SOFTWARE ENGINEERING

Module 2

Module-II Syllabus

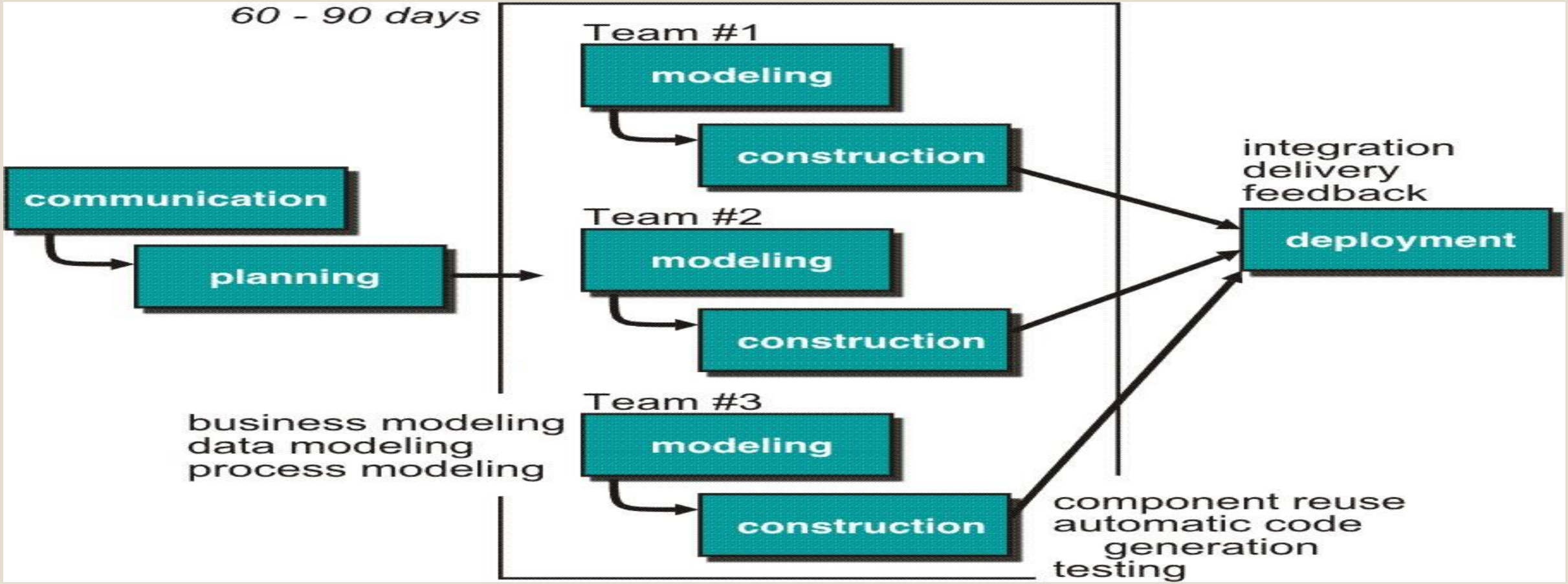
UNIT II Software development phases and processes

Software development and processes – RAD, **RUP, Agile: Scrum**, Prototyping

Development phases of Software in relation to Processes

What to develop? – Requirements gathering and Analysis, Types- functional, nonfunctional, system, **User Interface**, **quality requirements** and putting together– UML use cases, scenarios.

Rapid Application Development (RAD) Model



Makes heavy use of reusable software components with an extremely short development cycle

RAD model

- **Communication** – to understand problem information
- **Planning** – multiple s/w teams works in parallel on diff. system.
- **Modeling** –
  - **Business modeling** – Information flow among business is working.

  Ex. What kind of information drives?

  Who is going to generate information?

  From where information comes and goes?
  - **Data modeling**
  - **Process modeling**

RAD

- **Construction –** it highlighting the use of pre-existing software component
- Automatic code generation is done..
- **Deployment** – Deliver to customer basis for subsequent iteration.
- RAD model emphasize a short development cycle.
- "High speed" edition of linear sequential model.
- If requirement are well understood and project scope is constrained then it enable development team to create " fully functional system" within a very short time period

RAD Model **Drawback**

- Large projects more people are required & large teams
- If developer & customer are not committed RAD fails
- Performance is major issue then RAD fails
- RAD is suitable where risk is low
- RAD requires sufficient human resources
- Not appropriate when technical risks are there

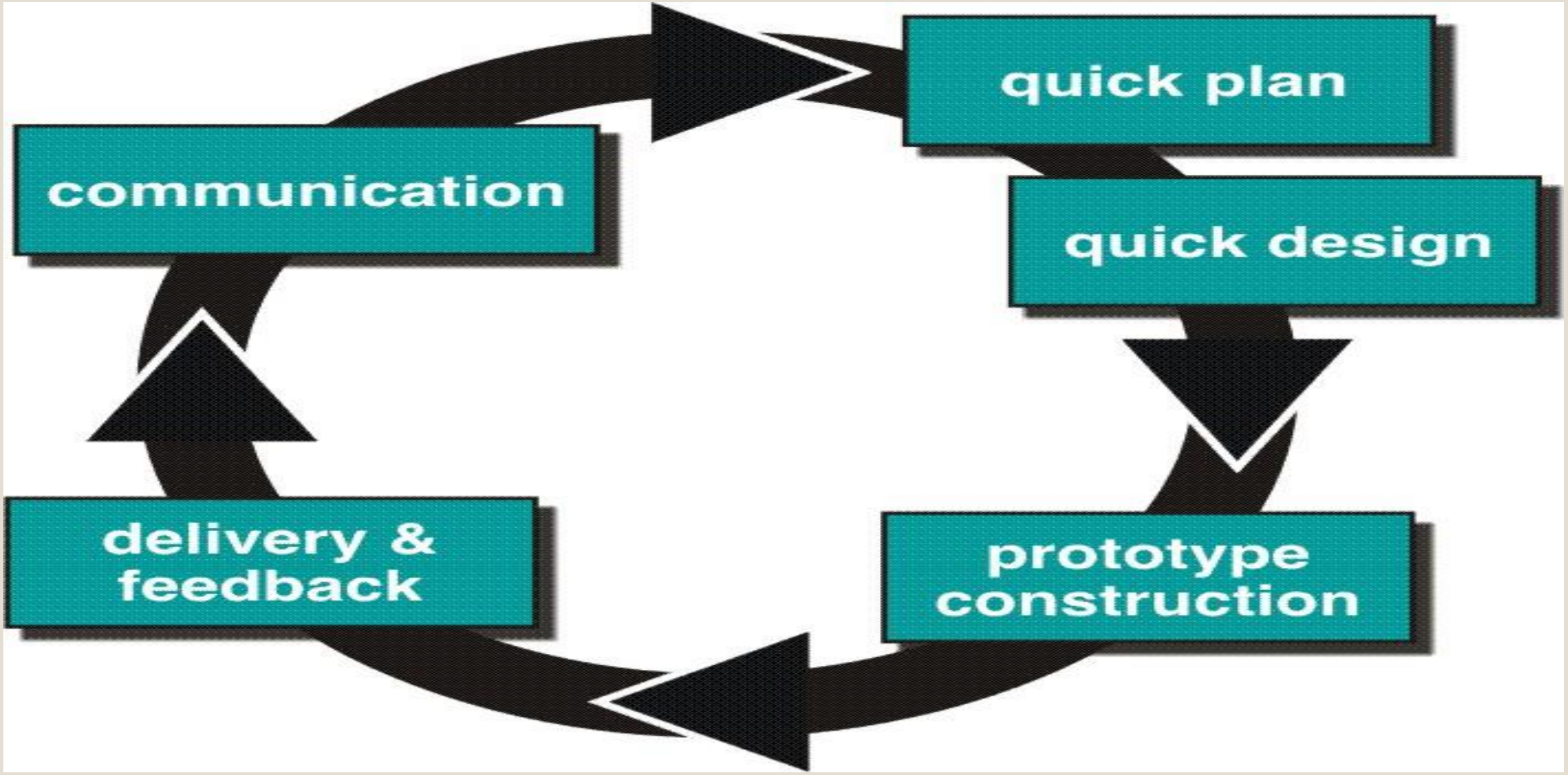
Evolutionary Process Model: Prototyping Model

- Starts with communication
- Requirement gathering is done
- Software developer use this model when requirements are fuzzy
- Quick design is done leads to prototype
- Prototype is refined until customer is satisfied
- Programs are built and thrown on it and we rebuild the system to improve quality

Drawbacks:

- When developer is unsure of efficiency algorithm
- When developer has some compromises like in appropriate programming languages, os used, algorithms used

Evolutionary Process Models : Prototyping



Prototyping Best approach when:

- customer Objectives are general but does not have details like input, processing, or output requirement.
- Developer may be unsure of the efficiency of an algorithm, O.S., or the form that human machine interaction should take.
- when requirement are fuzzy.
- Quick design leads to prototype construction.
- Prototype is deployed and evaluated by the customer/user.
- Feedback from customer/end user will refine requirement and that is how iteration occurs during prototype to satisfy the needs of the customer.

Prototyping

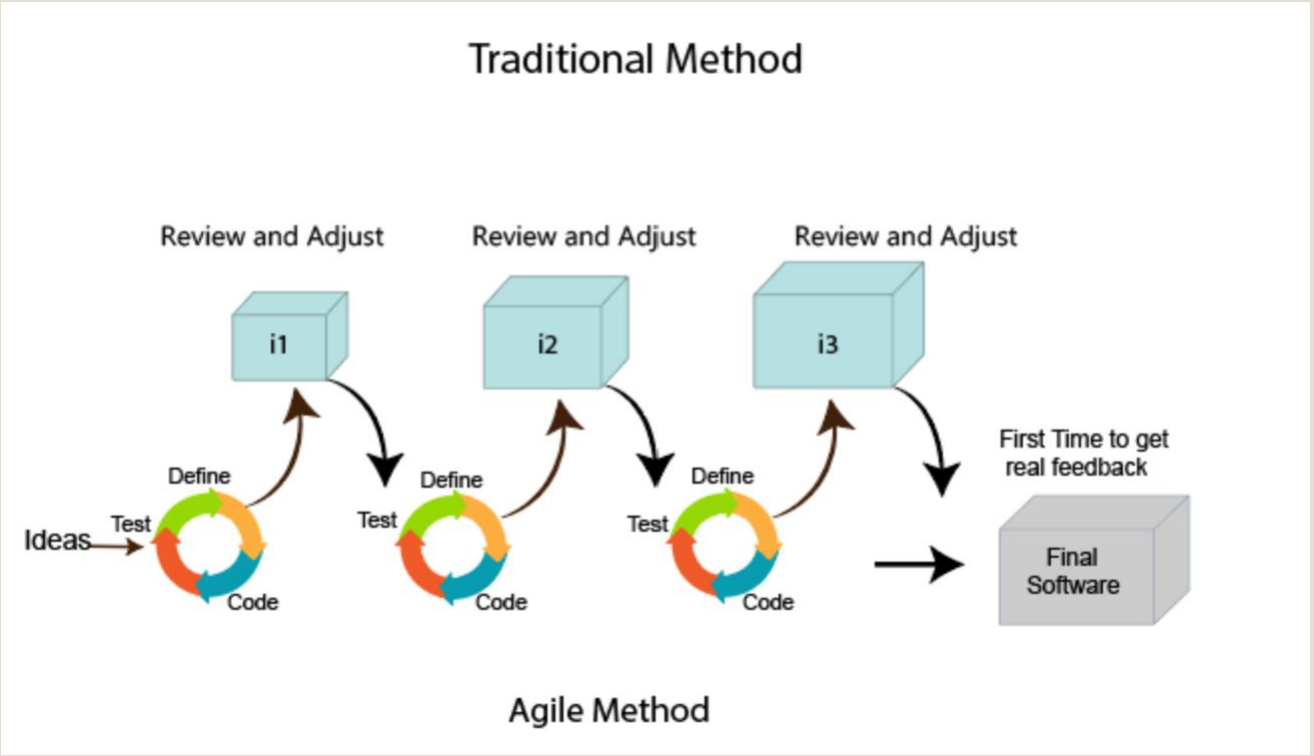
- Prototype can be serve as “the first system”.
- Both customers and developers like the prototyping paradigm.
  - Customer/End user gets a feel for the actual system
  - Developer get to build something immediately.

Limitations Of SDLC Process Models

- Difficulty To Adapt The Changes
- Less Flexibility
- Long Software Development Cycles.
- To overcome these challenges, the Agile method was introduced.
- The Scrum model is also one of the Agile frameworks

What is Agile Methodology?

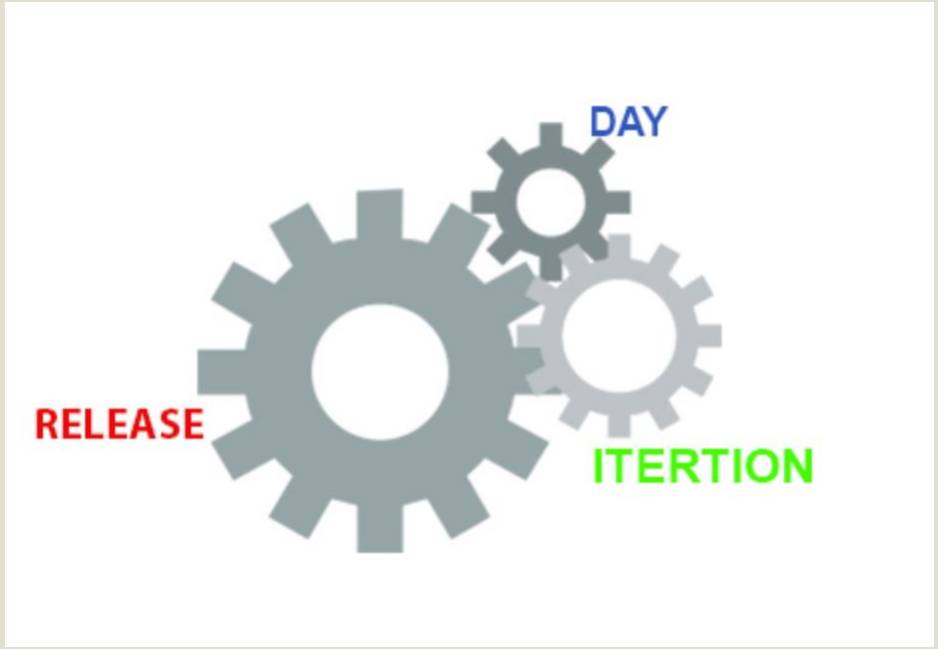
- An agile methodology is an iterative approach to software development.
- Each iteration of agile methodology takes a short time interval of 1 to 4 weeks.



Roles in Agile

There are two different roles in a Agile methodology.

- Scrum Master
- Product Owner.



Scrum Master

- The Scrum Master is a team leader
- It follow agile practices
- They enable the close co-operation between all the roles and functions.
- They remove all the blocks which occur.
- They safeguard the team from any disturbances.
- They work with the organization to track the progress and processes of the company.
- They includes
  - Planned meetings
  - Daily stand-ups
  - Demo
  - Review
  - Retrospective meetings, and
  - Facilitate team meetings and decision-making process.

Product Owner

- The Product Owner is one who runs the product from a business perspective

- He defines the requirements and prioritizes their values.
- He sets the release date and contents.
- He takes an active role in iteration and releasing planning meetings.
- He ensures that the team is working on the most valued requirement.
- He represents the voice of the customer.
- He accepts the user stories that meet the definition of done and defined acceptance criteria.

Cross-functional team

❖Every agile team contains self-sufficient team with 5 to 9 team members.

❖ The average experience of each member ranges from 6 to 10 years.

The agile team contains

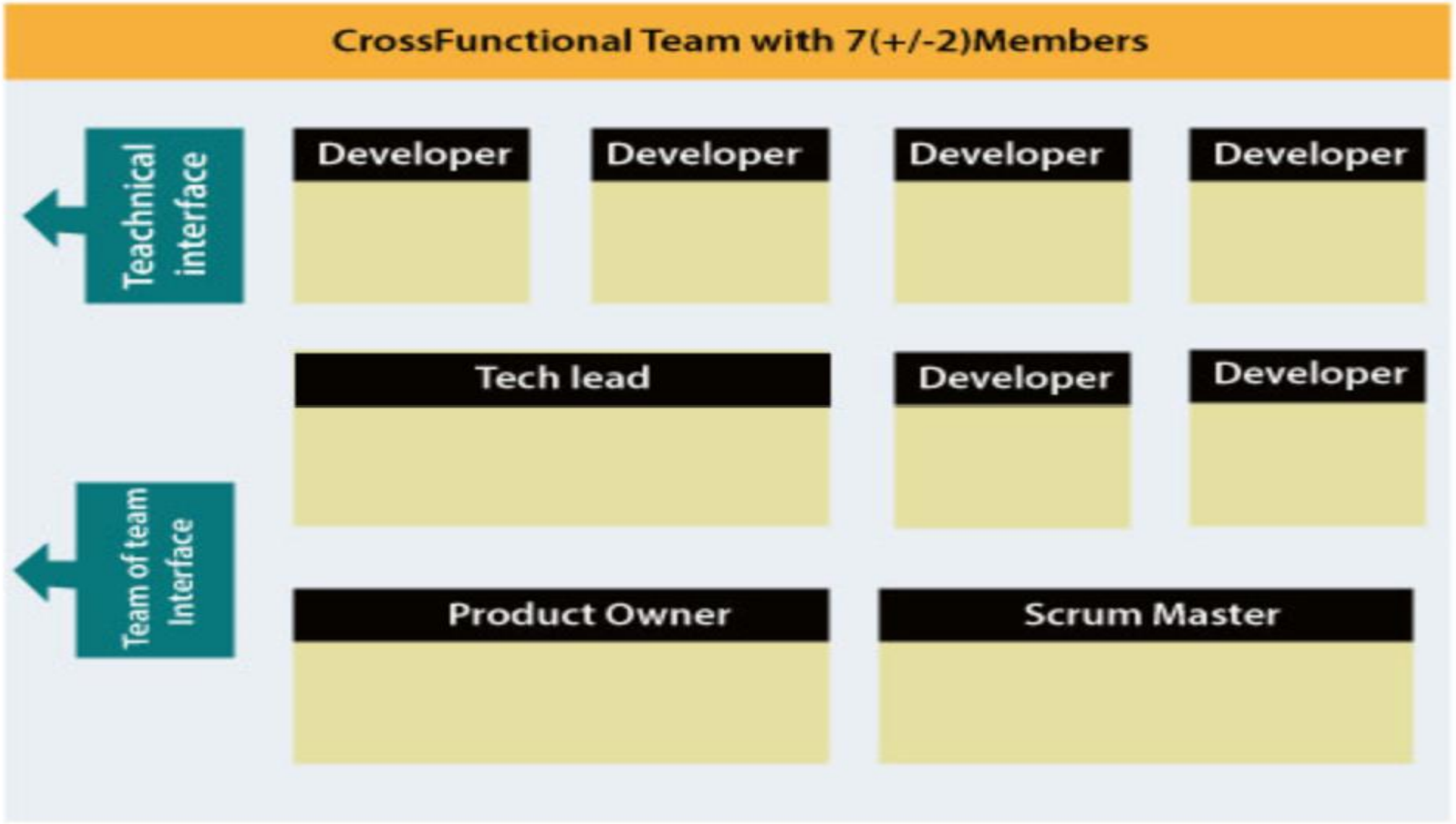
❖3 to 4 developers

❖ 1 tester

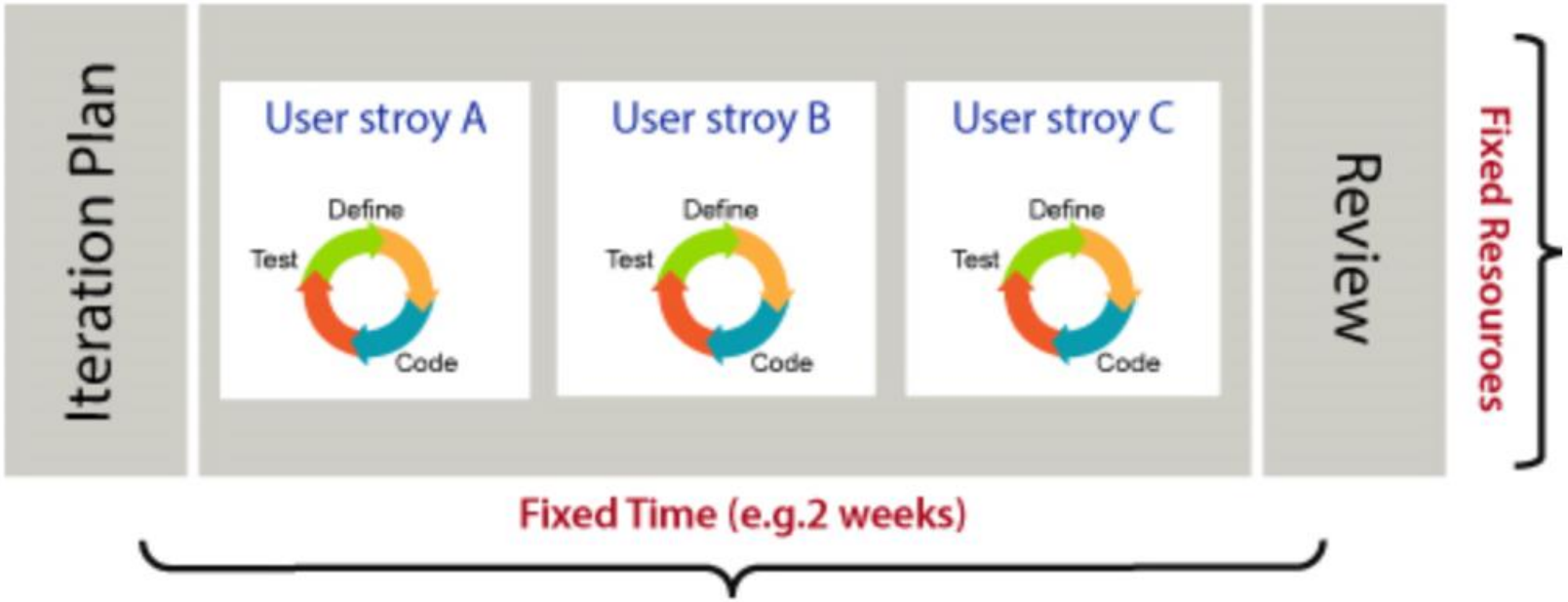
❖ 1 technical lead

❖ 1 scrum master

❖1 product owner

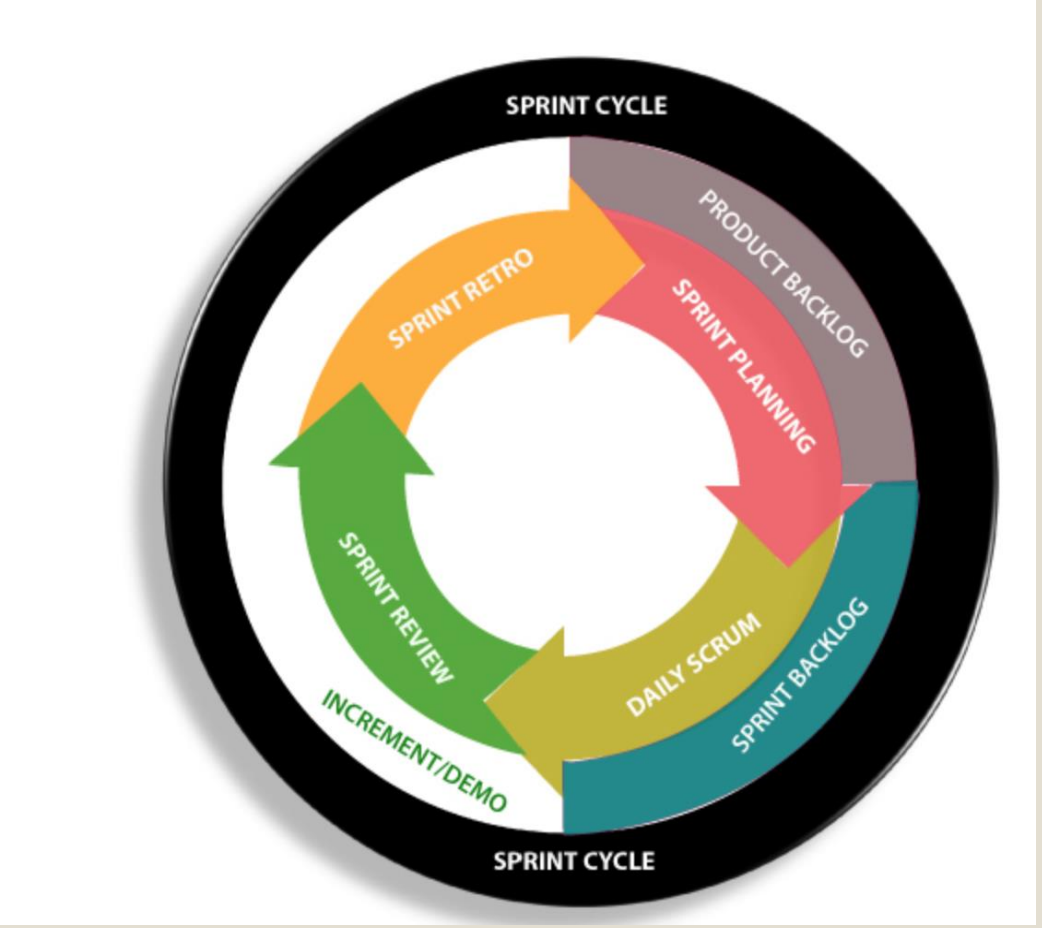

CrossFunctional Team with 7(+/-2)Members
Teachnical interface
Developer
Developer
Developer
Developer
Tech lead
Developer
Developer
Team of team Interface
Product Owner
Scrum Master

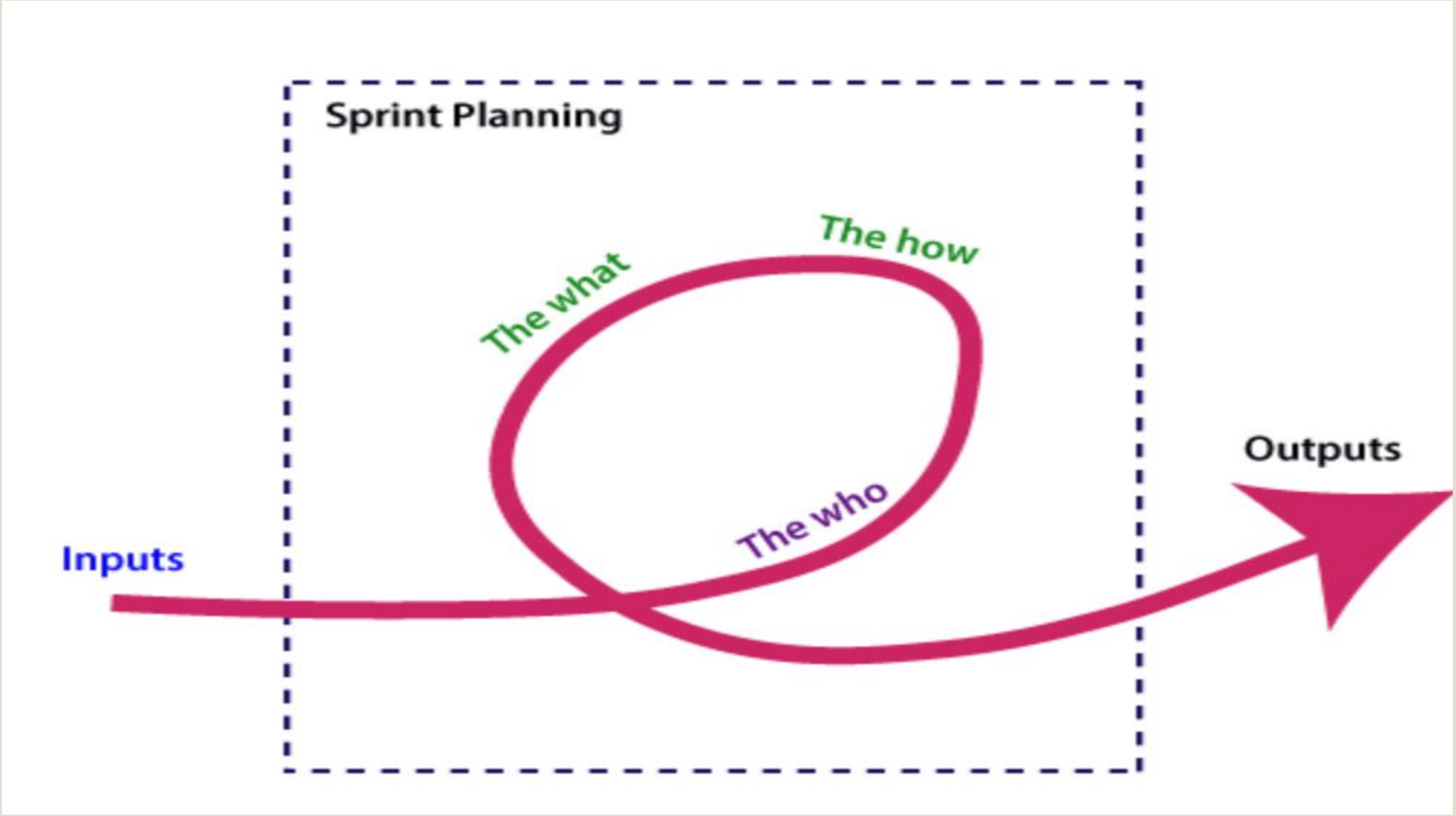
Each user requirement is a planned based and their backlog prioritization and size. The team decides, how much scope they have and how many hours available with each team to perform their planed task.



What are sprints?

- With scrum, a product is built in a series of repetition called **sprints**.
- It breaks down big complex projects into bite-size pieces.
- It makes projects more manageable




Sprint Planning
The how
The what
The who
Inputs
Outputs

What is Scrum Model?

- Scrum is an Agile framework
- product is built in a series of repetition called sprints.
- It manages complex projects
- It is applicable to various fields.
- It was originally formalized in the early 1990's
- It has gained widespread adoption in various industries.

Key components of the Scrum Development Model

1. Roles
2. Artifacts
3. Events
4. Rules

Key components of the Scrum Development Model : Roles

Product Owner: Represents the stakeholders

He is responsible for defining and prioritizing the product backlog.

Scrum Master: Facilitates the Scrum process

ensures that the team adheres to its practices.

Development Team: responsible for delivering the product increment.

Key components of the Scrum Development Model : Artifacts

- **Product Backlog:** A prioritized list of features, enhancements, and bug fixes that need to be addressed in the product.
- **Backlog**: A subset of the product backlog selected for a specific sprint, containing tasks the team commits to completing.
- **Product Increment:** The sum of all the completed product backlog items at the end of a sprint.

Key components of the Scrum Development Model : Events

Sprint: A time-boxed iteration **(usually 2-4 weeks)** during which a potentially shippable product increment is created.

Sprint Planning: A meeting at the beginning of each sprint where the team plans the work to be done.

Daily Scrum (Stand-up): A **short daily meeting** where team members discuss progress, plan for the day, and **identify and address** impediments.

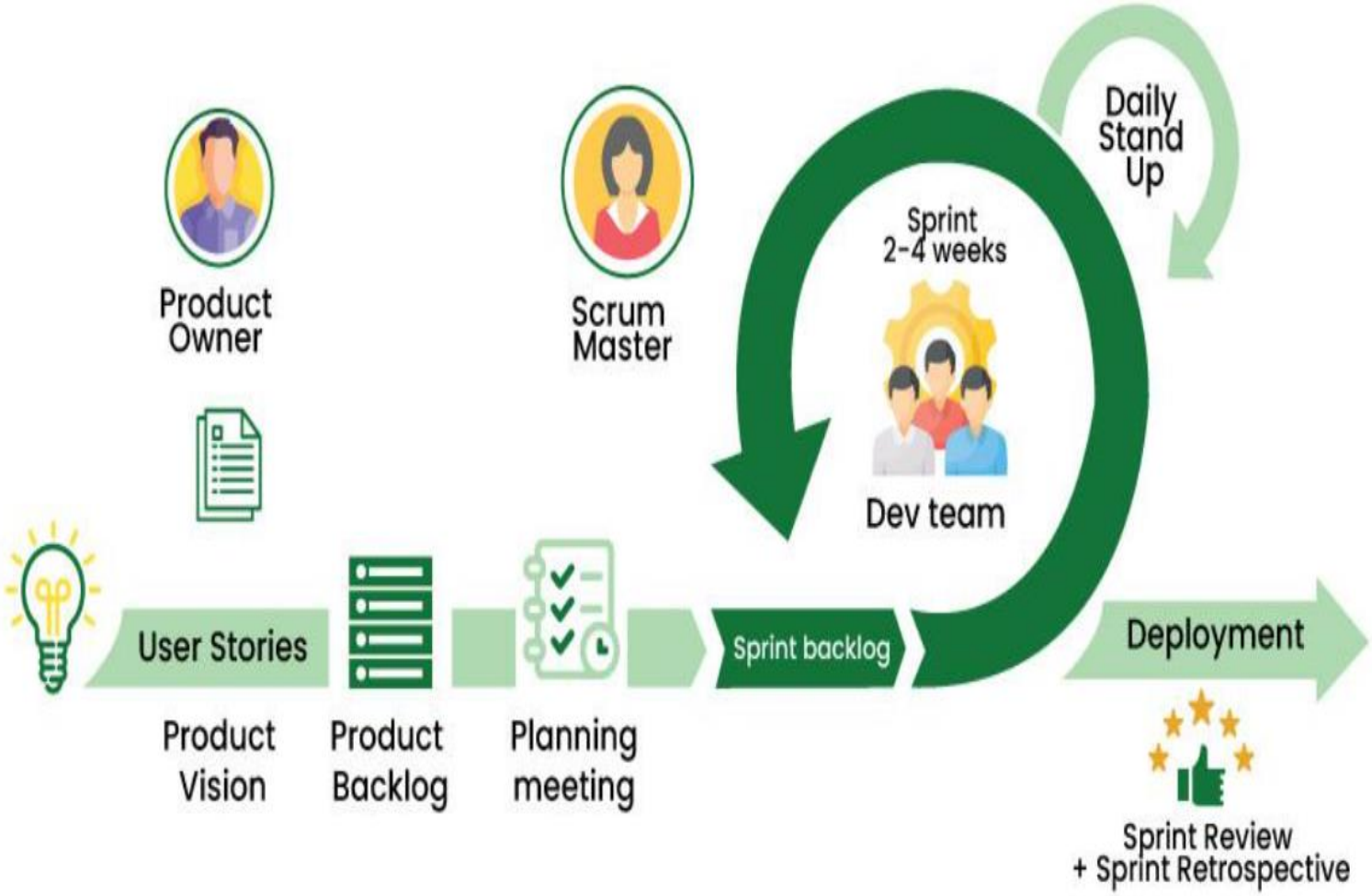
Sprint Review: A meeting at the **end of each sprint to review** the completed work and gather **feedback.**

Sprint Retrospective: A meeting at the end of each sprint for the team to reflect on their processes and **identify improvements.**

Key components of the Scrum Development Model : Rules

- Scrum emphasizes transparency, inspection, and adaptation.
- The product is built incrementally in fixed-length iterations (sprints).
- Changes are only made between sprints unless there is a compelling reason to make a change during a sprint.

How Scrum Model Works?



Scrum model steps

Product Backlog

Sprint Planning

Sprint Meeting

Sprint Review

Sprint retrospective

Repeat

Scrum model steps

Product Backlog:

It is a prioritized list of features, user stories, enhancements, and bug fixes that need to be addressed in the product.

It is managed by the Product Owner

Planning Sprint:

It is a key event at the beginning of each sprint.

During this meeting, the Scrum Team, including the Product Owner, Scrum Master, and Development Team, collaboratively selects items from the Product Backlog to work on during the upcoming sprint.

The team defines the Sprint Goal

creates the Sprint Backlog, detailing the tasks required to complete the selected items.

Scrum model steps

Sprint Meeting: often referred to as the Daily Scrum or Daily Stand-up

It is a brief daily meeting where team members provide updates on their progress

discuss what they plan to work on next, and highlight any impediments.

The goal is to synchronize the team's activities and ensure everyone is on the same page.

Scrum model steps

Sprint Review: It is held at the end of each sprint.

The Scrum Team, stakeholders, and the Product Owner come together to review the completed work.

The Development Team demonstrates the product increment, and stakeholders provide feedback.

This session informs future planning and adjustments to the Product Backlog.

Scrum model steps

Sprint retrospective: It occurs after the Sprint Review

The team discusses what went well, what could be improved, and any action items for enhancing their processes.

The focus is on continuous improvement.

Scrum model steps

- **Repeat:** the cycle repeats with a new Sprint Planning meeting, followed by another sprint of development, daily stand-ups, Sprint Review, and Sprint Retrospective.
- This iterative process continues throughout the project, allowing the team to adapt to changing requirements, continuously improve, and deliver increments of the product at the end of each sprint.

Benefits of Scrum Model in SDLC

Flexibility and Adaptability

Customer Satisfaction

Early and Predictable Delivery

Improved Collaboration

Increased Transparency

Reduced Time to Market

Continuous Improvement

Increased Product Quality

Risk Management

Empowered and Motivated Teams

Cost Control

Scalability

Agile methodology	Waterfall model
It follows the incremental approach.	It is a sequential design process.
It divides the project development lifecycle into a sprint.	The software development process is divided into distinct phases.
Agile methodology is a flexible methodology.	The Waterfall is a structured software development methodology.
Agile is the collection of many different projects.	It is completed as one single project.
The test plan is reviewed after each sprint	Test plan is reviewed after complete development.
Testing team can take part in the requirements change phase without problems.	It is difficult for the test to initiate any change in needs.

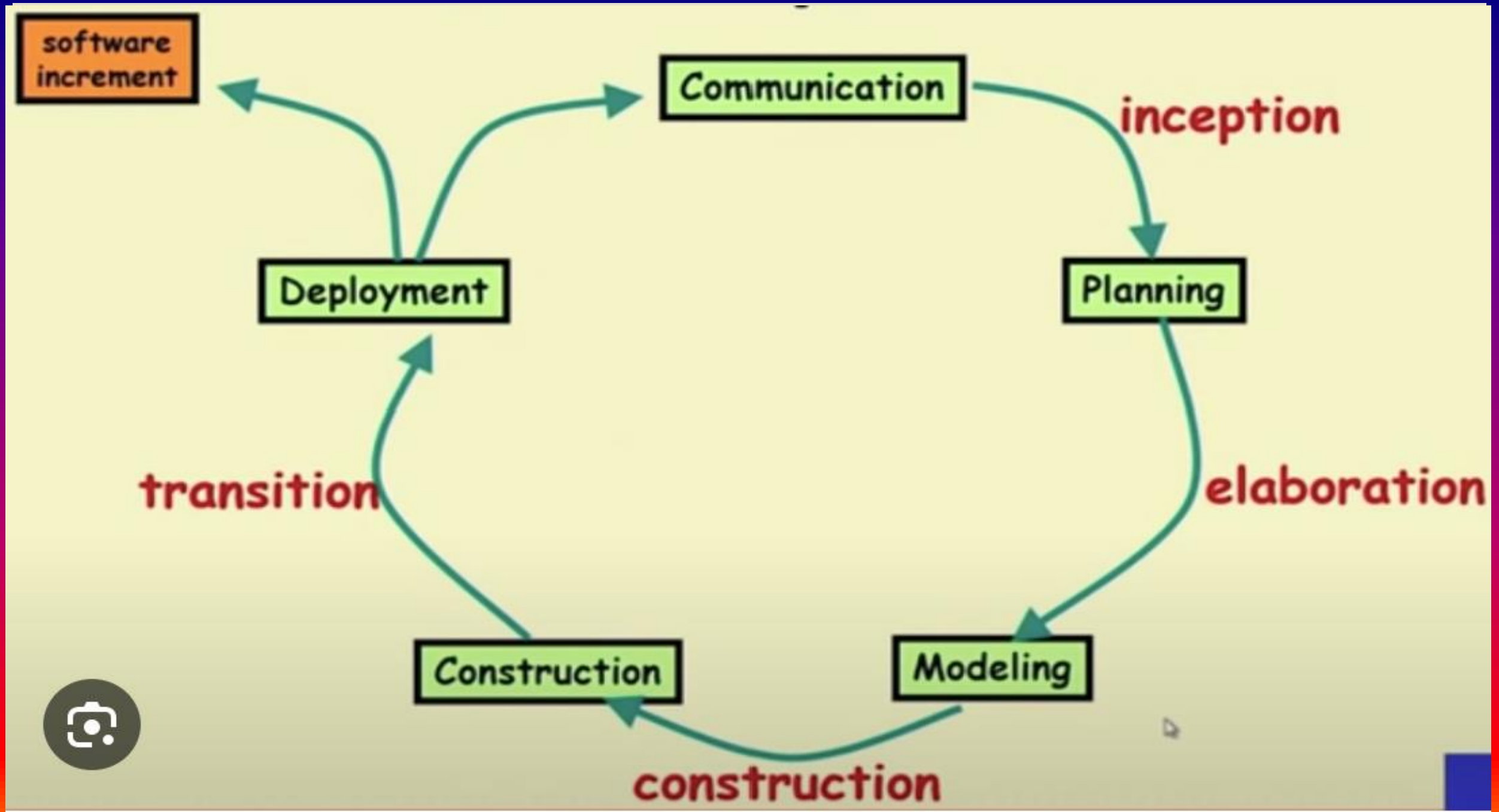
Unified Process Model

It is rational unified process model developed by grady booch

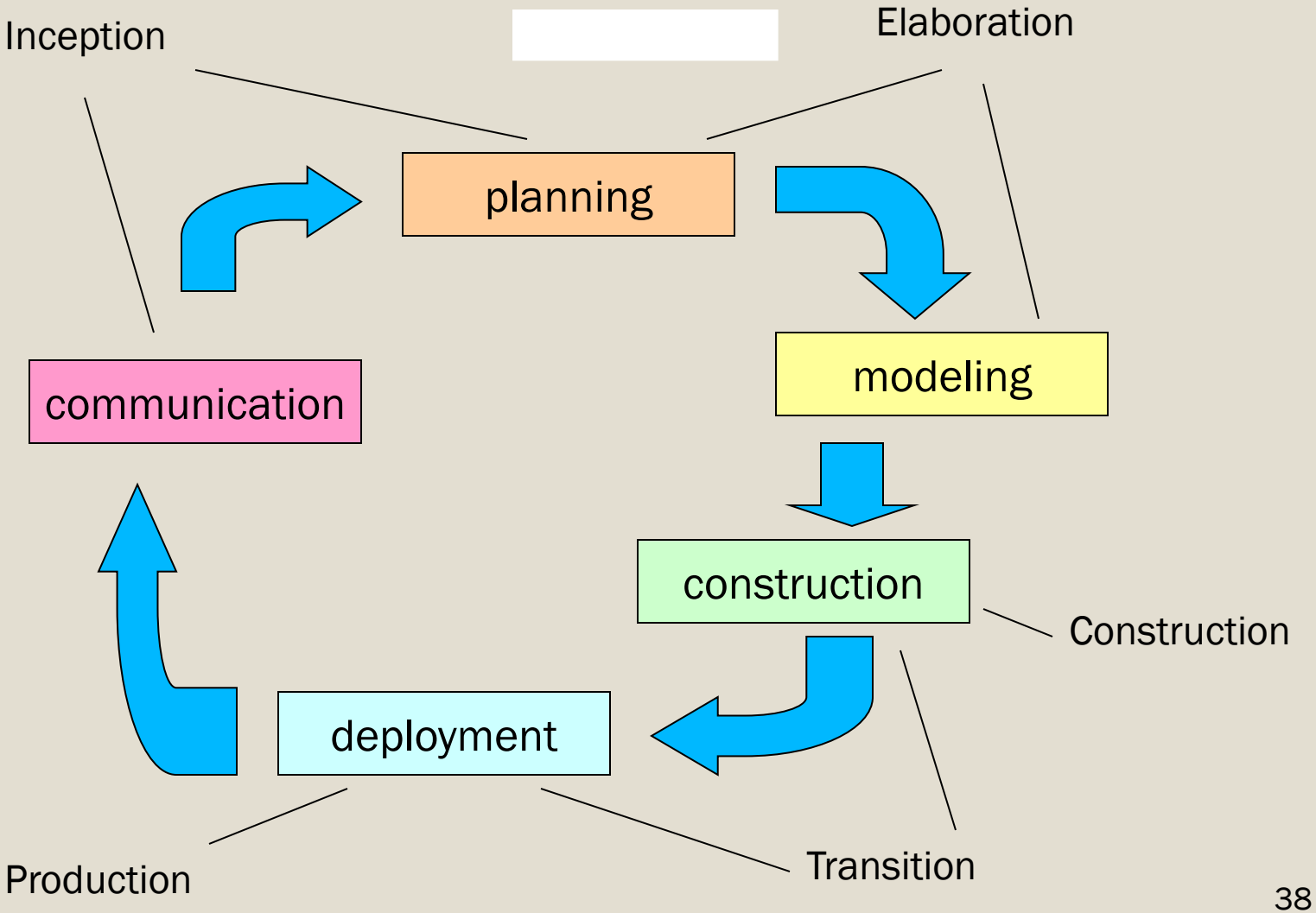
It follows iterative incremental approach

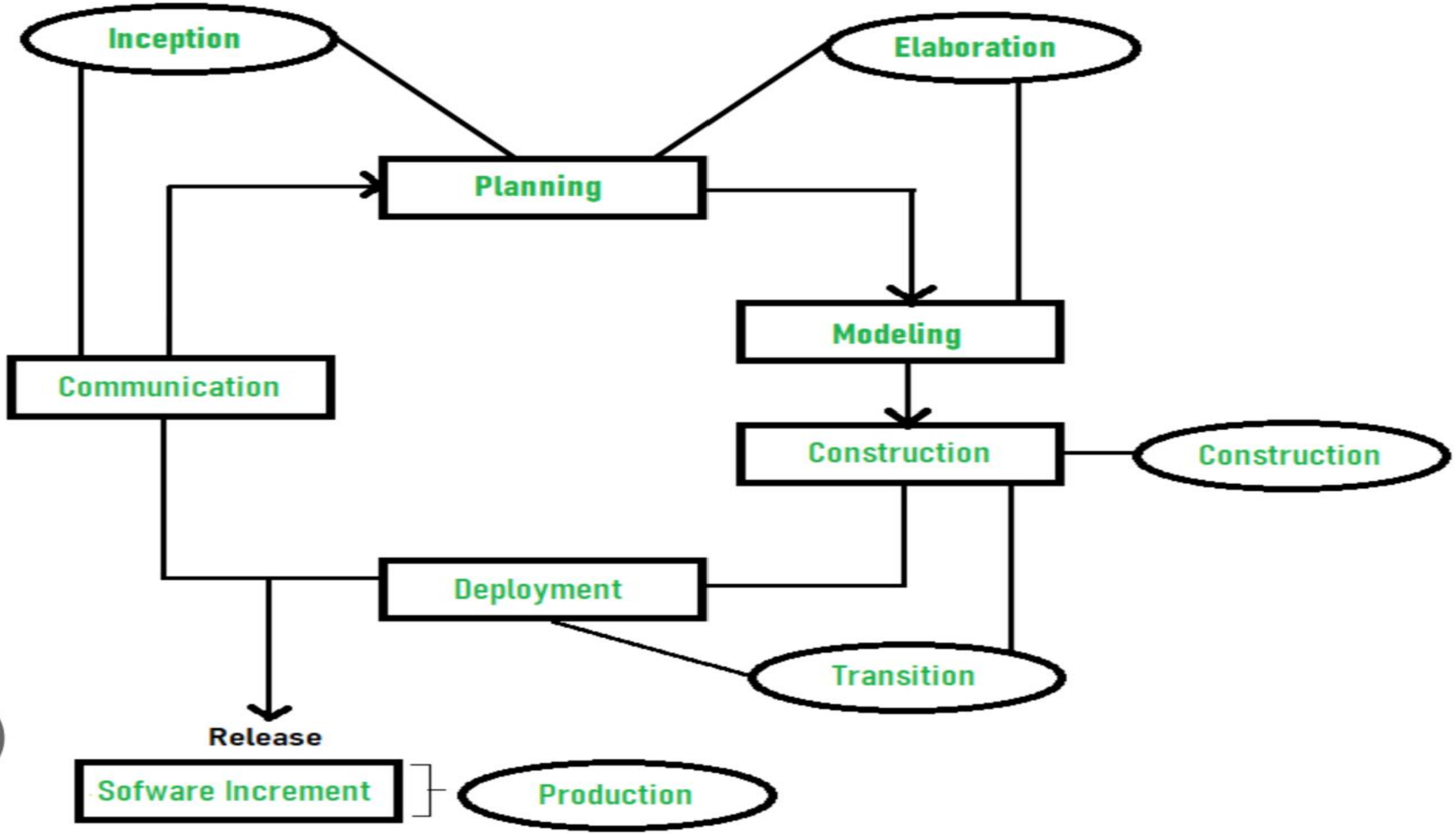
It has 5 phases

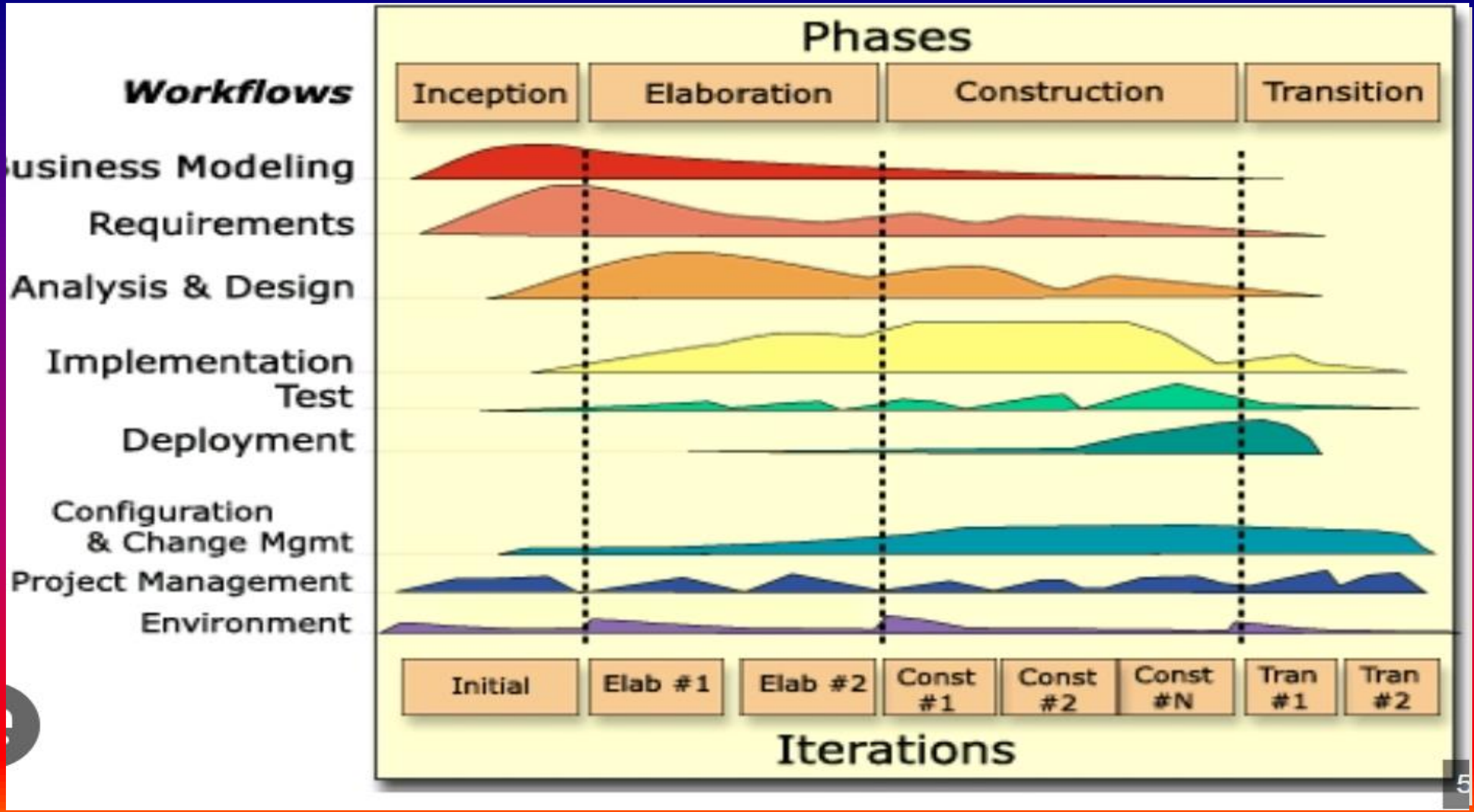
- Inception
- Elaboration
- Construction
- Transition
- production


software increment
Communication
inception
Deployment
Planning
transition
elaboration
Construction
Modeling
construction

Phases of the Unified Process




Inception
Elaboration
Planning
Modeling
Communication
Construction
Construction
Deployment
Transition
Release
Sofware Increment
Production


Phases
Workflows
Inception
Elaboration
Construction
Transition
usiness Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration
& Change Mgmt
Project Management
Environment
Initial
Elab #1
Elab #2
Const
#1
Const
#2
Const
#N
Tran
#1
Tran
#2
Iterations

Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
phases and iterations.
Business model,
if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
including non-functional
Analysis model
Software architecture
Description.
Executable architectural
prototype.
Preliminary design model
Revised risk list
Project plan including
iteration plan
adapted workflows
milestones
technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
increment
Test plan and procedure
Test cases
Support documentation
user manuals
installation manuals
description of current
increment

Transition phase

Delivered software increment
Beta test reports
General user feedback

Requirement Engineering Process

How to conduct a feasibility study (7 Steps)

1. Conduct a **Preliminary Analysis**
2. Prepare a Projected **Income Statement**
3. Conduct a **Market Survey**, or Perform Market Research
4. Plan Business Organization and Operations- start-up costs, fixed investments and operation costs.
5. Prepare an Opening Day Balance Sheet- estimate of the assets and liabilities
6. Review and **Analyze All Data-**
7. Make a Go/**No-Go Decision**

FOCUS OF Feasibility study

Questionnaire to people in organization is prepared.

- How will proposed system help
- What are integration problems
- What are current process models

FOCUS OF Feasibility study

- If system contributes to organizational objectives
- If system can be engineered to current technology
- If system is with in given budget
- If system can be integrated to other useful system
- Feasibility study should be done with the help project managers, software engineers, customers, technical experts.
- Feasibility study should be completed with in 2 to 3 weeks
- Finally feasibility report should be submitted

What is a Feasibility Study?

- project is Profitable for the company to undertake.
- This study is mandatorily
- Determine the viability of a project from an **economical, legal, and technical perspective.**
- whether a project is achievable or worth the investment.
- Description of the project
- resource allocation
- accounting statements
- financial data
- legal requirements
- tax obligations.

Types of Feasibility Studies

1. **Technical Feasibility**
2. **Economic Feasibility**
3. **Legal Feasibility**
4. **Operational Feasibility**
5. **Scheduling Feasibility**

Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

Technical Feasibility

- focuses on the technical resources available to the organization.
- evaluation of the hardware, software
- technical requirements of the proposed system.
- analyzes technical skills
- capabilities of technical team
- existing technology can be used or not
- maintenance and up-gradation is easy or not

Economic Feasibility

- cost/ benefits analysis of the project
- economical benefits associated with a project before financial resources are allocated.
- determine the positive economic benefits to the organization that the proposed project will provide.
- cost for final development like hardware and software resource
- design and development cost
- operational cost
- project will be beneficial in terms of finance for organization or not.

Legal Feasibility

- study to know if proposed project conform legal and ethical requirements.
- project is analyzed in legality point of view.
- project certificate, license, copyright etc.
- proposed project conflicts with legal requirements like
- zoning laws
- data protection acts or social media laws
- A feasibility study might reveal the organization's ideal location isn't zoned for that type of business.

Operational Feasibility

- determine whether—and how well
- organization's needs can be met by completing the project.
- how a project plan satisfies the requirements identified
- organization's goals can be satisfied by completing the project.
- how much easy product will be to operate and maintenance after deployment.
- usability of product
- Determining suggested solution is acceptable or not etc.

Scheduling Feasibility

- It is the most important for project success
- project will fail if not completed on time.
- organization estimates how much time the project will take to complete.
- timelines/deadlines is analyzed
- project may fail if it can’t be completed on time.

TECHNICAL FEASIBILITY

This assessment focuses on the organization's technical resources. It helps organizations determine if resources meet capacity and if the technical team can convert ideas into working systems

ECONOMIC FEASIBILITY

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability and benefits associated with a project

LEGAL FEASIBILITY

This assessment investigates whether any aspect of the proposed project conflicts with legal requirements like zoning laws, data protection acts, or social media laws

OPERATIONAL FEASIBILITY

This assessment involves undertaking a study to analyze and determine whether—and how well—the organization's needs can be met by completing the project

SCHEDULING FEASIBILITY

This assessment is the most important for project success. In scheduling feasibility, an organization estimates how much time the project will take to complete

Feasibility study

- It is a situation to check whether system generated is useful/not.
- A feasibility study decides whether or not the proposed system is worthwhile
- Once system is feasible then only REP begins.

Various activities involved in REP:

- Discovering requirements
- Converting requirements into some form
- Checking whether requirements are as per customer needs /not.

Requirements Introduction

Requirement is descriptions of

- what the system should do
- services that it provides
- constraints on its operation.
- The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE).

Requirements Engineering

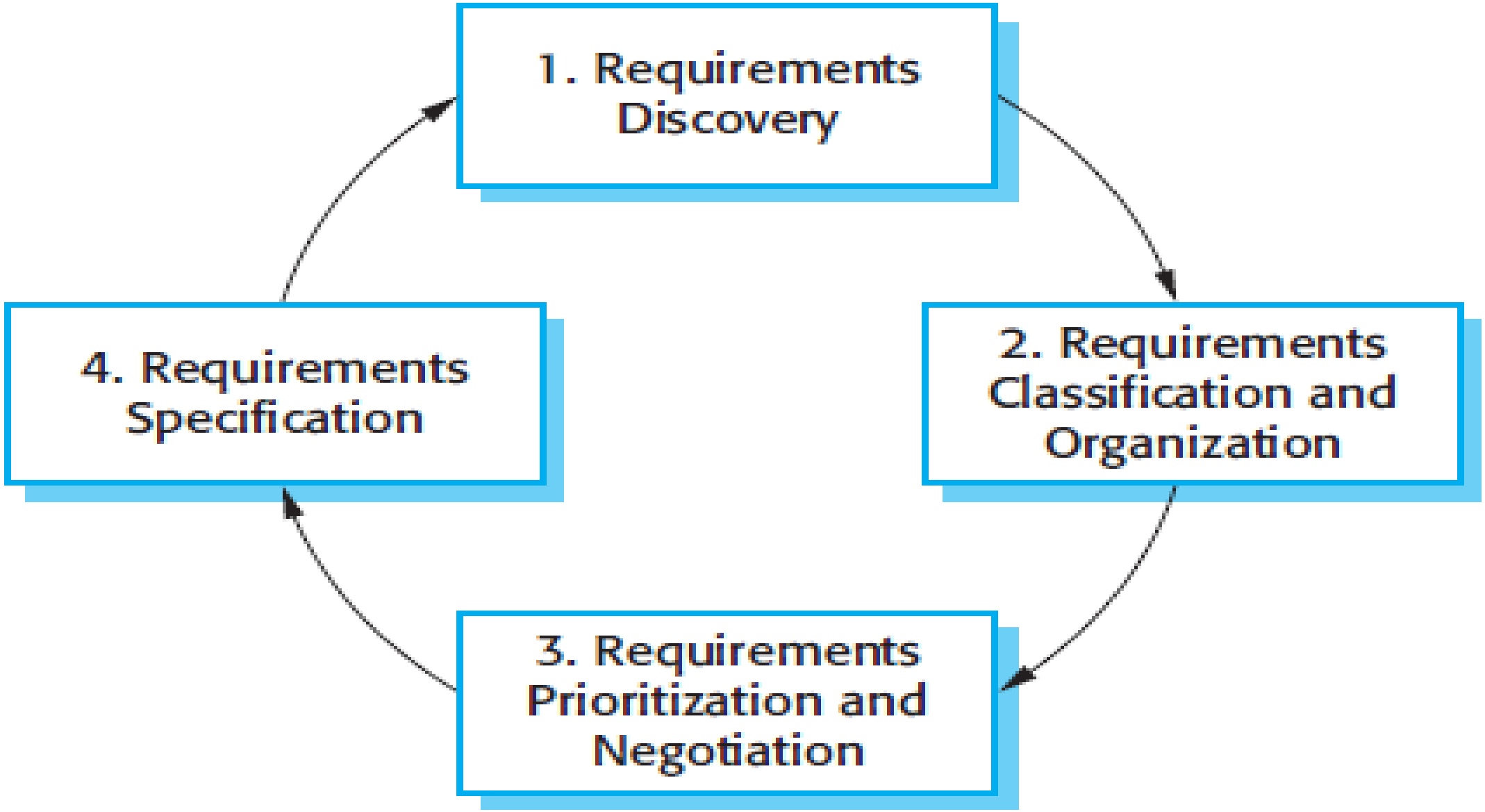
- Requirement: A function, constraint that the system must provide to fill the needs of the system's intended user(s)
- REP is a process that is used to discover, analysis & validate system requirements.
- The process vary widely.
- REP goal is to maintain & create SRS.

Requirement Engineering process

It is a process in which various activities like

- Discovery
- Analysis
- Validation

- It begins with feasibility study
- It ends with requirement validation.
- Finally requirements document is prepared


1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

Stakeholder Participation

Manage Requirements

Elicit Requirements

Analyze Requirements

Document Requirements

Validate Requirements

Requirements engineering processes

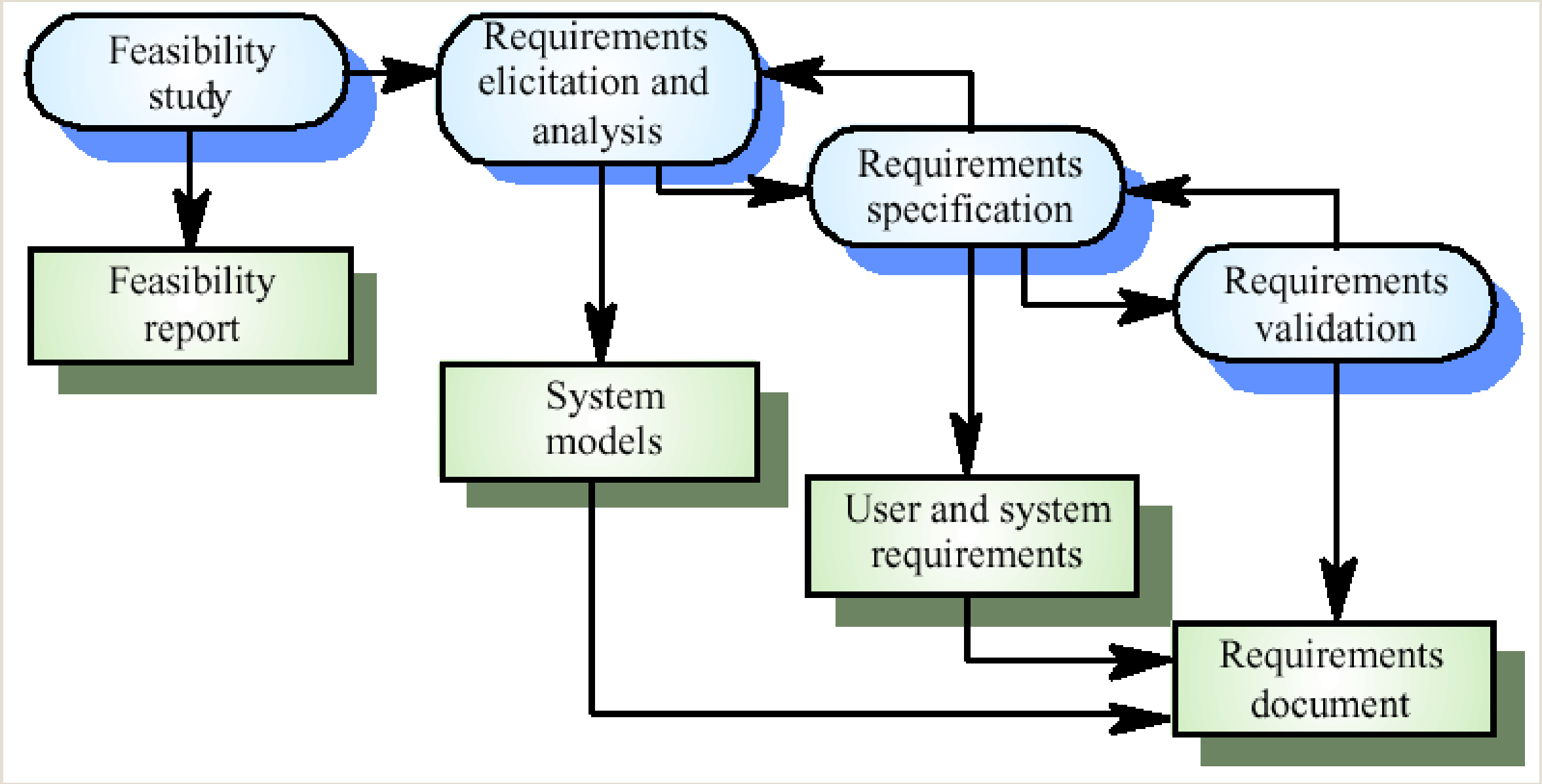
There are a number of generic activities common to all processes

- Requirements elicitation
- Requirements analysis
- Requirements validation
- Requirements management.

Requirements Engineering Tasks

- Inception —Establish a basic understanding of the problem and the nature of the solution.
- Elicitation —Draw out the requirements from stakeholders.
- Elaboration (Highly structured)—Create an analysis model that represents information, functional, and behavioral aspects of the requirements.
- Negotiation—Agree on a deliverable system that is realistic for developers and customers.
- Specification—Describe the requirements formally or informally.
- Validation —Review the requirement specification for errors, ambiguities, omissions, and conflicts.
- Requirements management —Manage changing requirements.

Requirements engineering process



REQUIREMENT Elicitation and analysis/ requirements discovery.

- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.
- They are called *stakeholders.*

Problems of requirements analysis

- After performing feasibility study REA is done
- Requirement elicitation means discovery of all requirements
- After identifying of requirements analysis is done

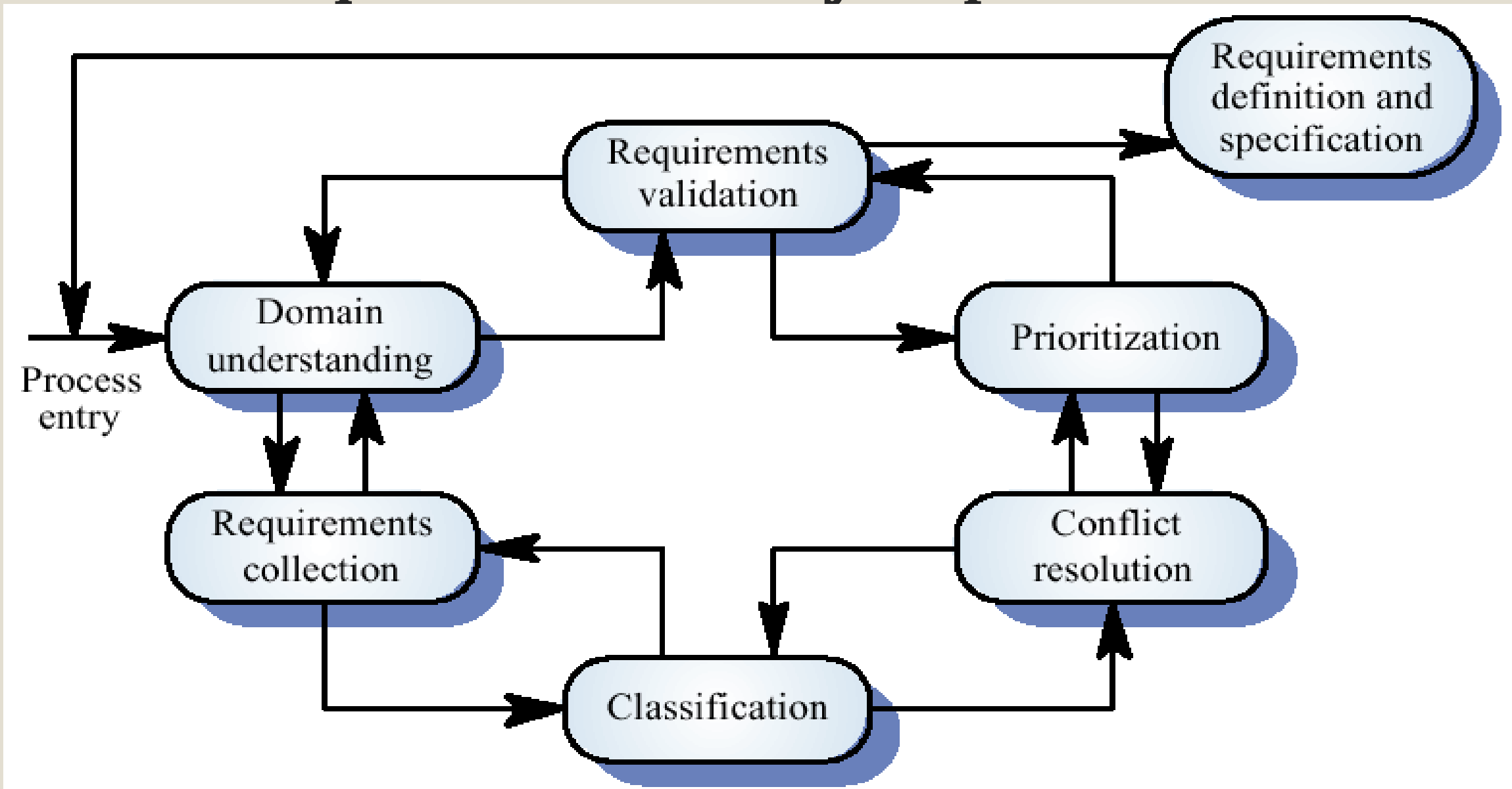
Stake holders

- Stakeholders don’t know what they really want.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.

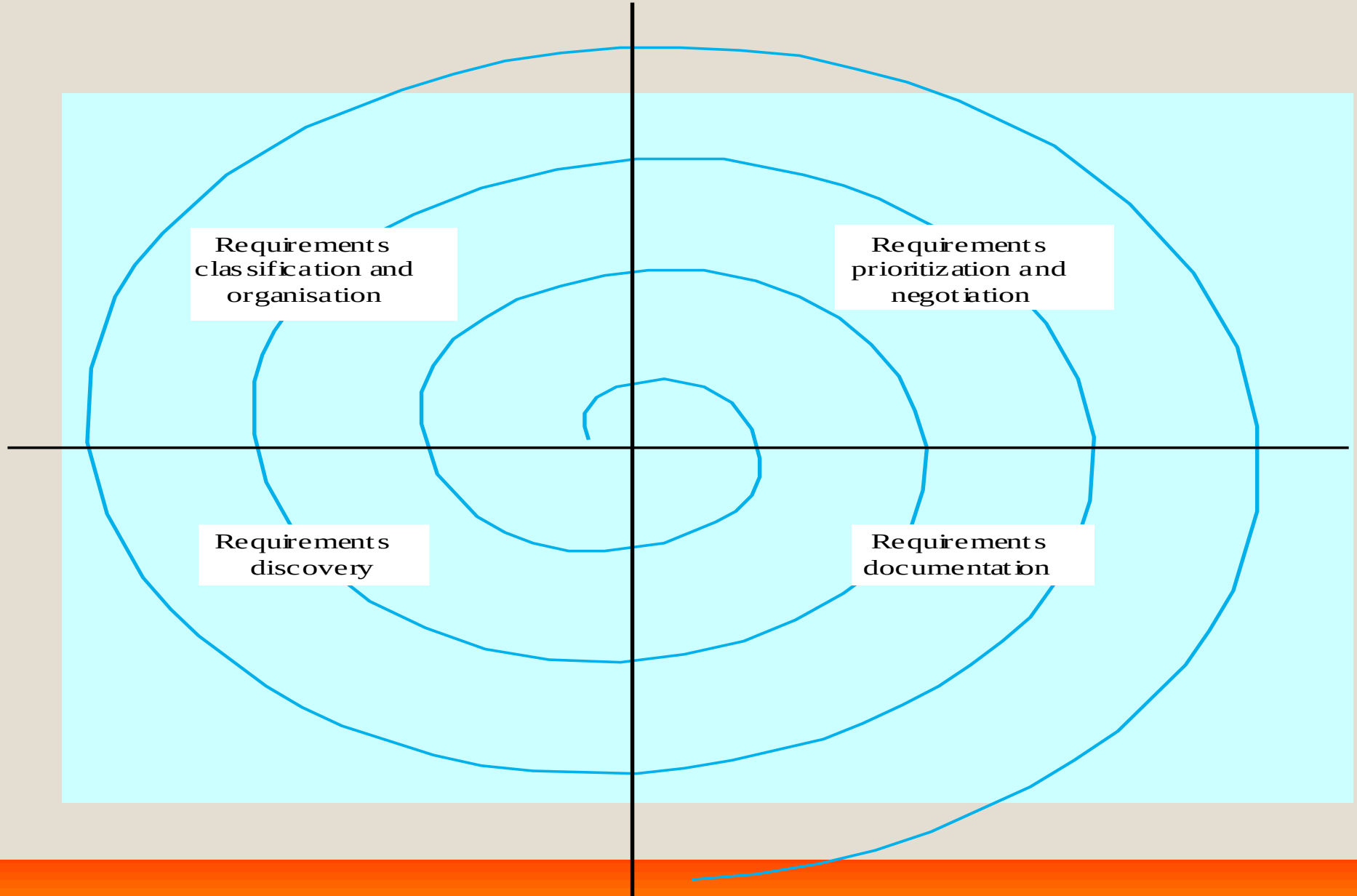
Problems encountered in Understanding Requirements

- **Unrealistic expectations: they** are unable to specify what they want.
- **Differences in requirements:** Different stake holders specify different requirements
- **Economic & business development**: due to change in economic & business environment is dynamic.
- **Political changes: some** times political factors affect heavily on system

The requirements analysis process



The REA PROCESS



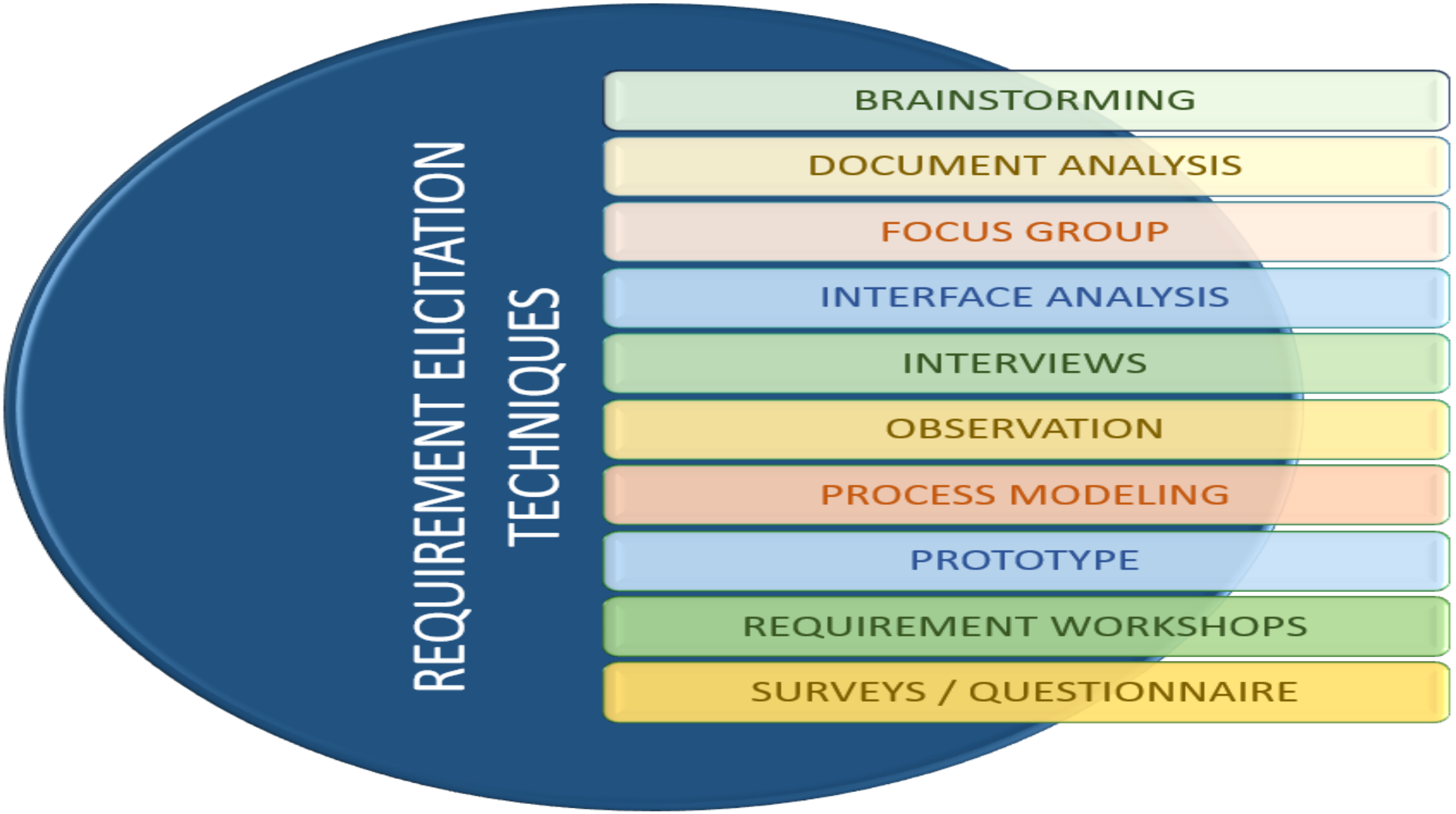
Process activities

- **Requirements discovery**
  - By having effective communication with customers requirements can be identified..
- **Requirements classification and organisation**
  - All unstructured requirements are categorized systematically depending on the nature.
- **Prioritisation and negotiation**
  - Prioritising requirements and resolving requirements conflicts.
- **Requirements documentation**
  - Requirements are documented and input into the next round of the spiral.

Requirements discovery

Methods for conducting requirement discovery

- Conducting interviews
- Observations
- Creating use case scenarios


REQUIREMENT ELICITATION TECHNIQUES
BRAINSTORMING
DOCUMENT ANALYSIS
FOCUS GROUP
INTERFACE ANALYSIS
INTERVIEWS
OBSERVATION
PROCESS MODELING
PROTOTYPE
REQUIREMENT WORKSHOPS
SURVEYS / QUESTIONNAIRE

Example of stakeholders for ATM

- Bank customers
- Representatives of other banks
- Bank managers
- Counter staff
- Database administrators
- Security managers
- Marketing department
- Hardware and software maintenance engineers
- Banking regulators

Interviewing

- This is effective method of requirement gathering
- REAteam communicates to stake holders by asking various questions about the system
- In formal or informal interviewing.

There are two types of interview

- **Closed interviews** where a pre-defined set of questions are answered.
- **Open interviews** where there is no pre-defined agenda and a range of issues are explored with stakeholders.

Characteristics of effective interview:

- Interview should be conducted in free environment
- It should be with open minded approach
- Requirement engineer should listen to stake holders with patience
- We should ask questions & gather requirements

Requirements checking

- Validity. Does the system provide the functions which best support the customer's needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements validation techniques

- **Requirements reviews**
  - 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.

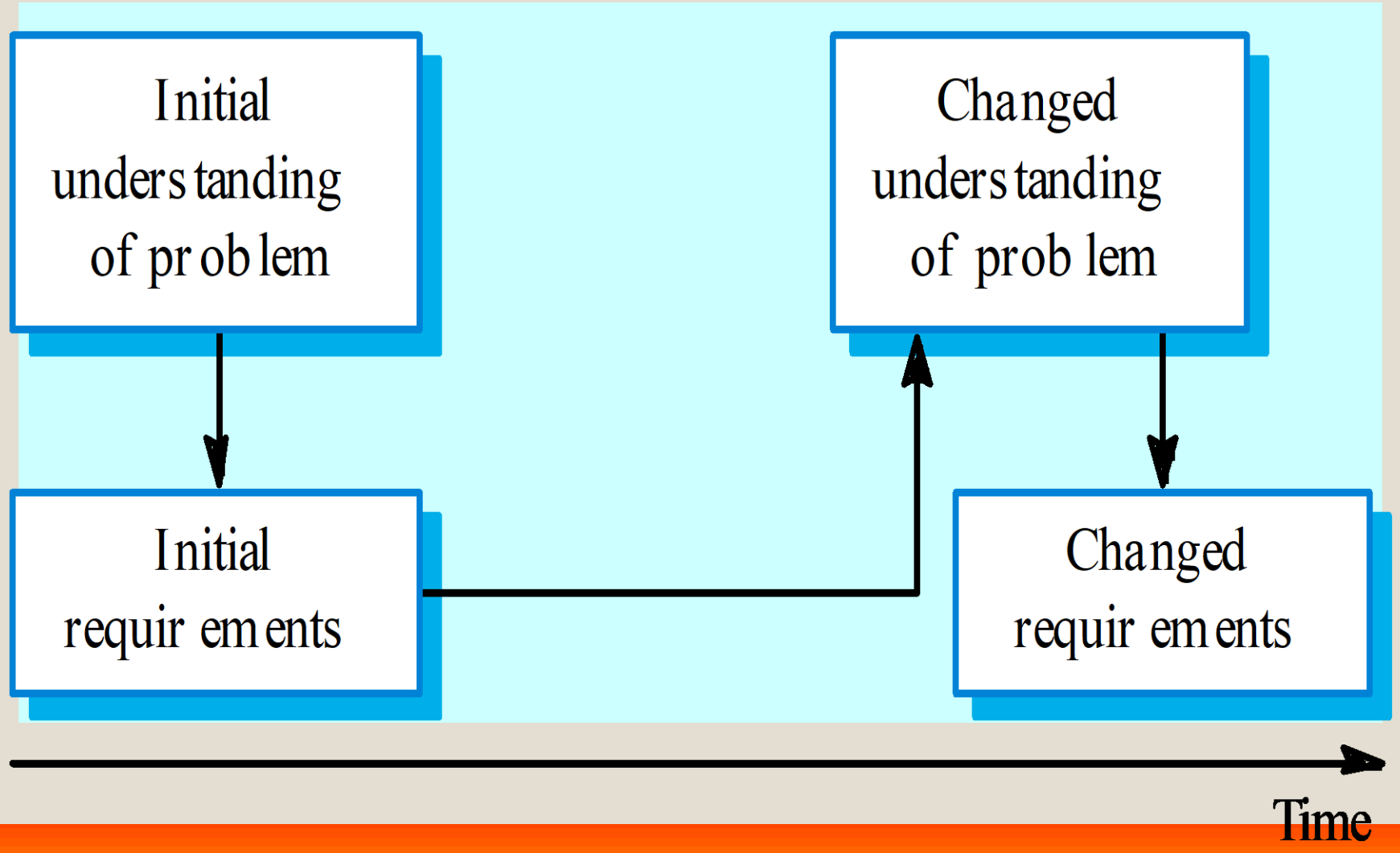
Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Requirements management

- It is the process of managing changing requirements during the requirements engineering process and system development.
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - Different viewpoints have different requirements and these are often contradictory.

Requirements evolution



Traceability

Traceability is concerned with the relationships between requirements, their sources and the system design

- **Source traceability**
- **Requirements traceability**
- **Design traceability**

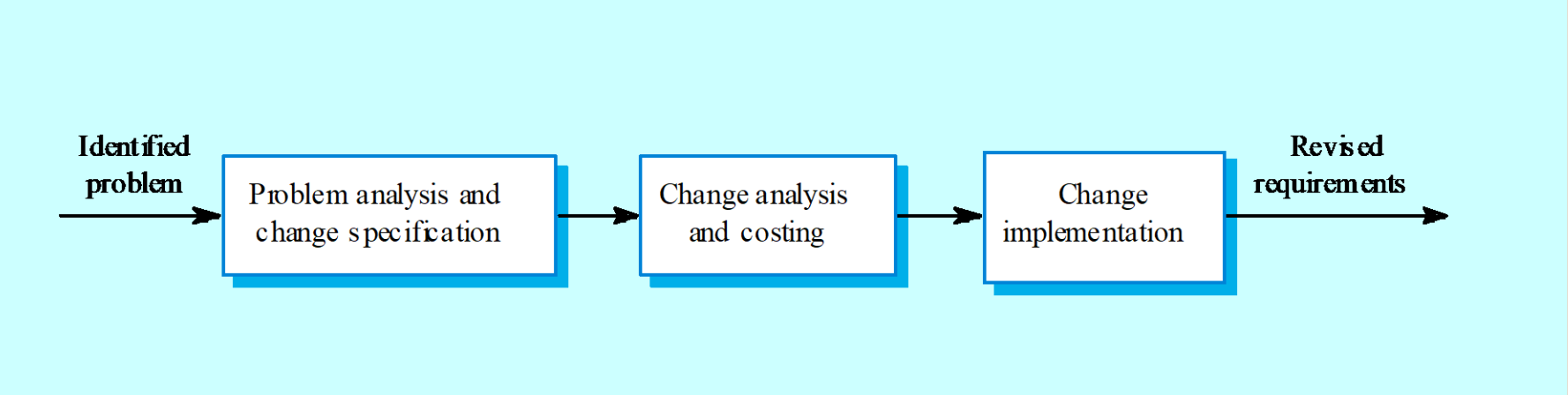
A traceability matrix

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

Requirements change management

- Should apply to all proposed changes to the requirements.
  - Problem analysis. Discuss requirements problem and propose change;
  - Change analysis and costing. Assess effects of change on other requirements;
  - Change implementation. Modify requirements document and other documents to reflect change.

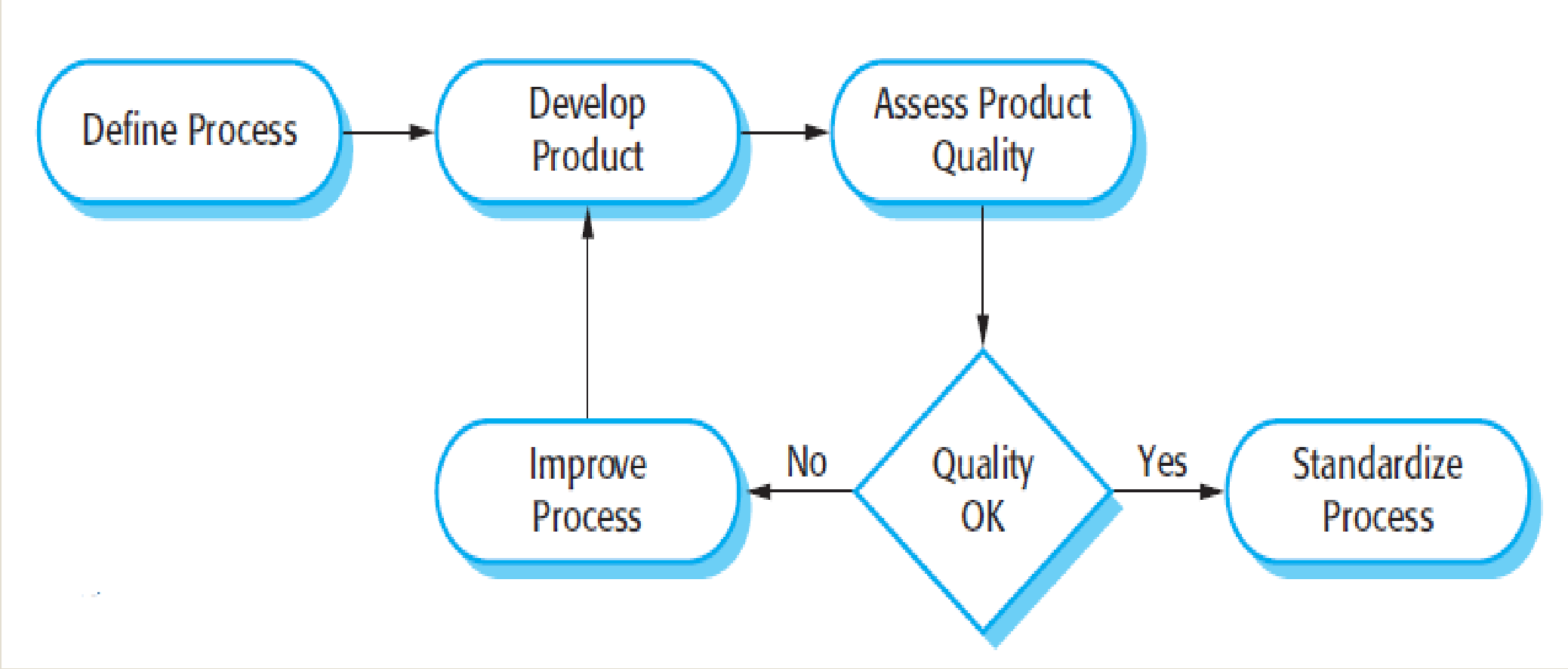
Change management



Software quality

- The terms ‘quality assurance’ & ‘quality control’ are widely used in manufacturing industry.
- **Quality assurance (QA)** is the process of defining standards that lead to high-quality products.
- **Quality control (QC)** is the application of these quality processes to weed out products that are not of the required level of quality.
- The QA team responsible for managing the release testing process.

Process-based quality



Software Requirement Specification

The software requirements document

- SRS is an official statement of what the system developers should implement.
- It include both the user & system requirements
- The user prioritizes requirements for next release
- IEEE standard for requirements documents (IEEE, 1998).
- comprehensive table of contents and document index is needed.

several reasons Why SRS required

- specific architecture to satisfy non-functional requirements may be necessary.
- An external regulator who needs to certify system

User & system requirements

- make a clear separation between these different levels of description.

user requirements

- high-level abstract requirements
- statements, in a natural language
- services the system is expected to provide
- constraints under which it must operate

system requirements

- description of what the system should do.
- contract between buyer and developers.

Example: Mental health care Patient management system (MHC-PMS)

User Requirement Definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System Requirements Specification

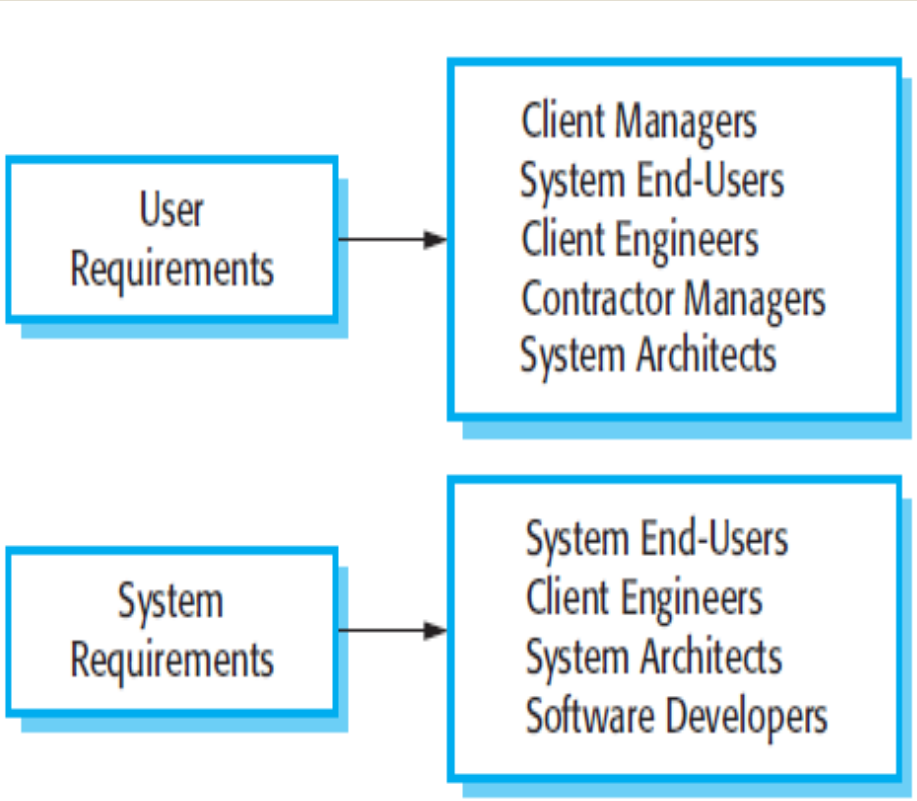
1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.

1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.

1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.

1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.



Functional and non-functional requirements

SRS are often classified as functional requirements or nonfunctional requirements:

Functional requirements

- how the system should react to inputs
- how the system should behave

Non-functional requirements

They include timing constraints

- constraints on the development process
- constraints imposed by standards.

Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

Non-functional requirements

- Constraints on the services or functions offered by the system such as time constraints, constraints on the development process, standards, etc.

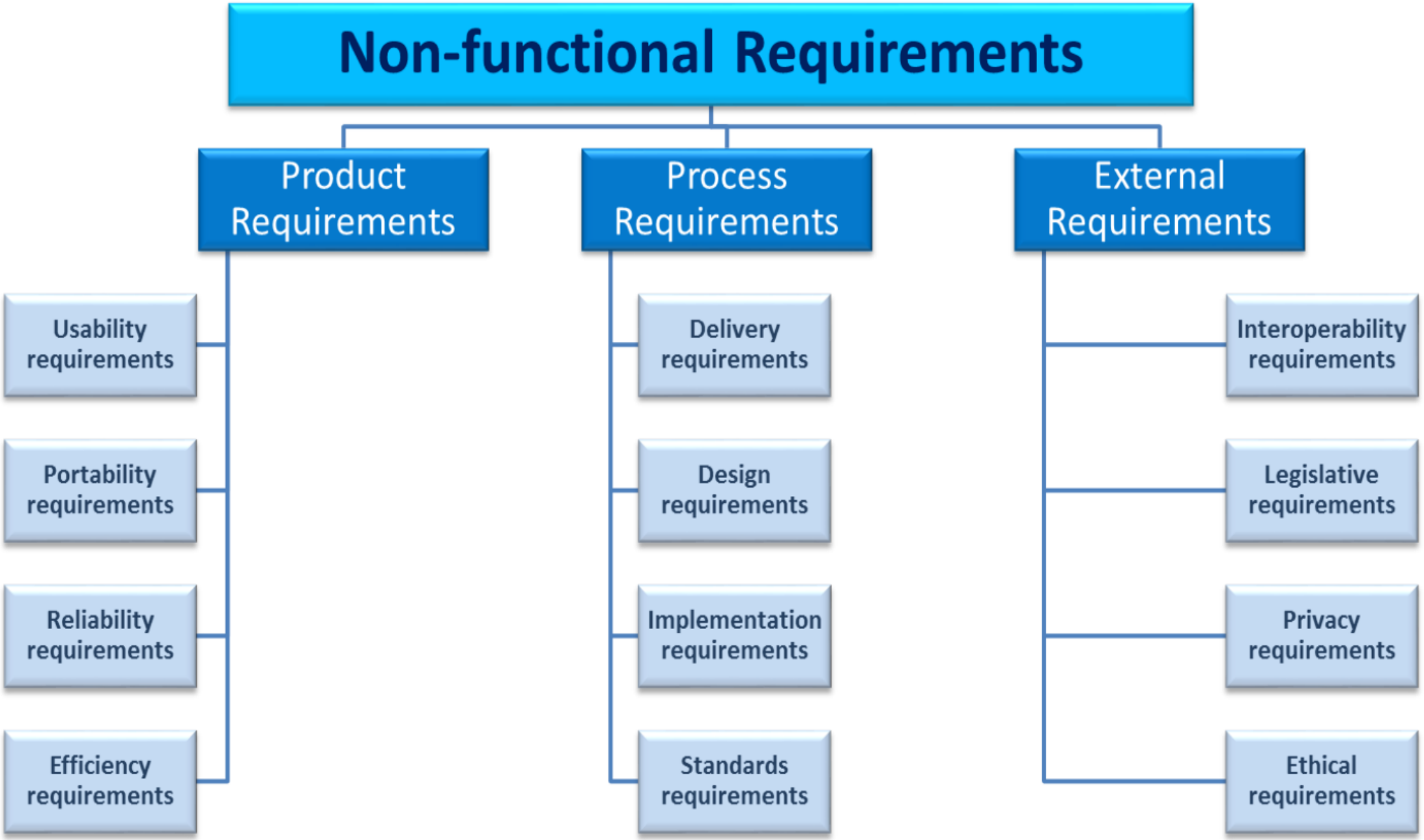

Non-functional Requirements
Product Requirements
Process Requirements
External Requirements
Usability requirements
Portability requirements
Reliability requirements
Efficiency requirements
Delivery requirements
Design requirements
Implementation requirements
Standards requirements
Interoperability requirements
Legislative requirements
Privacy requirements
Ethical requirements

Table 4-3 Example questions for eliciting nonfunctional requirements.

Category	Example questions
Usability	• What is the level of expertise of the user? • What user interface standards are familiar to the user? • What documentation should be provided to the user?
Reliability *(including robustness, safety, and security)*	• How reliable, available, and robust should the system be? • Is restarting the system acceptable in the event of a failure? • How much data can the system loose? • How should the system handle exceptions? • Are there safety requirements of the system? • Are there security requirements of the system?
Performance	• How responsive should the system be? • Are any user tasks time critical? • How many concurrent users should it support? • How large is a typical data store for comparable systems? • What is the worse latency that is acceptable to users?
Supportability *(including maintainability and portability)*	• What are the foreseen extensions to the system? • Who maintains the system? • Are there plans to port the system to different software or hardware environments?
Implementation	• Are there constraints on the hardware platform? • Are constraints imposed by the maintenance team? • Are constraints imposed by the testing team?
Interface	• Should the system interact with any existing systems? • How are data exported/imported into the system? • What standards in use by the client should be supported by the system?
Operation	• Who manages the running system?

Operation	• Who manages the running system?
Packaging	• Who installs the system? • How many installations are foreseen? • Are there time constraints on the installation?
Legal	• How should the system be licensed? • Are any liability issues associated with system failures? • Are any royalties or licensing fees incurred by using specific algorithms or components?

Ways of writing a system requirements specification

Notation	Description
Natural language sentences	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract.

Ways of writing a system requirements specification

- requirements are written in **natural language**
- System requirements **forms, graphical system models, or mathematical system models**
- **Graphical models** show how a state changes / describe a sequence of actions.
- **UML sequence charts and state charts** show the sequence of actions that occur in response to a certain message or event.
- **Formal mathematical specifications** describe the requirements for security-critical systems, but are rarely used in other circumstances.

SRS document Example

- In this document, flight management project is used as an example to explain few points.

https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database

https://krazytech.com/projects/software-requirements-specification-report

https://medium.com/@vincetran_28429/software-requirements-specification-srs-document-fd9ab103b18

What is UML?

UML stands for “Unified Modeling Language”

- It is a industry-standard graphical language .
- UML is a pictorial language used to make software blue prints
- It is used for specifying, visualizing, constructing, and documenting the artifacts of software systems
- UML is different from the other common programming languages
- It uses mostly graphical notations.
- Simplifies the complex process of software design

What is UML?

- It is a general purpose modelling language.
- Define a standard way to visualize the way a system has been designed.
- UML is not a programming language
- it is rather a visual language
- portray the behavior and structure of a system

Why UML for Modeling

- Use graphical notation to communicate more clearly than natural language (imprecise) and code(too detailed).
- Help acquire an overall view of a system.
- Tools can be used to generate code in various languages using UML diagrams
- UML is *not* dependent on any one language or technology.
- *A picture is worth than thousand words*
- UML can be defined as a simple modeling mechanism to model all possible practical systems in today's complex environment.

- The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997.
- International Organization for Standardization (ISO) published UML as an approved standard in 2005

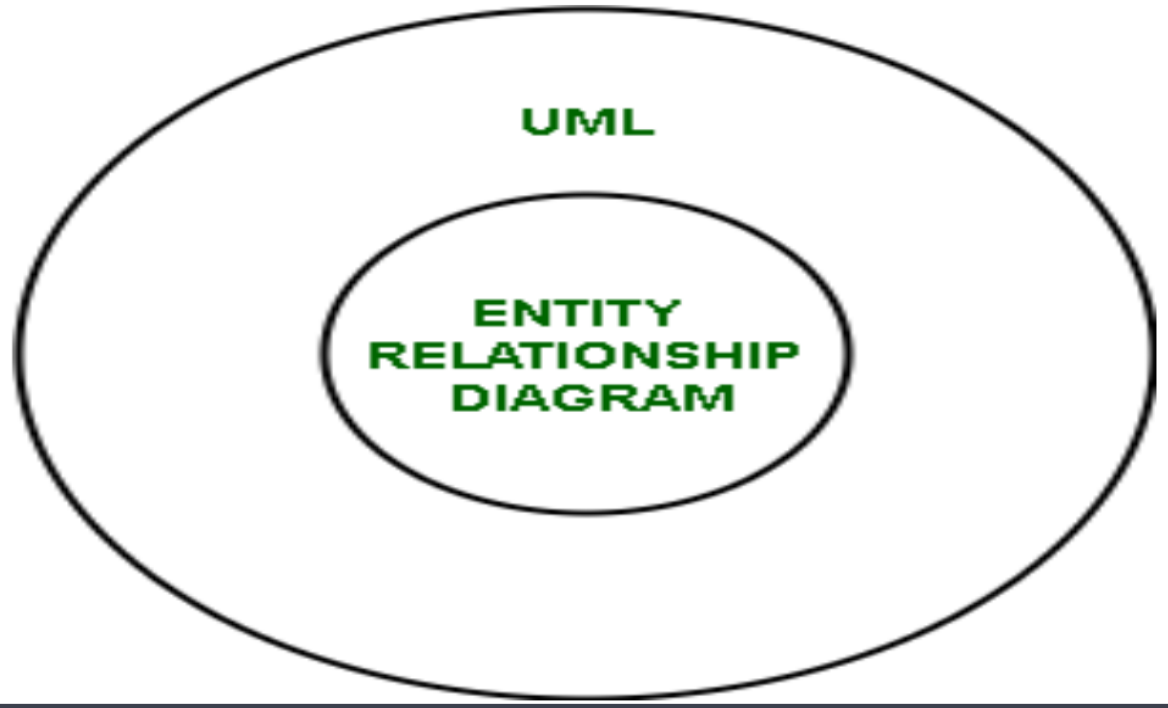
UML

1. Unified Modelling Language (UML) **:**

UML is a modelling language that visually represents a software system.

2. ER Diagram **:**

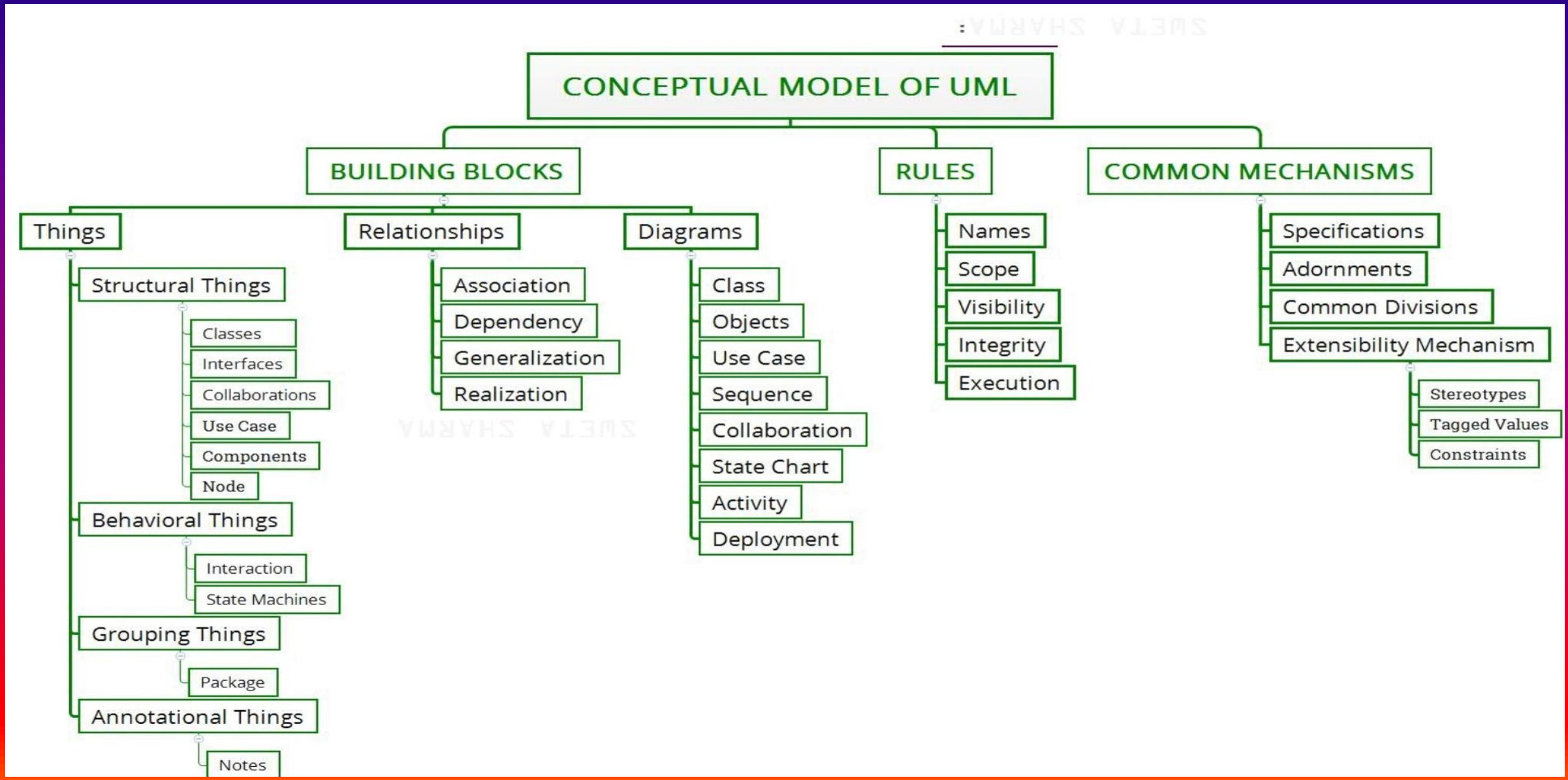
ER diagram is a pictorial representation of the real-world entities and their relationships with each other.



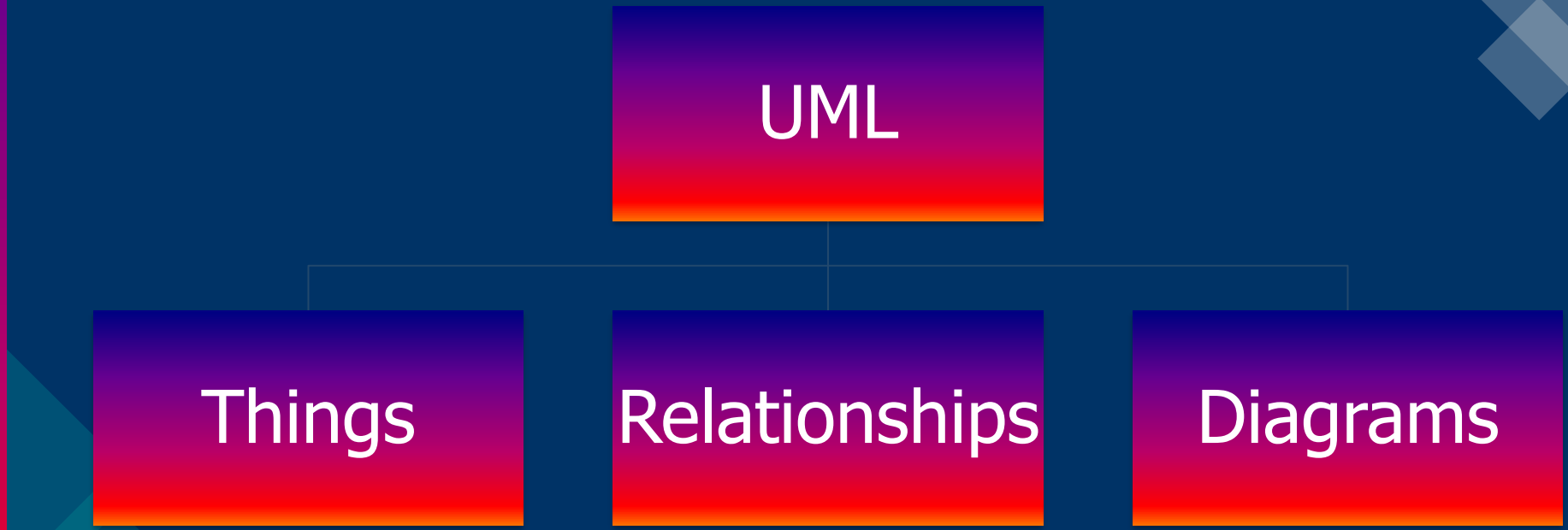
Conceptual Model of UML

- It is a model which is made of concepts and their relationships.
- It is the first step before drawing a UML diagram.
- It helps to understand the entities in the real world and how they interact with each other
- A conceptual model of the language underlines the three major elements:
- • The Building Blocks
- • The Rules
- • Some Common Mechanisms

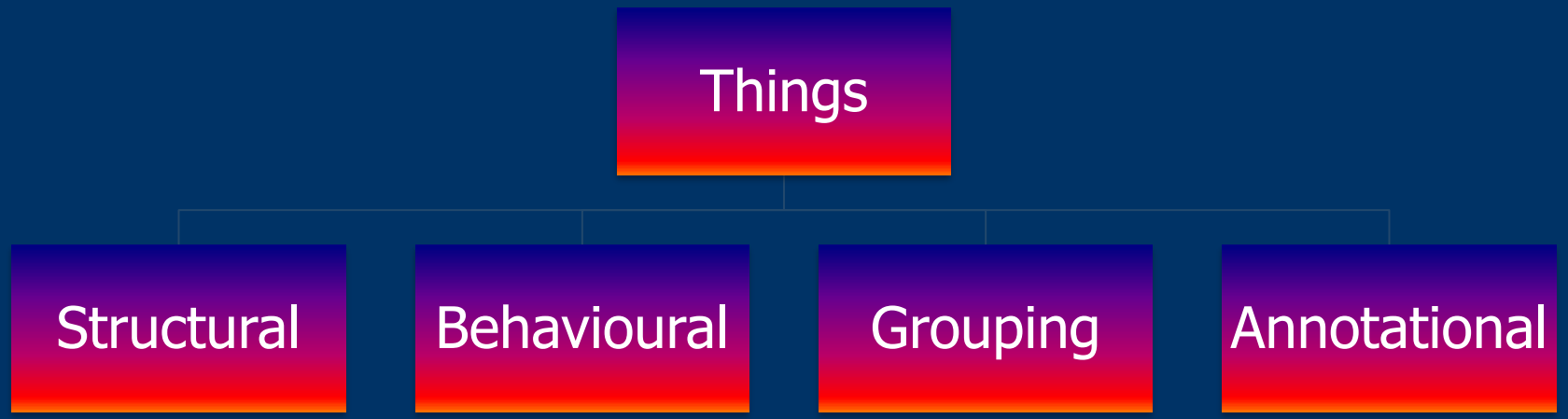
Conceptual model of the UML



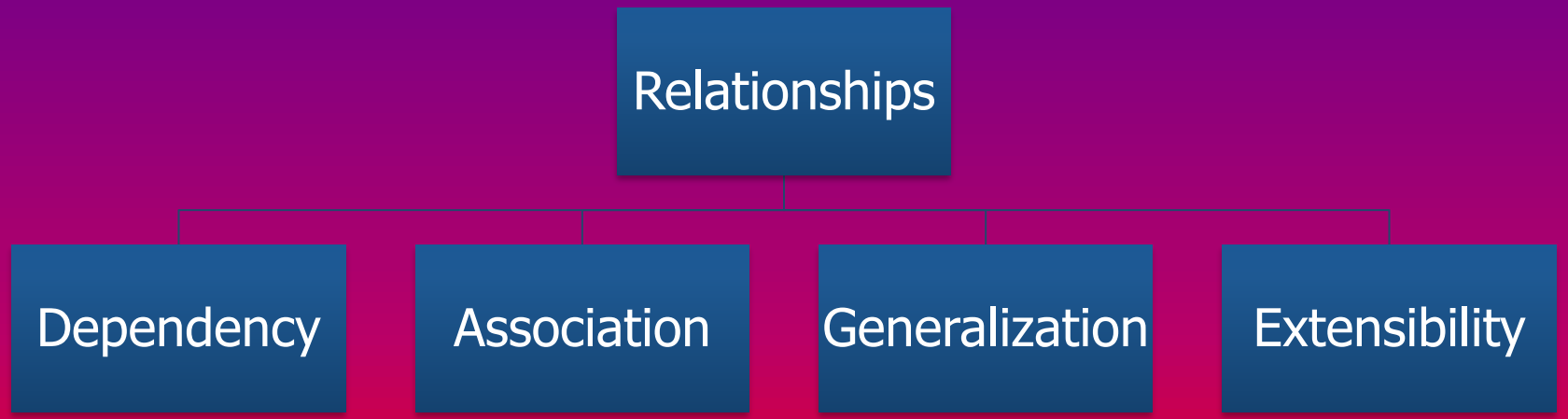
Conceptual model of the UML



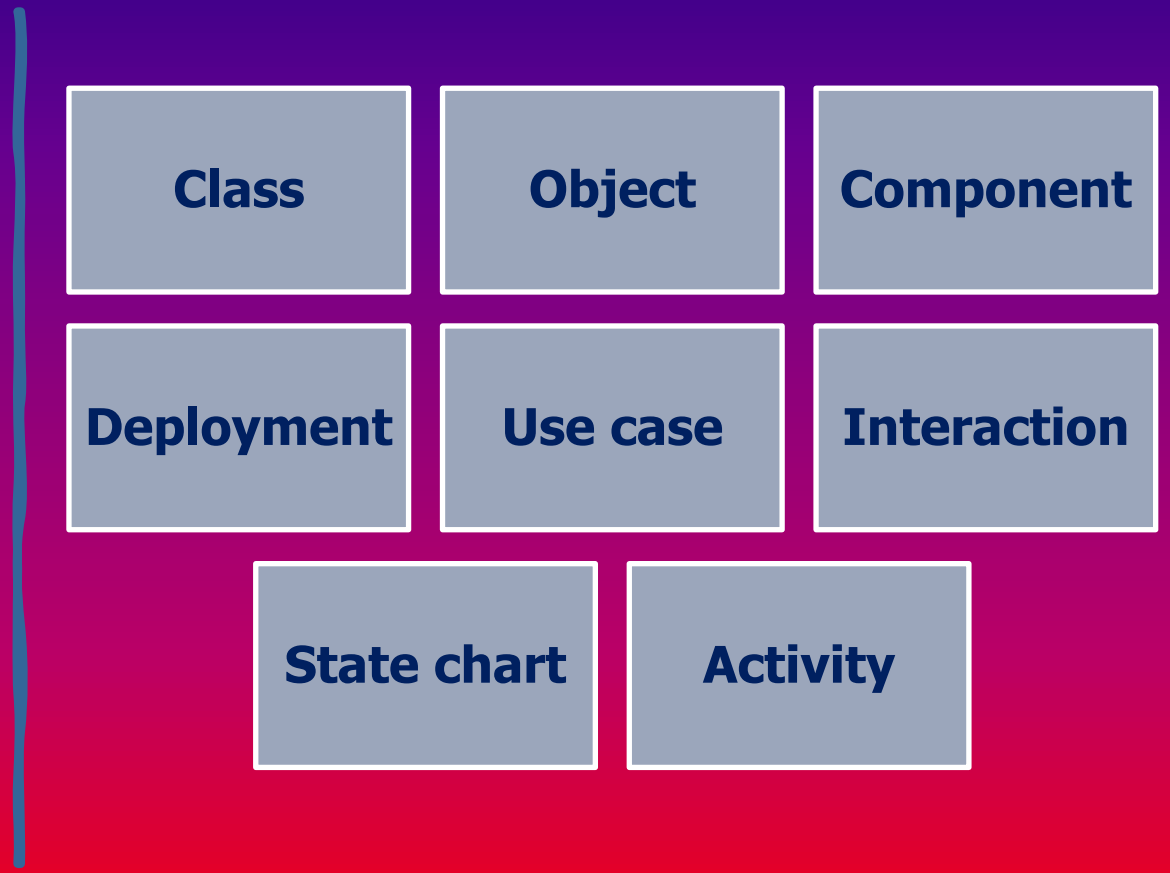
Conceptual model of the UML



Conceptual model of the UML



Conceptual model of the UML - Diagrams



Conceptual model of the UML

Structural Things

- Class
- Interface
- Collaboration
- Use case
- Components
- Nodes

Behavioral Things

- State machine
- Interaction

Grouping Things

- Package

Annotational Things

- Note

Things

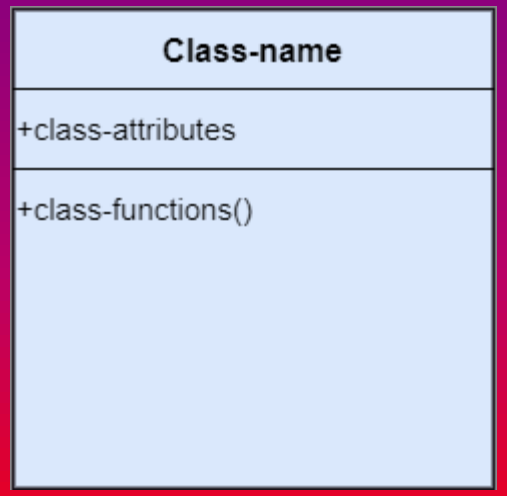
- Anything that is a real world entity or object is termed as things.
- It can be divided into several different categories:
- Structural things
- Behavioral things
- Grouping things
- Annotational things

Structural things

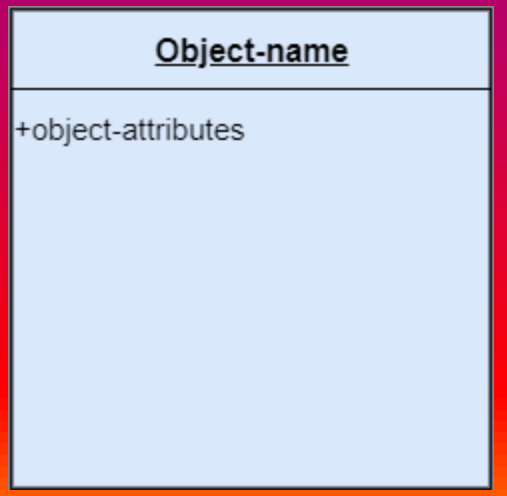
- It depicts the static behavior of a model is termed as structural things.They include
- Class
- Object
- Interface
- Node
- Collaboration
- component
- use case.

Structural things

Class: A Class is a set of identical things that outlines the functionality and properties of an object. class represents an object or a set of objects that share a common structure and behavior.

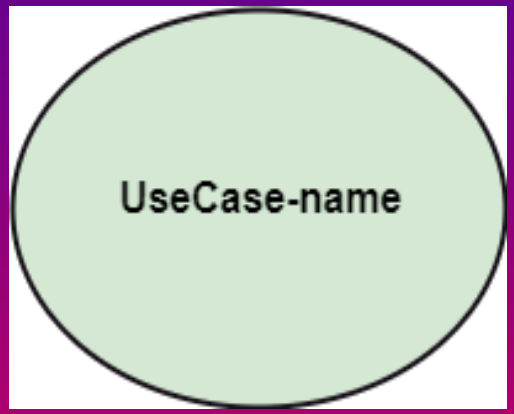


Object:: object is a specific instance of a class.
The notation of the object is similar to that of the class object name is always underlined

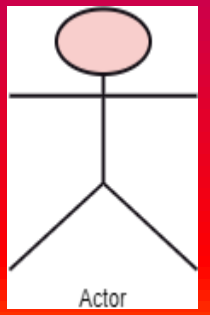


Structural things

Use case: It portrays a list of actions that a system performs to achieve a user's goal.

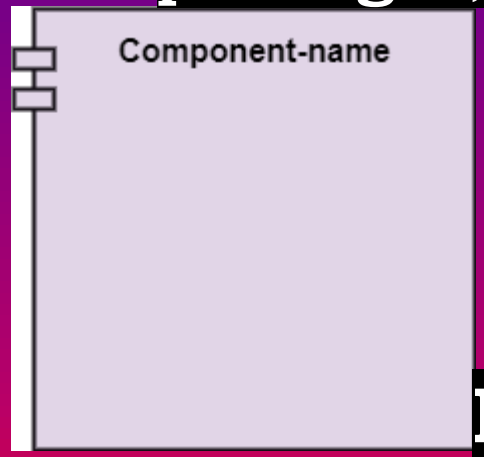


Actor: It Is an object that interacts with the system, for example, a user.

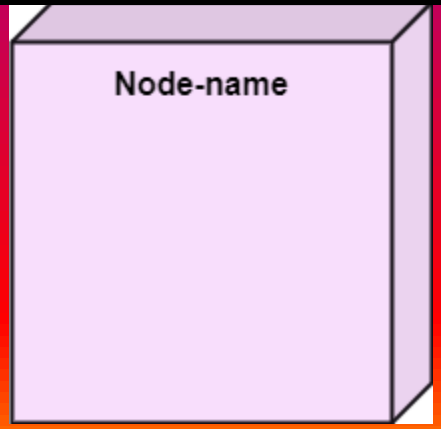


Structural things

Component: component is a modular part of a system that has a defined behavior. Components can be classes, packages, subsystems, or whole systems. .



Node: A physical element that exists at run time.



Behavioural Things

They are the verbs that encompass the dynamic parts of a model.

It depicts the behaviour of a system.

state machine

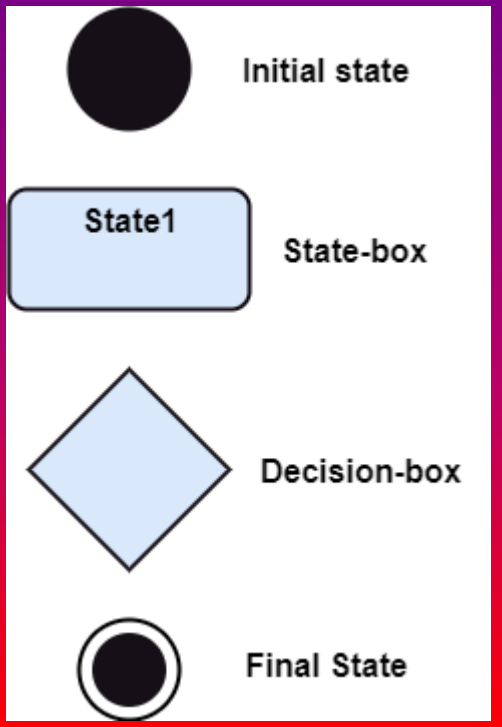
activity diagram

interaction diagram

Behavioural Things

State Machine: It defines a sequence of states that an entity goes through in the software development lifecycle.

record of several distinct states of a system component

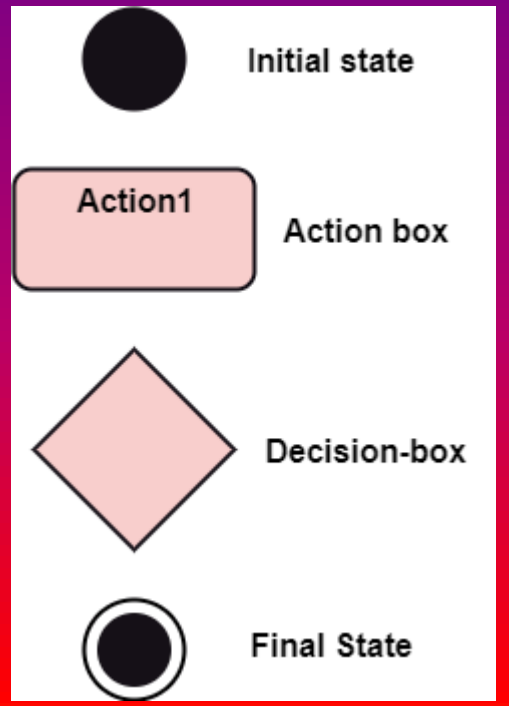


Behavioural Things

Activity Diagram: It portrays all the activities accomplished by different entities of a system.

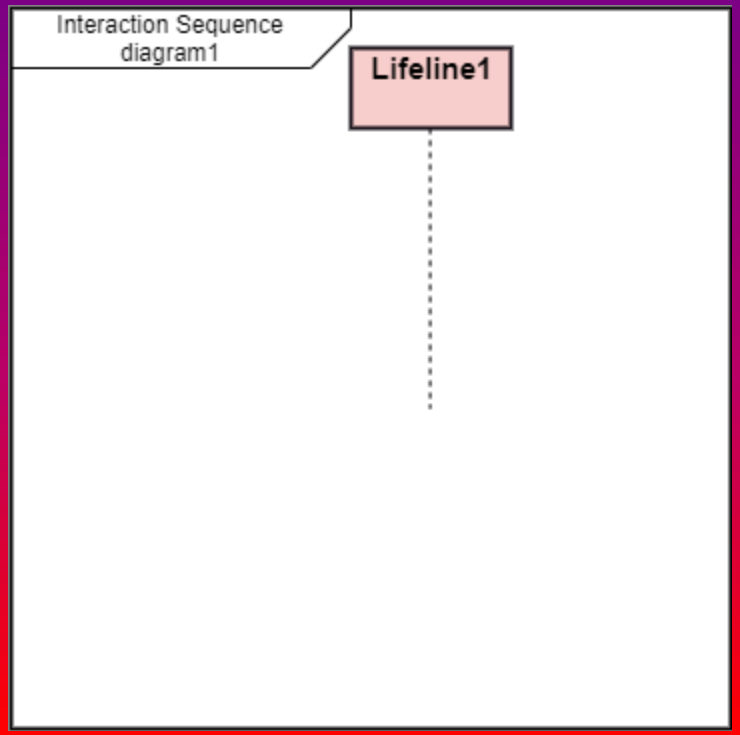
It is same as that of a state machine diagram.

It consists of an initial state, final state, a decision box, and an action notation.



Behavioural Things

Interaction Diagram: It illustrates the flow of messages between several components in a system.

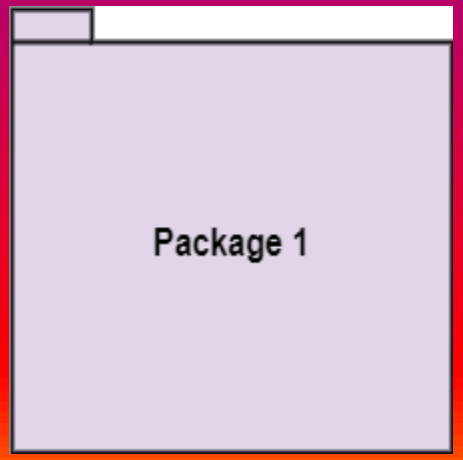


Grouping Things

It is a method that together binds the elements of the UML model.

In UML, the package is the only thing, which is used for grouping.

Package: Package is the only thing that is available for grouping behavioral and structural things.

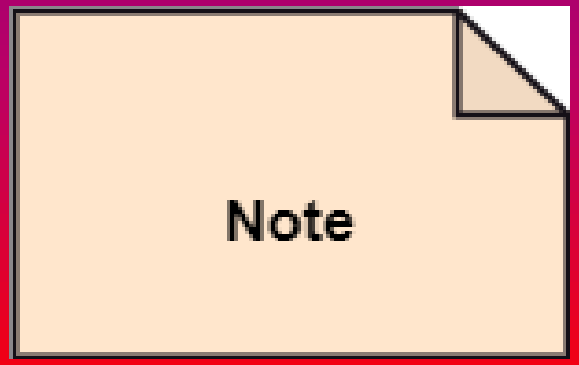


Annotation Things

It captures the remarks, descriptions, and comments.

In UML, a note is the only Annotational thing.

Note: It is a kind of yellow sticky note.



Relationships

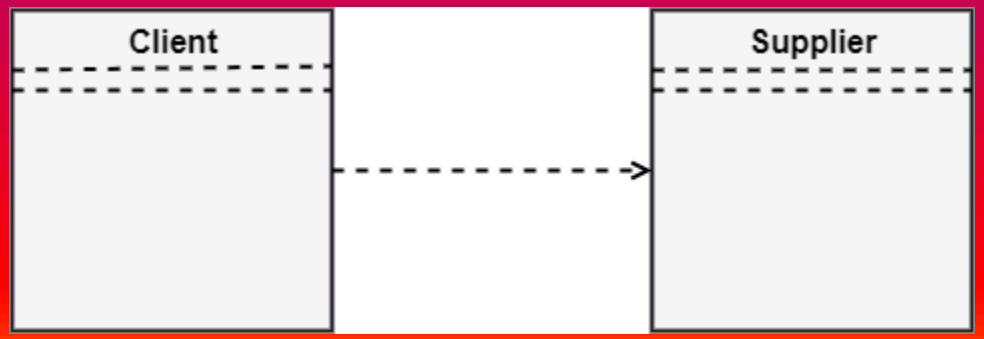
It illustrates the meaningful connections between things.

There are four types of relationships given below

Dependency: change in target element affects the source element

source element is dependent on the target element.

It is denoted by a dotted line followed by an arrow at one side as shown below,



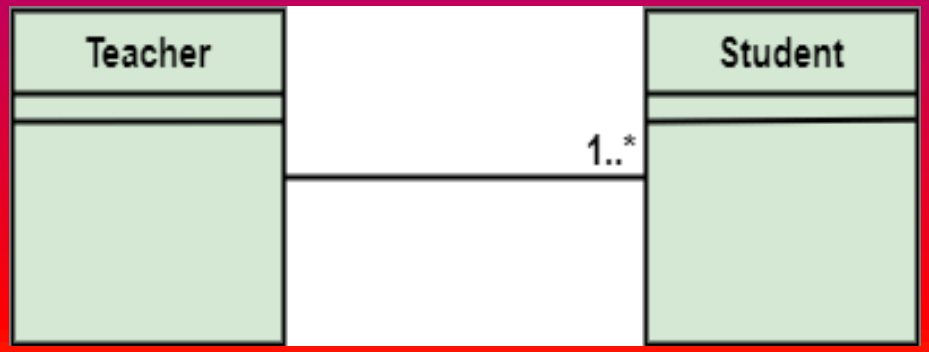
Relationships

Association: It associates the entities to the UML model.

It depicts the relationship between objects

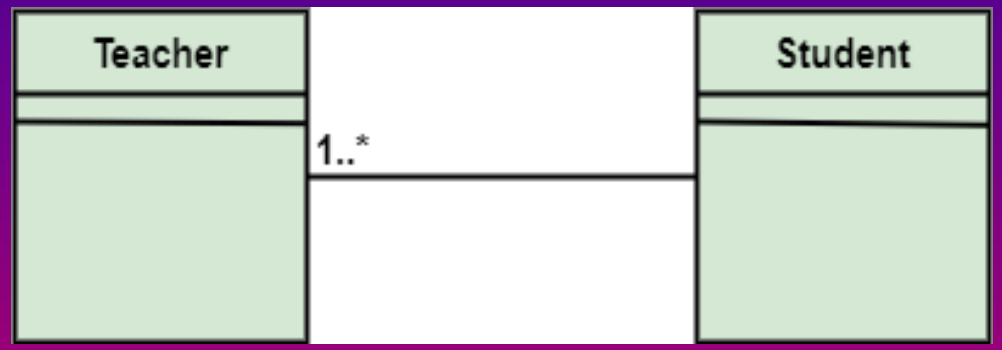
binary relationship between the objects representing an activity.

A single teacher has multiple students.



Relationships

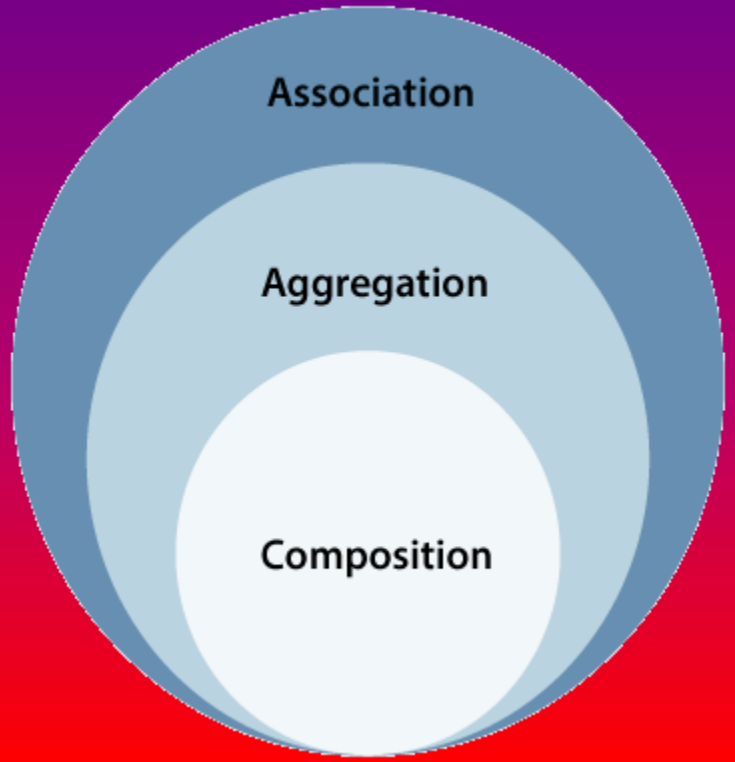
A single student can associate with many teachers.



The person works for the company. Here the person works for the company, and not the company works for a person.

Relationships

The composition and aggregation are two subsets of association.

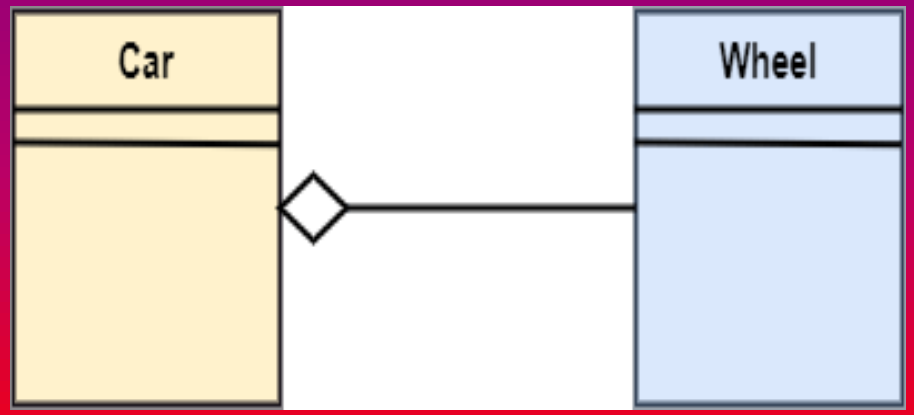


Relationships

Aggregation

It represents has a relationship.

It describes a part-whole or part-of relationship.



Relationships

Composition

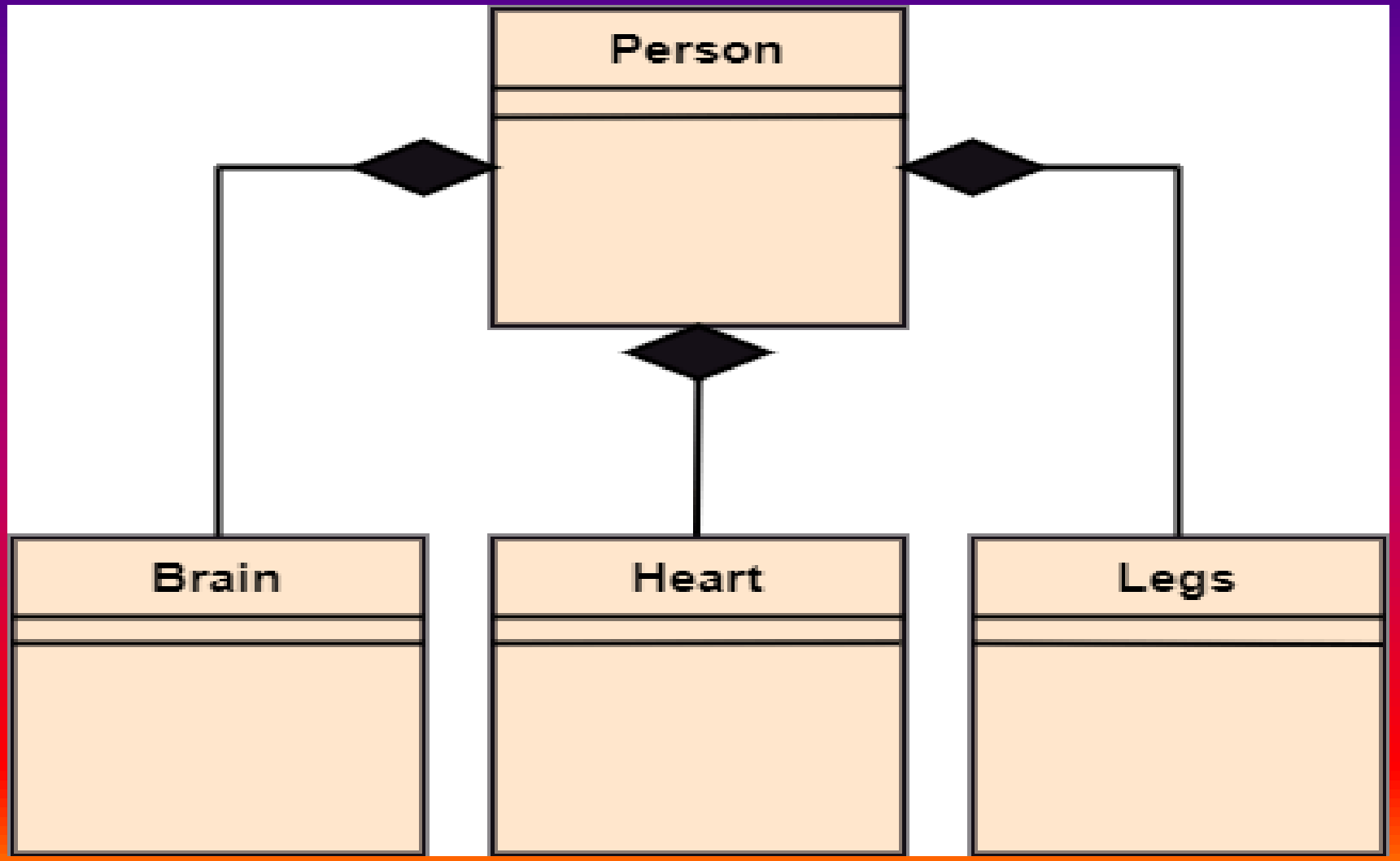
The composition is a part of aggregation

It depicts dependency between a composite (parent) and its parts (children),

if the composite is discarded, so will its parts get deleted.

If the person is destroyed, the brain, heart, and legs will also get discarded.

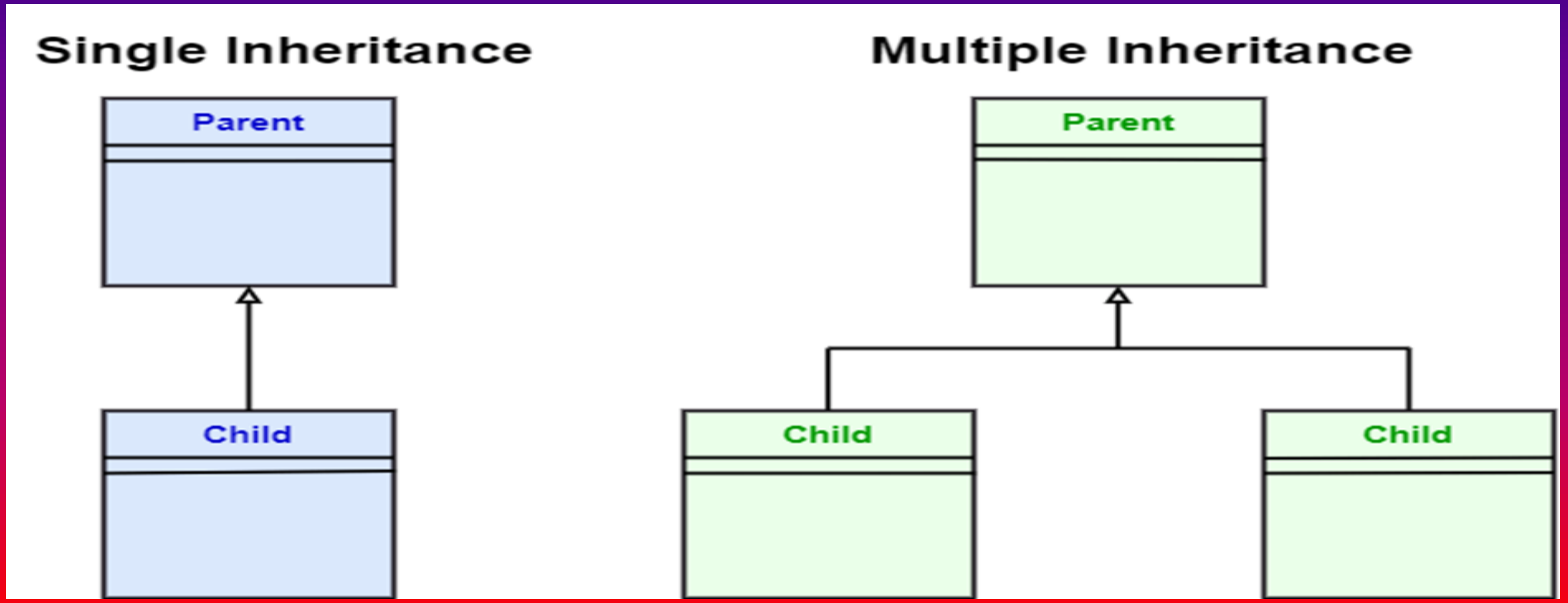
Composition Relationship



Generalization

- Generalization: It portrays a parent class or superclass and a specific kind of that thing (a child class or subclass).
- It is used to describe the concept of inheritance.
- The generalization relationship is represented by a solid line with a hollow arrowhead pointing towards the parent model element from the child model element.

Generalization



Diagrams in UML

It can be broadly classified as:

Structural Diagrams – Capture static aspects or structure of a system.

- Component Diagrams
- Object Diagrams
- Class Diagrams
- Deployment Diagrams

Behaviour Diagrams – Capture dynamic aspects or behavior of the system.

- Use Case Diagrams
- State Diagrams
- Activity Diagrams
- Interaction Diagrams

UML Diagrams

Class Diagram –

- widely use UML diagram is the class diagram.
- depict the static structure of a system by showing system's classes,their methods and attributes
- Object Diagram –
- referred to as a screenshot of the instances in a system and the relationship that exists between them

UML Diagrams

- Component Diagram – represent the how the physical components in a system have been organized.
- depict the structural relationship between software system elements
- Deployment Diagram – represent system hardware and its software.
- what hardware components exist and what software components run on them.

Behavioural Diagrams

- State Machine Diagrams : it represents the behavior using finite state transitions.
- State diagrams are also referred to as State machines and State-chart Diagrams .
- Activity Diagrams :illustrate the flow of control in a system.
- model sequential and concurrent activities using activity diagrams

Behaviuoral Diagrams

- Use Case Diagrams –depict the functionality of a system or a part of a system.
- illustrate the functional requirements of the system and its interaction with external agents(actors).
- high level view of what the system or a part of the system does without going into implementation details
- Sequence Diagram – A sequence diagram simply depicts interaction between objects in a sequential order
- Collaboration Diagram : sequenced messages exchanged between objects.
- A communication diagram focuses primarily on objects and their relationships

Behavioral diagrams

- Any system can have two aspects, static and dynamic.
- A model is considered as complete when both the aspects are covered fully.
- It basically capture the dynamic aspect of a system.
- Dynamic aspect can be further described as the changing/moving parts of a system.

UML has the following five types of behavioral diagrams:

- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram

UseCase Diagram

- Usecase diagrams are a set of usecases, actors and their relationships.
- They represent the use case view of a system.
- They are Used for describing a set of user **scenarios**
- Mainly used for capturing user requirements
- Work like a **contract** between end user and software developers

UseCase Diagram

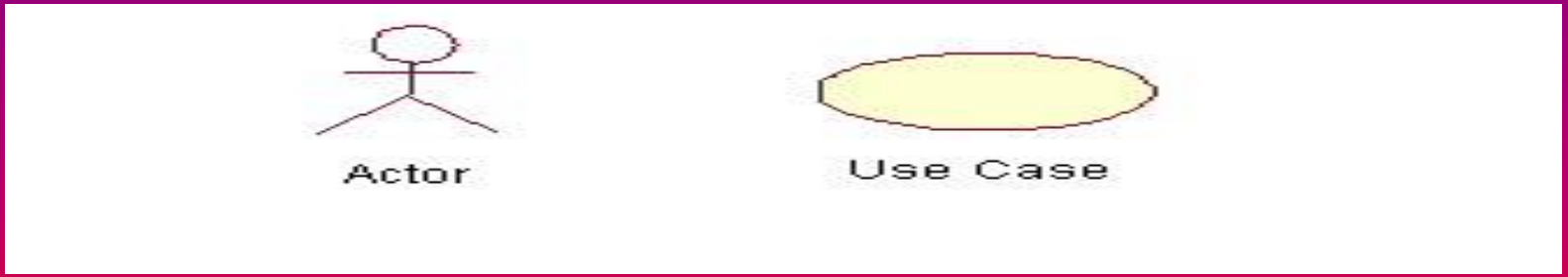
- Functionalities to be represented as an use case
- Actors
- Relationships among the use cases and actors
- The name of a use case is very important.
- So the name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Use note when ever required to clarify some important points.

Use Case Diagram (core components)

Actors: A role that a user plays with respect to the system,including human users and other systems.

e.g.,inanimate physical objects (e.g. robot);

Use case: A set of scenarios that describing an interaction between a user and a system, including alternatives.



System boundary: rectangle diagram representing the boundary between the actors and the system.

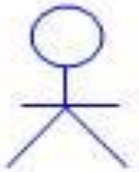
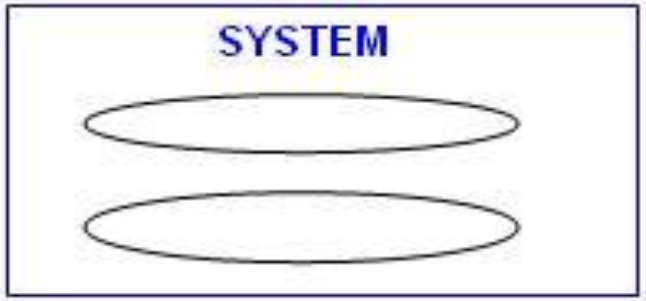
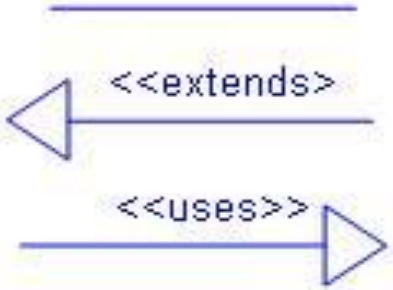
Usecase Diagram

- It represent the dynamic behavior of a system. It depicts the high-level functionality of a system

1. It gathers the system's needs.
2. It represents the interaction between the actors.

Use case diagrams consist of 4 objects.

- Actor
- Use case
- System

Symbol	Reference Name
	Actor
	Use Case
	System
	Relationship

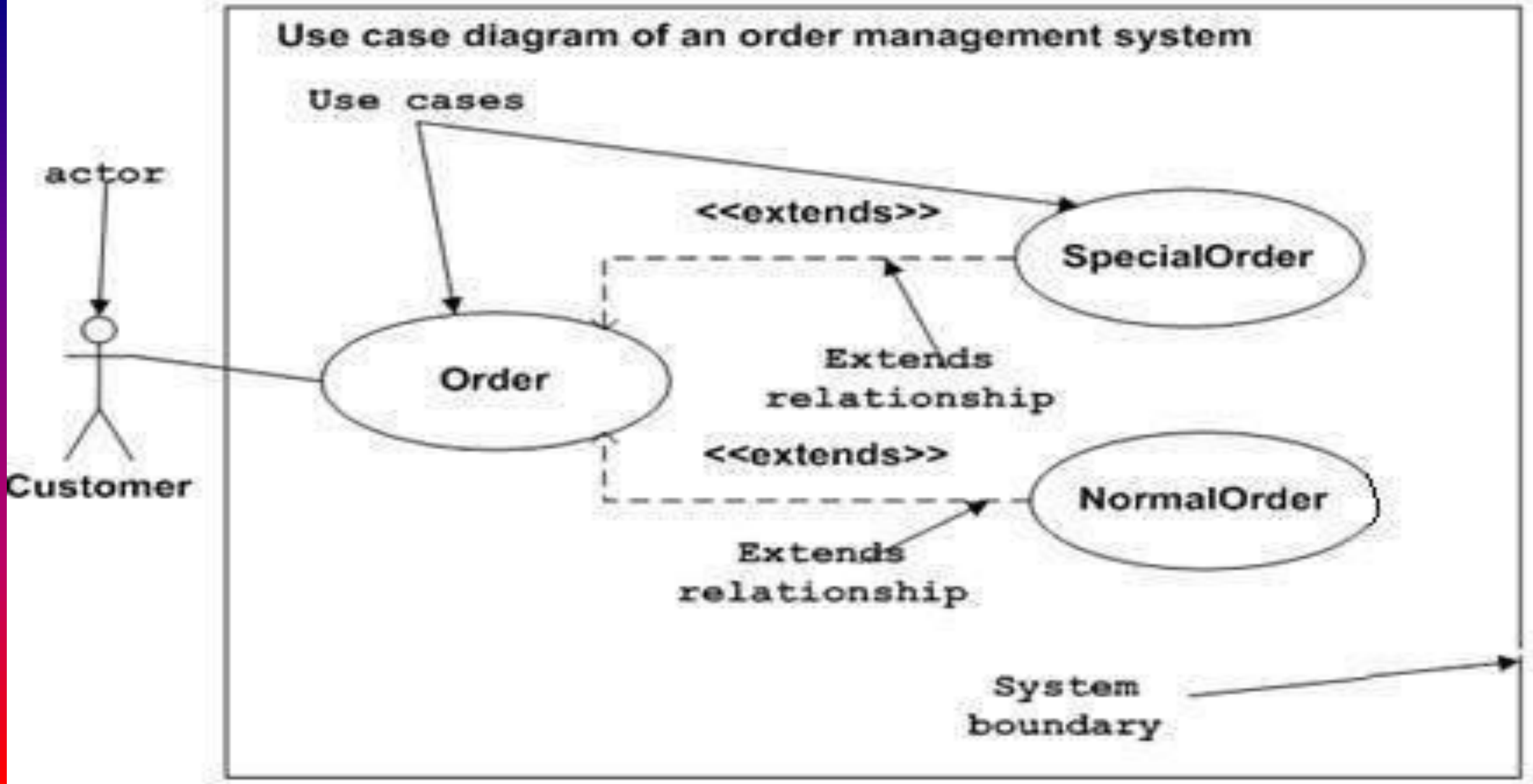
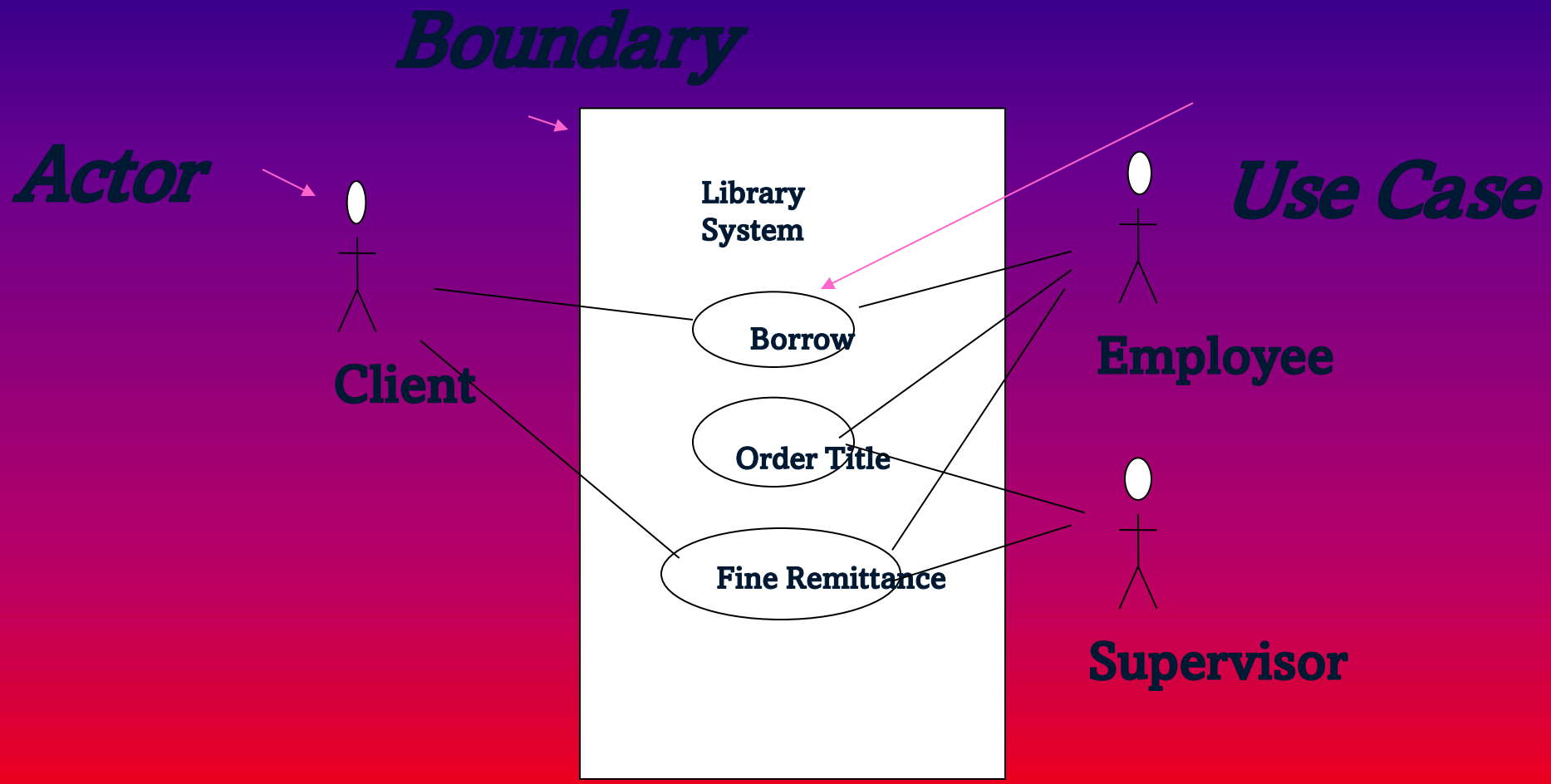
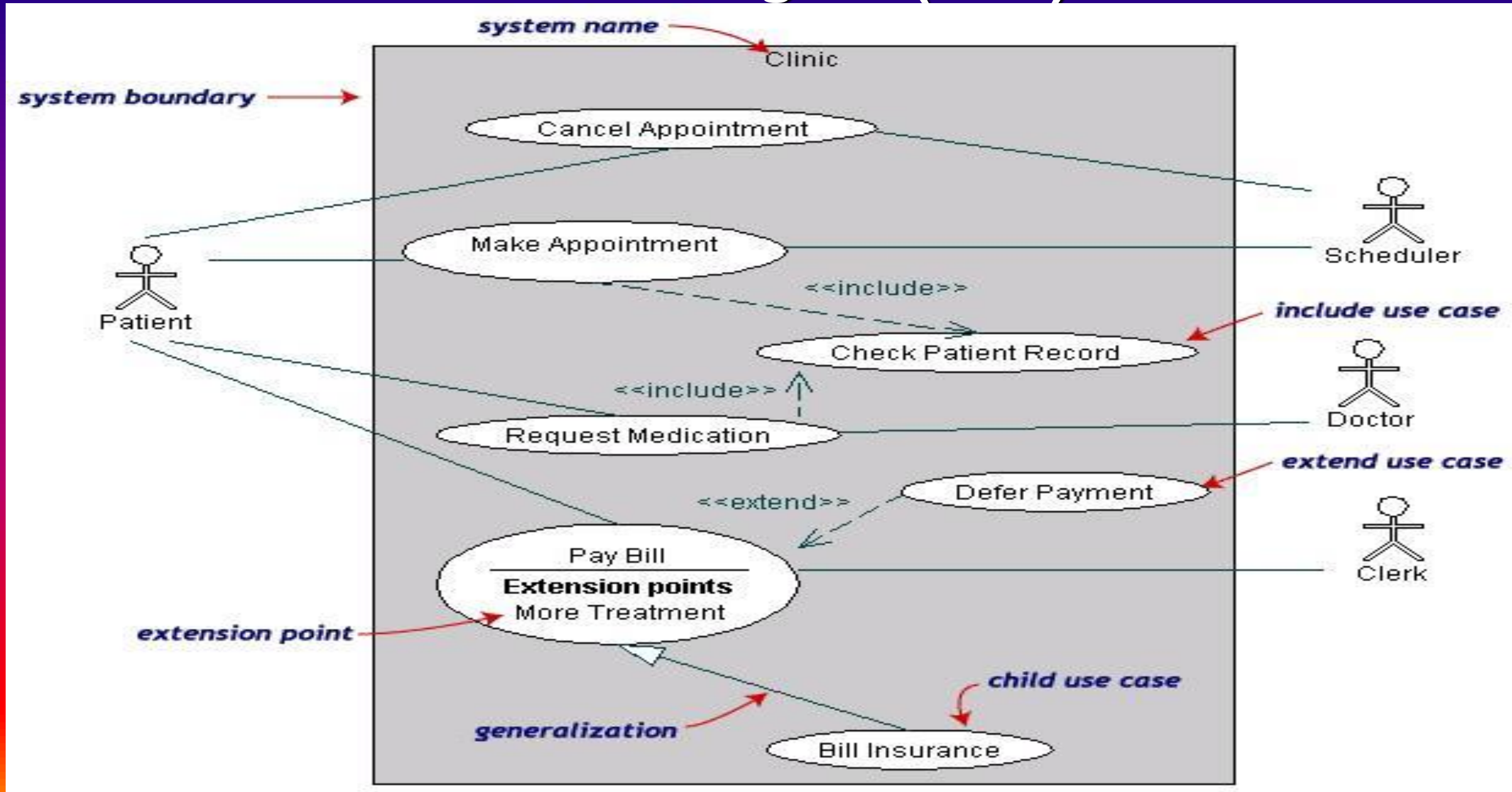


Figure: Sample Use Case diagram

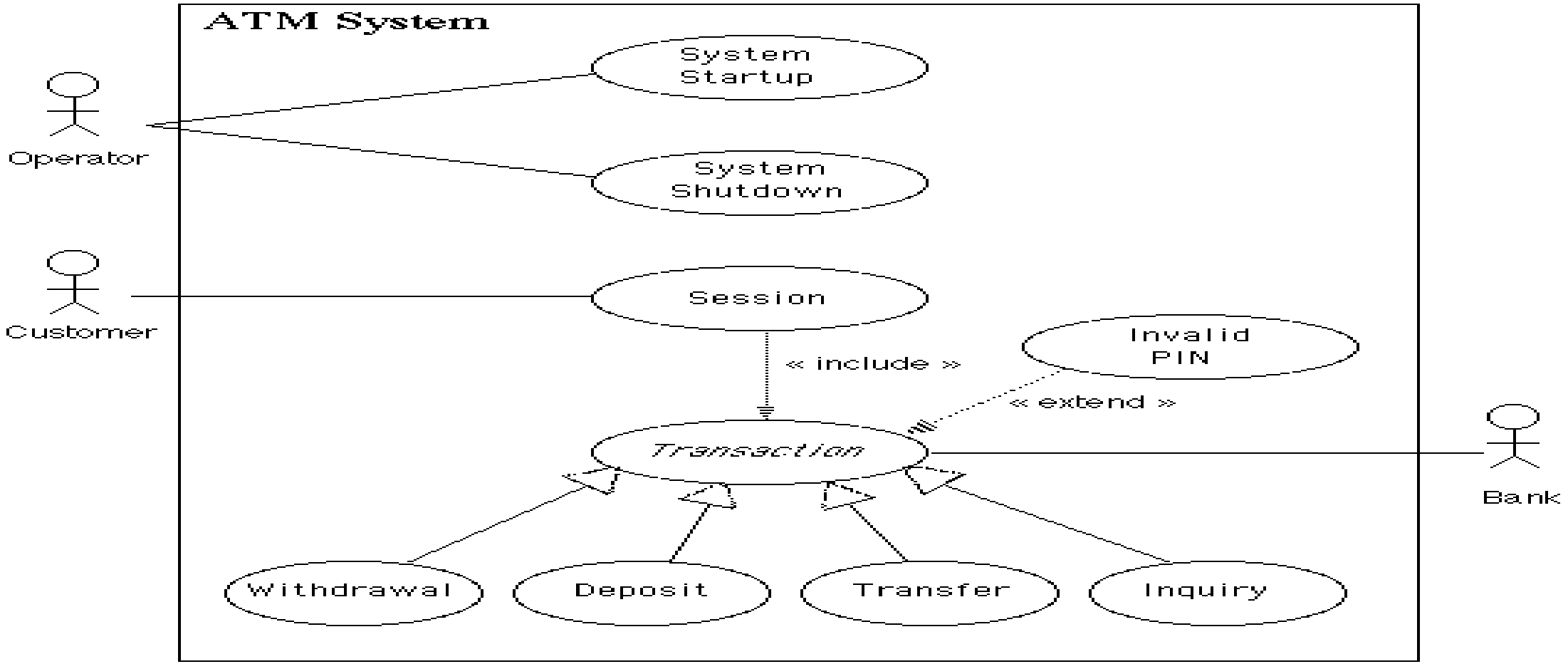
Use Case Diagrams



Use Case Diagrams(cont.)



USECASE DIAGRAM



USECASE DIAGRAM