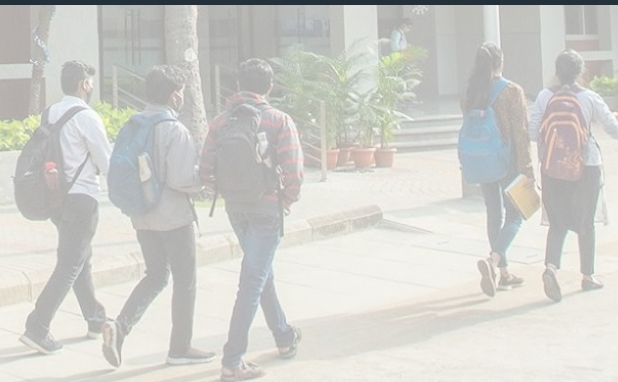


# NEW-AGE GLOBAL UNIVERSITY FOR LIBERAL EDUCATION



# Agile Software Engineering

## CS 2004

### Unit 1

#### Introduction to Agile

# Outline



- Faculty Profile
- Syllabus and Curriculum
- Assessment
- Course Introduction
- What is Software Engineering
- What is DevOps

# Syllabus



- Unit 1 - Introduction to Agile
- Unit 2 - Agile Methodologies & Practices
- Unit 3 - Sprint Management and Planning Techniques
- Unit 4 - DevOps Foundations
- Unit 5 - Agile Testing and Automation

# Unit 1: Introduction to Agile



- Introduction to Software engineering (SDLC)
- Software process models- waterfall, V model, Iterative model, Spiral model
- Introduction to Agile, Agile Manifesto, Agile development approach, traditional development models
- Benefits & pitfalls of Agile development
- Introduction to OOAD and UML.

# Unit 2 – Agile Methodologies & Practices



- Introduction to various Agile methodologies such as Scrum, XP, Lean, and Kanban
- Scrum: Scrum Roles – Product Owner, Scrum Master, Team, Release Manager, Project Manager, Product Manager, Architect, events, and artifacts
- Product Inception: Product vision, stakeholders, initial backlog creation;
- Agile Requirements – User personas, story mapping, user stories, 3Cs, INVEST Acceptance criteria, sprints, requirements, product backlog and backlog grooming
- Test First Development; Pair Programming and Code reviews; code coverage and tools

## Unit 3 - Sprint Management and Planning Techniques



- Sprint Planning, Sprint Reviews, Sprint Retrospectives Sprint Planning - Agile release and iteration (sprint) planning,
- Develop Epics and Stories, Estimating Stories, Prioritizing Stories (WSJF technique from SAFe),
- Iterations/Sprints Overview. Velocity Determination, Iteration Planning Meeting, Iteration
- Planning Guidelines, Development, Testing, Daily Stand-up Meetings, Progress Tracking, Velocity Tracking, Monitoring and Controlling: Burn down Charts, Inspect & Adapt (Fishbone Model), Agile Release Train

# Unit 4 – DevOps Foundations



- Overview of Devops and why it is needed;
- Continuous Integration, Build, Testing, Delivery and Deployment(CI and CD): Jenkins, Pylint, PyUnit.
- Overview of version control, branching and release using GIT -Git/Github/GitActions
- Creating pipelines, Setting up runners Containers and container orchestration (Docker, DockerHub and Kubernetes) for application development and deployment;

# Unit 5 – Agile Testing & Automation



- Testing: Functionality Testing, UI Testing(Junit, PyUnit, Sonar), Performance Testing, Security Testing, A/B testing;
- Agile Testing: Principles of agile testers; The agile testing quadrants, Agile automation, Test automation pyramid;
- Test Automation Tools - Selenium, Traceability matrix;

# Assessment

- In Class Activities & Quizzes
- CIE (70 Marks)
  - CP1: 20 Marks
    - 10 Marks (Quiz/In Class Test),
    - 5 Marks (Assignments & MOOC course),
    - 5 Marks (from exercises/course project of Lab Record)
  - CP2: 25 Marks (Pen & Paper)
  - CP3: 25 Marks
    - 20 Marks from Lab Record,
    - 5 Marks (Assignments & MOOC course)
- SEE (30 Marks) – Pen & Paper

# Reference Books



- Essential Scrum: A Practical Guide to the Most Popular Agile Process  
Kenneth S. Rubin 2012, published by Addison-Wesley Professional
- Software Engineering, Ian Sommerville

# What is a software?

*Software, in its most general sense, is a set of instructions or programs instructing a computer to do specific tasks.*

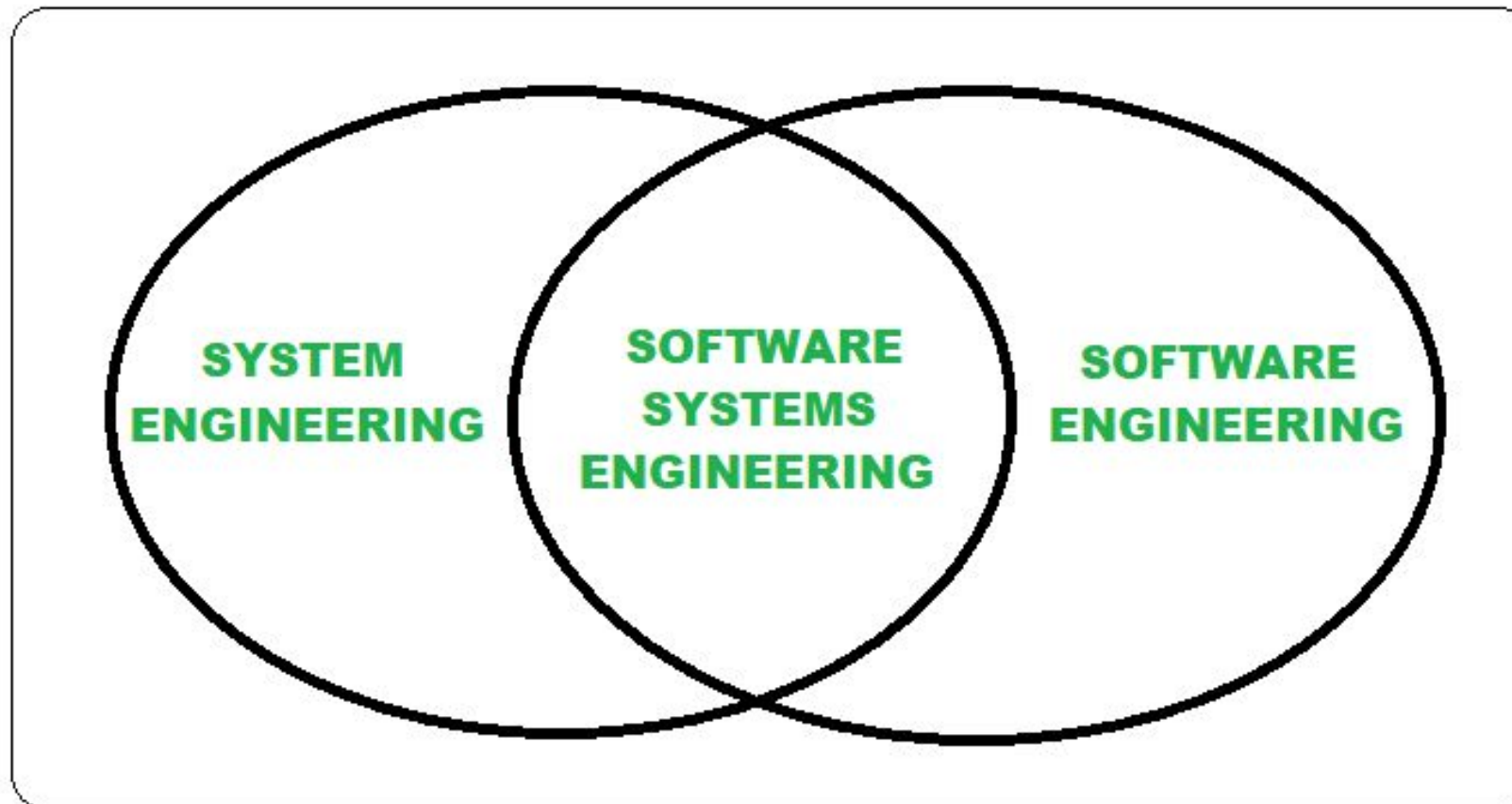
*Software is a generic term used to describe computer programs. Scripts, applications, programs and a set of instructions are the terms often used to describe software.*

# What is Software Engineering

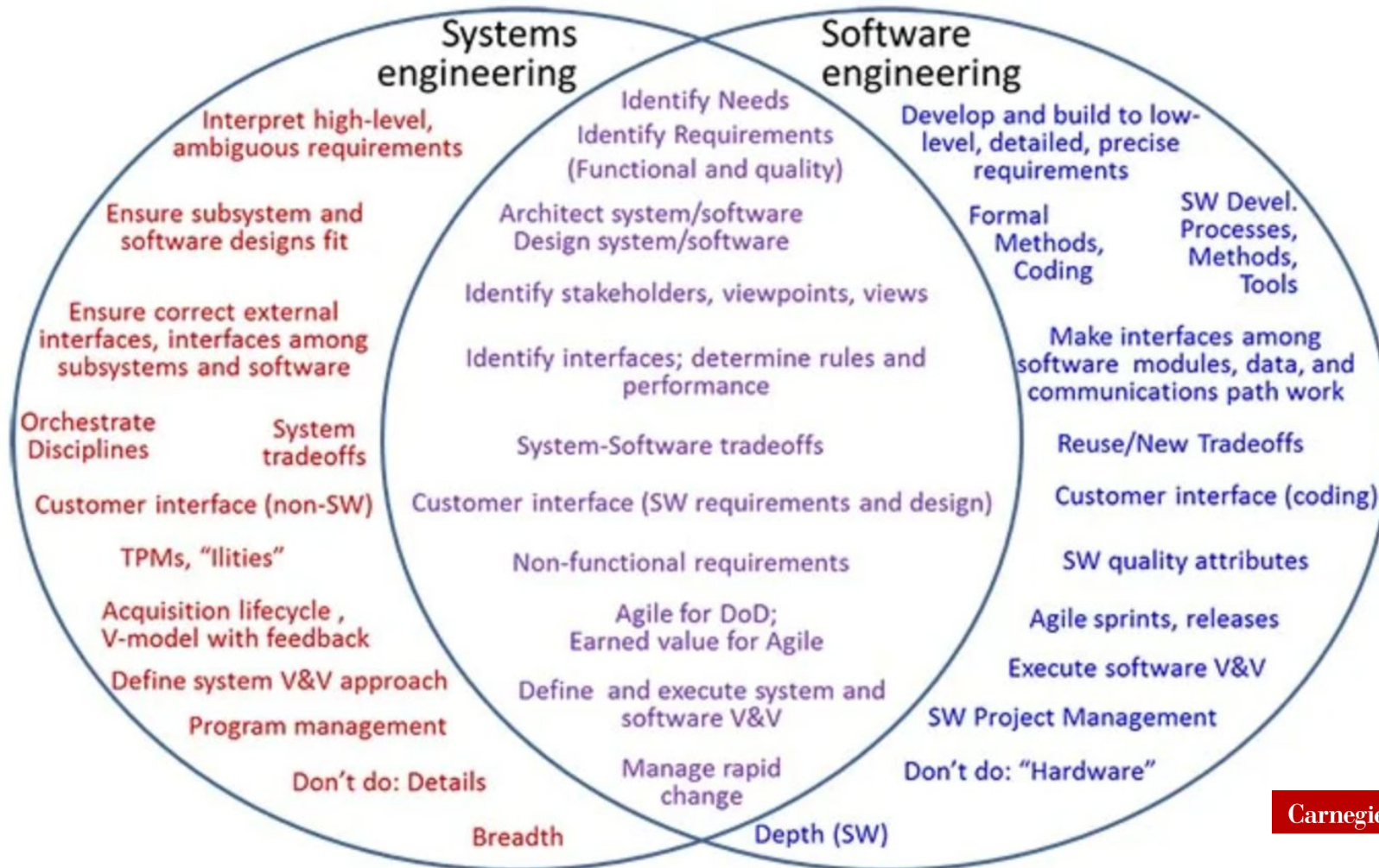


- Software Engineering is the process of designing, developing, testing, delivering and maintenance of software
- It is systematic and disciplined approach to software development
- It aims to deliver high quality, reliable and maintainable software
- It is discipline that is concerned with all aspects of software production from early stages of system specifications to maintenance

# Systems Engineering vs Software Engineering



# Systems Engineering vs Software Engineering



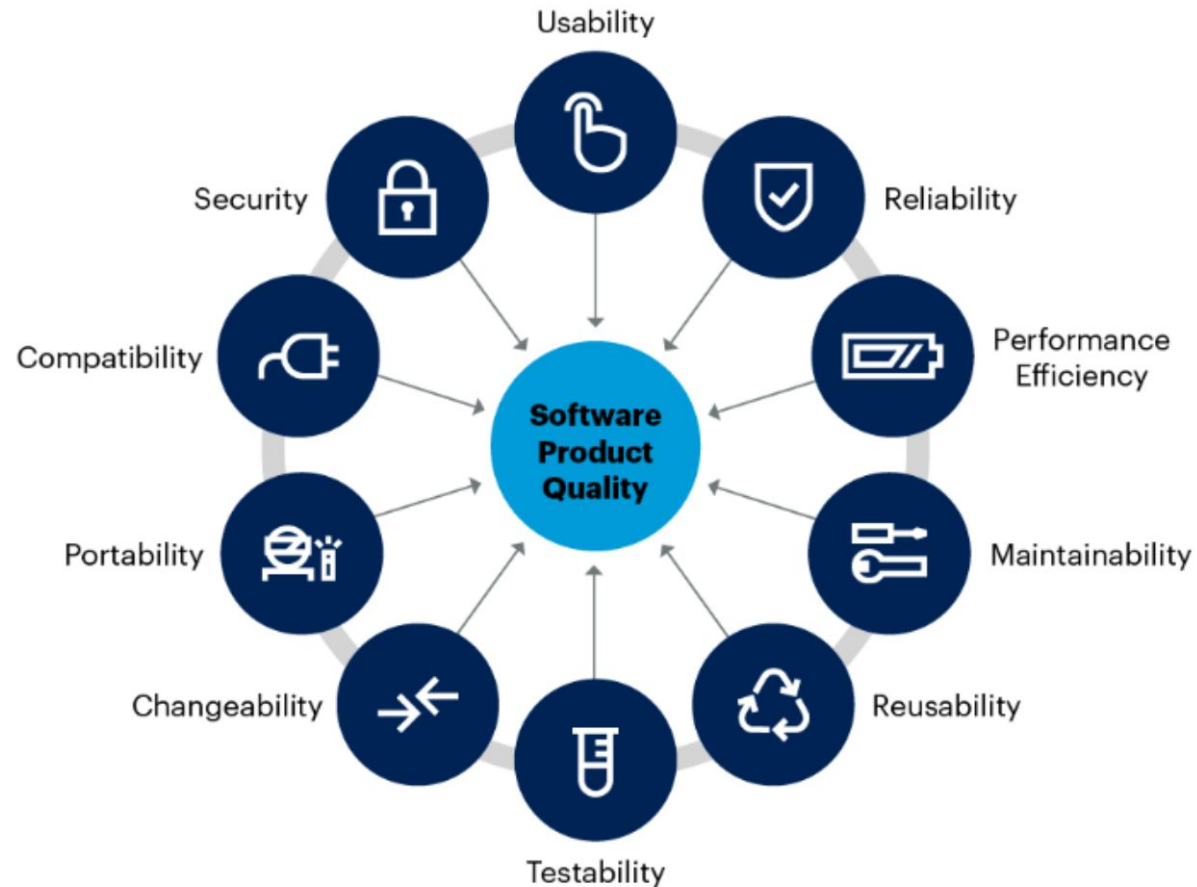
Carnegie Mellon University

# Fundamental Activities



- Software Specifications
- Software Development
- Software Validation
- Software Evolution

# Non-functional Quality Characteristics



Source: Gartner (August 2021)

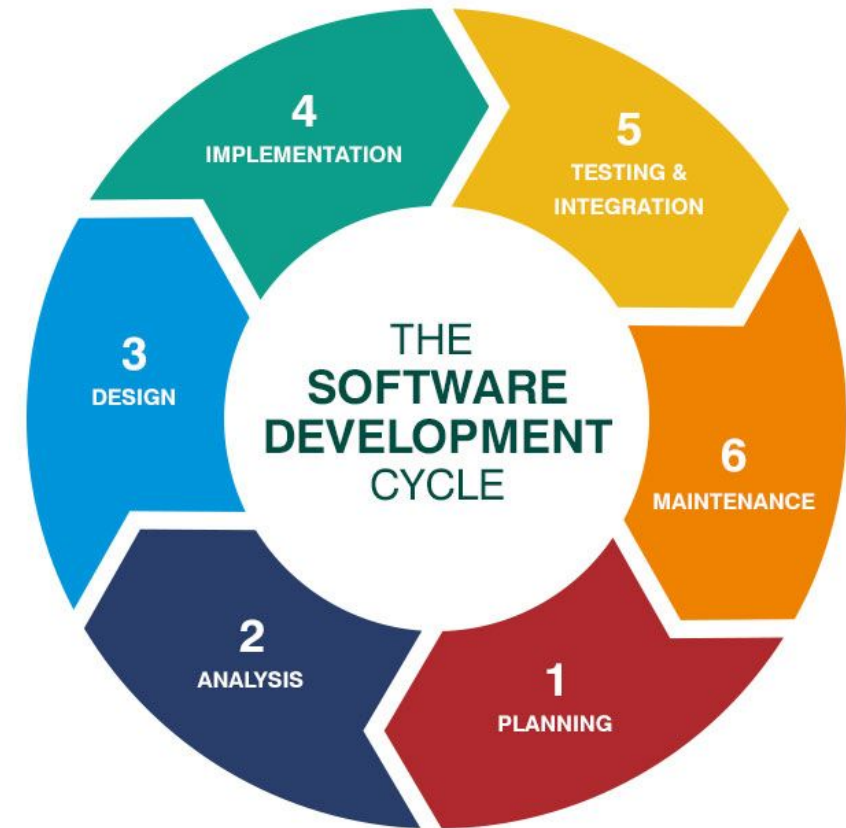
# Software Engineering Diversity

- Standalone
- Interactive / Transaction Based
- Embedded Control Systems
- Entertainment Systems
- Research & Modeling
- IOT / Data Collection
- System of Systems

- It highlights the needs of differentiated software engineering approaches for various kind of software projects

# Software Development Life Cycle (SDLC) – Key stages

- Planning
- Requirement gathering & Analysis
- Design
- Implementation (Coding/Development)
- Testing
- Deployment
- Maintenance



# SDLC Models

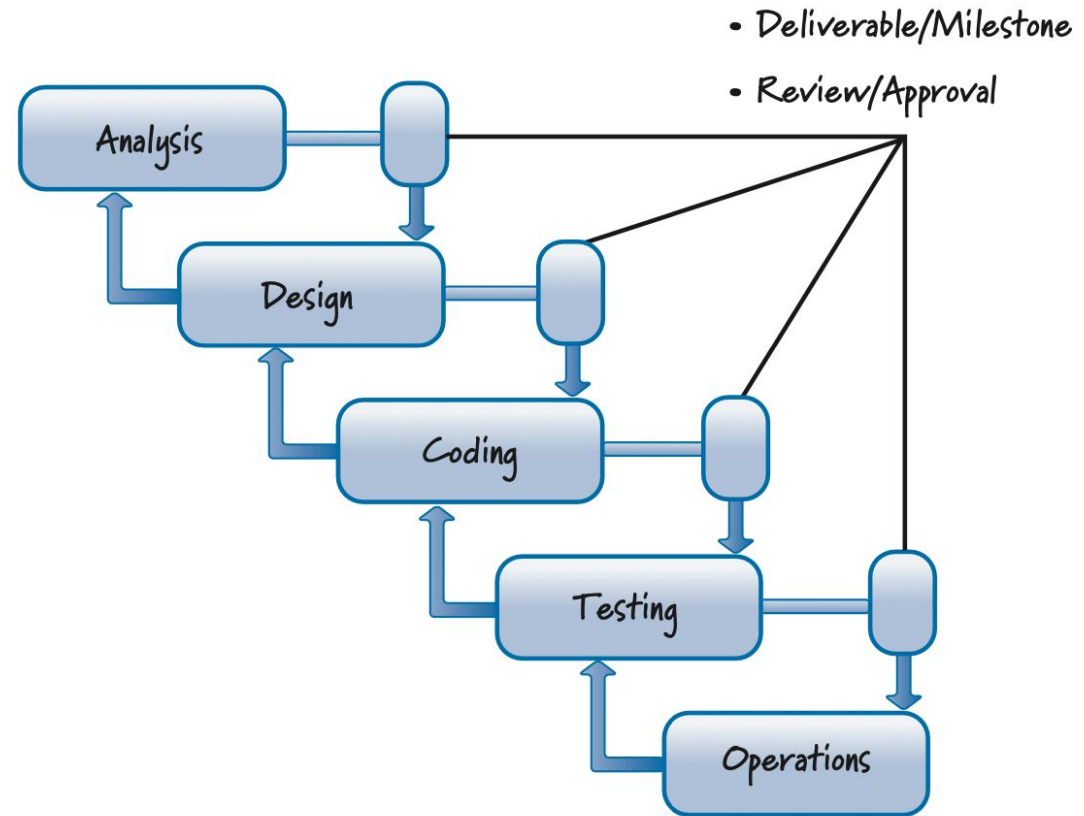
1. Waterfall Model
2. V-Model (Validation and Verification)
3. Iterative Model
4. Spiral Model
5. Agile Model

# Waterfall Model

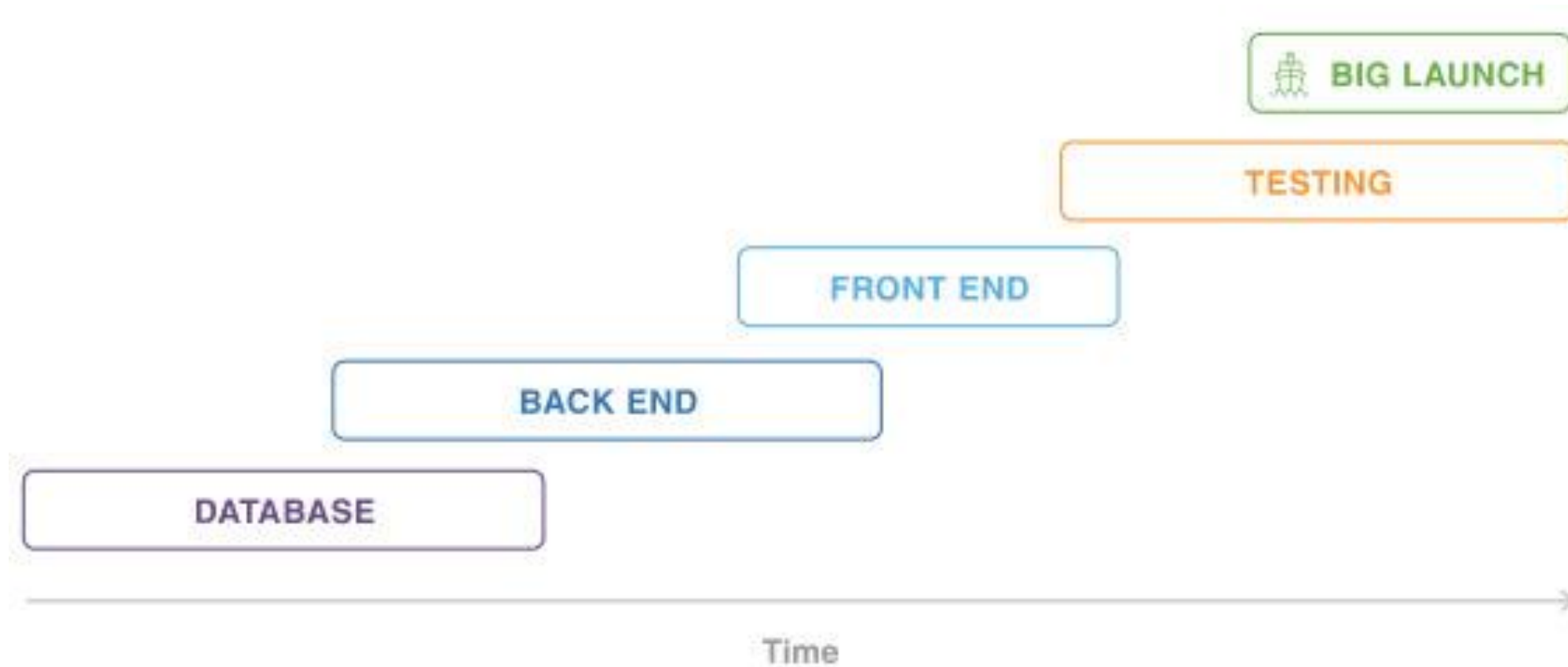
- The waterfall model is a linear, sequential approach to software development.
- It follows a set path with limited deviation and is best suited for projects with well-defined requirements and a clear understanding of the final product.
- Once a phase is completed, it can be difficult and costly to revisit a previous stage

# Waterfall or sequential development

- Works well if you are applying it to problems that are well defined, predictable, and unlikely to undergo any change.
- Problem is that most product development efforts are anything but predictable especially at the beginning



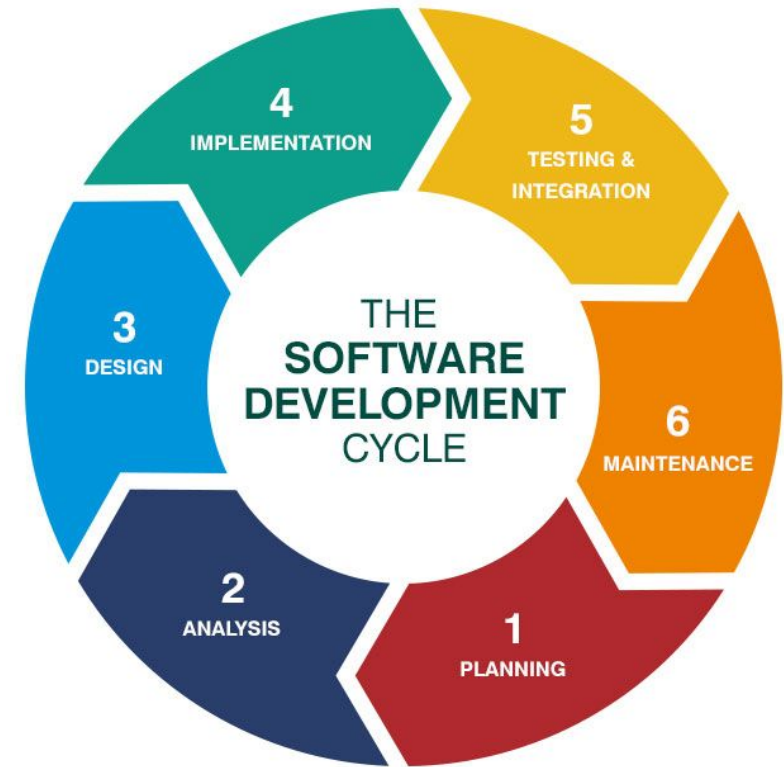
# Waterfall Model (Illustration)



# Software Development Life Cycle (SDLC)

## Key stages

1. Planning
2. Requirement gathering & Analysis
3. Design
4. Implementation (Coding/Development)
5. Testing
6. Deployment
7. Maintenance



# 1. Planning

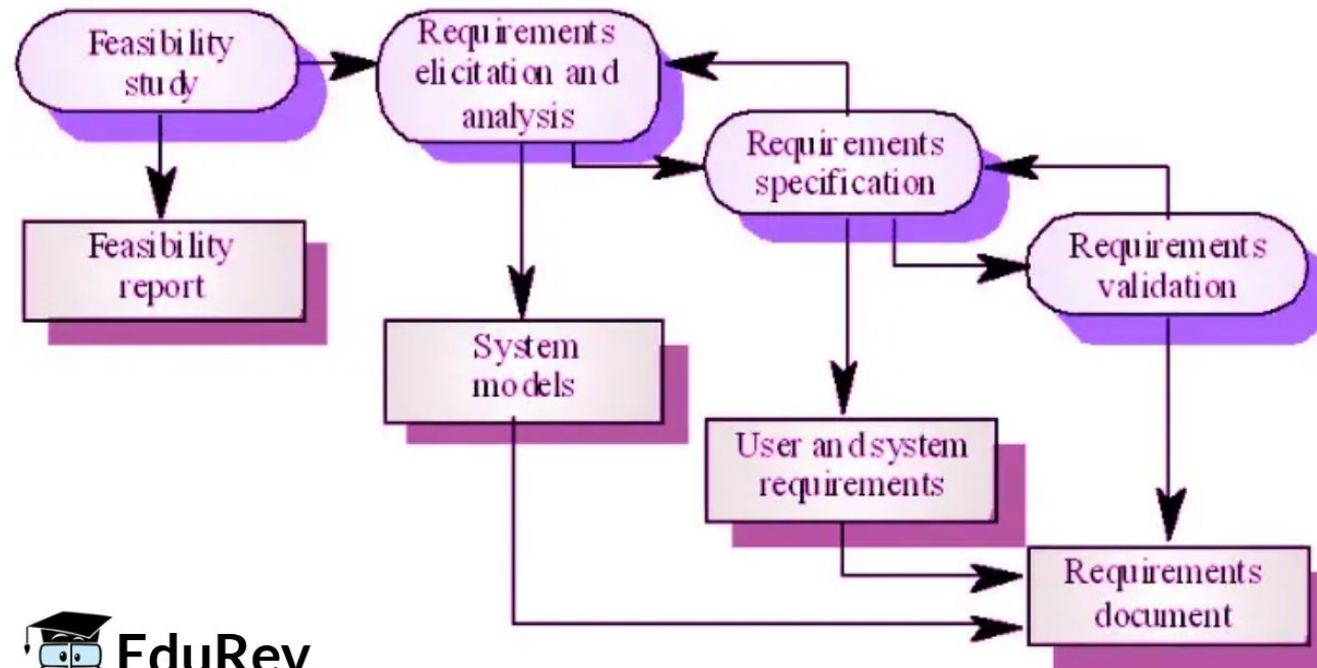
- **Objective:** Define the scope and purpose of the project.
- **Activities:**
  - Identify project goals,
  - perform feasibility studies,
  - estimate resources,
  - set timelines, and
  - develop a project plan.
- **Outcome:**
  - Project plan,
  - budget, and
  - resource allocation.

## 2. Requirement gathering & Analysis



- **Objective:** Understand and document what the software needs to accomplish.
- **Activities:**
  - Engage with stakeholders to gather functional and non-functional requirements, analyze the needs, and document them in a Software Requirements Specification (SRS).
- **Outcome:**
  - Clear and detailed requirements document

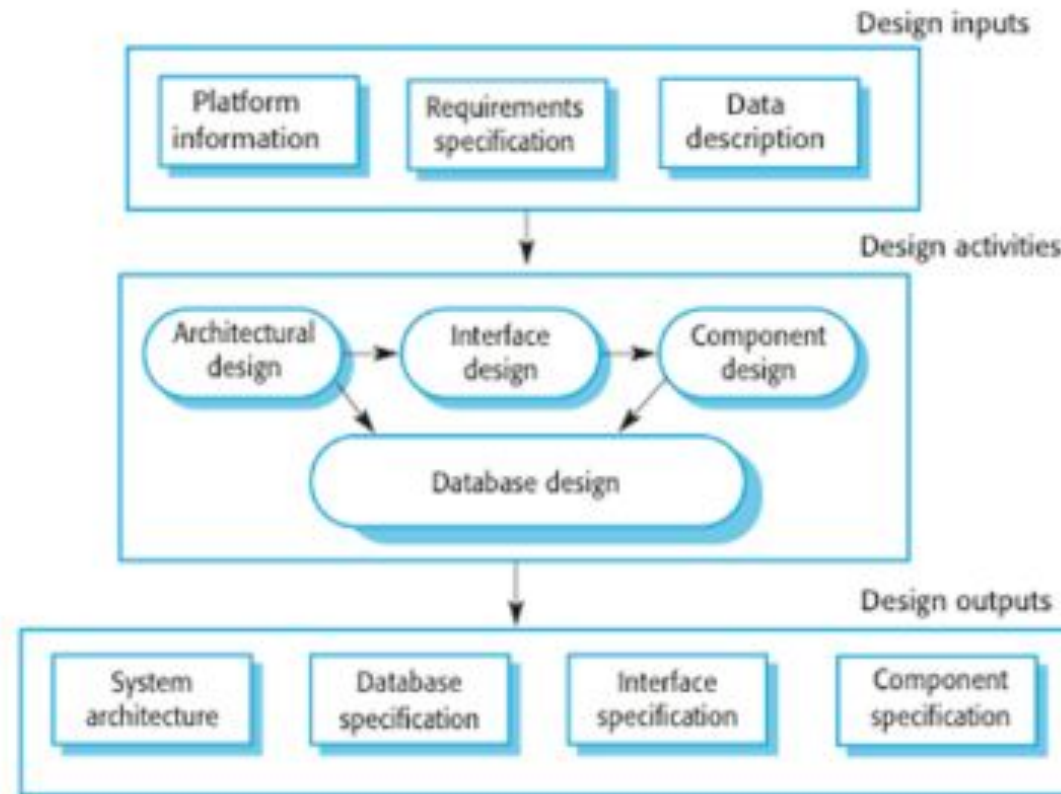
# The requirements engineering process



## 3. Design

- **Objective:** Create the architecture and design of the software.
- **Activities:**
  - Develop system architecture, database design, and interface design based on the requirements. Consider scalability, security, and performance.
- **Outcome:**
  - Design documents including system architecture, data models, and interface layouts.

# A general model of the design process



## 4. Implementation (Coding/Development)



- **Objective:** Write the actual code to build the software system.
- **Activities:**
  - Break the project into small coding tasks, assign them to developers, and begin coding according to the design.
- **Outcome:**
  - Functional software components ready for integration.

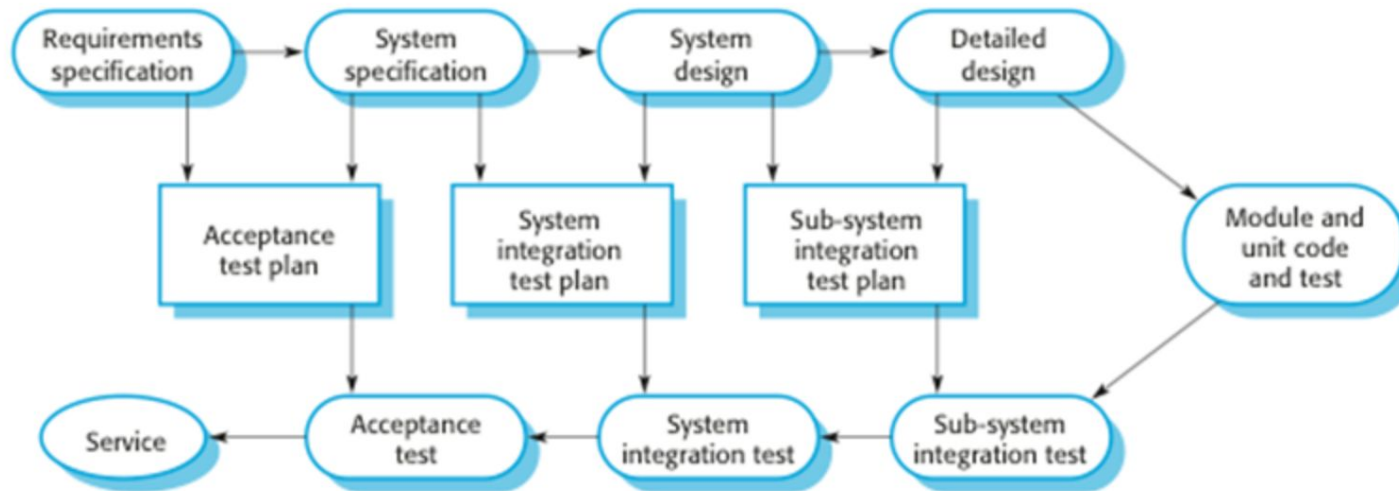
## 5. Testing

- **Objective:** Ensure the software works as expected and meets the requirements.
- **Activities:**
  - Perform various levels of testing (unit, integration, system, acceptance) to identify bugs, issues, or deviations from the requirements
- **Outcome:**
  - Bug-free, fully functional software.

# Stages of testing



# Testing phases in a plan-driven software process



# 6. Deployment

- **Objective:** Release the software for use in a real-world environment
- **Activities:**
  - Deploy the software to the production environment, ensure it is configured correctly, and make it available to end-users
- **Outcome:**
  - Updated, functional, and stable software over time.

# 7. Maintenance

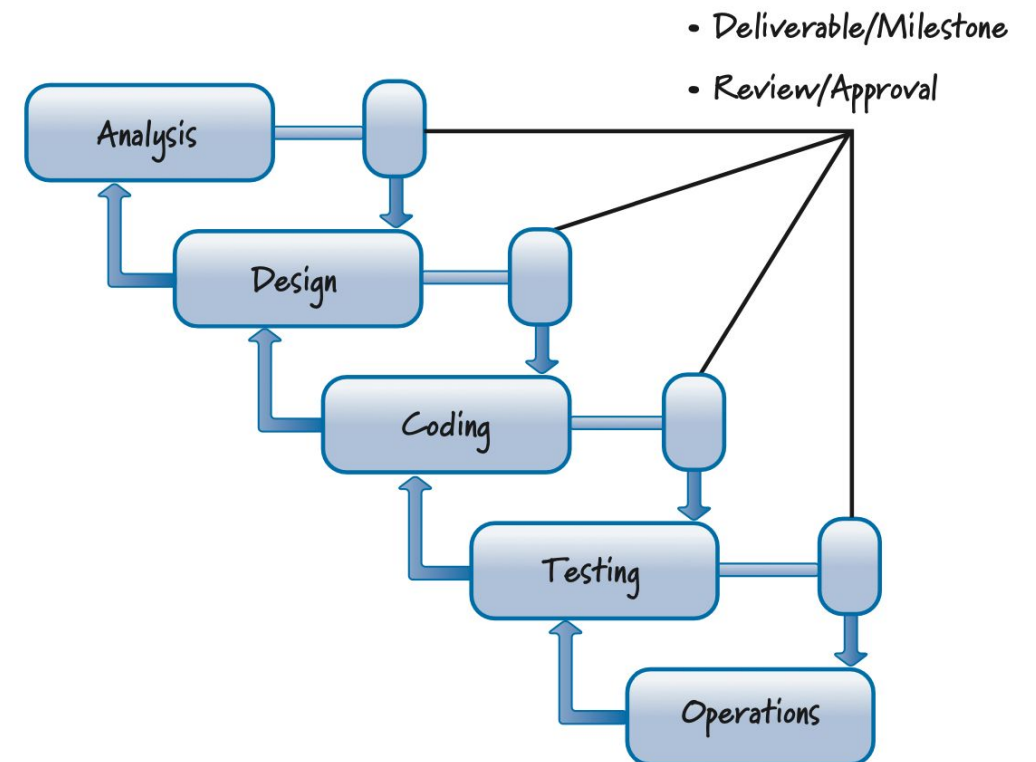
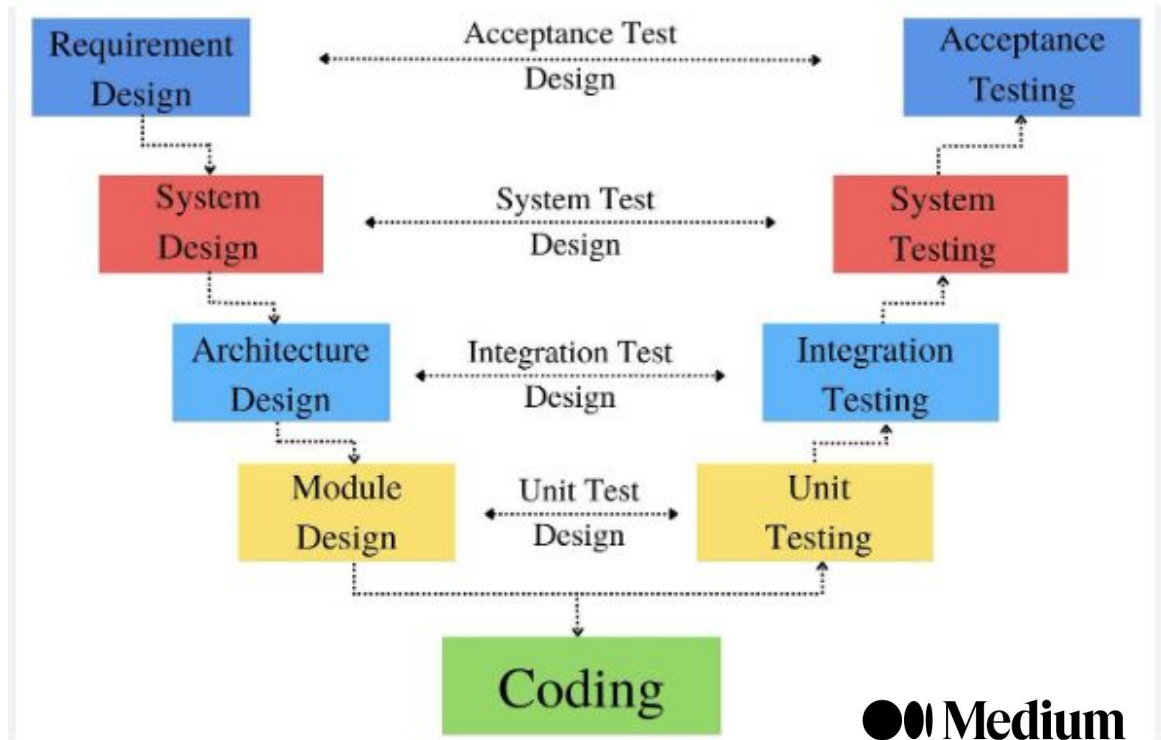
- **Objective:** Keep the software up-to-date and functioning after deployment
- **Activities:**
  - Address bugs reported by users, implement enhancements, update the software to meet new requirements, and perform routine maintenance
- **Outcome:**
  - Updated, functional, and stable software over time

# Importance of SDLC

- **Ensures project clarity:** Each phase has clear objectives, making the development process structured and organized.
- **Minimizes risk:** Regular feedback and testing reduce the chances of major issues.
- **Improves efficiency:** Planning and designing ahead helps avoid unnecessary delays and budget overruns.
- **Enhances software quality:** By following a systematic process, the final product is more likely to meet the user's expectations and be of higher quality.

Thus SDLC provides a clear roadmap for delivering high-quality software in a predictable, controlled manner.

## 2. V-Model Vs Waterfall Model



# V-Model (Verification and Validation Model)



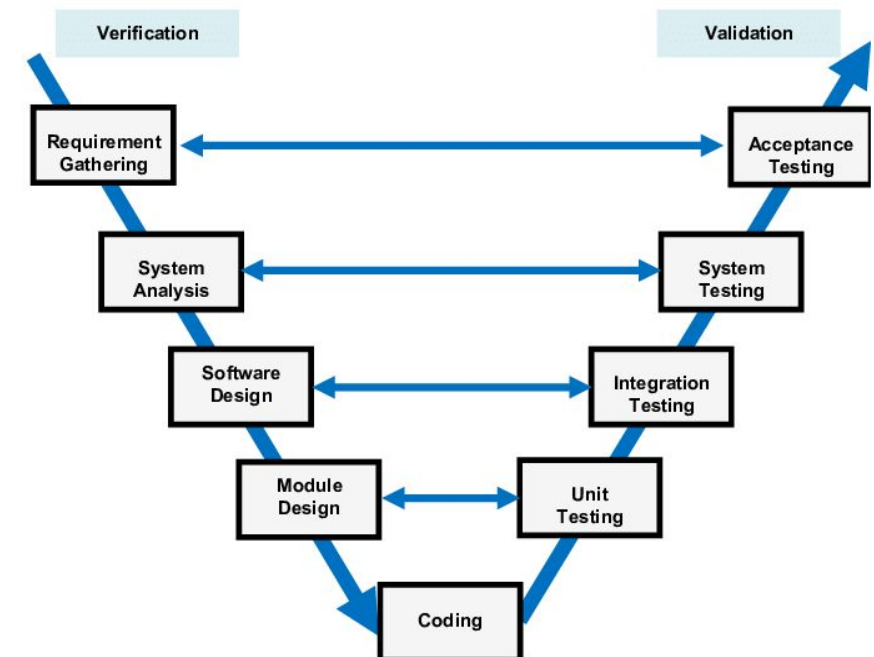
- A SDLC model that emphasizes a **sequential, step-by-step process** with a strong focus on validation and verification at each stage.
- It's often called an extension of the **Waterfall model**, but the key difference is that for each development stage in the V-Model, there is a corresponding testing phase, making it a more structured approach with continuous quality assurance.
- For eg., at Requirements Design Stage, the Acceptance criteria is defined, User Acceptance Test (UAT) cases are created, reviewed with the stake holder and a test plan is created. Each **requirement** from the Requirements Specification Document (SRS) is mapped to one or more acceptance test cases ensuring full **traceability** between requirements and the tests that will validate them.

# Key Characteristics of the V-Model

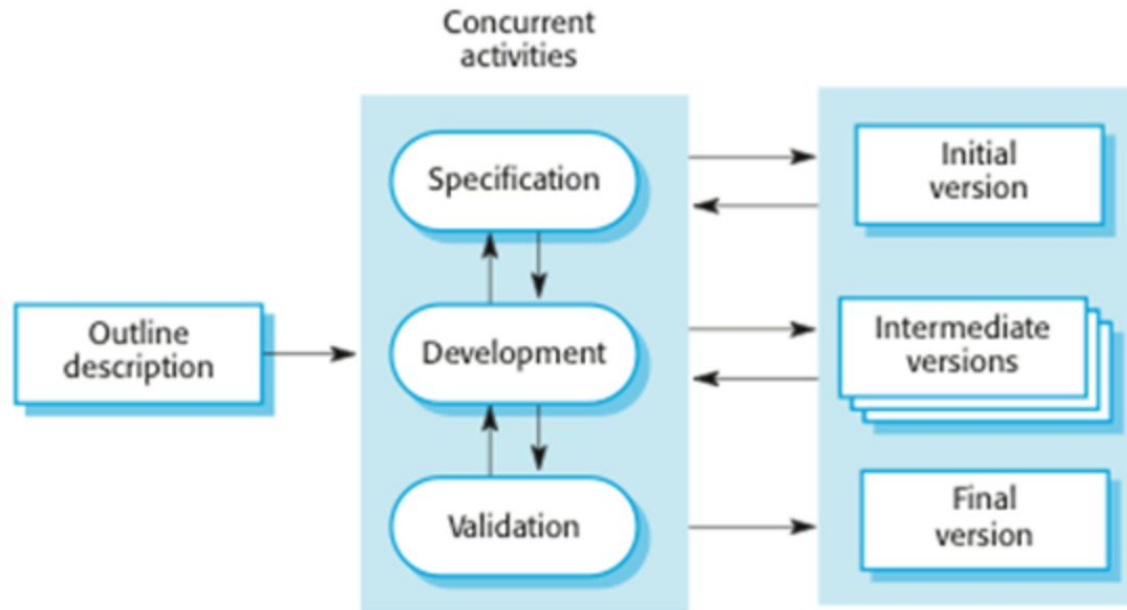
- **Sequential:** Like the Waterfall model, the V-Model follows a sequential process. Each phase must be completed before moving to the next.
- **Testing Phase for Every Development Stage:** Unlike the Waterfall model, testing is not deferred until after coding. For every development stage, there is a corresponding testing phase to ensure early defect detection
- **Verification and Validation Focus:** The V-Model splits the process into two tracks: verification (on the left side) and validation (on the right side), forming a "V" shape.

# Verification vs. Validation

- **Verification:** Ensures that the product is being built correctly (aligned with the requirements and design specifications). This happens through reviews, design inspections, and testing throughout the development stages
- **Validation:** Ensures that the product meets the user's needs and expectations. This is primarily done through various testing stages on the right side of the "V."

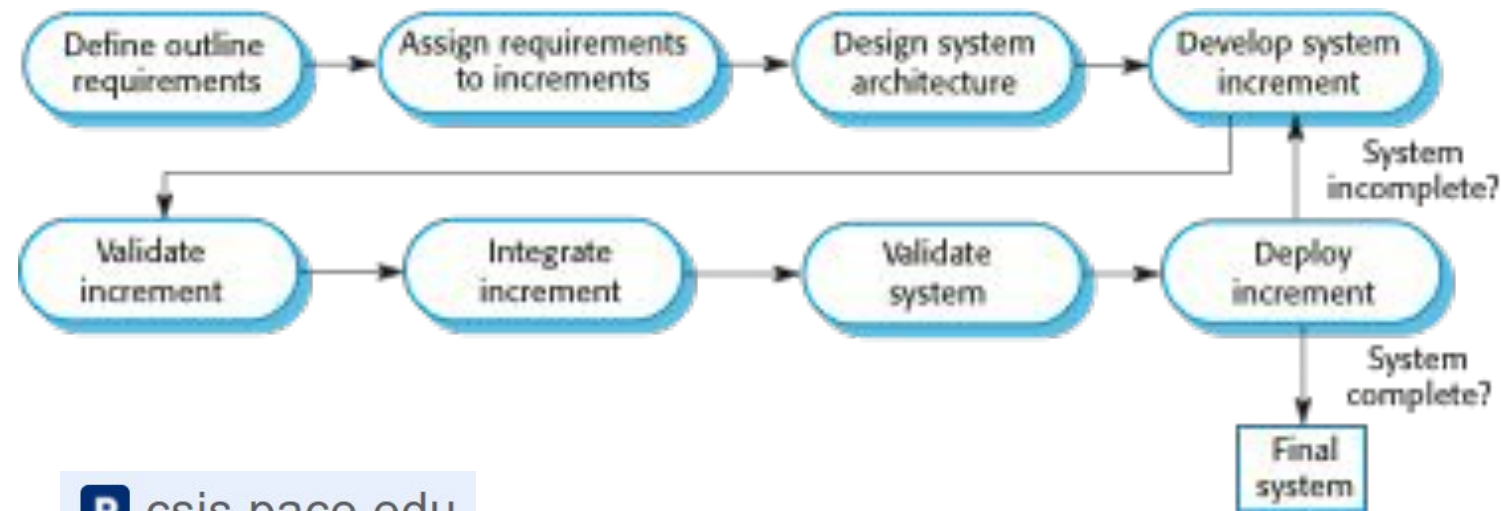


# Incremental development



 [csis.pace.edu](http://csis.pace.edu)

# Incremental delivery



 [csis.pace.edu](https://csis.pace.edu)

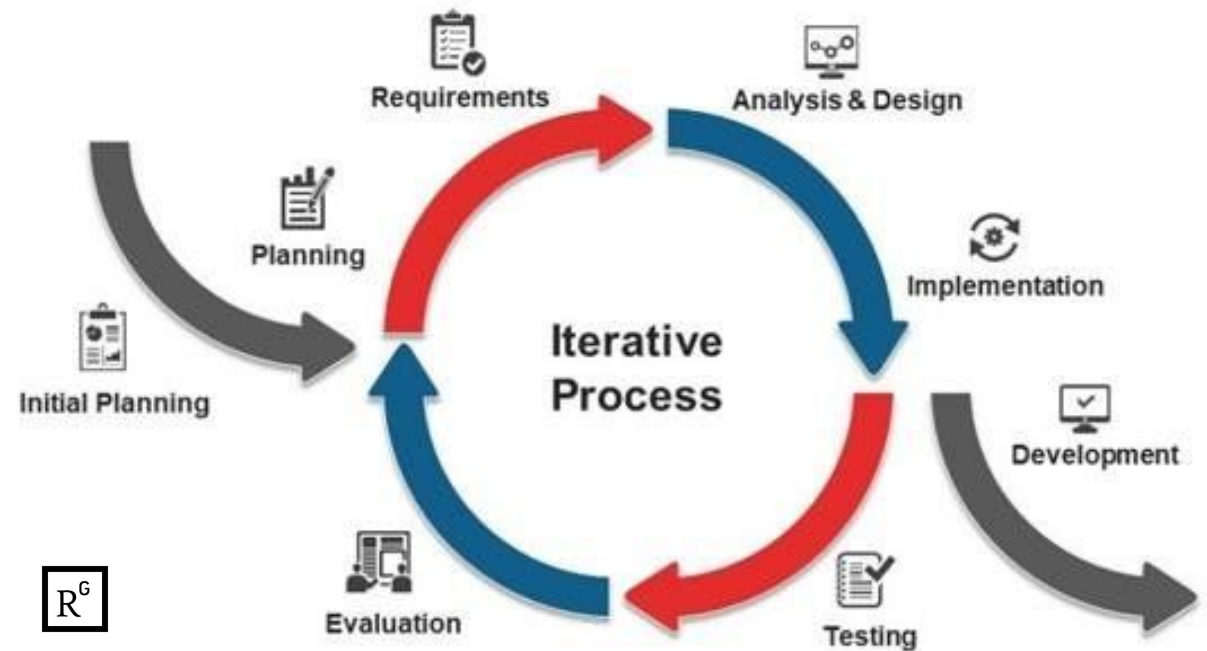
# Agile (or Iterative Model)



- The agile model is an iterative approach to software development that focuses on continuous releases and incorporating customer feedback.
- It allows for more flexibility and adaptability during the development process, making it better suited for projects with changing requirements or uncertain outcomes.

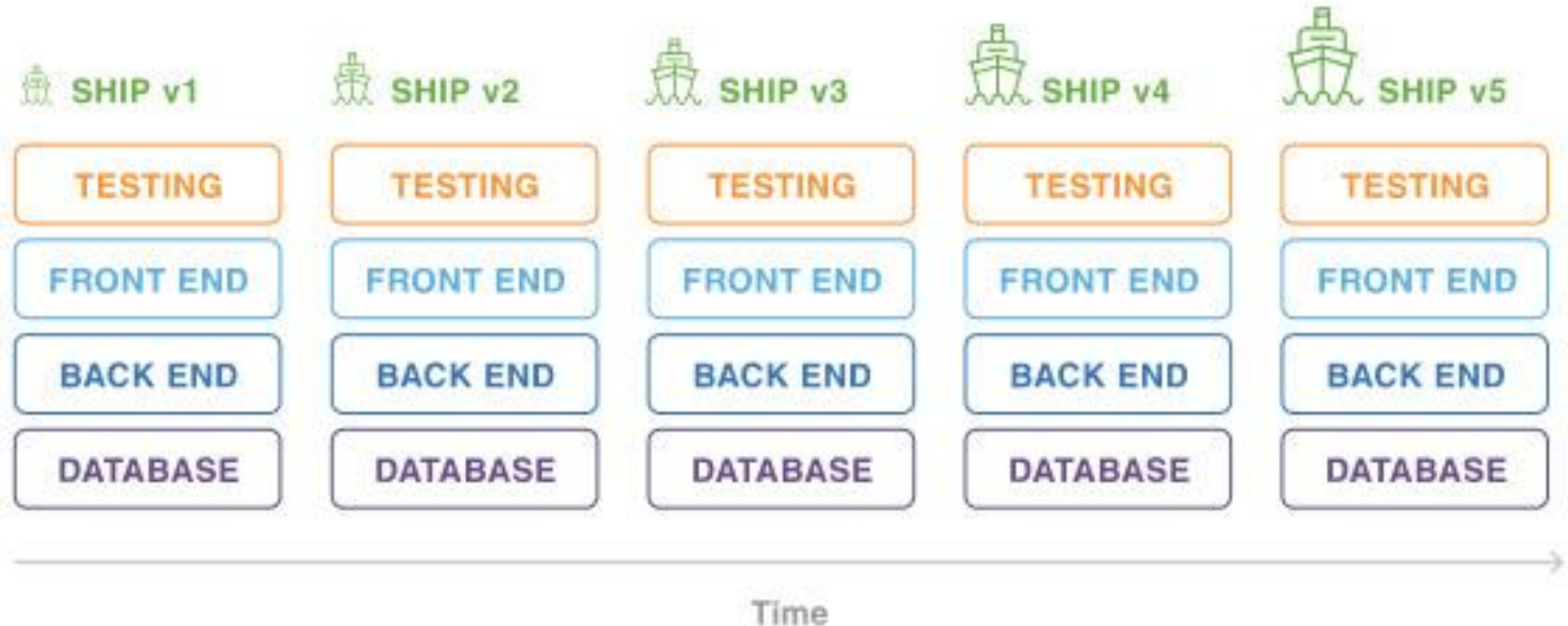
# 3. Iterative Model

**Develops the  
system  
incrementally  
through repeated  
cycles (iterations)**



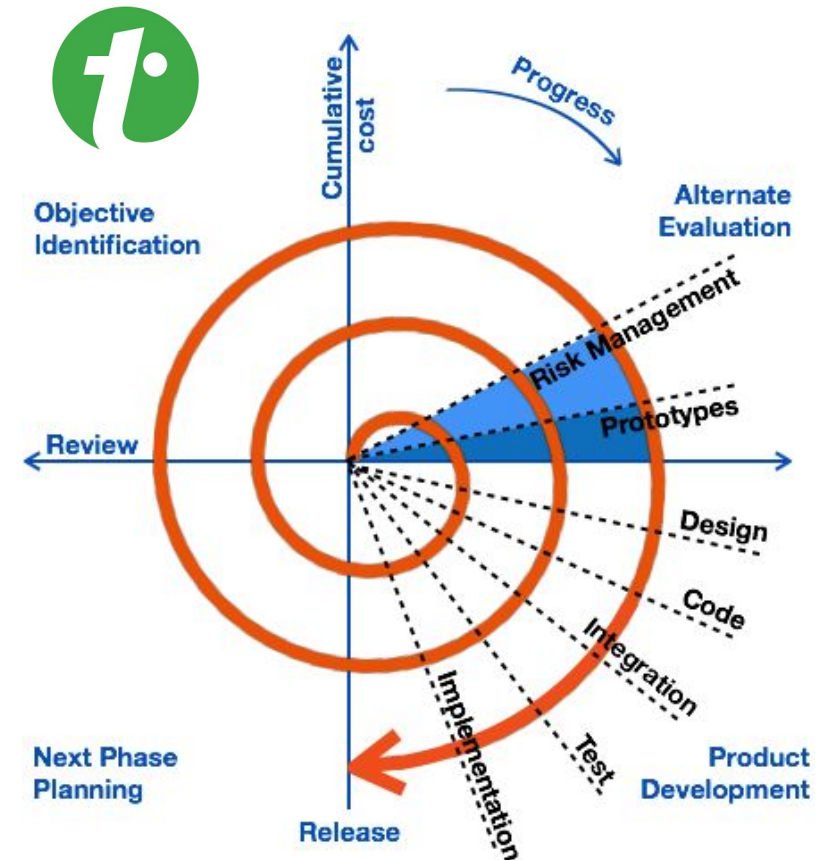
R<sup>6</sup>

# Iterative Model (Illustration)



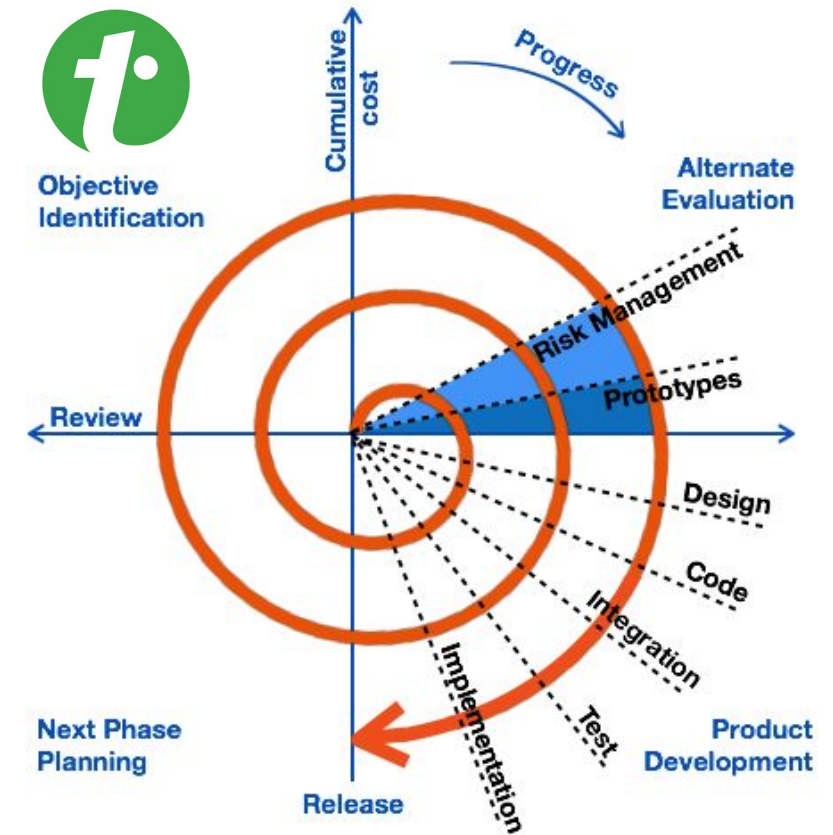
# 4. Spiral Model

- A unique model that combines iterative development with sequential model taking a risk-driven approach,
- Introduced by Barry Boehm in 1986 to address the shortcomings of other SDLC models, particularly the Waterfall and Iterative models.
- The Spiral Model is well-suited for large, complex, and high-risk projects because
- it emphasizes risk management throughout the software development process.

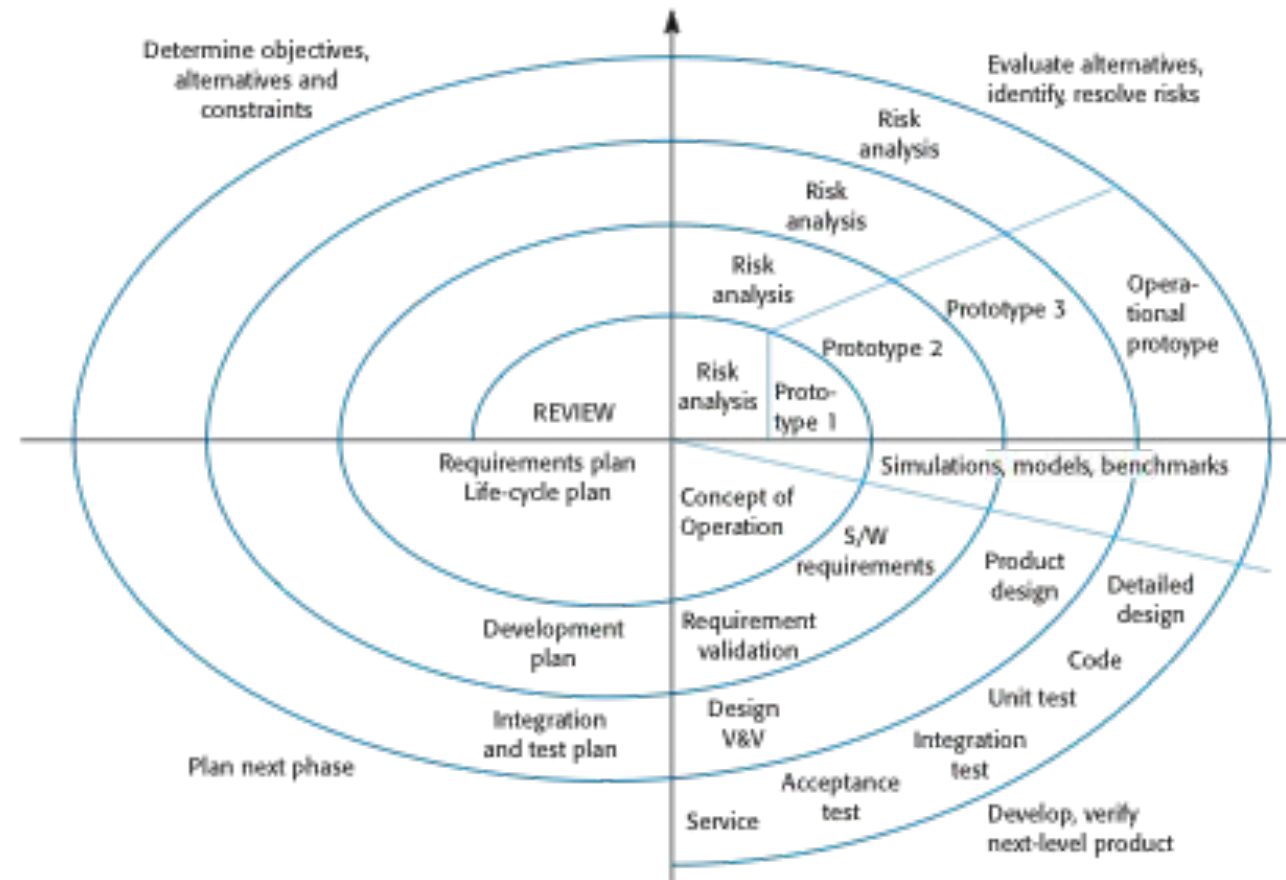


# Spiral Model - Contd.

- The spiral model has four phases:
  1. Identification
  2. Design
  3. Construct / Build
  4. Evaluation & Risk Analysis
- Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer

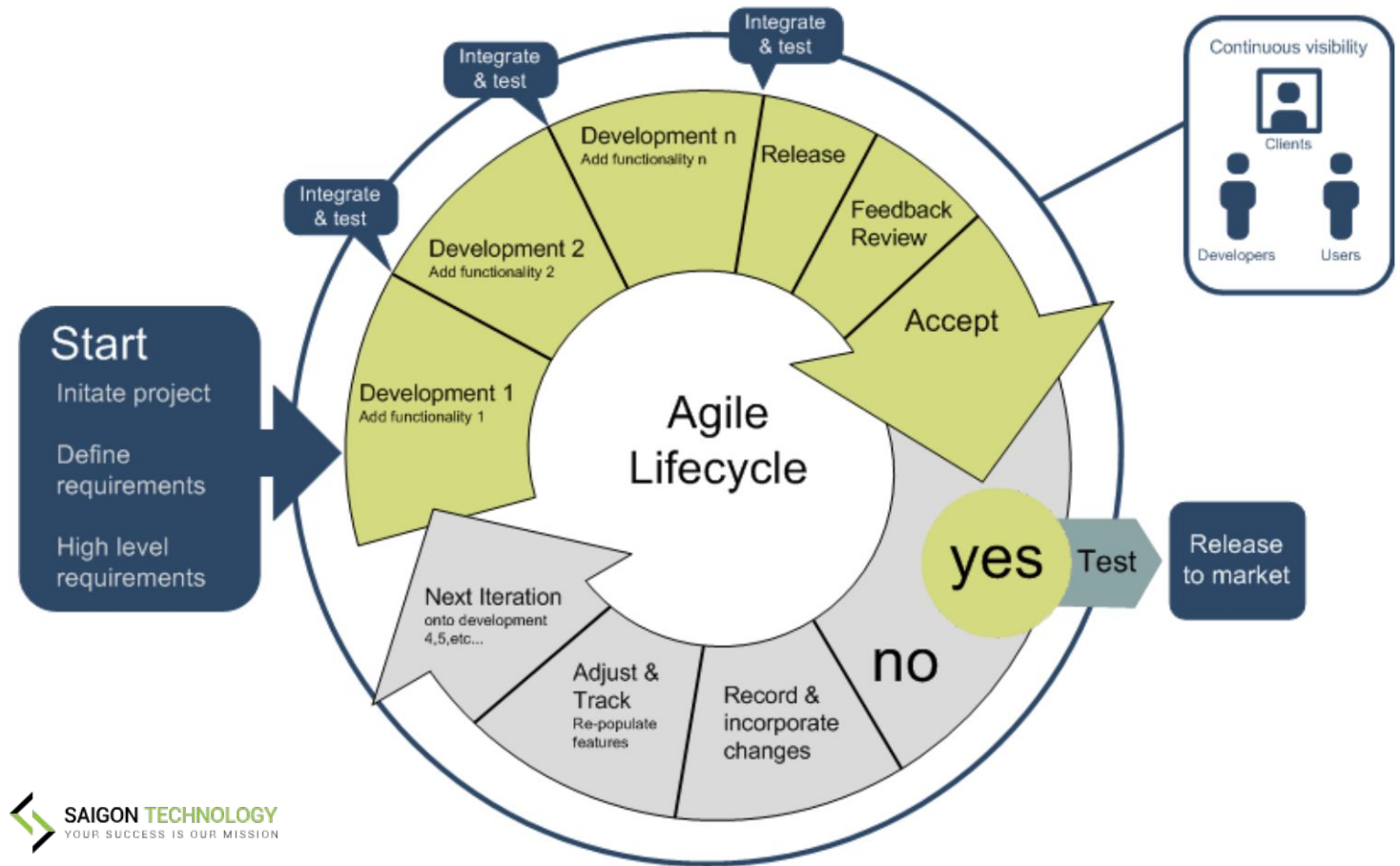


# Boehm's spiral model of the software process



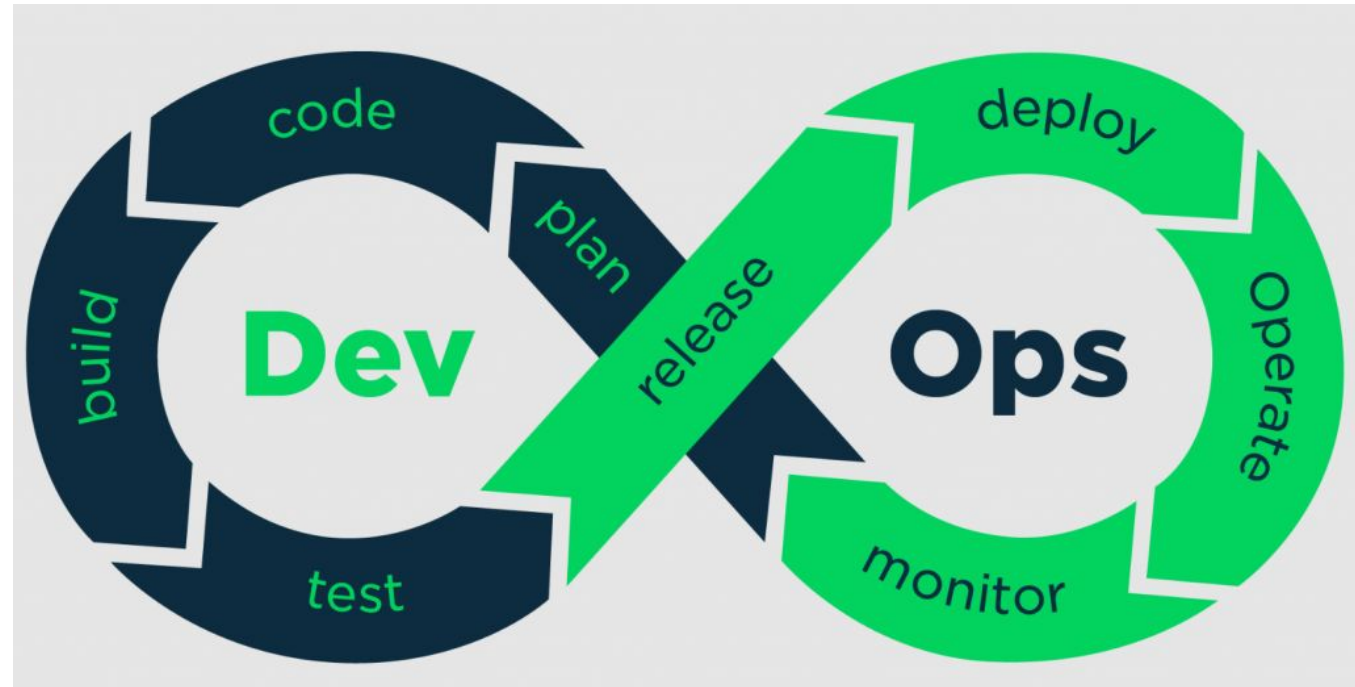
# 5. Agile Model

An iterative and incremental approach where the project is divided into small, manageable chunks (sprints) for continuous feedback and improvement.



# DevOps

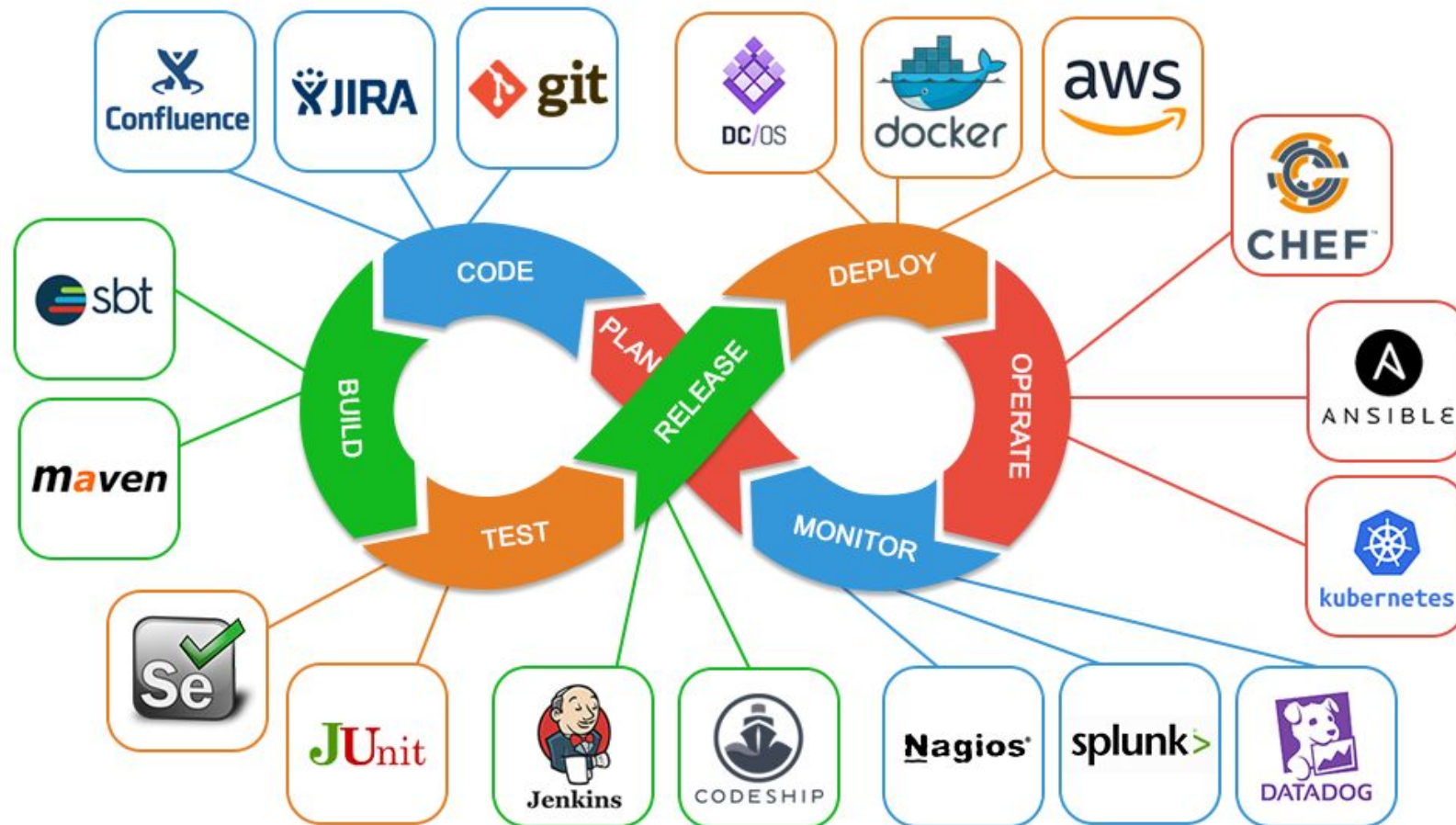
Focuses on **collaboration** between development and operations teams, emphasizing continuous integration, continuous deployment (CI/CD), and automation.



# What is DevOps

- DevOps is a compound of development (Dev) and operations (Ops). It is the union of people, process, and technology to continually provide value to customers. DevOps outlines a software development process and an organizational culture shift that speeds the delivery of higher quality software by automating and integrating the efforts of development and IT operations teams – two groups that traditionally practiced separately from each other, or in silos
- DevOps enables formerly siloed roles—development, IT operations, quality engineering, and security—to coordinate and collaborate to produce better, more reliable products. By adopting a DevOps culture along with DevOps practices and tools, teams gain the ability to better respond to customer needs, increase confidence in the applications they build, and achieve business goals faster

# DevOps Tools



# The Agile manifesto

- A foundational document that outlines the principles and values of Agile software development.
- Created in 2001 by a group of 17 software developers who sought a more flexible, collaborative, and efficient approach to delivering software than the traditional, rigid models such as Waterfall.
- Consists of two main components:
  - Four Core Values
  - Twelve Principles



# 4 Core Values of Agile manifesto

## **Individuals and Interactions over Processes and Tools:**

- Agile emphasises the importance of people working together effectively, prioritising communication and collaboration over strict adherence to processes and tools. Tools are useful, but they should not limit the ability of team members to collaborate and respond to challenges.

## **Working Software over Comprehensive Documentation:**

- While documentation is important, the priority in Agile is delivering software that works and meets customer needs. Agile favors minimal, necessary documentation that supports development rather than exhaustive specifications that might become outdated or irrelevant.

## 4 Core Values of Agile manifesto (contd-)

### **Customer Collaboration over Contract Negotiation:**

- Agile values ongoing collaboration with customers throughout the project rather than relying solely on a fixed contract. This allows for continuous feedback, ensuring the software aligns with changing requirements and user needs.

### **Responding to Change over Following a Plan:**

- Agile recognises that requirements evolve, and customer needs may shift over time. Rather than sticking rigidly to a predefined plan, Agile encourages teams to embrace change and adjust their approach as necessary to deliver value to the customer.

# 12 Principles of Agile

## **Customer satisfaction through early and continuous delivery of valuable software**

- The highest priority is to satisfy the customer by delivering functional software as soon as possible and continuously throughout the project.

## **Welcome changing requirements, even late in development.**

- Agile processes accommodate changes, even late in development, as they often lead to a better product that meets the customer's needs.

## **Deliver working software frequently, with a preference for shorter timescales.**

- Agile teams aim to deliver working software in short cycles (e.g., every 2-4 weeks), allowing for continuous feedback and faster delivery.

# 12 Principles of Agile - contd.

**Business people and developers must work together daily throughout the project.**

- Collaboration between business stakeholders and developers ensures alignment and understanding of project goals, fostering better outcomes.

**Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.**

- Agile emphasises the importance of self-organised, motivated teams that take ownership of their work.

**The most efficient and effective method of conveying information is face-to-face conversation.**

- While written communication is important, face-to-face conversation (or its virtual equivalent) is the most effective way to ensure clarity and resolve issues quickly.

# 12 Principles of Agile - contd.

**Working software is the primary measure of progress.**

- The success of an Agile project is measured by the delivery of working software that meets customer requirements, not by the completion of specific tasks or documents.

**Agile processes promote sustainable development.**

- Agile advocates for a pace that allows for consistent progress without burnout, ensuring the team can continue delivering value indefinitely.

**Continuous attention to technical excellence and good design enhances agility.**

- High-quality code and robust design practices lead to a more adaptable and maintainable system, making it easier to respond to changes.

# 12 Principles of Agile - contd.

**Simplicity—the art of maximizing the amount of work not done—is essential.**

- Agile encourages teams to focus on what is essential and avoid unnecessary complexity, features, or tasks that do not add value.

**The best architectures, requirements, and designs emerge from self-organizing teams.**

- Teams that are empowered to make decisions and collaborate effectively are better able to develop optimal solutions.

**At regular intervals, the team reflects on how to become more effective and adjusts accordingly**

- Agile teams continuously improve their processes by regularly holding retrospectives to identify and implement improvements.

# Comparison between Agile & Traditional methods

## -1

Aspect	Agile Methodology	Traditional Methodology(e.g.,Waterfall)
Approach	Iterative & Incremental	Linear & Sequential
Project Structure	Flexible, adaptive to change	Rigid, predefined phases
Customer Involvement	High, continuous collaboration with customers	Low, limited to the initial and final stages
Development Cycles	Short sprints (2-4 weeks), frequent releases	Single development cycle with one major release

# Comparison between Agile & Traditional methods

## -2

Aspect	Agile Methodology	Traditional Methodology(e.g.,Waterfall)
Planning	Adaptive, focuses on short-term goals	Upfront, extensive planning for the entire project
Requirement Handling	Evolving, flexible requirements	Fixed at the beginning of the project
Documentation	Lightweight, focuses on working software	Heavy, extensive documentation
Risk Management	Continuous risk assessment during iterations	Risk is assessed early, less focus later

# Comparison between Agile & Traditional methods

-3

Aspect	Agile Methodology	Traditional Methodology(e.g.,Waterfall)
Testing	Continuous testing throughout development	Testing occurs after the development phase
Delivery	Frequent, incremental deliveries (working software)	Delivered only at the end of the project
Team structure	Cross-functional, self-organized teams	Hierarchical, with defined roles and responsibilities
Response to Change	Embraces change, even late in development	Change is discouraged or requires re-planning
Feedback	Continuous feedback from stakeholders	Feedback is often received after project completion

# Key differences between Agile & Traditional methods

- **Approach**

- **Agile** is highly **iterative and incremental**. Projects are broken into small, manageable cycles (sprints), with each iteration delivering a functional part of the software.
- **Traditional methods**, like **Waterfall**, follow a **linear and sequential** approach. The project progresses through predefined stages (e.g., planning, design, development, testing) with each stage needing to be completed before moving to the next.

## Key differences between Agile & Traditional methods (2)



- **Flexibility and Adaptability**

- **Agile** is designed to be flexible, with the expectation that requirements may change even during development. Teams adapt to these changes through regular reviews, re-prioritisation, and continuous feedback
- **Traditional methods** tend to be more rigid, assuming that requirements are fixed from the start. Changes are more costly, as they often require going back to previous phases.

# Key differences between Agile & Traditional methods (3)



- **Customer Collaboration**
  - Agile is designed to be flexible, with the expectation that requirements may change even during development. Teams adapt to these changes through regular reviews, re-prioritisation, and continuous feedback.
  - In traditional methods, customer involvement is typically higher at the beginning (during the requirements phase) and at the end (during product delivery), which can lead to a gap between what is built and what the customer really needs.

# Key differences between Agile & Traditional methods (4)

- **Delivery Cycle:**
  - Agile delivers working software frequently—after each sprint or iteration. This allows stakeholders to see progress and use parts of the software early in the process.
  - Traditional models deliver the software only once, at the end of the project. This can result in delays if the product doesn't meet user expectations or has issues that weren't discovered until later stages.

# Key differences between Agile & Traditional methods (5)



- **Risk Management:**

- In Agile, risks are continuously assessed as the product evolves. This incremental approach reduces overall project risk since issues can be identified and resolved early.
- In traditional models, risks are assessed mostly during the initial stages. If risks are not properly identified, they may lead to problems later in the development cycle, often when it is costly or difficult to address them.

# Key differences between Agile & Traditional methods (6)

- **Documentation:**

- Agile emphasizes working software over comprehensive documentation. While documentation is still important, it is kept lightweight to focus more on delivering functional products.
- Traditional methods rely heavily on detailed documentation at every stage. This extensive paperwork is often time-consuming and can delay progress.

# Key differences between Agile & Traditional methods (7)



- **Testing:**

- **In Agile**, testing is integrated into every iteration, ensuring continuous quality checks as features are built.
- **Traditional methods** usually perform testing after development is complete. This can lead to the discovery of bugs late in the project, which can be expensive and time-consuming to fix.

# Process tailoring



**Process tailoring** refers to adapting and customising a development methodology to fit the specific needs of a project or organisation.

It is particularly relevant when choosing between **Agile** and **Traditional** methods or combining elements from both to create a hybrid approach.

**Process tailoring** allows teams to maximise the effectiveness of their chosen methodology and to ensure that it aligns with the unique needs of each project.

**Improving efficiency** by tailoring processes to optimize team performance and reduce unnecessary steps.

**Scaling Agile** for larger projects by using frameworks like SAFe.

# Importance of Process tailoring

- **Project uniqueness:** Each project has different requirements, risks, team compositions, and constraints. Tailoring allows teams to align processes with the project's specific demands.
- **Hybrid approaches:** In some cases, projects may benefit from a mix of Agile and Traditional practices, depending on project phases or teams. For example, an organisation may use a Waterfall approach for high-level planning and documentation but adopt Agile for development and testing phases.
- **Efficiency:** Tailoring optimises processes, ensuring that teams focus on what's most valuable to their specific situation rather than following a one-size-fits-all approach.

# Process tailoring Examples

- **Scaling Agile for large projects:** In large projects, Agile processes may need scaling frameworks such as **SAFe (Scaled Agile Framework)**
- **Blending Agile and Waterfall (Hybrid):** Some organisations adopt a hybrid model, where Agile is used for development while Waterfall principles are applied for planning and high-level design.
- **Adjusting sprint length:** Depending on the project, sprints in Agile may be tailored to last longer (e.g., 4-6 weeks) or shorter (e.g., 1 week) based on team capacity and complexity.
- **Risk Management in Agile:** For projects with high risks, Agile may incorporate more detailed upfront planning or risk analysis, blending some aspects of Traditional methodologies

# Process tailoring Summary

- **Customization** of methodology to fit specific project requirements and constraints.
- **Blending methods** such as combining Waterfall planning with Agile execution.
- **Improving efficiency** by tailoring processes to optimize team performance and reduce unnecessary steps.
- **Scaling Agile** for larger projects by using frameworks like SAFe

# OOAD and UML

# What is OOAD?

- OOAD stands for Object-Oriented Analysis and Design
- It is a software engineering methodology that focuses on the analysis of a problem domain and the design of a solution using object-oriented principles
- OOAD is used to identify the objects, classes, and relationships that are relevant to the problem domain, and to model the behavior of the system using these elements

# What is OOAD?



- OOAD typically involves several stages, including requirements gathering, analysis, design, implementation, and testing
- During the analysis stage, developers use techniques such as use case analysis and class-responsibility-collaboration (CRC) cards to identify the objects and classes that are relevant to the problem domain
- During the design stage, developers use tools such as UML (Unified Modeling Language) diagrams to represent the relationships between these objects and classes and to model the behavior of the system.

# Why OOAD?

- **Reusability:** Emphasizes the use of reusable components and design patterns, which can save time and effort in software development
- **Scalability:** Helps developers design software systems that are scalable and can handle changes in user demand and business requirements over time
- **Maintainability:** Emphasizes modular design and can help developers create software systems that are easier to maintain and update over time
- **Flexibility:** Helps developers design software systems that are flexible and can adapt to changing business requirements over time

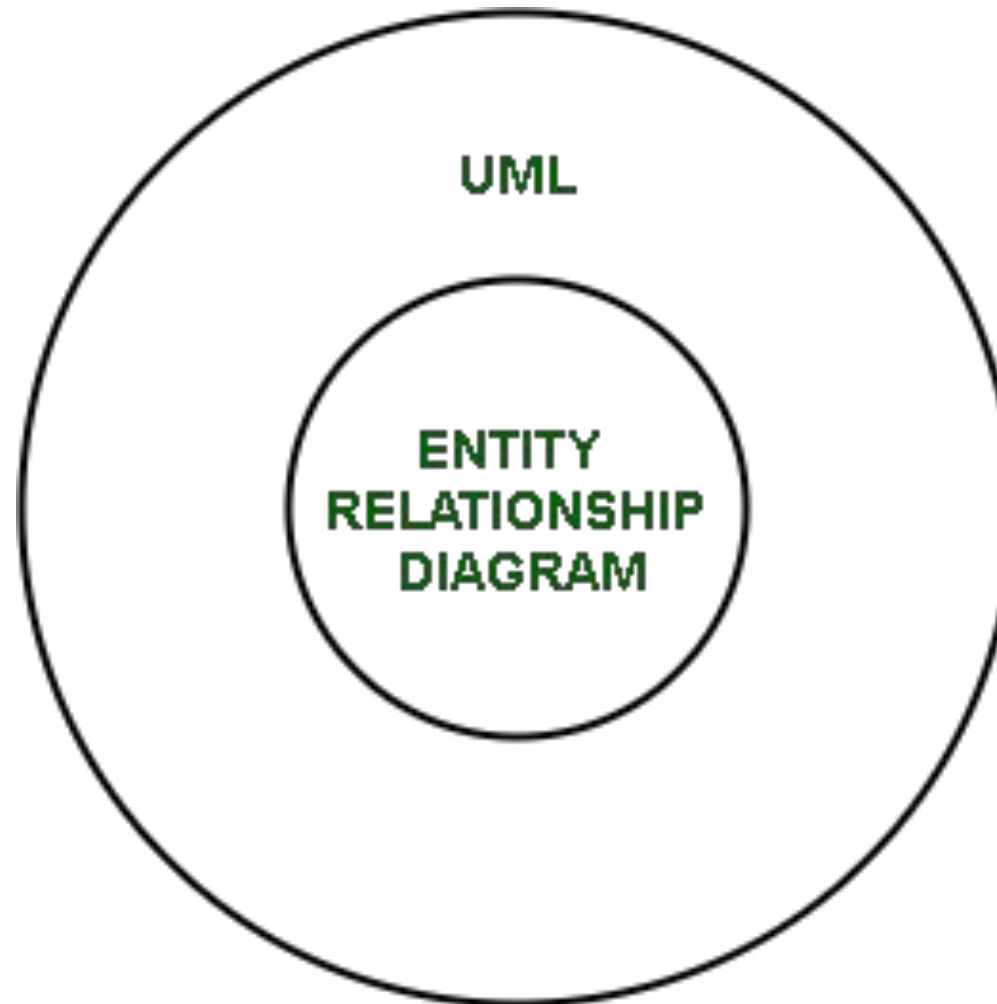
# What is UML



- UML, on the other hand, is a graphical language for OOAD (Object Oriented Analysis and Design) that provides a standard way to write a software system's blueprint
- It helps to visualize, specify, construct, and document the artifacts of an object-oriented system

# Why UML?

- **Improved communication:** UML provides a standardized notation for creating diagrams that represent different aspects of a software system. This can improve communication between team members and stakeholders by providing a common visual language for discussing the design of the system<sup>2</sup>.
- **Better documentation:** UML diagrams can serve as a form of documentation for the design of a software system. This can make it easier for developers to understand and maintain the system over time<sup>2</sup>.
- **Easier problem-solving:** UML diagrams can help developers identify problems and inconsistencies in the design of a software system. This can make it easier to find and fix issues before they become major problems<sup>2</sup>.



# UML vs ER

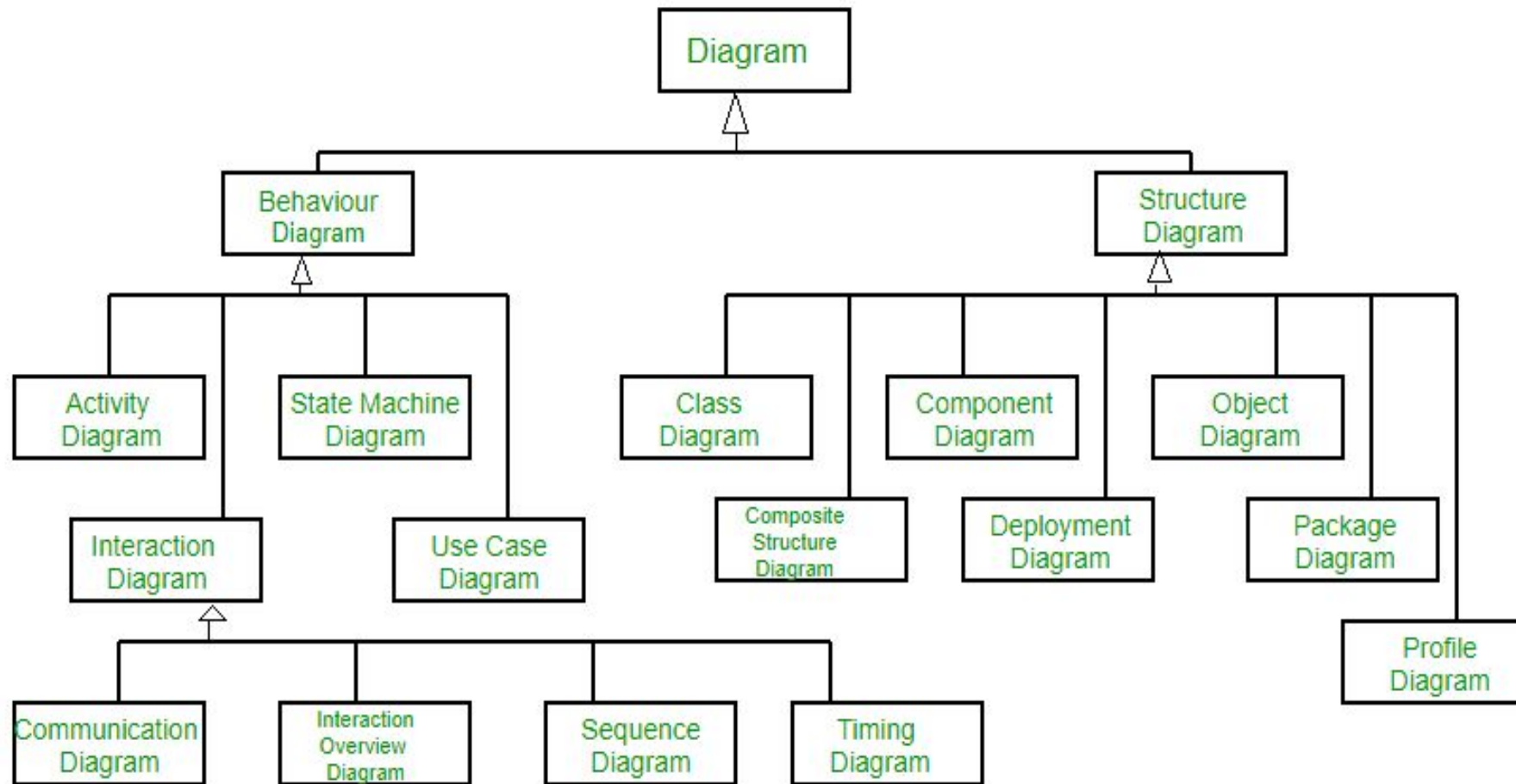
## UML

- UML stands for Unified Modelling Language
- It is parent of ER
- It is used to design the entire software
- It has use cases and workflows

## ER

- ER Diagram stands for Entity Relationship Diagram
- It is child of UML
- It is used to design only the databases
- It has entities, attributes and relationships

# UML Example



# How OOAD Methodology Works



- **Analyze the problem domain:** Start by analyzing the problem domain and identifying the requirements of the software system. This will help you understand the scope of the project and the features that need to be implemented.
- **Identify objects and classes:** Use OOAD principles to identify the relevant objects and classes in the problem domain. Think about the real-world entities that the software system needs to represent and how they interact with each other.
- **Model the behavior of the system:** Once you have identified the objects and classes, use OOAD to model the behavior of the system. Think about the actions that users can perform and how the system should respond.
- **Create UML diagrams:** Use UML to create diagrams that represent the design of the system. These diagrams could include class diagrams, sequence diagrams, use case diagrams, and others. Use these diagrams to visualize the relationships between objects and classes, and to document the behavior of the system.
- **Implement and test:** Once you have completed your analysis and design, you can start implementing your software project. Use an object-oriented programming language to write code that implements the features you have designed. Test your code to ensure that it meets the requirements of the project.

# UML



**Inheritance**



**Aggregation**



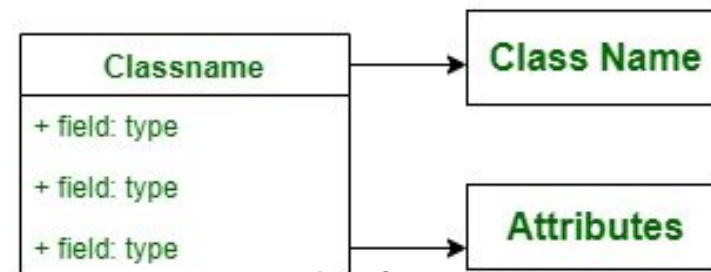
**Composition**



**Association**



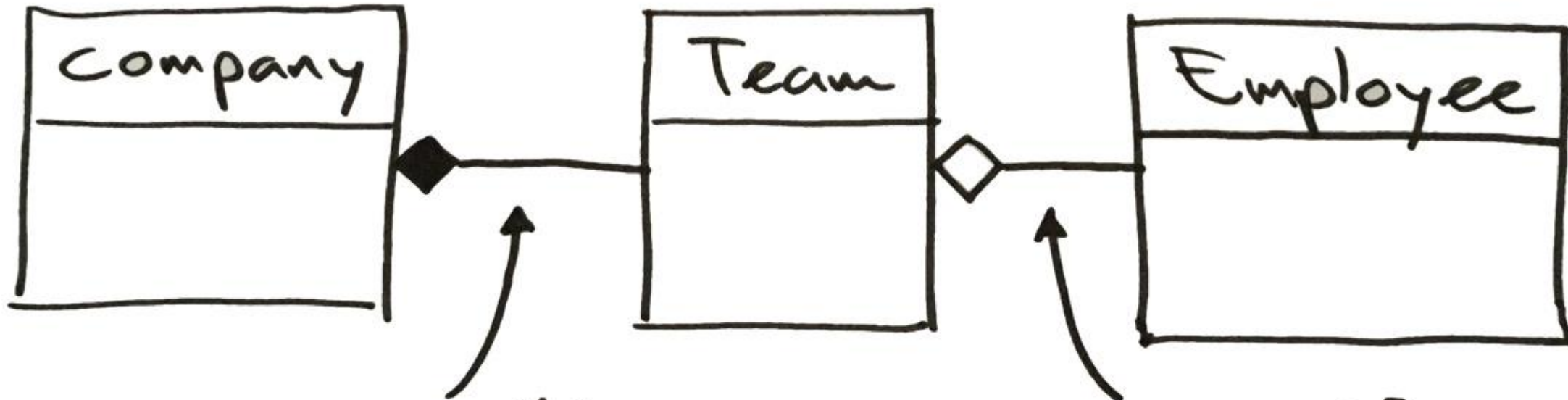
**Unidirectional  
Association**



1 company

multiple teams

multiple employees



composition:

Team can not exist  
without company  
company "owns" team

aggregation:

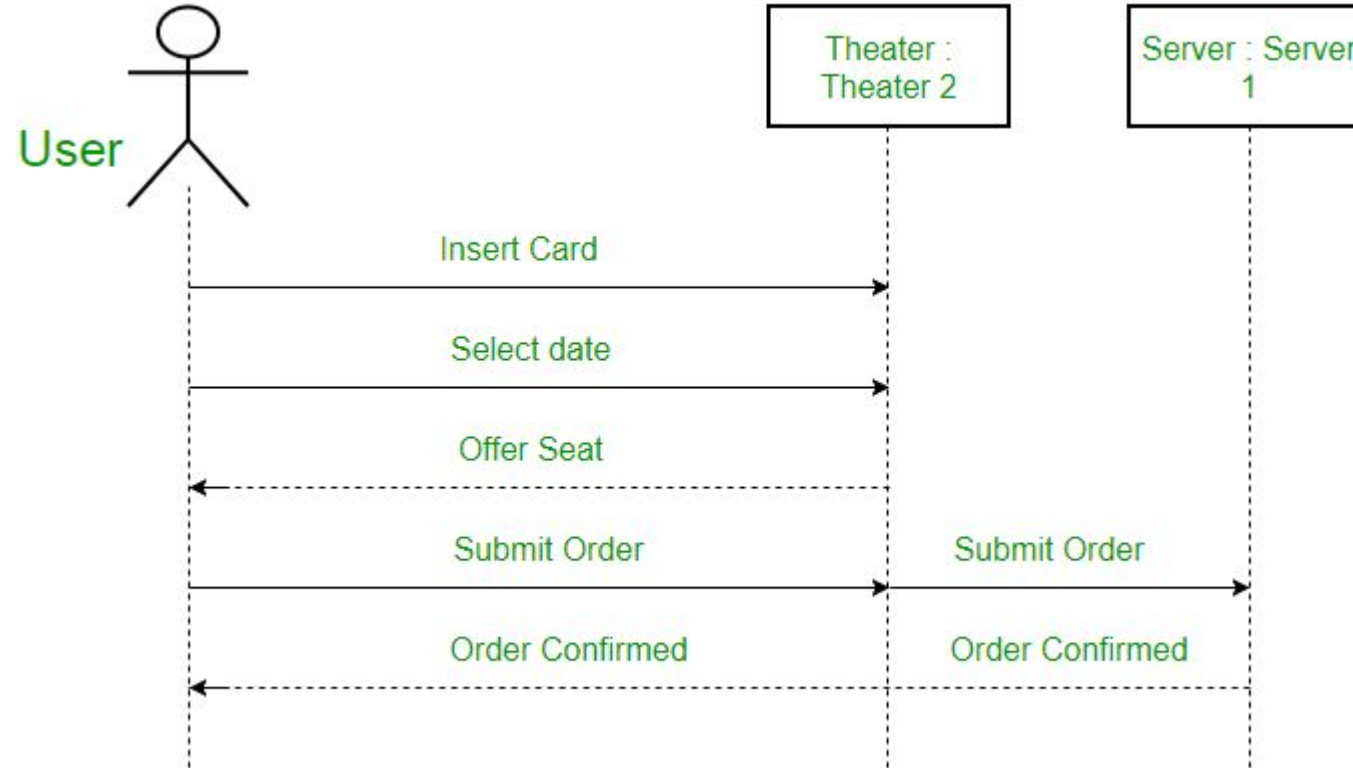
throw away the  
team and the employee  
can be connected to another  
team

# Aggregation

- In an aggregation relationship, the contained classes can exist independently of the container class
- To show aggregation in a diagram, you can draw a line from the parent class to the child class with a diamond shape near the parent class
- **Library and Books:** A library is made up of one or more books, among other materials. In aggregation, the contained classes (books) are not strongly dependent on the lifecycle of the container (library). Even when the library is dissolved, the books will still remain
- **Pond and Ducks:** A pond can have zero or more ducks, and a duck can have at most one pond at a time. A duck can exist separately from a pond, for example, it can live near a lake. When a pond is destroyed, it usually does not kill all the ducks

# Composition

- In a composition relationship, the contained classes cannot exist independently of the container class
- To show composition in a diagram, you can draw a line from the parent class to the child class with a filled diamond shape near the parent class
- **Car and Carburetor:** A car has exactly one carburetor, and a carburetor is a part of one car. Carburetors cannot exist as separate parts, detached from a specific car
- **House and Rooms:** A house is made up of one or more rooms. The rooms cannot exist independently of the house. If the house is destroyed, the rooms will also be destroyed



# Activity Diagram

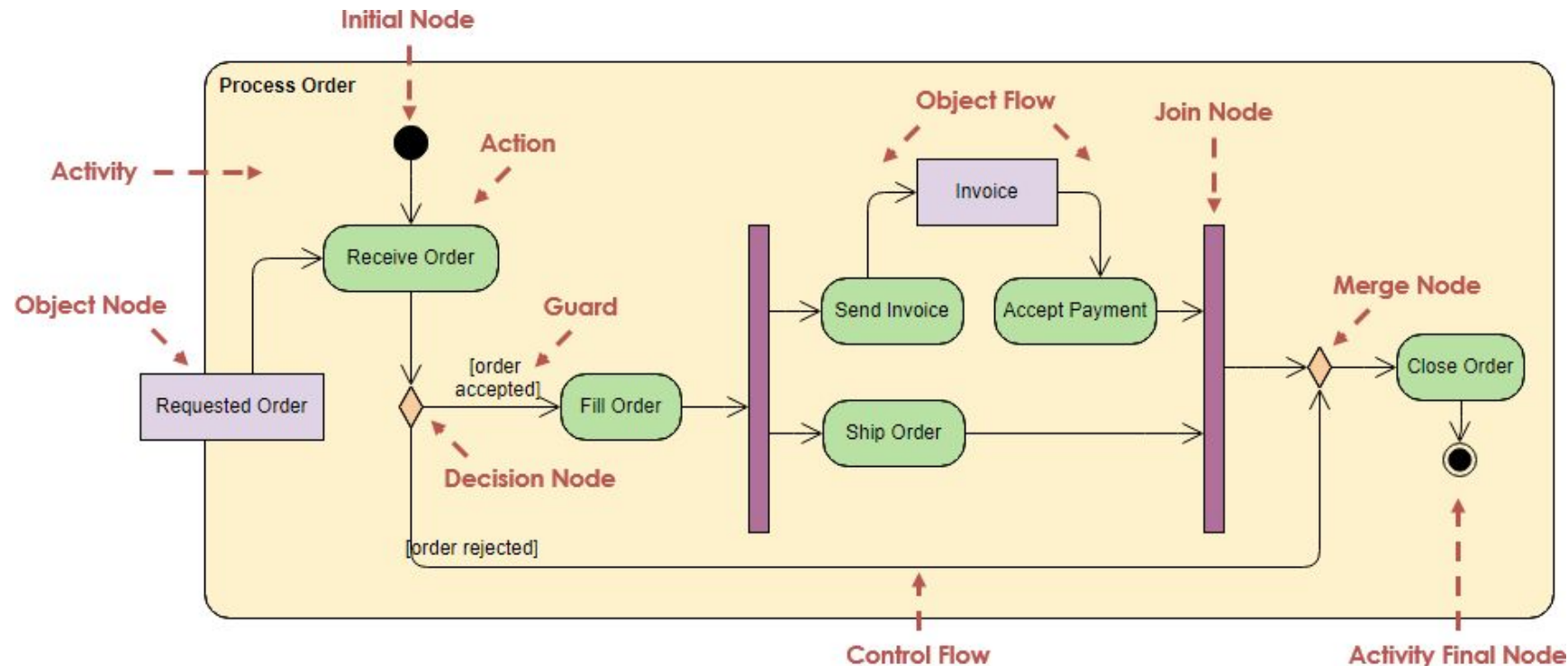
- An activity diagram is a type of interaction diagram that shows the flow of control in a system. It visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram
- Activity diagrams are often used in business process modeling and can describe the steps in a use case diagram. Activities modeled can be sequential and concurrent



# Activity Diagram

[Activity Diagram Tutorial \(visual-paradigm.com\)](http://visual-paradigm.com)

**Order Processing:** An example of an activity diagram showing the business flow activity of order processing is given below. The input parameter is the Requested order, and once the order is accepted, all of the required information is then filled, payment is also accepted, and then the order is shipped

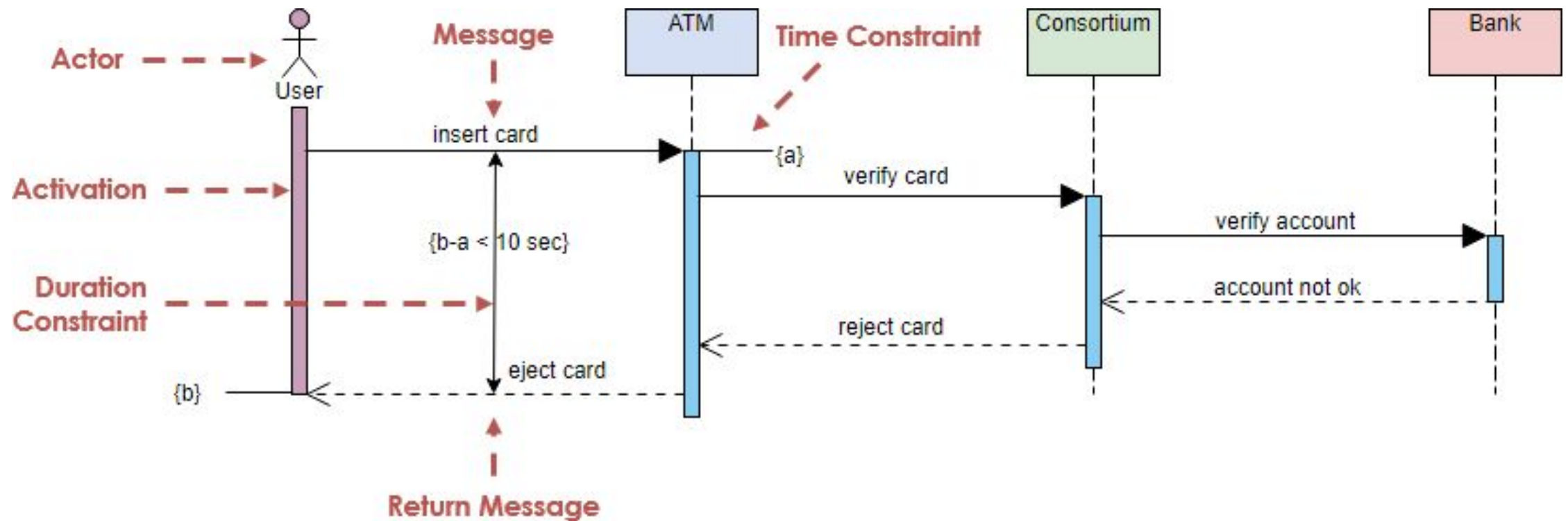


# Sequence Diagram

- A sequence diagram is a type of interaction diagram that shows how objects and components interact with each other to complete a process
- It represents the interactions between objects in a single use case and illustrates how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed

# Sequence Diagram

[Sequence Diagram Tutorial \(visual-paradigm.com\)](http://visual-paradigm.com)

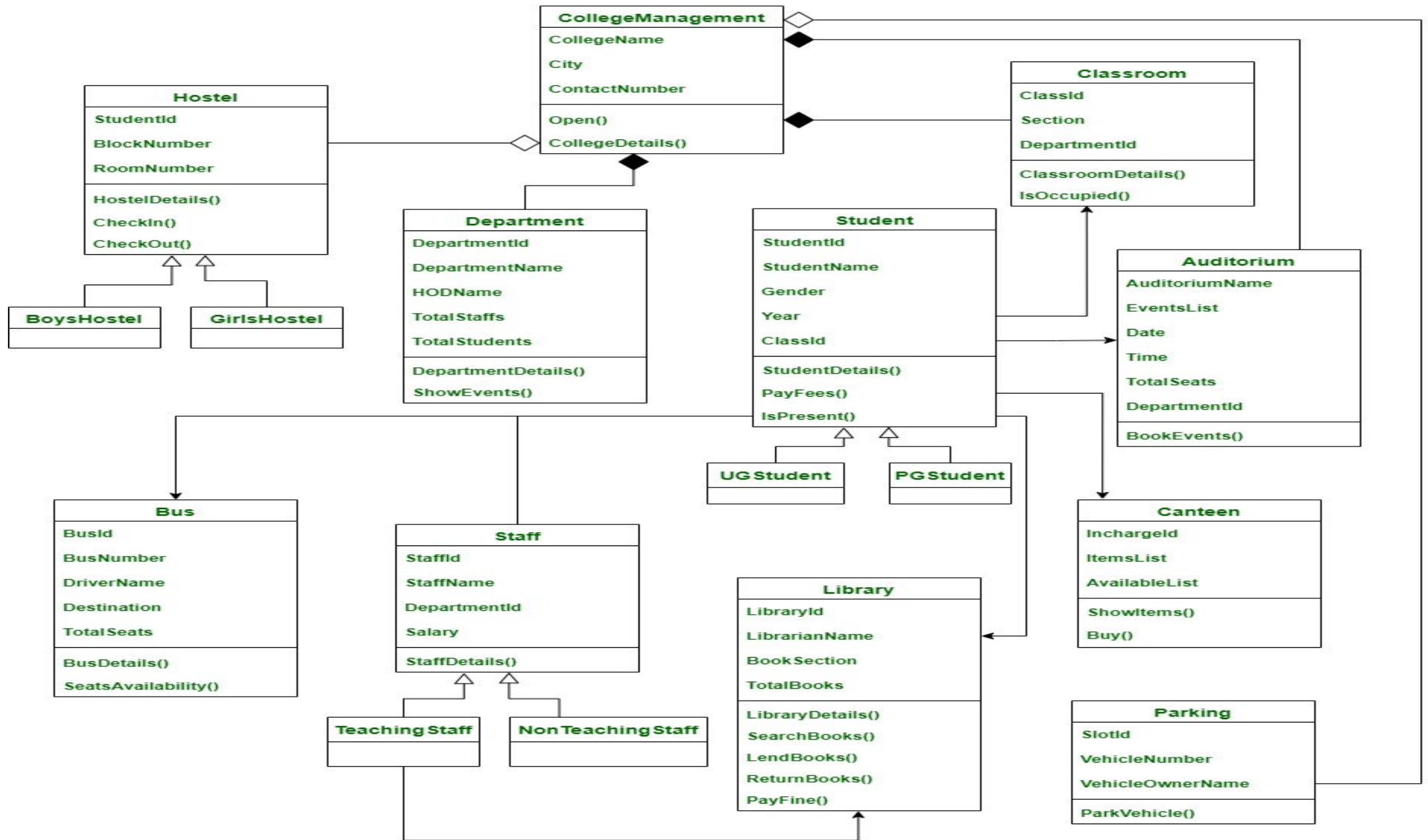


# UML Example (Dice Game)

1. Define Use Case
2. Define Domain Model
3. Define Interaction Diagrams
4. Define Design Class Diagrams

# Class in UML

ClassName
attribute1: type attribute2: type
method1(parameter1 : type) : returnType method2(parameter1 : type) : returnType



# Mini Project Activity



- Each team is assigned with software project
- They will need to submit and present (team lead) the following
  - Class Diagram
  - Activity Diagram
  - Sequence Diagram

## In Class Activity – How to write Product Requirements



- Make Groups consisting of 4 Students (total of 15-16 Groups to be formed)
- Allocate software projects to each group
- Group need to document – Top 5-7 Product Requirements
- Review and Feedback

## Sample topics

1. Online Learning Platform
2. E-commerce Website
3. Mobile Banking App
4. Social Media Platform
5. Fitness Tracking Application
6. Smart Home Automation System
7. Healthcare Management System
8. Travel Booking Website
9. Food Delivery Service App
10. Ride-Sharing Application
11. Project Management Tool
12. Online Grocery Shopping Platform
13. Virtual Event Management System
14. Music Streaming Service
15. Remote Work Collaboration Tool

# Sample topics (ML & DL )

1. **Predicting House Prices:** Using a dataset like the Ames Housing Dataset, students can build a model to predict house prices based on various features like square footage, number of bedrooms, and location.
2. **Sentiment Analysis:** Develop a sentiment analysis model that can classify the sentiment of tweets or product reviews as positive, negative, or neutral.
3. **Image Classification:** Create a model that can classify images into categories (e.g., cats vs. dogs) using a dataset like CIFAR-10 or ImageNet.
4. **Recommendation System:** Build a recommendation system for movies, books, or products based on user preferences and past behavior.
5. **Handwritten Digit Recognition:** Use the MNIST dataset to create a model that can recognize handwritten digits.
6. **Spam Email Detection:** Develop a machine learning model to classify emails as spam or non-spam.
7. **Face Recognition:** Implement a face recognition system that can identify individuals from a given set of images.
8. **Weather Forecasting:** Create a model that predicts weather conditions based on historical weather data.
9. **Customer Churn Prediction:** Build a model to predict whether a customer will cancel a subscription or leave a service based on their behavior and usage patterns.
10. **Stock Price Prediction:** Develop a model to predict future stock prices based on historical stock market data.
11. **Object Detection:** Create a model that can detect and classify objects within an image or video using a dataset like COCO.
12. **Language Translation:** Implement a neural machine translation model to translate text from one language to another.
13. **Chatbot Development:** Develop a conversational AI chatbot that can answer questions and assist users with various tasks.
14. **Medical Diagnosis:** Build a model to assist in diagnosing diseases based on medical images or patient data (e.g., detecting pneumonia from chest X-rays).
15. **Time Series Forecasting:** Create a model to forecast future values in a time series dataset, such as electricity consumption or traffic flow.

# In Class Activity – Quiz 1



- Quiz on Testing Process and Various Types of testing