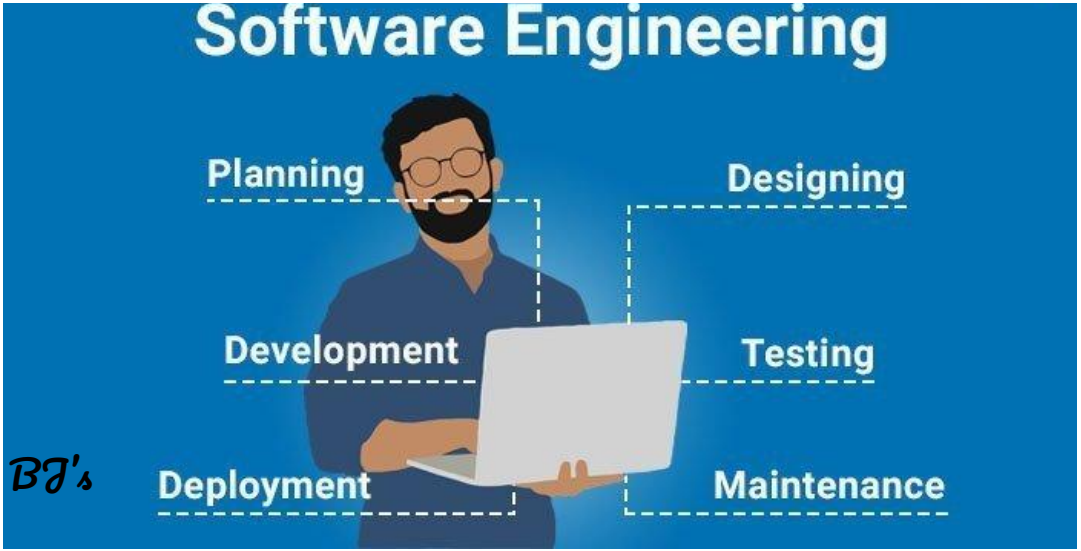



Dr. Biswajeet Sethi
Assistant Professor II
School of Computer Engineering, KIIT
PhD - Department of CSE, IIT Kharagpur
Former NPTEL TA - Cloud Computing
Former SSE - Infosys LTD

Class Participation/ Attendance - 5
90% - 5
85% - 4
80% - 3
75% - 2
Class Note (5).
Mapped with attendance - 5
85%- 5
80% - 4
75% - 3
70% - 2

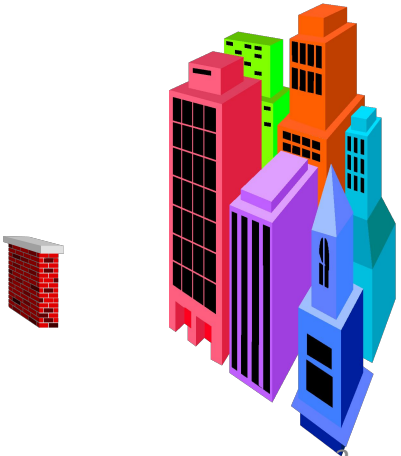
BJ's

Assignments / Quiz/ Projects - 20

# What is Software Engineering?

- Engineering approach to develop software.
  - Building Construction Analogy.
- Systematic collection of past experience:
  - Techniques,
  - Methodologies,
  - Guidelines.

BJ's




Deployment
Updates
Testing
Maintenance
Software Product
Code Design
BJ's
System Design
System Analysis
Requirements

# Use Case: Developing a Fitness Tracker Mobile App

## 1. Requirement Gathering (Systematic Approach)

You don't start coding right away. First, you gather clear **requirements** from the client or users.

- Users want to:
  - Track steps
  - Log workouts
  - Set fitness goals
  - Sync data with wearable devices

BJ's

✅ **Engineering Principle:** Understand the problem clearly before building a solution (like an architect understanding the purpose before drawing building plans).

# Use Case: Developing a Fitness Tracker Mobile App

## 2. Design (Disciplined Approach)

Next, software engineers design how the app will work.

- Create **UI/UX mockups**
- Design the app's **architecture** (backend, database, APIs)
- Choose the **tech stack** (React Native, Firebase, etc.)

BJ's

# Use Case: Developing a Fitness Tracker Mobile App

### 🔹 3. Development (Systematic + Disciplined)

Now developers start coding the app in **phases** (e.g., login system, workout logging module, sync feature).

- Follow version control using **Git**
- Stick to **coding standards**
- Code in **sprints** (Agile method)

✅ **Systematic:** Work in stages
✅ **Disciplined:** Stick to plan, team coordination, clean code

BJ's

Git helps you keep track of every change you make to your project — so you can go back, fix mistakes, and collaborate with others without losing anything.

A **sprint** is a short, fixed period of time (usually **1–2 weeks**) where a development team works to complete a set of tasks (called **user stories** or **features**)

## 🛠️ What Git Does:

| Feature | What It Means |
| --- | --- |
| 📝 Tracks changes | Git saves every version of your file |
| ⏪ Revert to old versions | You can go back to any previous state |
| 👥 Team collaboration | Multiple people can work on the same code without conflict |
| 🌿 Branches | Create separate lines of work (e.g., test features without breaking main code) |
| 🔀 Merging | Combine changes from different team members or branches |

### 🧱 Basic Git Concepts:

| Term | Meaning |
| --- | --- |
| **Repository (repo)** | A Git-managed project directory |
| **Commit** | A snapshot of changes (like saving progress) |
| **Branch** | A separate version of the code to work on independently |
| **Merge** | Combine one branch into another (e.g., add a new feature to the main code) |
| **Clone** | Download a Git project from the internet (e.g., from GitHub) |
| **Push** | Upload your changes to a remote server (like GitHub) |
| **Pull** | Download others' changes from the remote server |

BJ's

## 📱 Example (in a Mobile App Project)

1. You create a **Git repository** for your mobile app.
2. You make a change (e.g., add a login screen) → **commit**
3. Your teammate adds a profile screen on a separate **bran**
4. You later **merge** their changes with yours.

BJ's

## 🧠 Why Git is Important in Software Engineering:

- ✅ Prevents **loss of code**
- ✅ Makes **teamwork safer** (no one overwrites someone else's work)
- ✅ Keeps a **history** of every change
- ✅ Helps with **code reviews** and tracking bugs
- ✅ Widely used in industry (GitHub, GitLab, Bitbucket)

BJ's

## 📱 Example: Mobile Fitness App in Agile Sprints

| Sprint | Duration | Goals / Features to Code |
|---|---|---|
| Sprint 1 | 1 week | User Login + Signup screen |
| Sprint 2 | 1 week | Step counter UI + tracking logic |
| Sprint 3 | 1 week | Workout logging + local database |
| Sprint 4 | 1 week | Push notifications + reminders |

BJ's

Each sprint results in part of the app that **actually works** — not just designs or placeholders.

# Guess if you know (2 marks)

BJ's

# Use Case: Developing a Fitness Tracker Mobile App

### 4. Testing (Quantifiable Approach)

The app is thoroughly tested for functionality, performance, and security.

- Write **unit tests** and **UI tests**
- Measure **code coverage**
- Use tools to track **number of bugs**, **crash reports**, **response times**

BJ's

✅ **Quantifiable:** Bugs per module, crash rate, battery usage — all measurable metrics

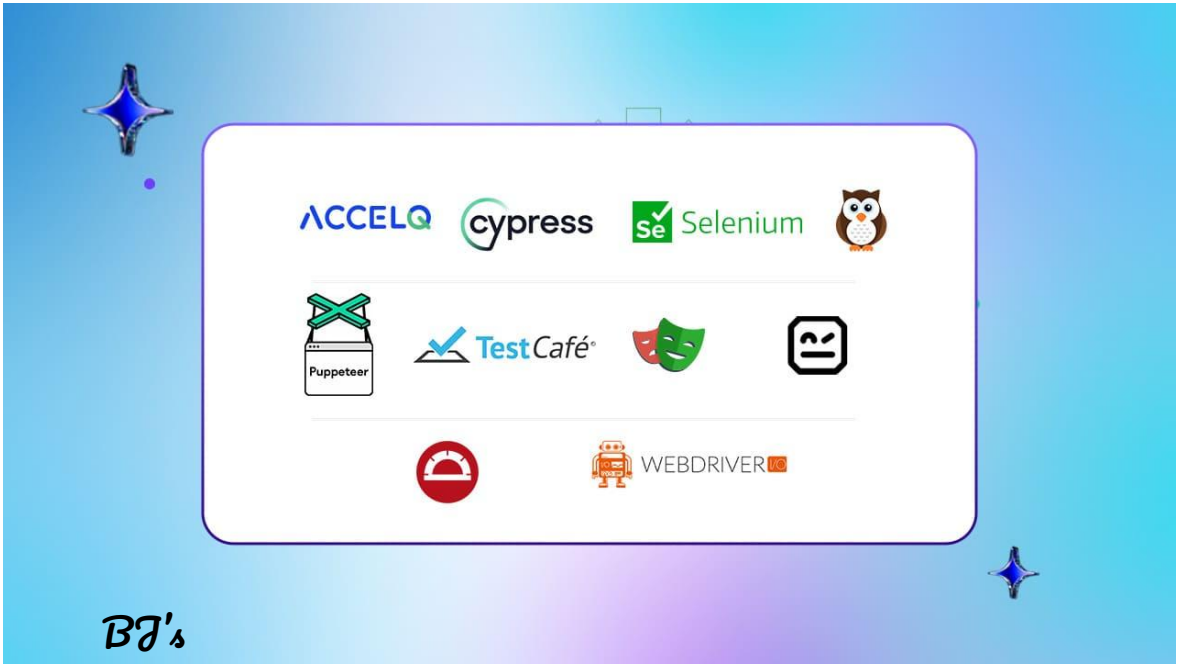
Assignment: Name some tools used to report software bug

## 🔁 Comparison Table

| Feature | Unit Test | UI Test |
|---|---|---|
| 🧪 Tests what? | One function or component | Full user interface (buttons, screens) |
| 👩‍💻 Done by | Developers | Testers or QA automation tools |
| ⚙️ Tools | JUnit, pytest, NUnit | Selenium, Appium, Espresso |
| ⚡ Speed | Very fast | Slower |
| 📱 Real app interaction | No | Yes |
| 🧩 Dependency on UI | No | Yes |

BJ's

# Know Some UI Testing tools



# Use Case: Developing a Fitness Tracker Mobile App

🔹 **5. Deployment & Operation**

App is published to Play Store/App Store.

- Use CI/CD pipelines for deployment
- Monitor with analytics tools (e.g., Firebase Crashlytics, Google Analytics)

✅ Engineers monitor how users interact, crashes, and retention — again **measurable**

BJ's

## 🚀 What is a CI/CD Pipeline?

**CI/CD** stands for:

- **CI = Continuous Integration**
- **CD = Continuous Delivery** or **Continuous Deployment**

A **CI/CD pipeline** is a series of **automated steps** that help developers **build, test, and deploy code faster and more reliably**.

### ⚙️ What Happens in a CI/CD Pipeline?

| Stage | What Happens |
| --- | --- |
| 🧪 **Continuous Integration (CI)** | Developers push code to a shared repository (e.g., GitHub). CI automatically:<br>- Merges the code<br>- Builds the app<br>- Runs unit tests to check for bugs |
| 🚀 **Continuous Delivery (CD)** | After successful tests, the code is packaged and **ready to deploy**. A human may trigger the release. |
| ⚡ **Continuous Deployment (CD)** | Even the deployment is **automatic**. Code goes live right after passing all checks — **no human involvement**. |

BJ's

# 🔥 1. Firebase Crashlytics

## 🧩 What is it?

**Firebase Crashlytics** is a **real-time crash reporting tool** that helps you **track, prioritize, and fix app crashes and errors**.

### 💉 Key Features:

BJ's

| Feature | What it does |
|---|---|
| 🚨 Crash Reports | Shows where and why your app is crashing |
| 📍 Stack Traces | Tells you the exact line of code that caused the crash |
| 📱 Device Info | Captures OS version, phone model, app version |
| 🔁 Real-time Alerts | Notifies you instantly when a new crash happens |
| 📊 Crash Frequency | Tells how many users are affected and how often |

# 📊 2. Google Analytics (via Firebase Analytics)

## 🧩 What is it?

Firebase Analytics (aka Google Analytics for Firebase) helps you track user behavior inside your app — what users click, how long they stay, where they drop off, etc.

### 📈 Key Features:

| Feature | What it tracks |
|---|---|
| 👥 User Engagement | How many users opened the app and for how long |
| 📍 Screen Tracking | Which screens users visit most |
| ➡️ User Journeys | Flow of users across different screens |
| 🛠️ Custom Events | Track specific actions like "completed workout", "shared on social" |
| 📊 Demographics | User locations, languages, devices |
| 🎯 Audiences | Segment users (e.g., paid vs free, new vs returning) |
| 💰 Conversion Tracking | Track goals like signups, in-app purchases |

# Use Case: Developing a Fitness Tracker Mobile App

### 🔹 6. Maintenance and Updates

User feedback leads to updates:

- Add new features (e.g., social sharing)
- Fix bugs
- Ensure compatibility with new OS versions

BJ's

✅ **Engineering mindset**: Continuously improve product quality and performance over time

# IEEE Definition

- "Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

BJ's

- **"Systematic" -** Instead of jumping directly to coding, a team creates a project plan and follows development phases in order.

- **"Disciplined" -** Following coding standards, writing unit tests, and performing code reviews are part of a disciplined approach.
- **"Quantifiable" -** Measuring how many defects are found in testing or how long a system takes to respond.

- **Development** – creating the software
- **Operation** – running the software in the real world
- **Maintenance** – updating and fixing the software after deployment

BJ's

Example: A mobile app is designed and developed, deployed to users, and regularly updated with bug fixes and new features.

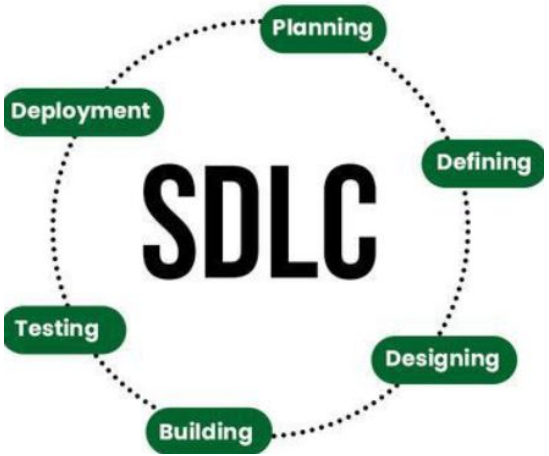
# Software Crisis

- It is often the case that software products:
  - Fail to meet user requirements.
  - Expensive.
  - Difficult to alter, debug, and enhance.
  - Often delivered late.
  - Use resources non-optimally.

# Software Development Life Cycle (SDLC)

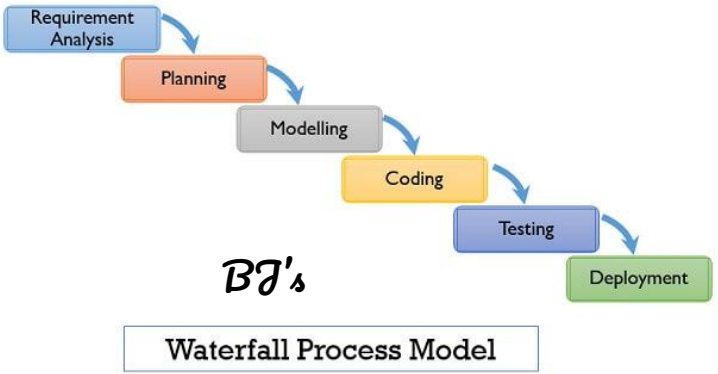
SDLC is a process followed for software building within a software organization. SDLC consists of a precise plan that describes how to develop, maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.

BJ's



- “The basic idea… take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system… Start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions. At each version design modifications are made along with adding new functional capabilities. “ Victor Basili

# Classical Waterfall Model



Waterfall Process Model

Each phase must be **completed before** the next begins.

Works well **only when requirements are fixed and clear** from the beginning.

**Low Flexibility**

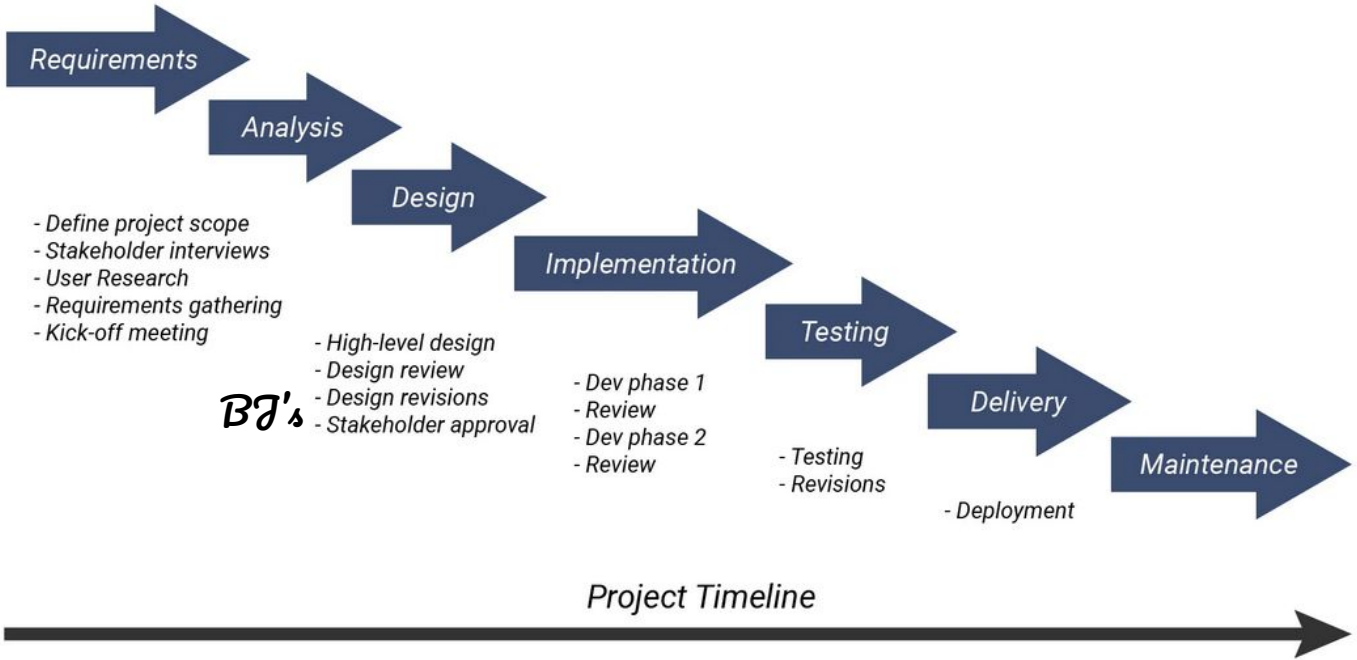
- Very hard to accommodate **changes** once the development moves past the requirement or design phase.

**Late Testing**

- The product is tested **only after coding is complete**, so bugs might be discovered **very late**.

**High Risk**

- A working software product is available **only at the end**, which can lead to **high risk** if something goes wrong.


Requirements
Analysis
Design
Implementation
Testing
Delivery
Maintenance
- Define project scope
- Stakeholder interviews
- User Research
- Requirements gathering
- Kick-off meeting
- High-level design
- Design review
- Design revisions
- Stakeholder approval
BJ's
- Dev phase 1
- Review
- Dev phase 2
- Review
- Testing
- Revisions
- Deployment
Project Timeline

# Iterative Waterfall Model

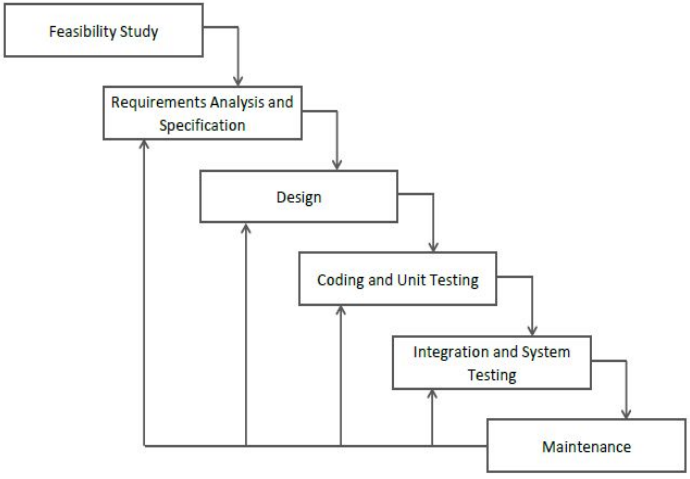
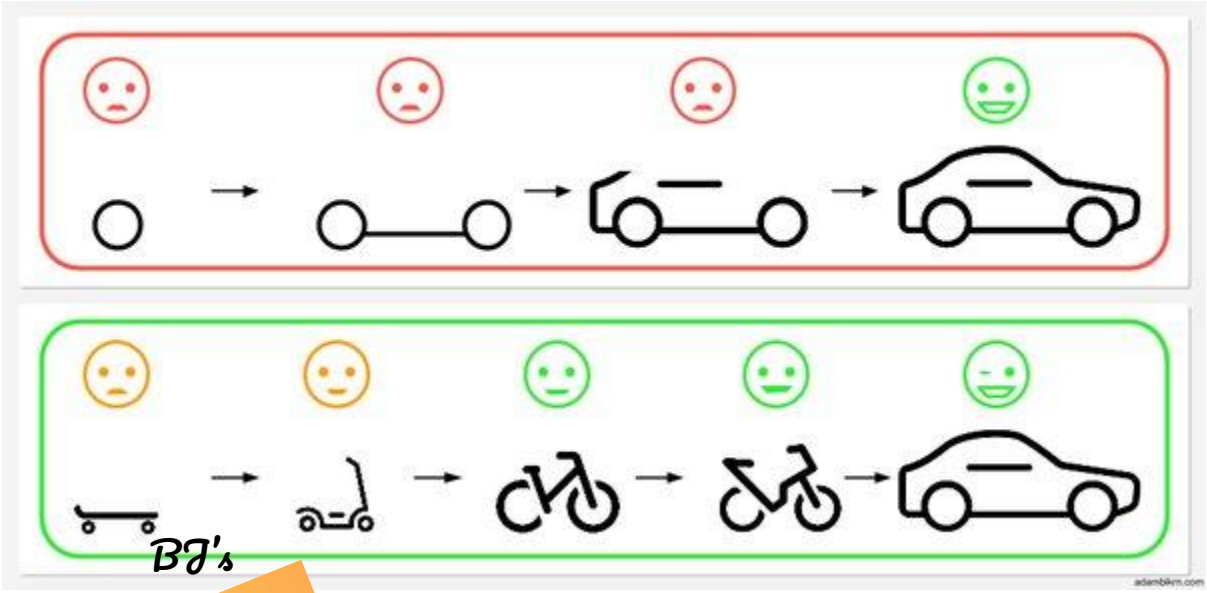



Figure 2: Iterative Waterfall Model

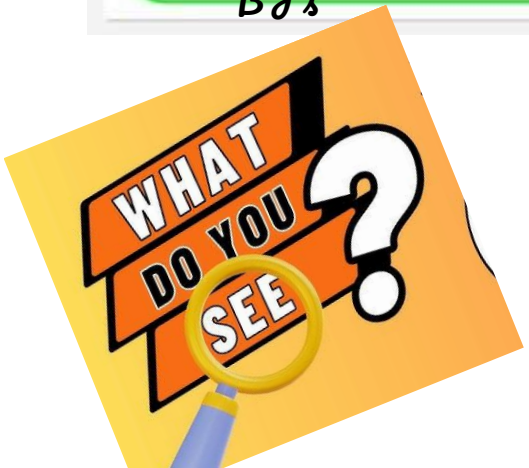
Unlike the classical model, this model supports **revisiting earlier phases** based on feedback or errors found in later stages (e.g., during testing or coding).

The **curved backward arrows** labeled **"Feedback"** show that if a problem is found in a later phase (e.g., during testing), it may require going back to an earlier phase (like design or even requirements), which is what makes it an **iterative** model—unlike the pure waterfall model which strictly avoids going backward.

As errors are corrected in earlier phases through iterations, it helps in **reducing risk and rework**.



BJ's



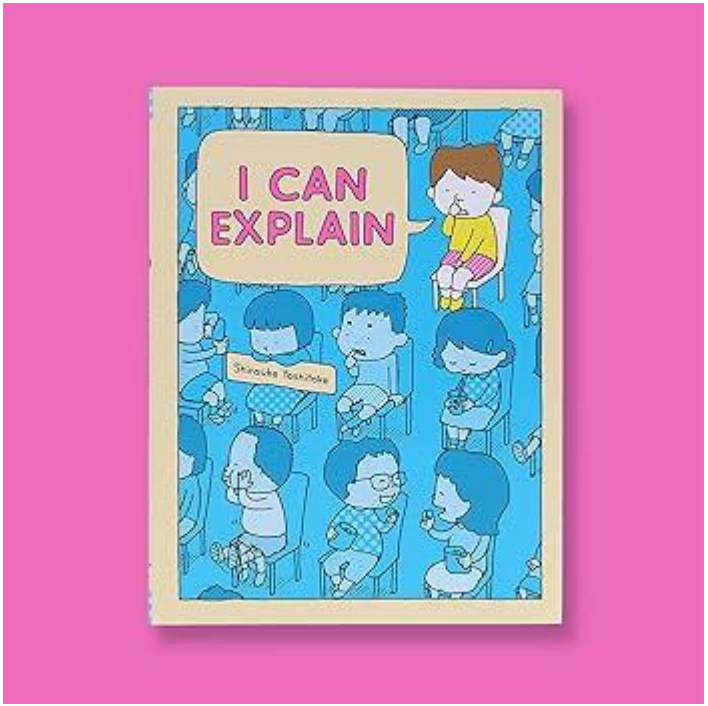
# Incremental and Iterative Development (IID)

- **Key characteristics**
  - Builds system incrementally
  - Consists of a planned number of iterations
  - Each iteration produces a working program
- **Benefits**
  - Facilitates and manages changes
- **Foundation of agile techniques and the basis for**
  - Rational Unified Process (RUP)
  - Extreme Programming (XP)

BJ's

# Iterative vs. Iteration (1 mark)

BJ's



# Iterative vs. Iteration

## What is Iterative Development?

Iterative development involves repeating the development cycle multiple times, with each iteration adding new features or refining existing ones. Each iteration builds upon the previous one, incorporating feedback and changes to improve the software. This approach allows for flexibility and adaptability, as requirements can evolve over time.

## What is Incremental development?

Incremental development involves delivering the software in small, incremental releases. Each release includes a subset of the final features, allowing the software to be tested and used by stakeholders early in the development process. This approach enables faster feedback and validation of requirements, reducing the risk of delivering a product that does not meet user needs.

BJ's

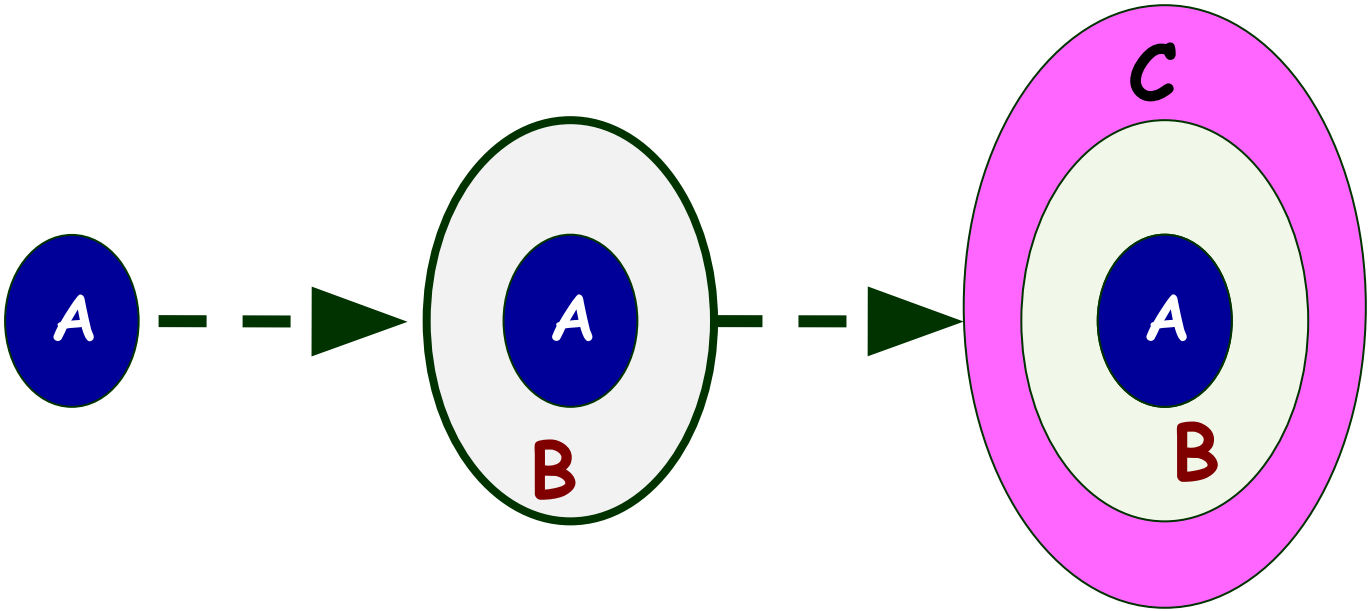
**Iterative** An adjective; describes a **process** that repeats steps.

**Iteration** A noun; refers to **one cycle/round** of the process.

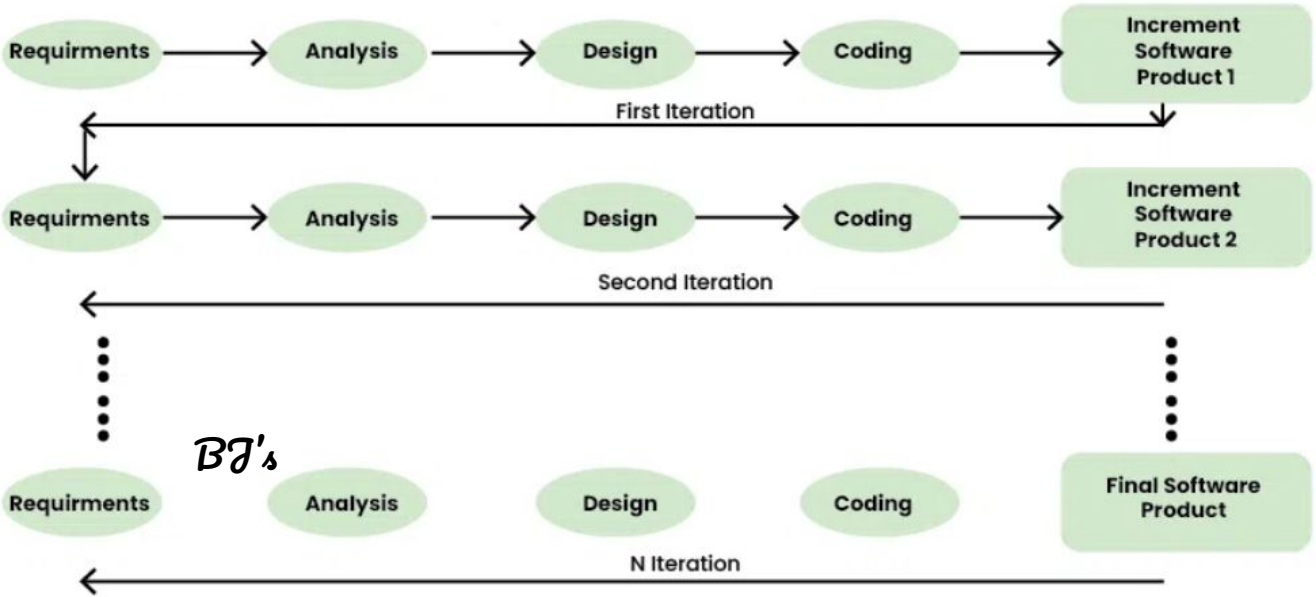
**Example:**

- "The **iterative** approach improves the software step by step."
- "We completed the **first iteration** of development last week."

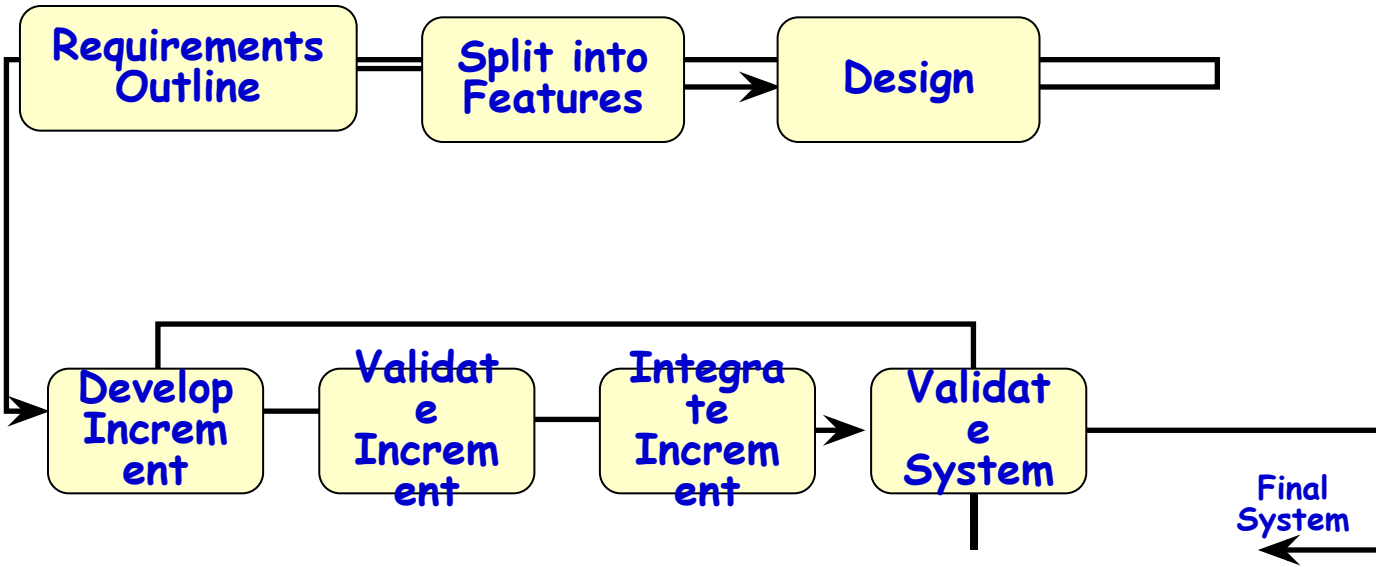
# Customer's Perspective



# What is Iterative Incremental Model?

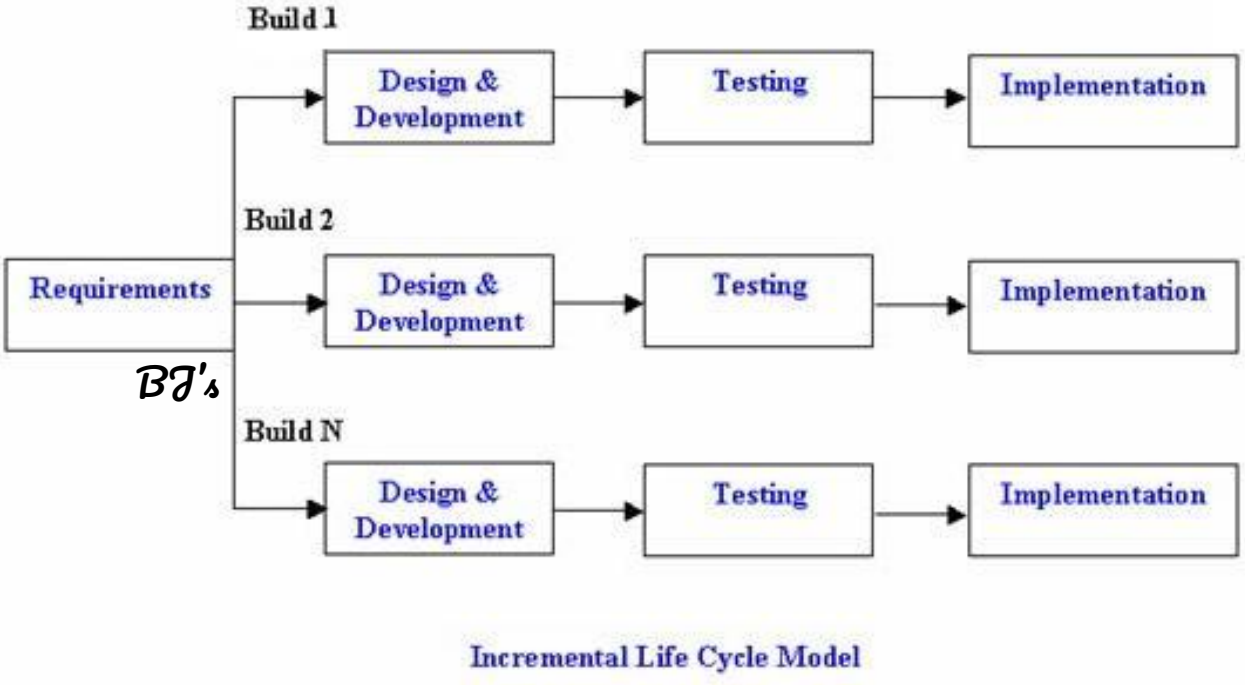


# Incremental Model



# Incremental Model: Requirements

Split into Features




Incremental Life Cycle Model

Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." This process may be described as an "evolutionary acquisition" or "incremental build" approach."

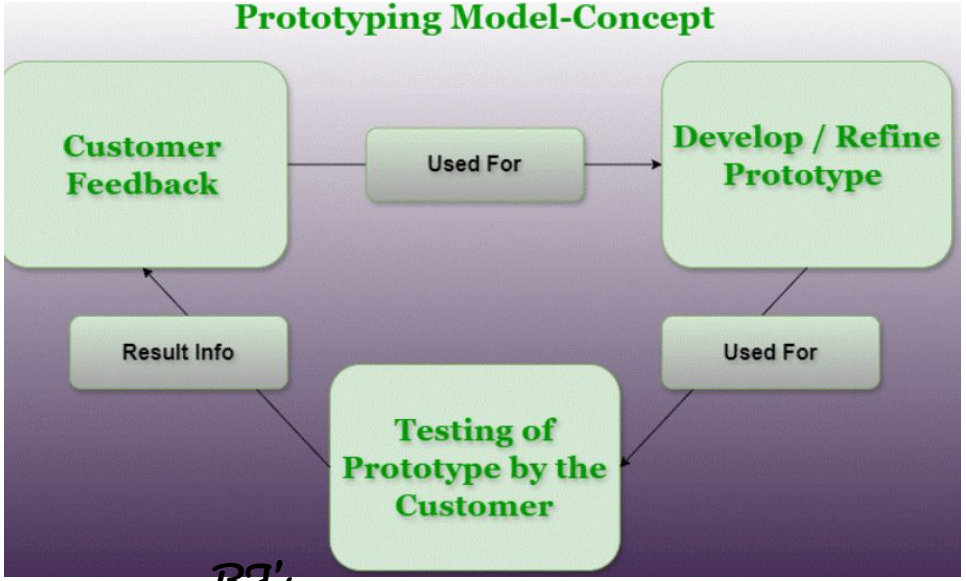
In this incremental model, the whole requirement is divided into various builds.

# Incremental Model

- Waterfall: single release
- Iterative: many releases (increments)
  - First increment: core functionality
  - Successive increments: add/fix functionality
  - Final increment: the complete product

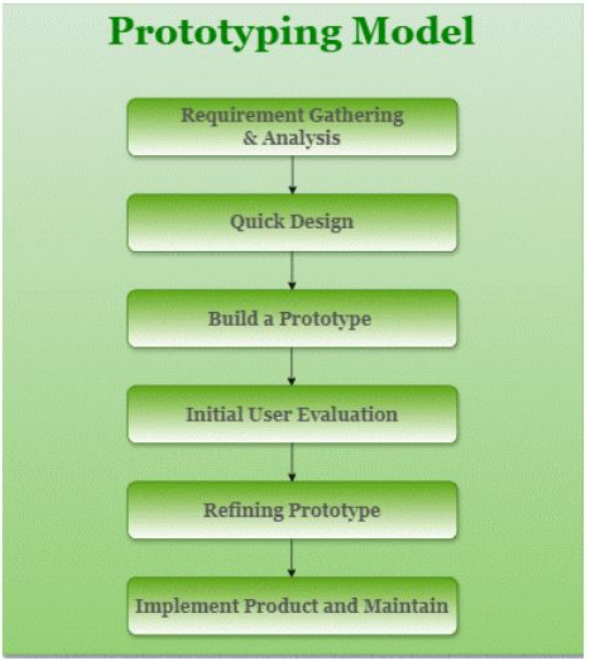
BJ's

- Each iteration: a short mini-project with a separate lifecycle
  - e.g., waterfall




1. When the customers do not know the exact project requirements beforehand.

1. a prototype of the end product is first developed, tested, and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.



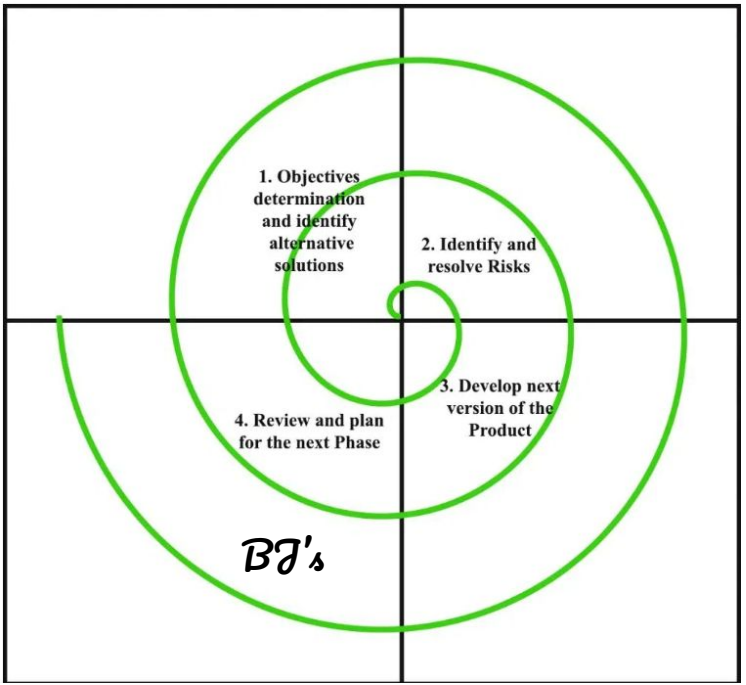

second phase could consist of a preliminary design or a quick design. During this stage, the system's basic design is formed. However, it is not a complete design.

During 3rd stage, an actual prototype is intended to support the knowledge gained from quick design.

## Assignment:

## Types of Prototyping Model

# Spiral Model




The **spiral** is divided into loops (or cycles), each representing a **phase** in the software development process.

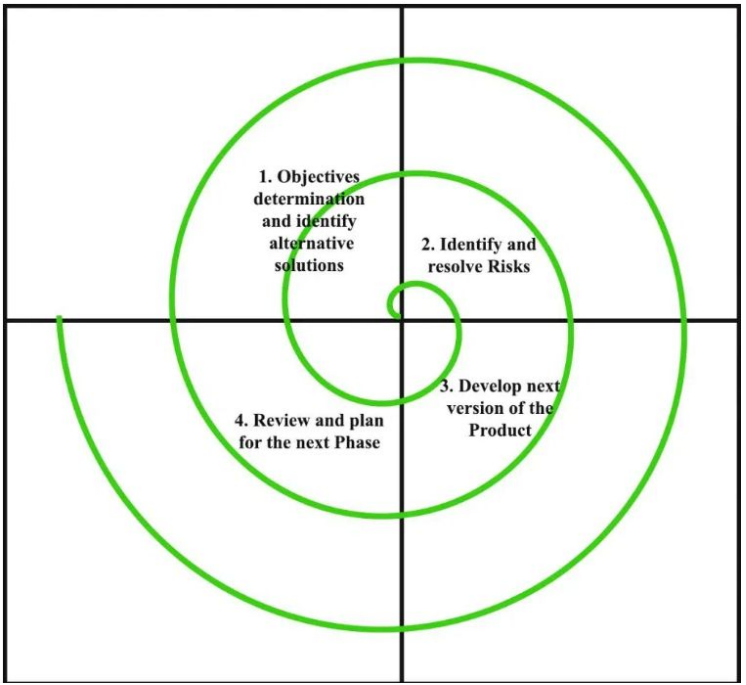
As you **move outward**, each loop corresponds to:

- A more **complete and mature version** of the software.

## Key Features of the Spiral Model:

- **Risk Handling**: Main focus is on identifying and reducing risks early.
- **Iterative Development**: The product is built and improved in repeated cycles.
- **Flexibility**: Allows changes based on feedback and risk discovery.
- **Customer Involvement**: Stakeholders are involved at every iteration.

# Spiral Model

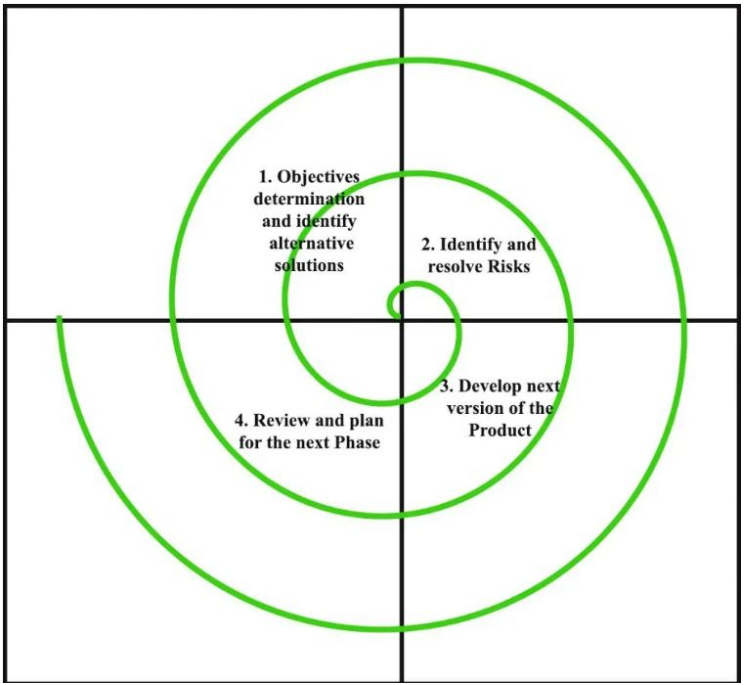



**Spiral Loops (Concentric Circles)**

Each loop or spiral represents **one iteration** of the software product with:

- Increasing levels of detail and completeness.
- From **concept development** to **final product**.

# Spiral Model



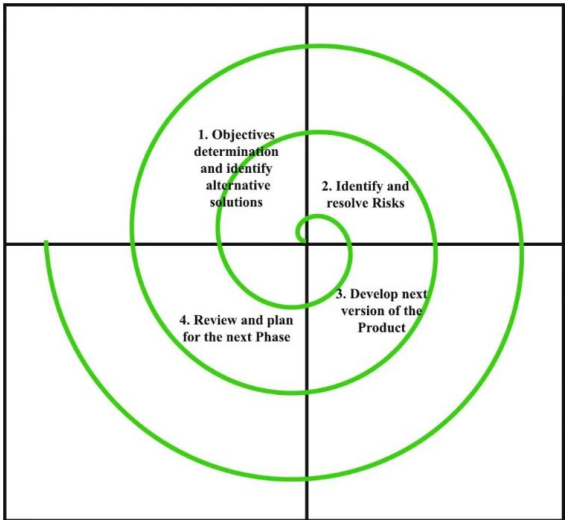

the development process begins at the **center of the spiral** and progresses outward in loops

## 🔁 Loop 1 (First Cycle) – Starts at the center

- **Start Point**: The **innermost circle** of the spiral (center).
- **What happens**:

1. Initial **objectives are determined**.
2. **Risks are identified** (e.g., feasibility, tech challenges).
3. A **prototype or rough version** of the system is developed.
4. The team **reviews** and plans the next steps.

# Spiral Model




Each loop = one **phase of development**.

**First loop** = early planning and risk analysis.

**Second loop** = improved version with more features, deeper design, or broader risk handling.

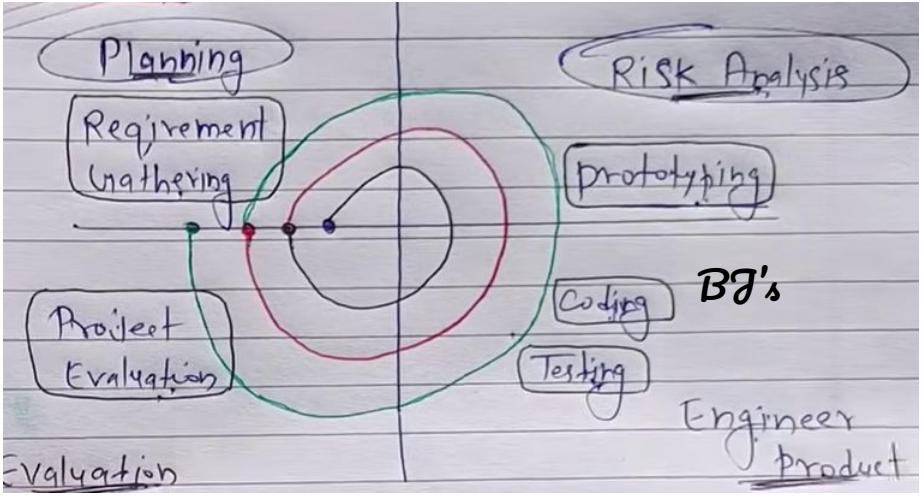
The **center** of the spiral is the **starting point**, and each **loop outward** shows **progression over time**.

## 🔁 Loop 2 (Second Cycle) – Just outside the first loop

- **Start Point**: After completing the first loop, you move outward to the **next ring** of the spiral.
- **What happens**:

1. Build on the previous version.
2. Define more refined **objectives and features**.
3. Handle new or remaining **risks**.
4. Create a **more complete prototype** or functional version.
5. Review and plan again.

✅ Loop 2 focuses on improving and expanding the product, guided by what was learned in loop 1.




🖤 **Black Loop (Innermost) – First Iteration / Initial Prototype**

Focus: **Concept development**

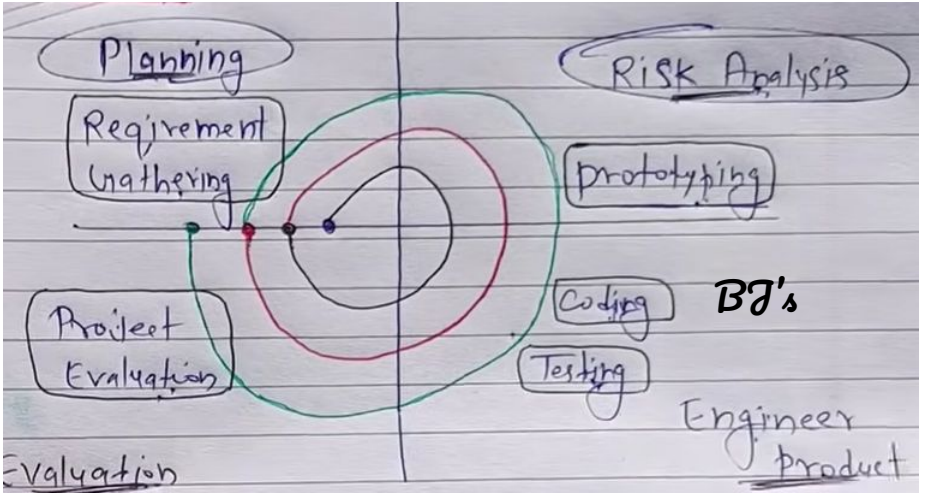
Activities:

Initial planning

Basic risk identification

Early prototyping

Outcome: A **very basic prototype** or **proof of concept**




🔴 **Red Loop (Middle) – Second Iteration / Refined Prototype**

Focus: **Refinement and Risk Handling**
Activities:

Deeper requirement analysis
More detailed risk assessment
Improved prototype
Possibly some initial coding and testing

- Outcome: A **refined version** of the product incorporating feedback from the first loop

## 🟢 Green Loop (Outer) – Final Iteration / Product Development

- Focus: **Finalization and Delivery**
- Activities:
  - Full-scale development
  - Final coding and testing
  - Final risk evaluation
  - Complete project evaluation
- Outcome: **Deliverable system or product**

**1.** Which model is characterized by **rigid sequential phases**, where each phase must be completed before the next begins?

A. Iterative Waterfall Model
B. Prototype Model
C. Waterfall Model
D. Spiral Model

**2.** Which model includes **risk analysis as a core component** in every phase of development?

A. Spiral Model
B. Waterfall Model
C. Iterative Waterfall Model
D. Prototype Model

BJ's

**3.** In which model is **user feedback on a mock version** of the software used to improve requirements?

A. Waterfall Model
B. Prototype Model
C. Iterative Waterfall Model
D. Spiral Model

BJ's

**4.** Which model improves upon the traditional waterfall by **allowing revisiting of earlier phases**?

A. Prototype Model
B. Spiral Model
C. Iterative Waterfall Model
D. Agile Model

BJ's

**6.** In which model is **a throwaway version of the software** created first to clarify requirements?

A. Waterfall Model
B. Iterative Waterfall Model
C. Prototype Model
D. Spiral Model

*BJ's*

**7.** Which model is most suitable for **complex and high-risk projects**, especially where requirements are unclear?

A. Waterfall
B. Spiral
C. Prototype
D. Iterative Waterfall

BJ's

i8. Which model includes **looping back** from any phase to a previous one but still follows a largely linear approach?

A. Iterative Waterfall Model
B. Prototype Model
C. Waterfall Model
D. Spiral Model

BJ's