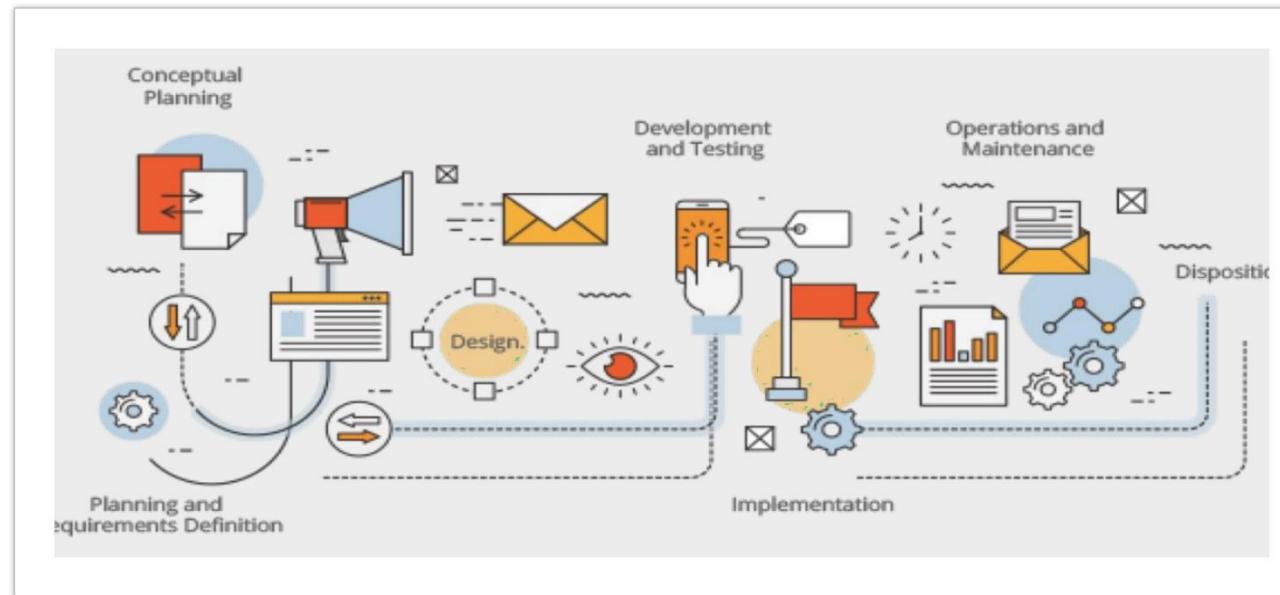


# Software Engineering

## Software Process Models Lecture#03

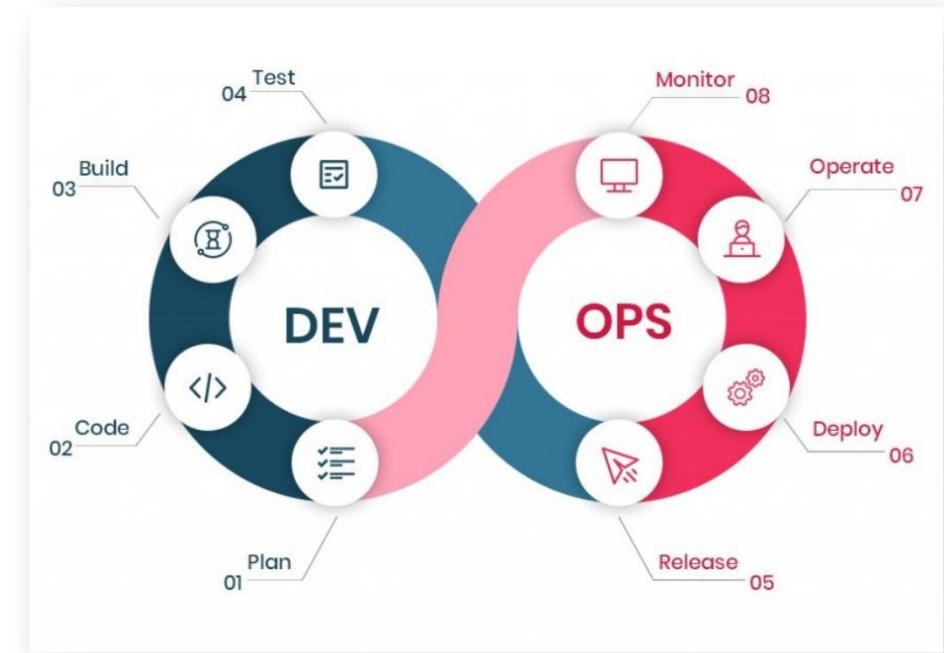
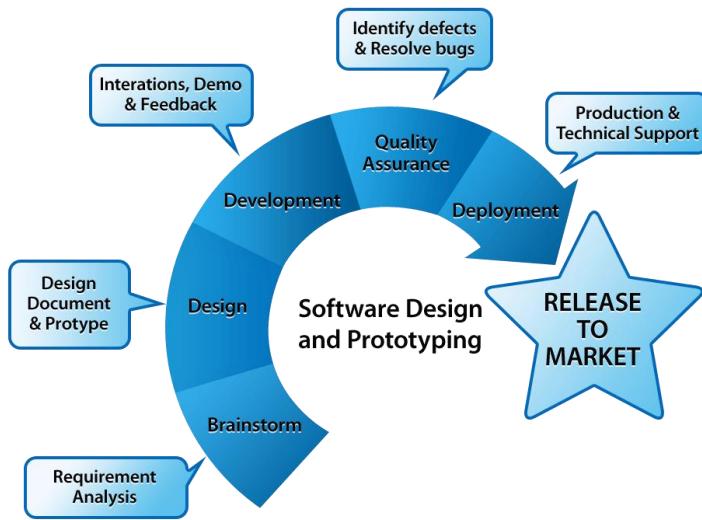


# Revision Lecture 02

1. What are the different software myths?
2. Why software development is still in crises?
3. Software engineering is about having good programming skills?
4. What is software ?
5. What is a product?
6. What are the two rules of software engineering?



# Software Process Models



- What is your understanding of a software process?
- Have you used any software Process model in your practice?



## **Definition of a Software Process:**

A software process is a **structured set of activities or steps** that are followed in order to develop, maintain, and support software. It provides a roadmap for developers to create software in a systematic and organized way.



# Software Dev Process

- Process is distinct from product
  - products are outcomes of executing a process on a project
- SW Eng. focuses on process
- Premise: Proper processes will help achieve project objectives of high QP

# Introduction to Processes

- Dictionary definition of process:
  - A series of actions to produce a result
- Our definition of *software development process*:
  - A systematic method for meeting the goals of a project by:
    - (**who**): determining the roles and skills needed to do the project
    - (**what**): defining and organizing development activities
    - (**when**): setting criteria and deadlines for progressing from one activity to the next
- Note: activities in a software process may be sequential, concurrent, or iterative.

# Fundamental activities of Software Process

---

- **Requirement Analysis:** Understanding what the users or stakeholders need.
- **Design:** Planning how the software will work and how the pieces will fit together.
- **Implementation (Coding):** Writing the actual code to make the software function.
- **Testing:** Checking if the software works as intended and is free from bugs.
- **Deployment:** Releasing the software to users.
- **Maintenance:** Keeping the software updated and fixing any issues that arise after deployment.



# Software process = Building a house

---

- **Requirement Analysis:** You talk to the people who will live in the house and figure out what they need.
- **Design:** You draw blueprints for how the house will look and function.
- **Implementation:** Builders start constructing the house.
- **Testing:** Inspect the house for any issues, leaks, or areas that don't work.
- **Deployment:** The house is ready for the people to move in.



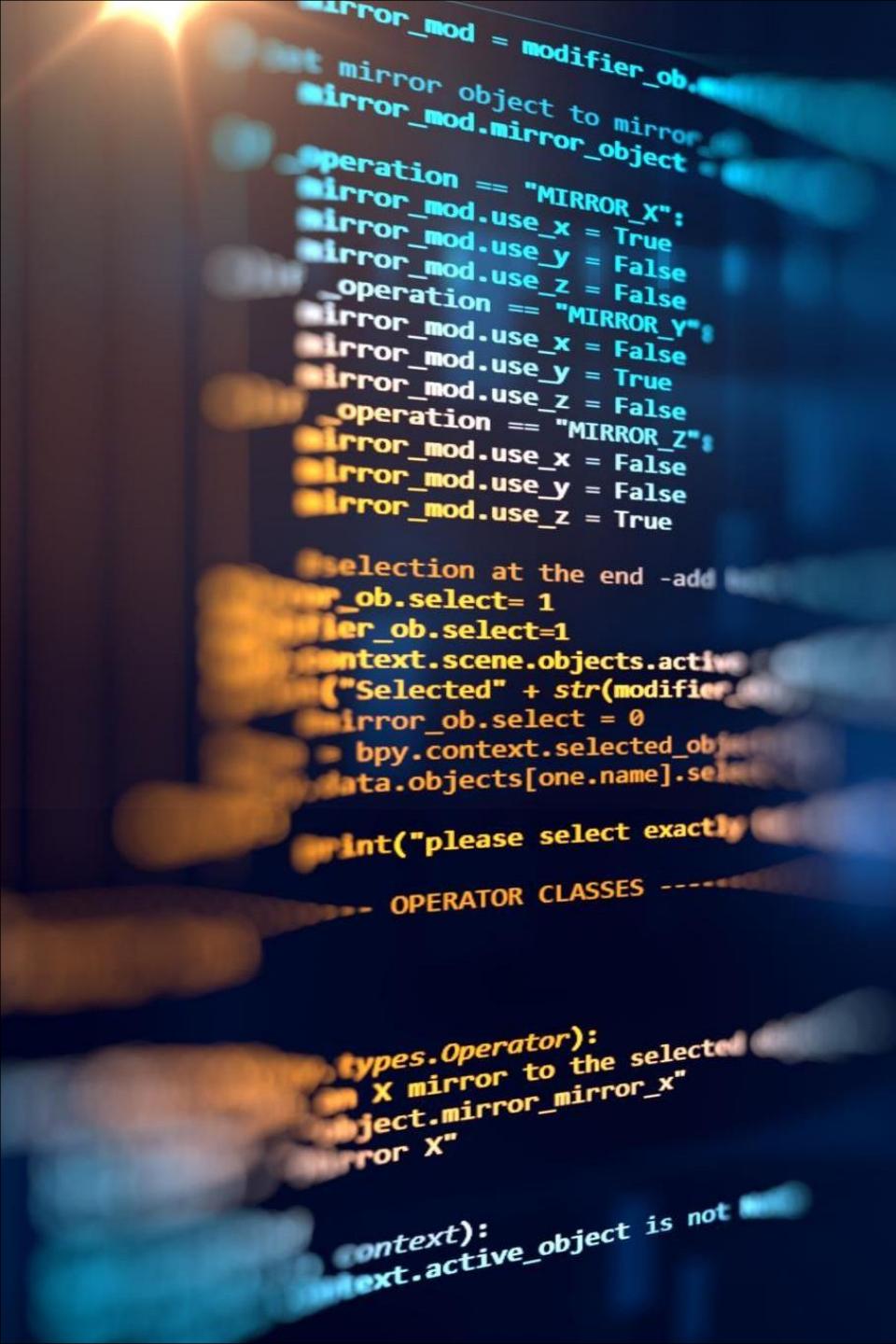
A photograph showing a person's hand holding a modern smartphone with a black frame. The phone is held vertically, facing forward. The background is a whiteboard covered with various colored sticky notes (yellow, red, blue) and some handwritten text, suggesting a planning or development environment.

# Software Process

---

Fundamental Assumption:

- ▼ Good processes lead to good software
- ▼ Good processes reduce risk
- ▼ Good processes enhance visibility
- ▼ Good processes enable teamwork

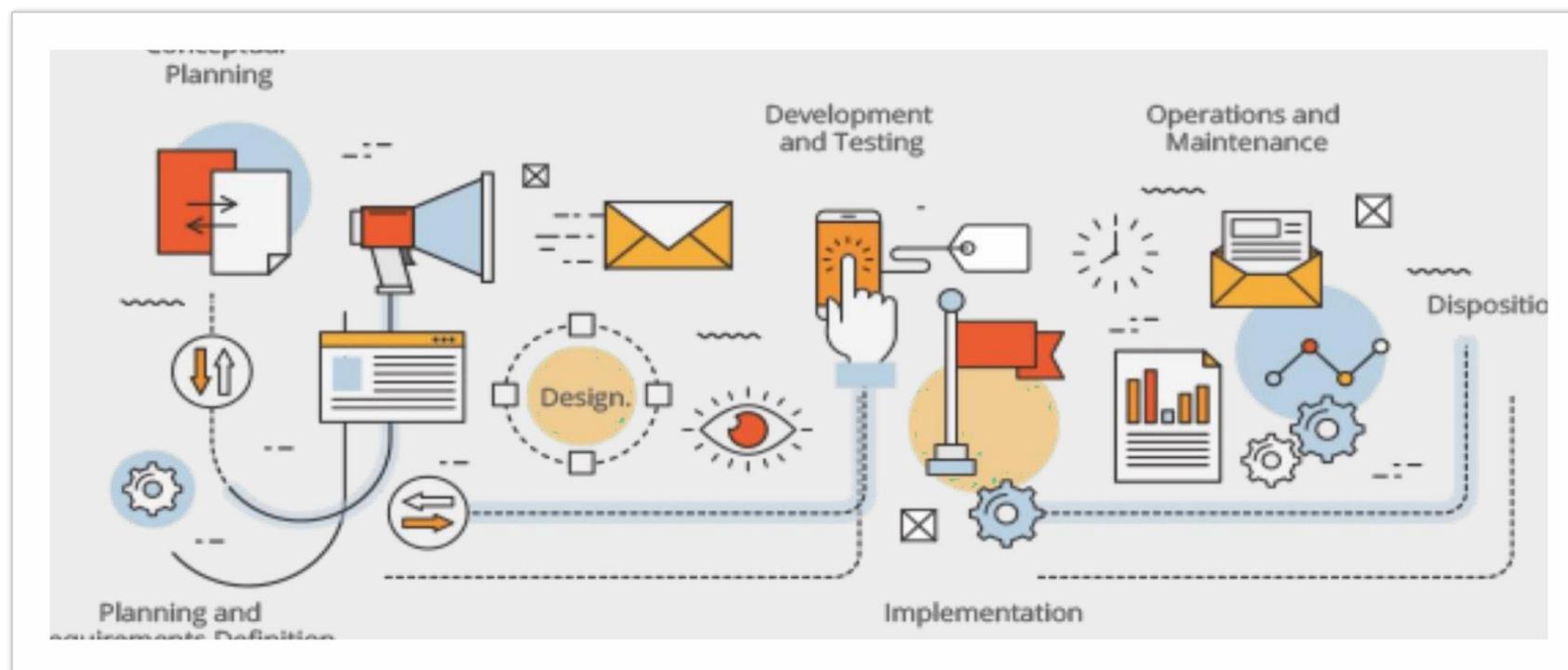


# Variety of Software Processes

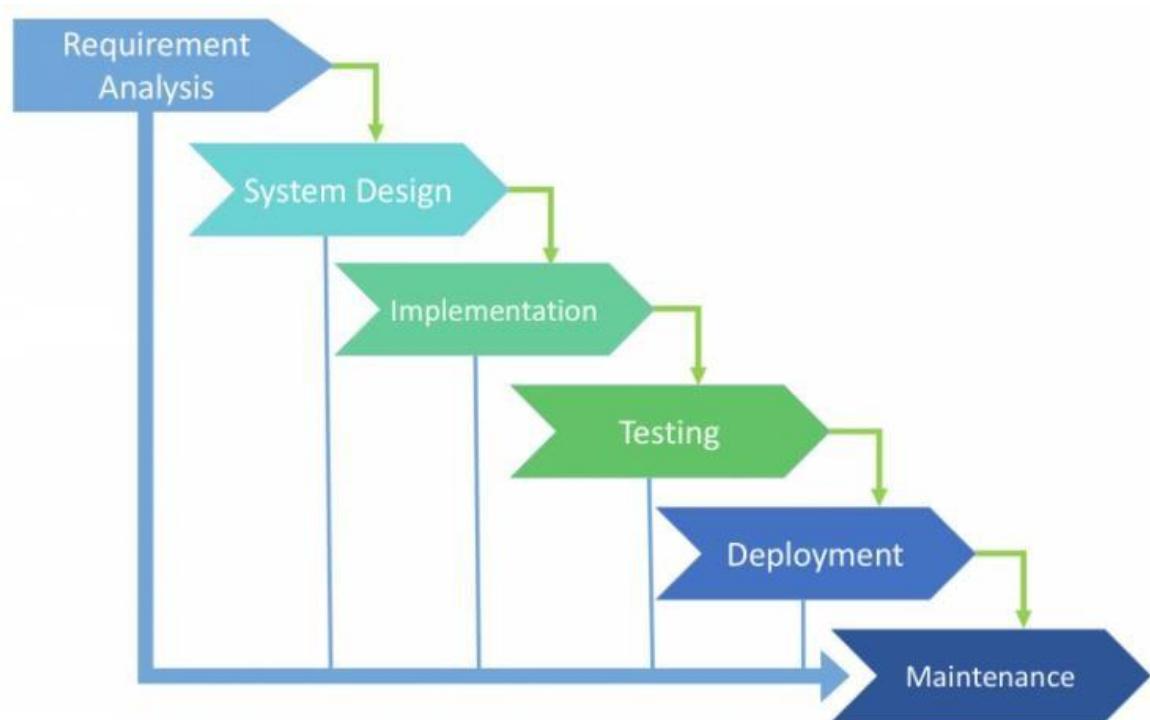
---

- Software products are very varied...
- Therefore, there is **no standard process** for all software engineering projects
- BUT successful software development projects all need to address similar issues.
- This creates a number of process steps that should be part of all software projects.

# Software Process Models



# Software development life cycle



# Process Assurances

- Ensuring that teams follow the defined processes and guidelines.
- Identifying inefficiencies or bottlenecks in the process and suggesting improvements.
- Risk Management: Identifying and mitigating risks early in the process.
- Quality Assurance (QA): Ensuring the process leads to a high-quality end product.
- Ensuring the process complies with industry standards or organizational policies.
- Tracking key metrics to measure the success and health of the process.
  - Example: Monitoring defect density, time to deliver, or process cycle efficiency.

# **How to Find the Right Software Process Model**

- 1. Understand Your Project Characteristics** (size, budget, complexity, timeline)
- 2. Identify Your Requirements and Their Stability** (stable vs evolving requirements)
- 3. Assess the Level of Risk** (High risk safety critical systems vs Low risk systems)
- 4. Determine the Level of Client Involvement** (High vs low client involvement)
- 5. Analyze Time-to-Market Needs** (rapid delivery etc)
- 6. Consider the Team's Skills and Experience**
- 7. Factor in Industry and Compliance Requirements** (Regulated Industries (Medical, Aerospace, Finance))
- 8. Evaluate Maintenance Needs** (Long maintenance or minimal maintenance)
- 9. Match with Stakeholder Expectations** (Prefer predictability vs flexibility)

# How to Find the Right Process?

TEAM-RELATED

**Development**  
Team: experienced?  
Co-located or distributed?

## CUSTOMER-RELATED **Requirements**

Application: critical or discretionary?  
Requirements: fluid or fixed?

**Process?**

**Constraints**

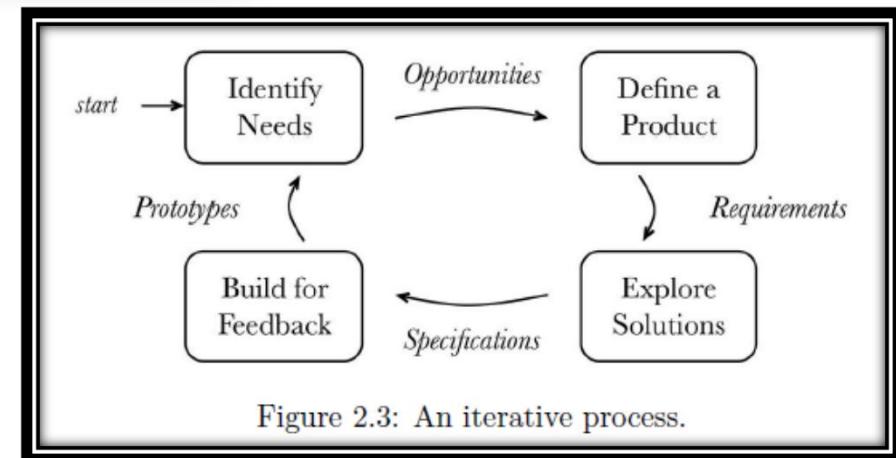
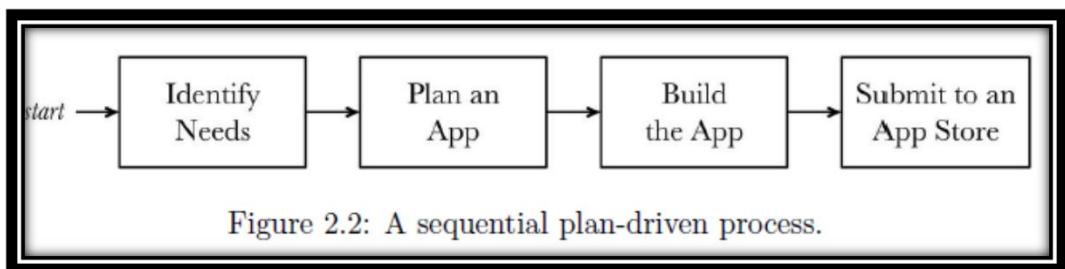
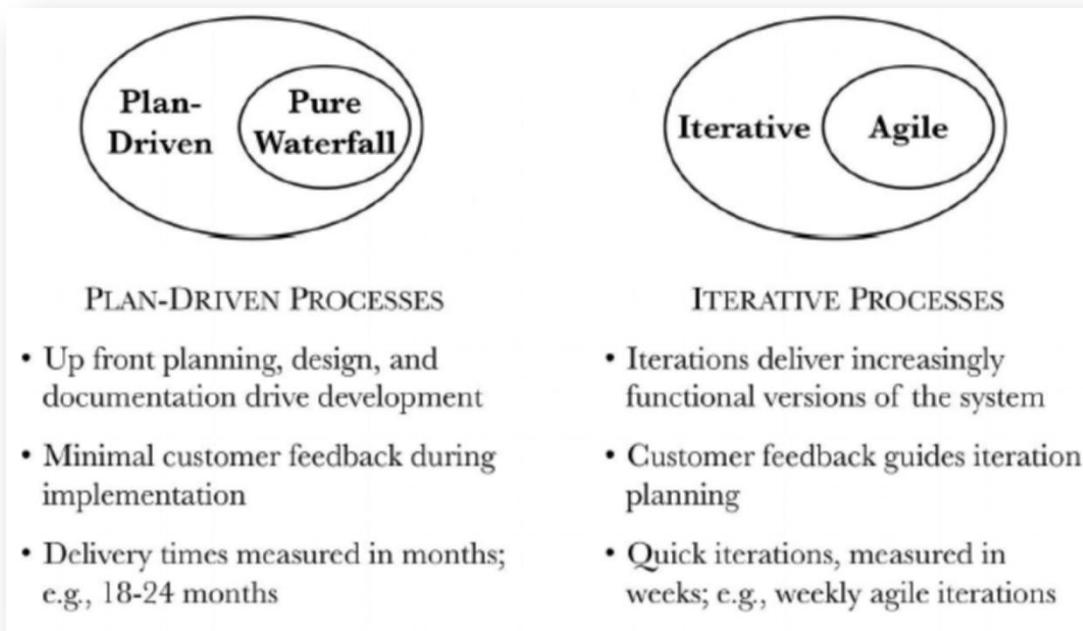
Priority: cost, schedule, or quality?  
Regulations or standards?

PRODUCT-RELATED

**Technology**  
Architecture: flexible?  
Tools: proven or risky?

ORGANIZATION-RELATED

# Organization of process activities



# Process Classes

- Plan and document
  - Assumes that well-defined stable requirements, documented in advance, guide all phases of development through maintenance (Sequential)
  - Infrequent, often indirect, interactions among stakeholders
- Iterative and incremental
  - Go quickly through all process steps to create a rough system, then repeat them to improve the system.
- Agile: quick Iterations, in week(s), not months
  - Early and continuous delivery of small increments of valuable software
  - Welcome changing requirements, even late in development
  - Frequent direct interactions among stakeholders

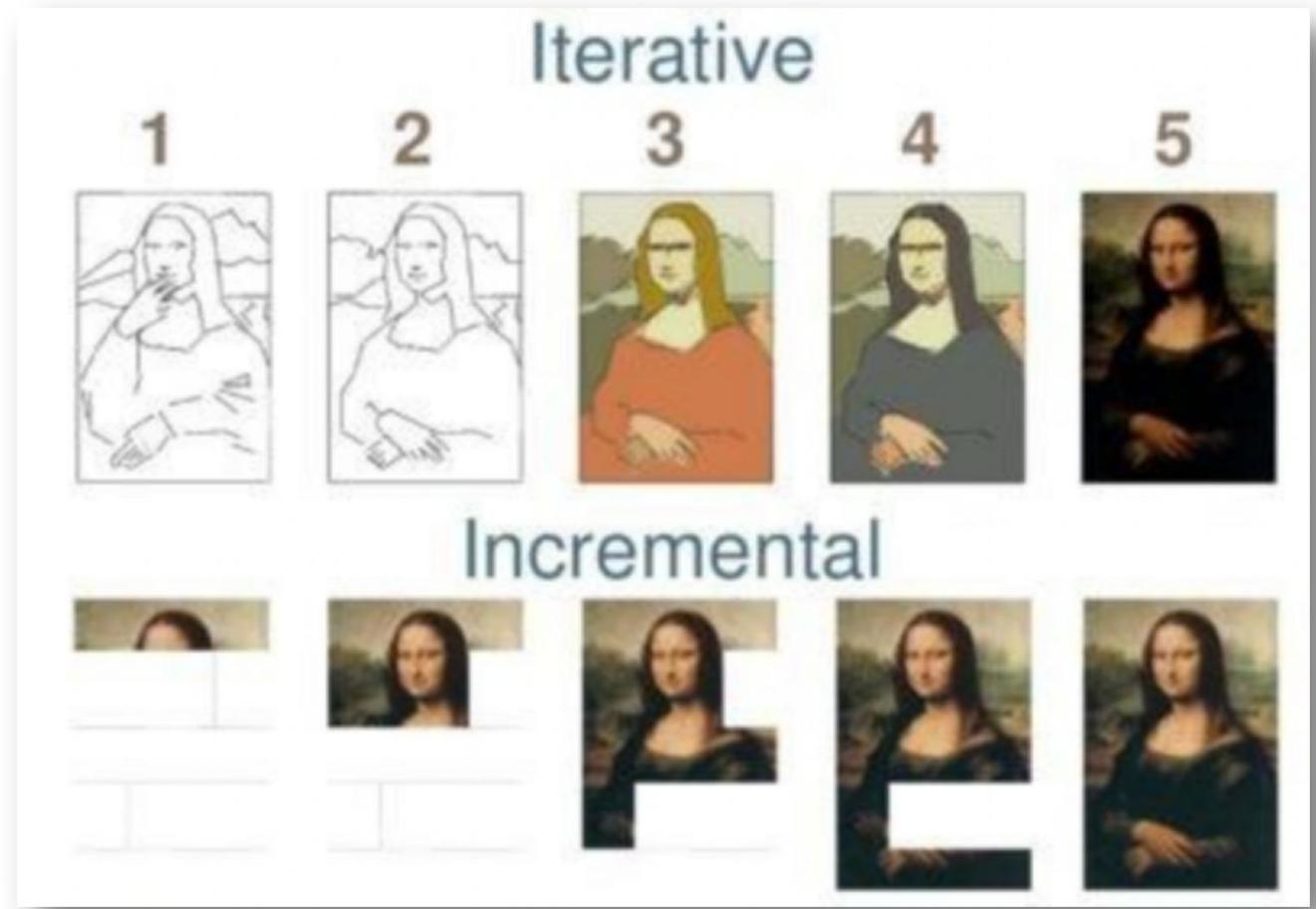
# 1. Software Process Models - Plan-driven

- **Waterfall:** A sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in **linear order**.
- **Prototyping:** A version of a system or part **developed quickly** to check the customer's requirements or feasibility of some design decisions. In prototyping, the **client is involved** throughout the development process.
- **Incremental Development:** developing an **initial implementation**, exposing this to **user feedback**, and **evolving it through several versions** until an acceptable system has been developed.
- **Spiral:** a risk-driven where the process is represented as **spiral rather than a sequence of activities**. best features from the waterfall and prototyping models, and introduces a new component; **risk-assessment**. Each loop represents a phase.
- **Iterative development:** Iterative development model aims to develop a system through building small portions of all features across all components.



## 2. Incremental Versus Iterative

- **Increment:** complete feature of a software product
- **Iterative:** small portions of a product
- Agile incorporates both

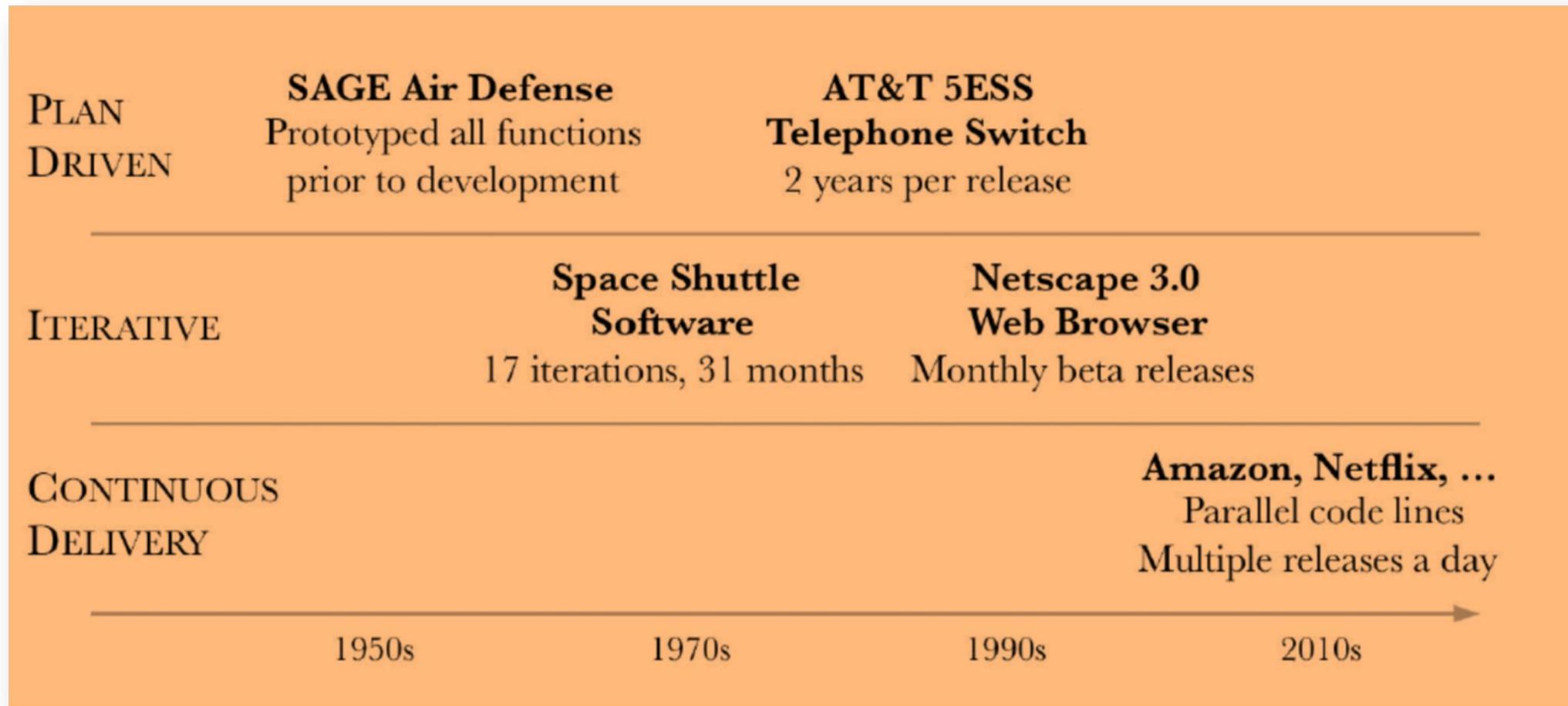


# 3. Software Process Models - Agile

- Agility is flexibility, it is a state of dynamic, adapted to the specific circumstances.
- The agile methods refers to a group of software development models based on the **incremental and iterative** approach, in which **the increments are small** and typically, new releases of the system are created and **made available to customers every few weeks**.
- involve customers in the development process to **propose requirements changes**. They **minimize documentation** by using informal communications rather than formal meetings with written documents.
- Best suited when **requirements change rapidly** during the development.
- There are a number of different agile methods available such as: Scrum, Crystal, Agile Modeling (AM), Extreme Programming (XP), etc

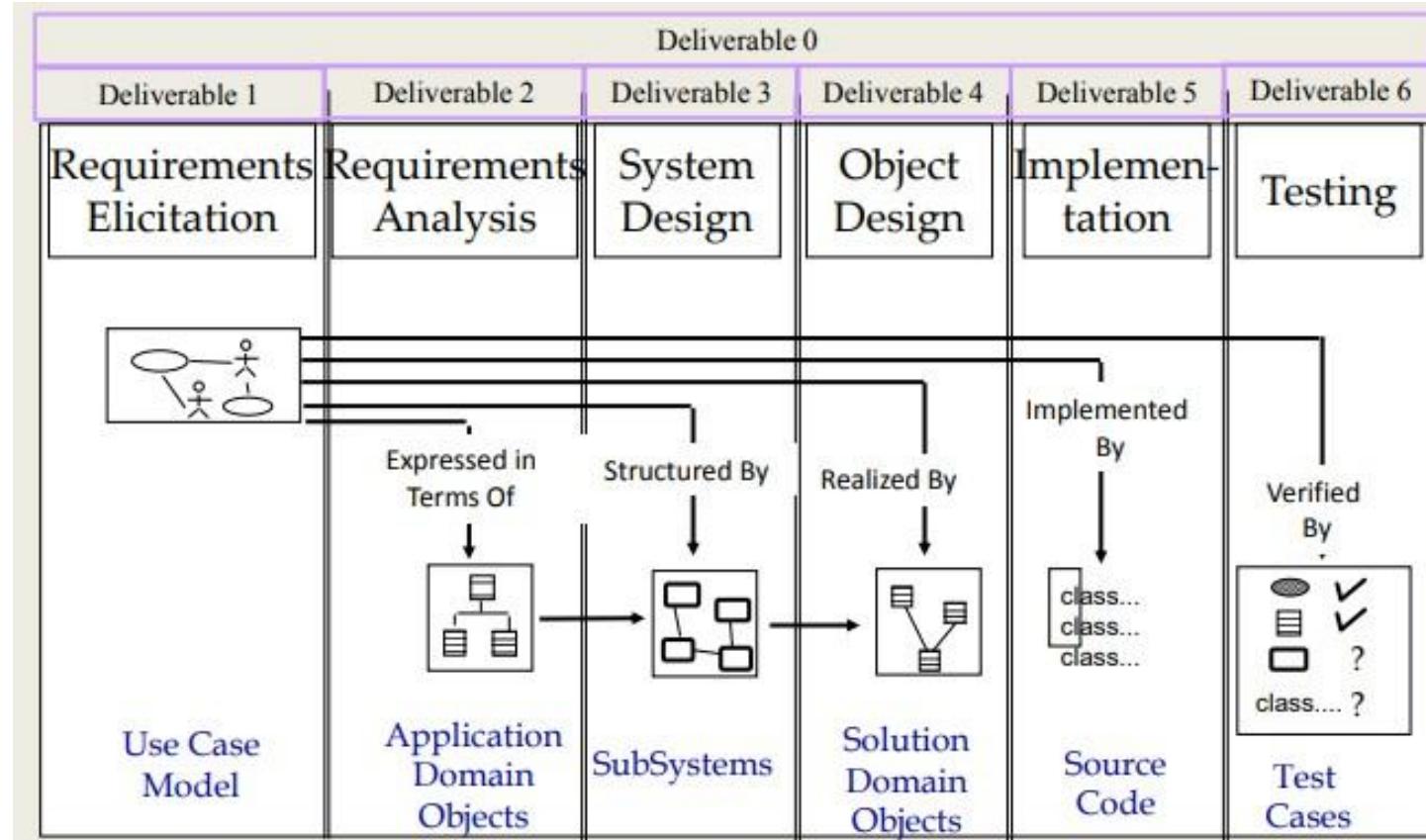


Rough timeline with examples of successful software projects using three classes of processes: plan-driven, iterative, and continuous delivery.



# Software Lifecycle Activities

---



# Basic Process Steps in all Software Development

## Feasibility and planning

---

Requirements

System and program design

Implementation

Acceptance and release

---

These steps may be repeated many times during the development cycle.

## Operation and maintenance

It is essential to distinguish among these process steps and to be clear which you are doing at any given moment.

# Quality Control Steps in all Software Development

- Validating the requirements
- Validating the system and program design
- Usability testing
- Program testing
- Acceptance testing
- Bug fixing and maintenance

Some of these steps will be repeated many times during the life of the system

# Process Step: Feasibility

A **feasibility study** precedes the decision to begin a project.

- What is the scope of the proposed project?
- Is the project technically feasible?
- What are the projected benefits?
- What are the costs, Nmetable?
- Are the resources available?
- What are the risks and how can they be managed?

**A feasibility study leads to a decision: go or no-- go.**

# Process Step: Requirements

- Requirements define the function of the system from the client's viewpoint.
- The requirements establish the system's functionality, constraints, and goals by consultation with the client, customers, and users.
- They must be developed in a manner that is understandable by both the client and the development staff.

# Process Step: Requirements

- This step is sometimes divided into:
  - Ø Requirements analysis
  - Ø Requirements definition
  - Ø Requirements specification
- The requirements may be developed in a self-- contained study, or may emerge incrementally.
- Failure to agree on the requirements and define them adequately is one of the biggest cause of software projects failing.

## High-Level Requirements

*HR01*

The underlined character in each menu selection shall be a shortcut key. When control and the shortcut key are pressed, the menu selection should be loaded.

*HR02*

The system shall have an address book available to store contacts.

*HR03*

The system shall have a help system that offers tips and explanation for each screen and each item on the screens upon demand.

.....

## Low-Level Requirements

*UC01*

Use case name: store a contact's information

Summary:

the address book should store a contact's name, email, address and phone number

Description:

1. enter "pine" command in terminal
2. either enter "a" or use arrows to make "address book" line highlighted and enter "enter"
3. enter "@"
4. enter nickname, fullname, fcc, comment and addresses. may leave some fields blank
5. press ctrl+x to save the entry

*UC02*

Use case name: access help system

Summary:

user accesses help system

Description:

user presses help key

.....

## Relationship Between HLRs and LLRs

Aspect	High-Level Requirements (HLRs)	Low-Level Requirements (LLRs)
Focus	What the system should do (goals).	How the system achieves those goals.
Audience	Stakeholders, managers, business analysts.	Developers, testers, and technical teams.
Detail Level	Generalized and abstract.	Specific and detailed.
Traceability	Can break down into multiple LLRs.	Maps directly to an HLR.
Example	"The system shall allow user login."	"Login API shall validate credentials within 2 seconds."

# Requirement gathering techniques

## 1 Interviews

Interviews are a crucial component of requirement gathering,

## 3 Workshops

Workshops serve as collaborative forums

## 5 Prototyping

Prototyping transforms abstract ideas into tangible models

## 7 Document Analysis

Document analysis involves scrutinizing existing documentation to extract valuable insights

## 2 Surveys & Questionnaires

Surveys and questionnaires provide a scalable approach to gather diverse stakeholder insights

## 4 Observation

Observation brings a hands-on approach to requirement gathering

## 6 Use Cases and Scenarios

Use cases and scenarios provide narrative context

# Process Step: System Design

Design describes the system from the software developers' viewpoint

- **System design:**

Establish a system architecture, both hardware and software, that matches the requirements

- **Program design:**

Represent the software functions in a form that can be transformed into one or more executable programs

Preliminary user testing is often carried out as part of the design step. Models are used to represent the requirements, system architecture, and program design.

# Process Step: Implementation

- **Implementation (coding)**
  - Ø The software design is realized as a set of programs or program units.
  - Ø These software components may be written by the development team, acquired from elsewhere, or modified from existing components.
- **Program testing**
  - Ø Program testing by the development staff is an integral part of implementation. Individual components are tested against the design.
  - Ø The components are integrated and tested against the design as a complete system.

# Process Step: Acceptance and Release

- **Acceptance testing**

The system is tested against the **requirements** by the **client**, often with selected customers and users.

- **Delivery and release**

After successful acceptance testing, the system is delivered to the client and released into production or marketed to customers.

# Process Step: Operation and Maintenance

- **Operation:**

The system is put into practical use.

- **Maintenance:**

Errors and problems are identified and fixed.

- **Evolution:**

The system evolves over time as requirements change, to add new functions, or adapt to a changing technical environment.

- **Phase out:**

The system is withdrawn from service.

- **This is sometimes called the Software Life Cycle**

# Heavyweight and Lightweight Software Development

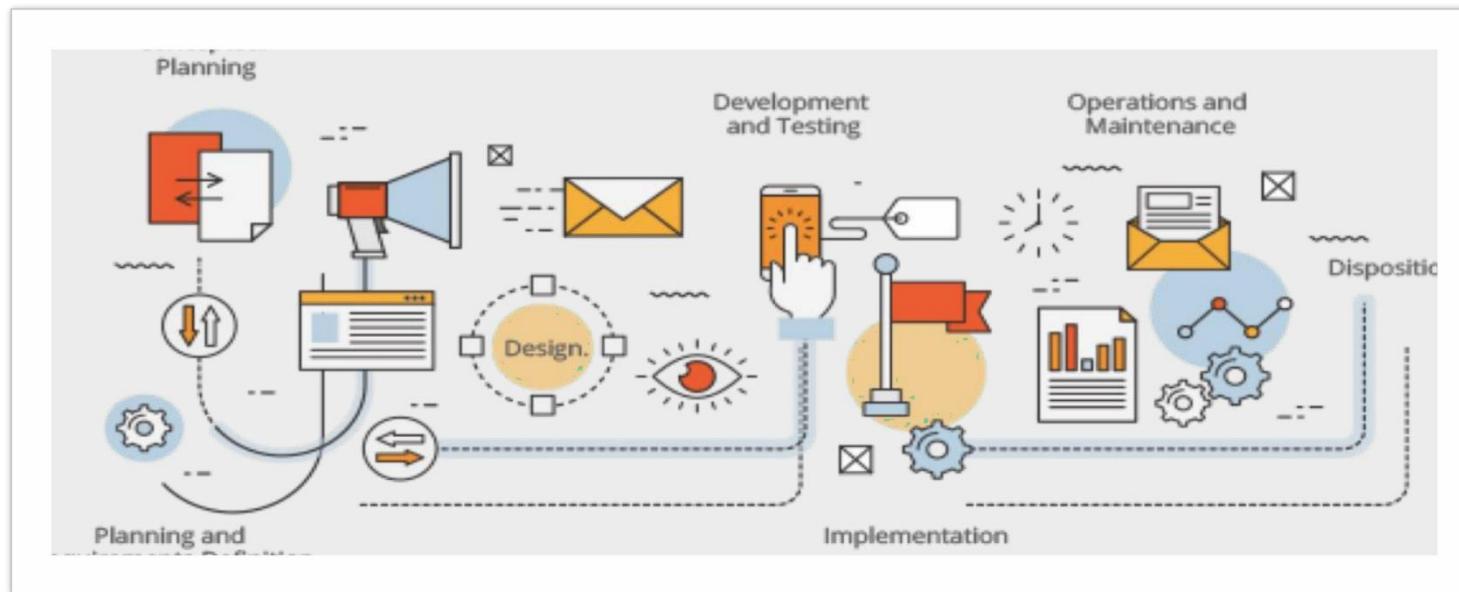
- **Heavyweight methodologies** are structured, formal, and process-intensive. They emphasize extensive planning, comprehensive documentation, and sequential workflows. These approaches are commonly associated with traditional software development models, such as Waterfall and V-Model.
- **Lightweight methodologies** are flexible, iterative, and collaboration-focused, with minimal emphasis on formal processes and documentation. These approaches are typically associated with Agile frameworks, such as Scrum, Kanban, and Extreme Programming (XP).

# Deliverables

A deliverable is some work product that is delivered to the client.

- In a **heavyweight process**, each process step creates a deliverable, usually documentation, e.g., a requirements specification.
- In a **lightweight process**, the deliverables are incremental working code, with minimal supporting documentation.

# Project Management Tools



# Software Management Tools

---

- Software configuration management
  - Github
- Project planning
  - Scrumwise
  - Trello



# Class Activity

- **Case Study:** Select a real-world example of a software project and how the process was applied.
- **Group Discussion:** Which phase is the hardest or most important and why.
- **Role Play:** Have students role-play different phases (like a developer coding or a tester finding bugs).