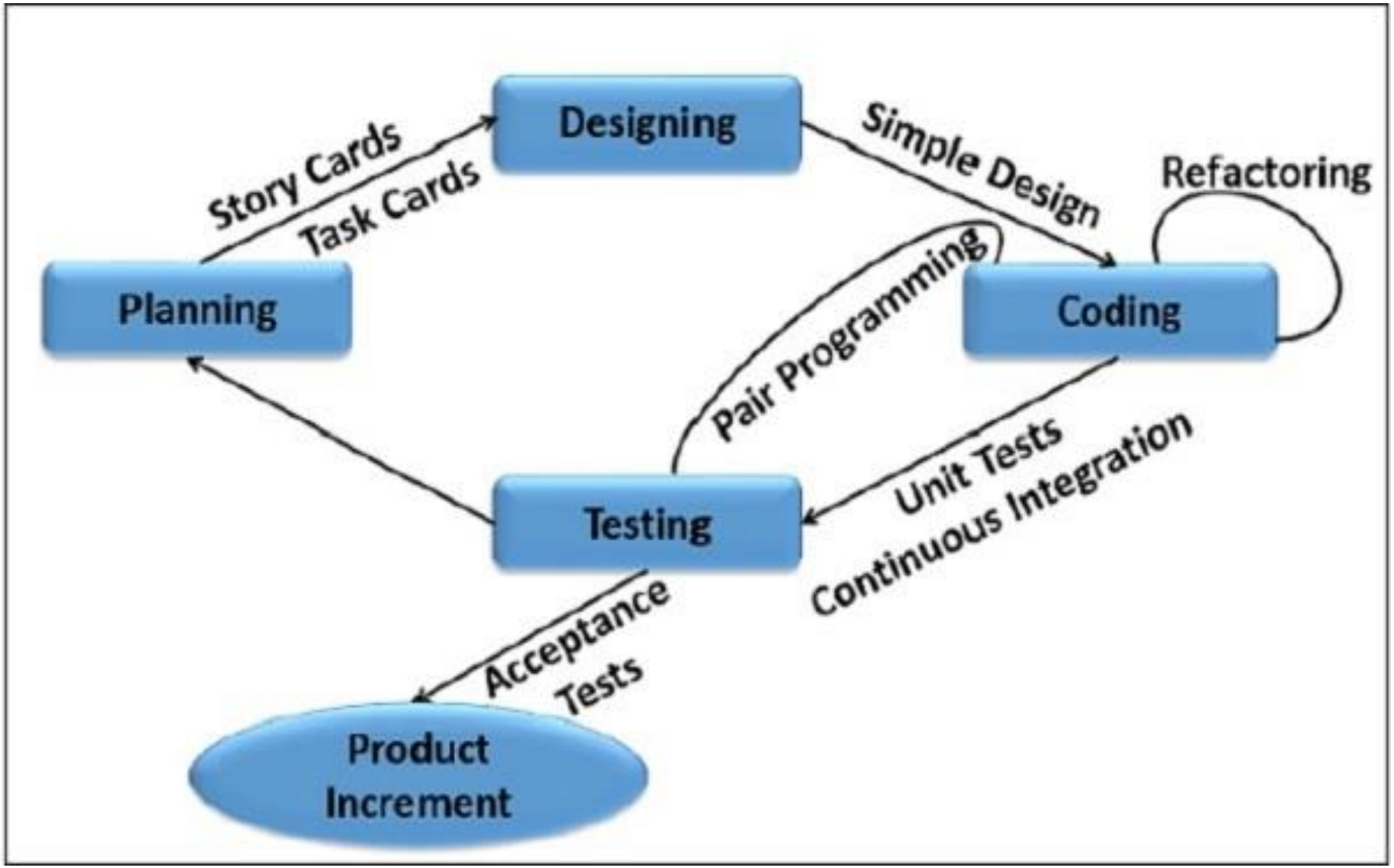
- Difficult to plan, structure and organize a project with no clear mission and vision
- Frequent changes in the project lead to a delay in the delivery time of the project
- Utilizes more resources and stakeholder's involvement in every small detail change and discussion

Extreme Programming (XP)

- Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team.
- XP is the appropriate engineering practices for software development.
- Extreme programming is an Agile project management methodology that targets speed and simplicity with short development cycles and less documentation.
- The process structure is determined by five guiding values, five rules, and 12 XP practices.
- XP is much more disciplined, using frequent code reviews and unit testing to make changes quickly.
- It's also highly creative and collaborative, prioritizing teamwork during all development stages.

When Applicable

- Dynamically changing software requirements
- Risks caused by fixed time projects using new technology
- Small, co-located extended development team
- The technology you are using allows for automated unit and functional tests
- **Manage a smaller team.** Because of its highly collaborative nature, XP works best on smaller teams of under 10 people.
- **Are in constant contact with your customers.**
- **Have an adaptable team that can embrace change (without hard feelings).**
- **Are well versed in the technical aspects of coding.** XP isn't for beginners. You need to be able to work and make changes quickly.


Designing
Story Cards
Task Cards
Simple Design
Refactoring
Planning
Coding
Pair Programming
Unit Tests
Continuous Integration
Testing
Acceptance
Tests
Product
Increment

Lifecycle of XP

- The XP lifecycle encourages continuous integration where the team member integrates almost constantly, as frequently as hourly or daily.

❖ **Pull unfinished work from user stories**
❖ **You prioritize the most important items**
❖ **Begin iterative planning**
❖ **Incorporate honest planning**
❖ **Stay in constant communication with all stakeholders and empower the team**
❖ **Release work**
❖ **Receive feedback**
❖ **Return to the iterative planning stage and repeat as needed.**

5 values of extreme programming

Simplicity
Communication
Feedback
Courage (honest updates on your progress)
Respect

5 Rules

Planning
Managing
Designing
Coding
Testing

12 extreme programming practices

- **The planning game:** XP planning is used to guide the work. The results of planning should be what you're hoping to accomplish and by when, and what you'll do next.
- **Customer tests:** When you finish a new feature, the customer will develop an acceptance test to determine how close it is to their original user story.
- **Small releases:** XP uses small, routine releases to gain insights throughout the process.
- **Simple design:** The XP system is designed for simplicity—you'll produce only what is necessary and nothing more.
- **Pair programming:** All programming comes from a pair of developers who sit side by side. There is no solo work in extreme programming.
- **Test-driven development (TDD):** XP's reliance on feedback requires heavy testing. Through short cycles, programmers release automated tests and then immediately react.

12 extreme programming practices

- **Refactoring:** This is where you'll pay attention to the finer details of the codebase, removing duplicates and making sure that the code is cohesive.
- Coll**ective ownership:** Any coding pair can change the code at any time, whether or not they developed it.
- **Continuous integration:** XP teams don't wait for completed iterations, they integrate constantly.
- **Sustainable pace:** Teams should decide how much work they can produce in this way per day and per week, and use that to set work deadlines.
- **Metaphor:** It's decided as a team, and uses language to describe how the team should function.
- **Coding standard:** XP teams follow one standard. XP developers code in the same, unified way so that it reads like one developer.

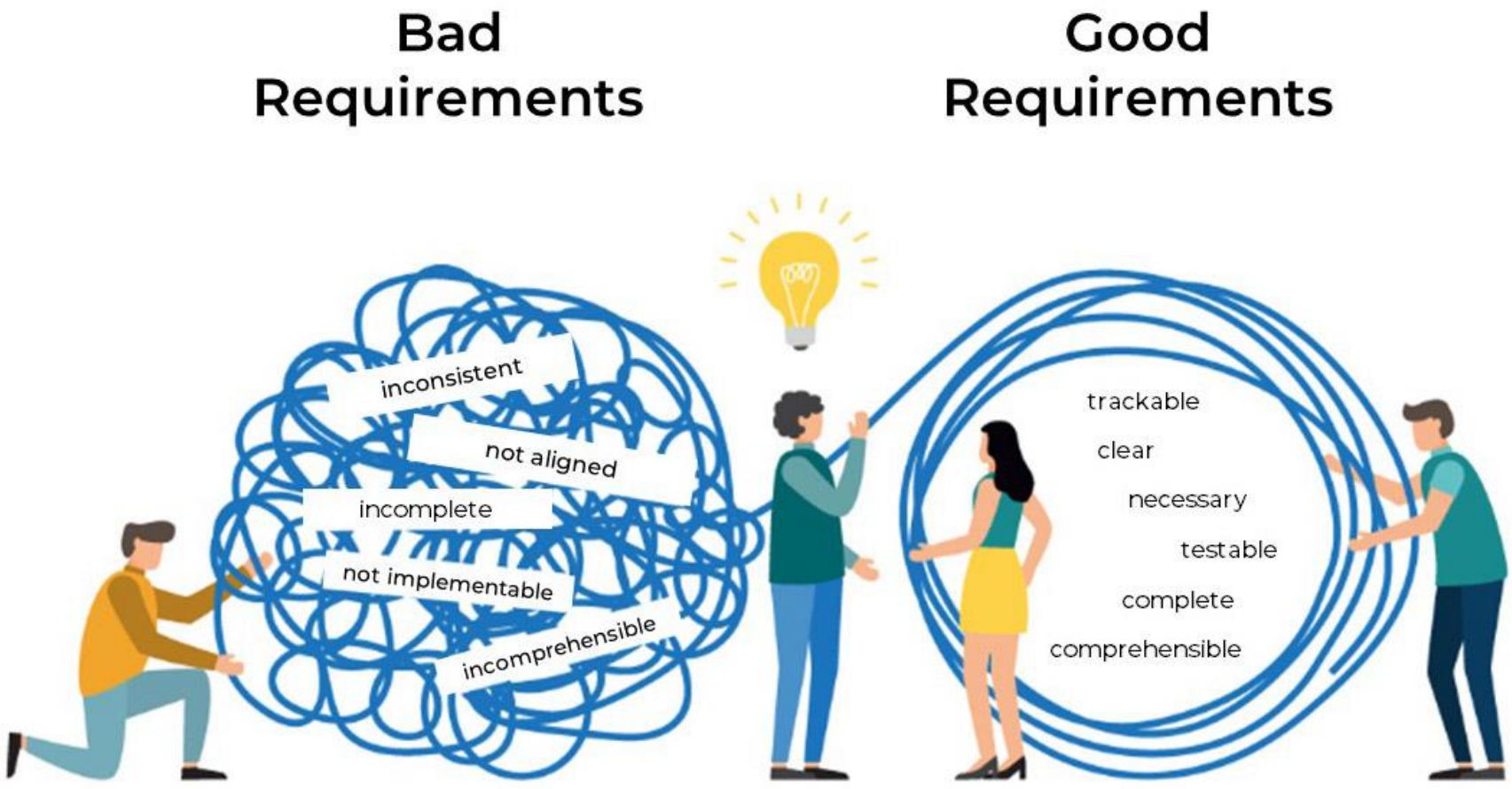
THANKS

MOUDLE-2

S/W Requirements

- Description of features and functionalities of the target system. Requirements convey the expectations of users from the software product.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.
- ***Describes in detail how the software will behave and the functionality it offers the user.***
- Defined as a high-level abstract statement or a detailed mathematical functional specification of a system's services, functions, and constraints.
- They are depictions of the characteristics and functionalities of the target system.

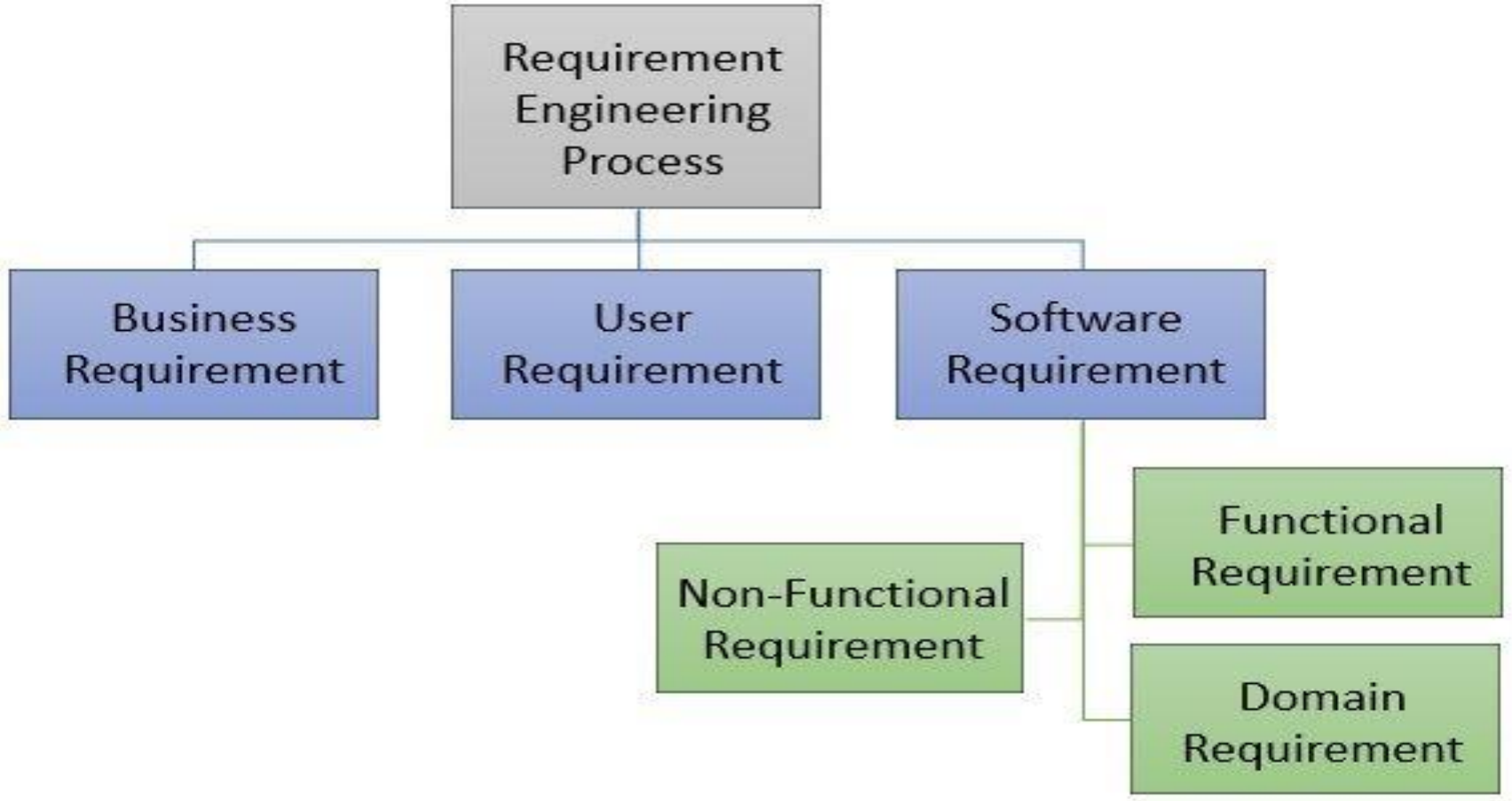
S/W Requirements



Software Requirements Characteristics

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Types of Software Requirements



Functional Requirements

- Functional requirements are such software requirements that are demanded explicitly as basic facilities of the system by the end-users.
- It describes system behavior under specific conditions.
- These are represented as inputs to the software system, its behavior, and its output.
- It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.
- Functional software requirements help us to capture the intended behavior of the system.
- ***Defines what a product must do, what its features and functions are.***

Input:
System behavior:
Output:

Types of Functional Requirements

Authentication
Authorization levels
Compliance to laws or regulations
External interfaces
Transactions processing
Reporting
Business rules, etc.

Formats:
Software requirements specification document
Use cases
User stories
Work Breakdown Structure (WBS), or functional decomposition
Prototypes
Models and diagrams

Functional Requirements Examples

- An operating system requires that users enter a password and username when logging in so that the system can authenticate their identity.
- A system runs regular checks to ensure their company meets the proper legal standards for their specific software.
- An operating system provides a receipt to users upon performing a transaction, and the system records information about the transaction in a saved file.
- Whenever a user logs into the system, their authentication is done.
- In case of a cyber attack, the whole system is shut down
- Whenever a user registers on some software system the first time, a verification email is sent to the user.

Non-Functional Requirements

- Not related to the system functionality,
- Defined as the quality constraints that the system must satisfy to complete the project contract.
- Describe how a system must behave and establish constraints of its functionality.
- This type of requirements is also known as the system's *quality attributes*.

Non-Functional Requirements

Deal with issues like:

Performance
Reusability
Flexibility
Reliability
Maintainability
Security
Portability

Non-Functional Requirements Example

- An operating system translates all foreign messages to the language of the system's current location.
- An operating system shuts down automatically when detecting a potential security threat.
- A system has added features that enhance usability, which include a large cursor so that viewers can easily identify it on the screen and speech-enabled text that allows users to type using their voices.

SRS

- A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform.
- It also describes the functionality the product needs to fulfill the needs of all stakeholders (business, users).

➢ An SRS gives you a complete picture of your entire project.

➢ It provides a single source of truth that every team involved in development will follow.

➢ It is your plan of action and keeps all your teams — from development and testing to maintenance — on the same page.

SRS

- **Main aim of requirement specification:**
  - Systematically organize the requirements
  - Document requirements properly
- The SRS document is useful in various contexts:
- It serves as:
  - **Statement of user needs**
  - **Contract document**
  - **Reference document**
  - **Definition for implementation**

WHY SRS

- SRS document is not only a **reference** document :
- It is also a **contract** between the development team and the customer
  - Once the SRS document is approved by the customer,
  - Any subsequent controversies are settled by referring the SRS document

- The SRS document is known as black-box specification:
  - The system is considered as a black box whose internal details are not known
  - Only its visible external behaviour (i.e. input/output) is documented.

WHY SRS

- SRS document **concentrates** on:
  - What needs to be done
  - Carefully avoids the solution (“how to do”) aspects
- The requirements are documented:
  - Using end-user terminology

SRS OUTLINE

1. Introduction

- 1.1 Purpose
- 1.2 Intended Audience
- 1.3 Intended Use
- 1.4 Product Scope
- 1.5 Definitions and Acronyms

2. Overall Description

- 2.1 User Needs
- 2.2 Assumptions and Dependencies

3. System Features and Requirements

- 3.1 Functional Requirements
- 3.2 External Interface Requirements
- 3.3 System Features
- 3.4 Nonfunctional Requirements

Properties of a Good SRS

Concise

- The **SRS** report should be concise yet unambiguous, consistent, and comprehensive.
- Verbose and irrelevant descriptions reduce readability and increase the possibility of errors.

Well-Structured

- It must be well-structured.
- A well-structured document is easy to comprehend and alter.
- The *SRS* document is revised multiple times to meet the users' needs.

Properties of a Good SRS

Black-Box View

- The **SRS** document should only define what the system should do, not how it should accomplish it.
- This means that the *SRS* document should describe the system's outward behavior rather than discussing implementation details.
- The *SRS* report should treat the system to be developed as a black box and define the system's externally visible behavior.
- As a result, the *SRS* report is often known as a system's black-box specification.

Properties of a Good SRS

Conceptual Integrity

- It should demonstrate conceptual integrity so that the reader may simply understand it.
- Response to undesired events.
- It should define permissible responses to unfavorable events.
- This is referred to as the system's response to unusual conditions.
- **It should specify what the system must do**
  - Not say how to do it

Properties of a Good SRS

- It should be traceable
  - One should be able to trace which part of the specification corresponds to which part of the design and code and vice versa
- It should be verifiable
  - **Ex.** “system should be user friendly” is not verifiable

Properties of Bad SRS

- Unstructured Specifications:
  - Narrative essay one of the worst types of specification document
- Noise:
  - Presence of text containing information irrelevant to the problem
- Silence:
  - Aspects important to the solution of the problem are omitted

Properties of Bad SRS

- Over-specification:
  - Addressing “how to” aspects
  - Over specification restricts the solution space for the designer
- Contradictions:
  - Contradictions might arise
    - if the same thing described at several places in different ways

Properties of Bad SRS

- Ambiguity:
  - Unquantifiable aspects, e.g. “good user interface”
- Forward References:
  - References defined only later on in the text
- Wishful Thinking:
  - Descriptions of aspects
  - for which realistic solutions will be hard to find

Requirement Engineering Process

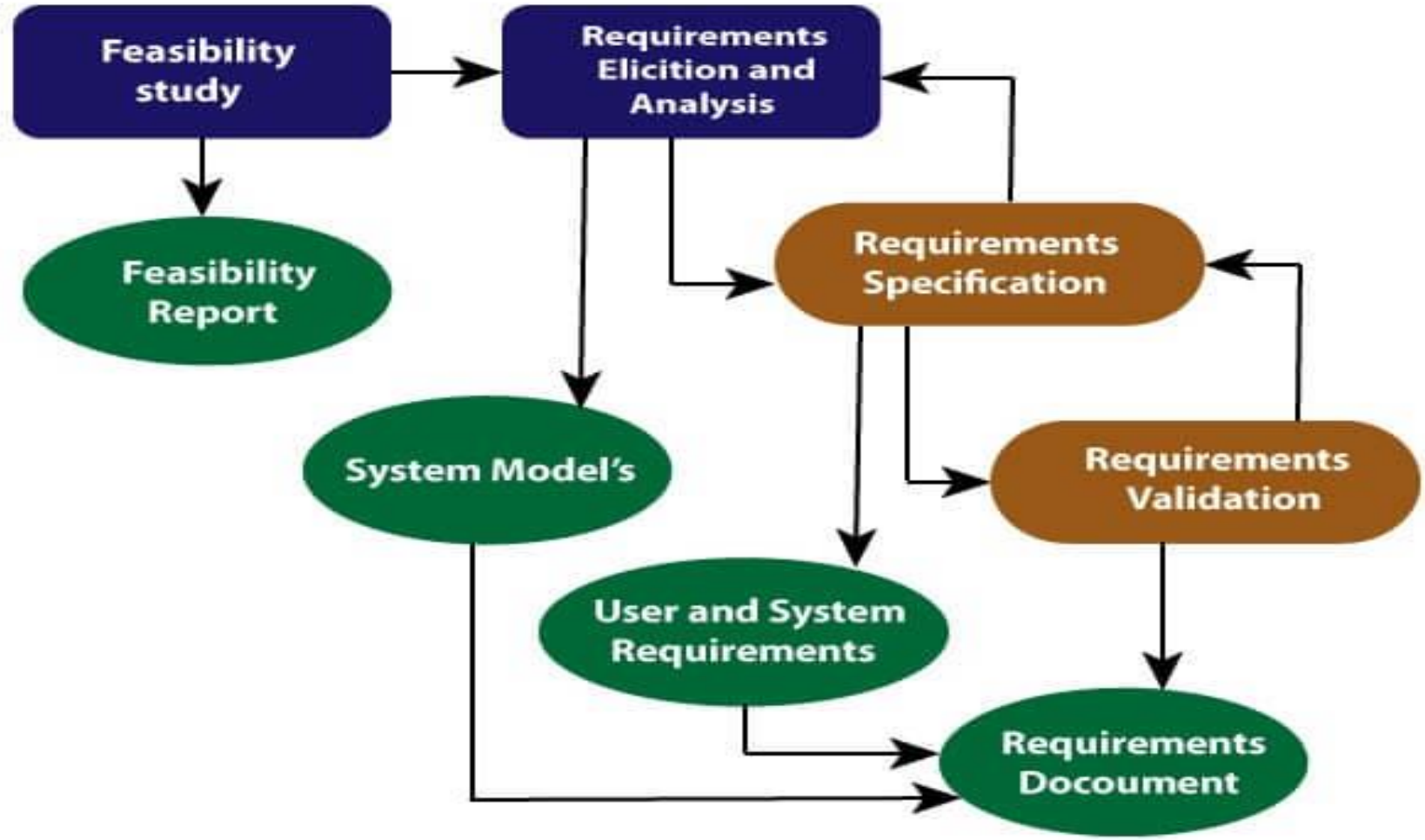
- The process of defining, documentation, and maintenance of requirements in the design process of engineering is called requirements engineering.
- Provides an apt mechanism to understand the customer's desires, analysis of needs of the customer, feasibility assessment, negotiations for a reasonable solution, clarity in the specification of the solution, specifications validation, and requirements management.
- In contrast, the requirements are being transferred to the working system.

Requirement Engineering Process

Four-step process, followed:

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

Requirement Engineering Process



Requirement Engineering Process

Requirement Engineering Process

1-Feasibility Study:

Technical Feasibility - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

Operational Feasibility - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.

Economic Feasibility - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

Requirement Engineering Process

1-Feasibility Study:

Technical Feasibility - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

Operational Feasibility - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.

Economic Feasibility - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

Requirement Engineering Process

2. Requirement Elicitation and Analysis:

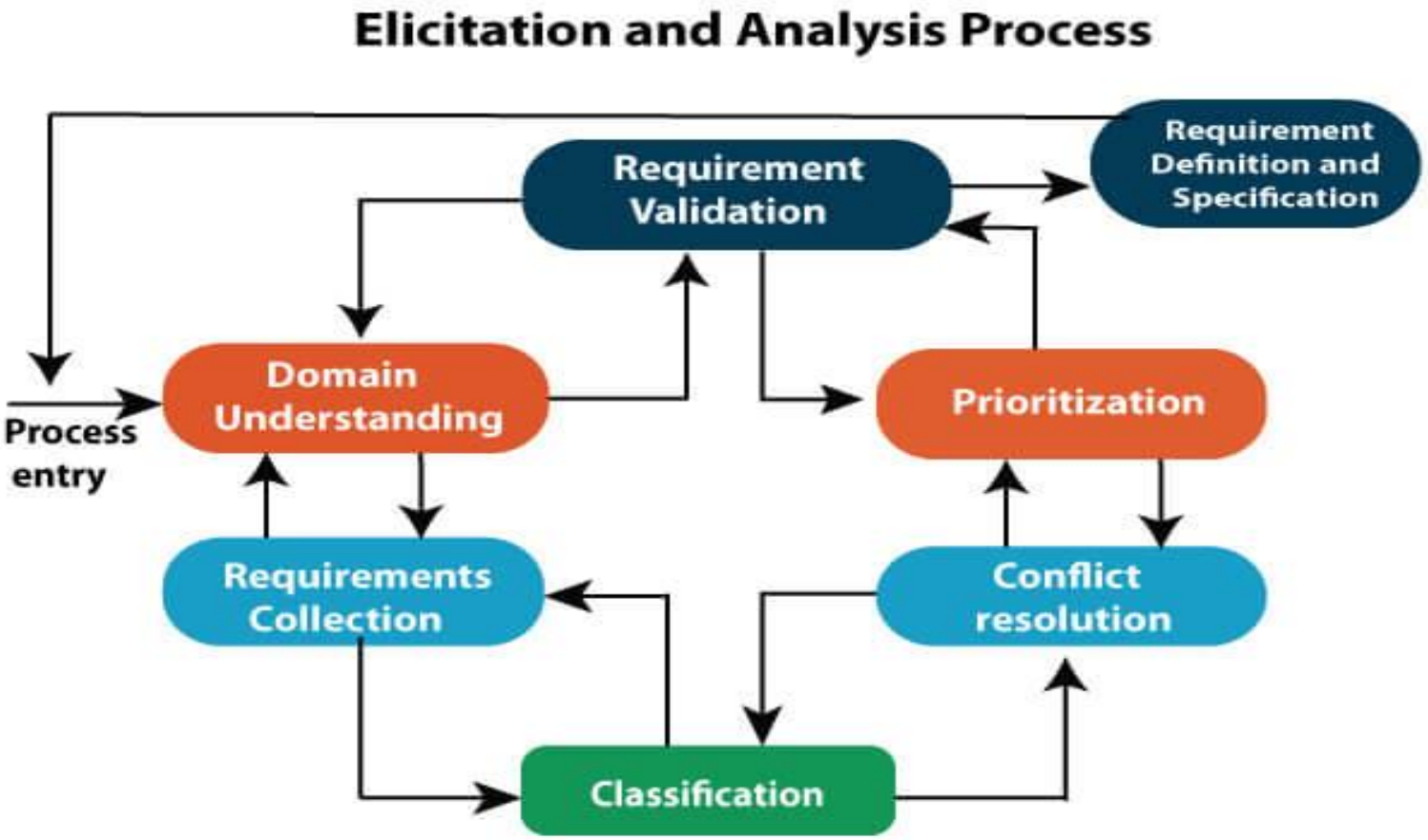
This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes:

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.

Requirement Engineering Process

2. Requirement Elicitation and Analysis:



Requirement Engineering 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.

Can be represented as:

Data Flow Diagrams: Data Flow Diagrams (DFDs) are used widely for modeling 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.

Requirement Engineering Process

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.

Requirement Engineering Process

4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirement Engineering Process

Requirements Validation Techniques

Requirements 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.

Requirement Engineering Process

5. Management of Software Requirements

- The process of managing the requirements that keep changing during the process of requirements engineering and development of the system is called management of software requirement.
- During the process of software management, there are new requirements with the change in the needs of business, and there is the development of a better understanding of the system.
- During the process of development, the requirements priority changes from different views.
- During the process of development, the technical and business environment of the system changes.

Advantages Requirement Engineering Process

Lesser chances of process overhead.

Requirements capture using requirements engineering are quicker and more precise.

Easy to identify and implement requirements in the initial stage.

Activities, communication is more efficient between the stakeholders and the software engineers.

There is a quick response to the changes in requirements.

Response to the changes in the requirements can be quicker and can be done at a relatively lower cost.

Decision Tree & Table

Representation of complex processing logic:

Decision Tree:

- A Decision Tree is a graph that uses a branching method to display all the possible outcomes of any decision.
- It helps in processing logic involved in decision-making, and corresponding actions are taken.
- It is a diagram that shows conditions and their alternative actions within a horizontal tree framework.
- It helps the analyst consider the sequence of decisions and identifies the accurate decision that must be made.

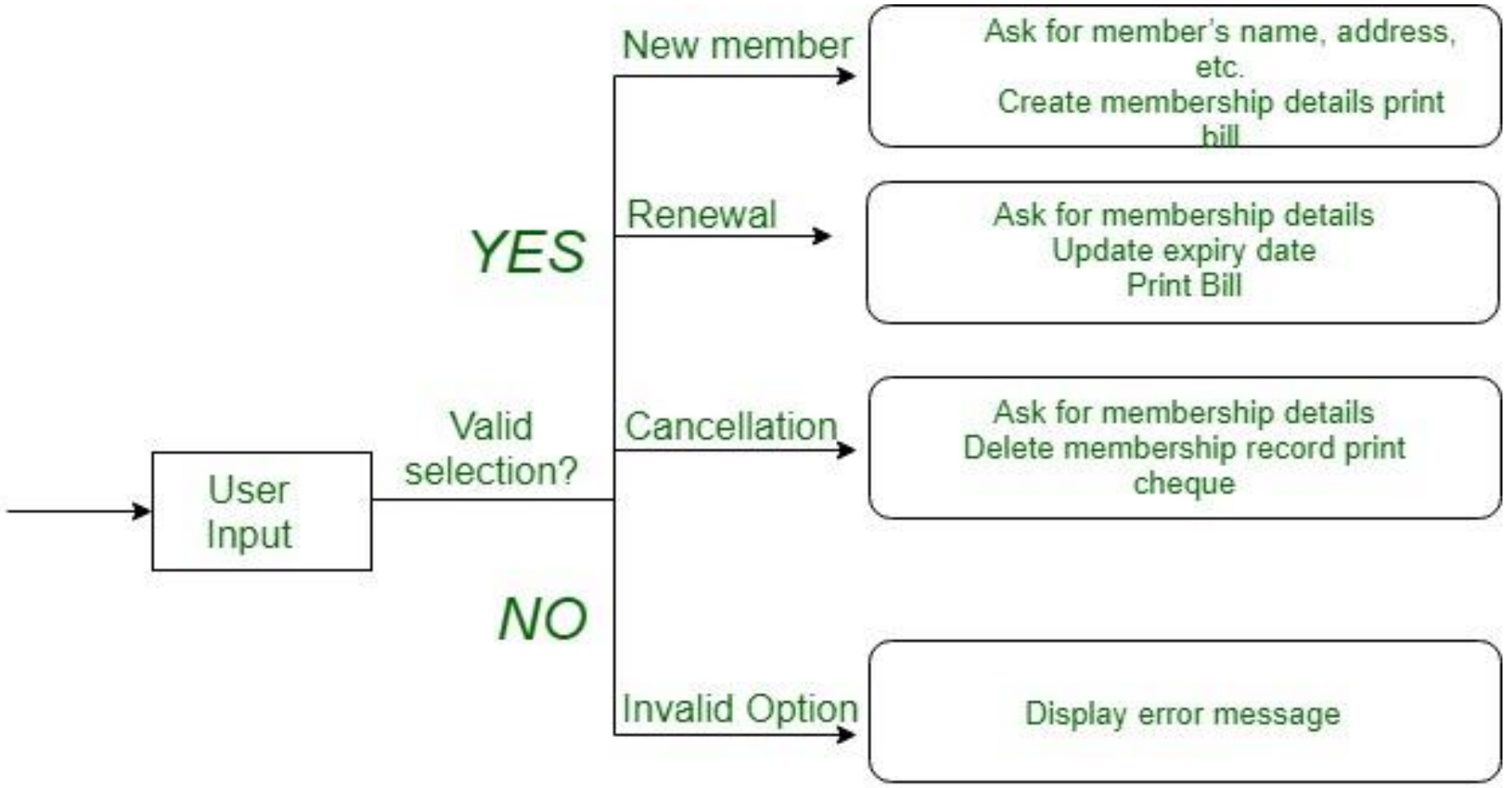
Decision Tree & Table

Advantages of Decision Tree:

- Decision trees represent the logic of If-Else in a pictorial form.
- Decision trees help the analyst to identify the actual decision to be made.
- Decision trees are useful for expressing the logic when the value is variable or action depending on a nested decision.
- It is used to verify the problems that involve a limited number of actions.

Decision Tree & Table

Decision Tree:



Decision tree for LMS

Decision Tree & Table

Decision Table:

- Data is stored in the tabular form inside decision tables using rows and columns.
- A decision table contains condition entries, condition stubs, action entries, and action stubs.
- The upper left quadrant contains conditions.
- The upper right quadrant contains condition alternatives or rules. The lower right quadrant contains action rules, and the lower-left quadrant contains actions to be taken.
- Verification and validation of the decision table are much easy to check, such as Inconsistencies, Contradictions, Incompleteness, and Redundancy.

Decision Tree & Table

Decision Table:

- Data is stored in the tabular form inside decision tables using rows and columns.
- A decision table contains condition entries, condition stubs, action entries, and action stubs.
- The upper left quadrant contains conditions.
- The upper right quadrant contains condition alternatives or rules. The lower right quadrant contains action rules, and the lower-left quadrant contains actions to be taken.
- Verification and validation of the decision table are much easy to check, such as Inconsistencies, Contradictions, Incompleteness, and Redundancy.

Decision Tree & Table

Decision Table:

Requirement Number					
Condition	1	2	3	4	5
User is authorized	F	T	T	T	T
Chemical is available	—	F	T	T	T
Chemical is hazardous	—	—	F	T	T
Requester is trained	—	—	—	F	T
Action					
Accept request			X		X
Reject request	X	X		X	

Formal System Specification

A formal technique is **a mathematical method to specify a hardware or software system**, verify whether a specification is realizable, verify that an implementation satisfies its specification, prove properties of a system Without necessarily running the system.

It consists two sets:

Syn and sem and a relation set sat.

Syn: syntactic domain

Sem: semantic domain

Sat: satisfaction relation

Sat(syn,sem)

Formal System Specification

Syn: syntactic domain

Consists of an alphabet of symbols and set of formation rules to construct well formed formulas from the alphabets.

Sem: Semantic domains

Consists of ADT to specify algebras, theories and programs.

Thank You