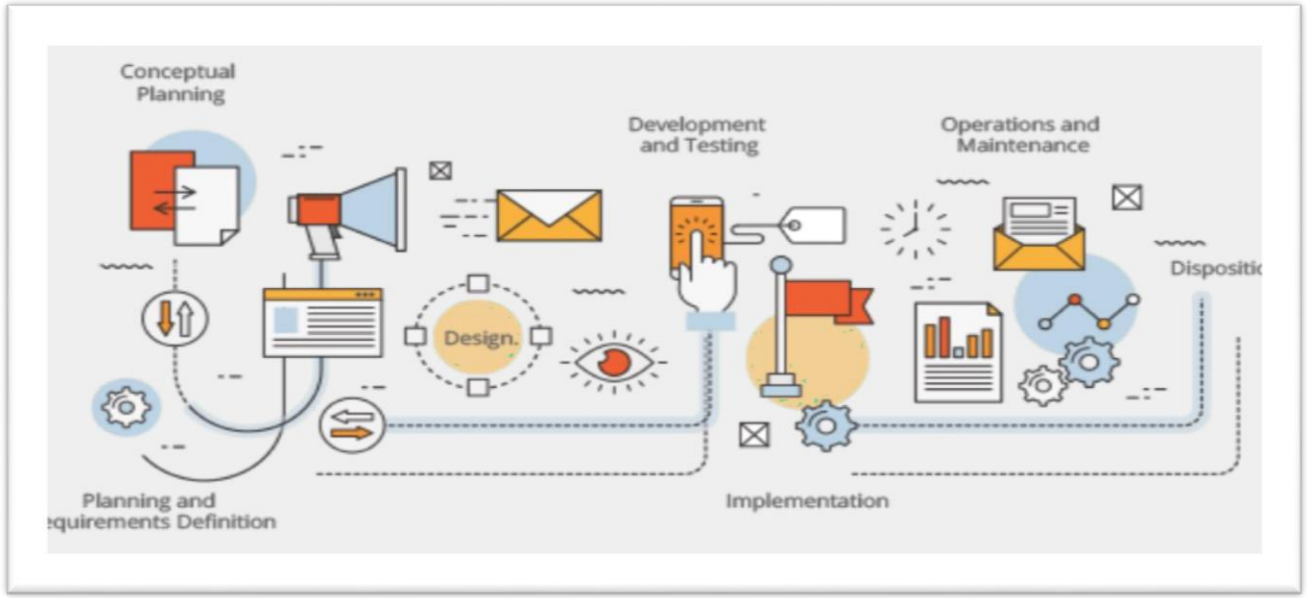
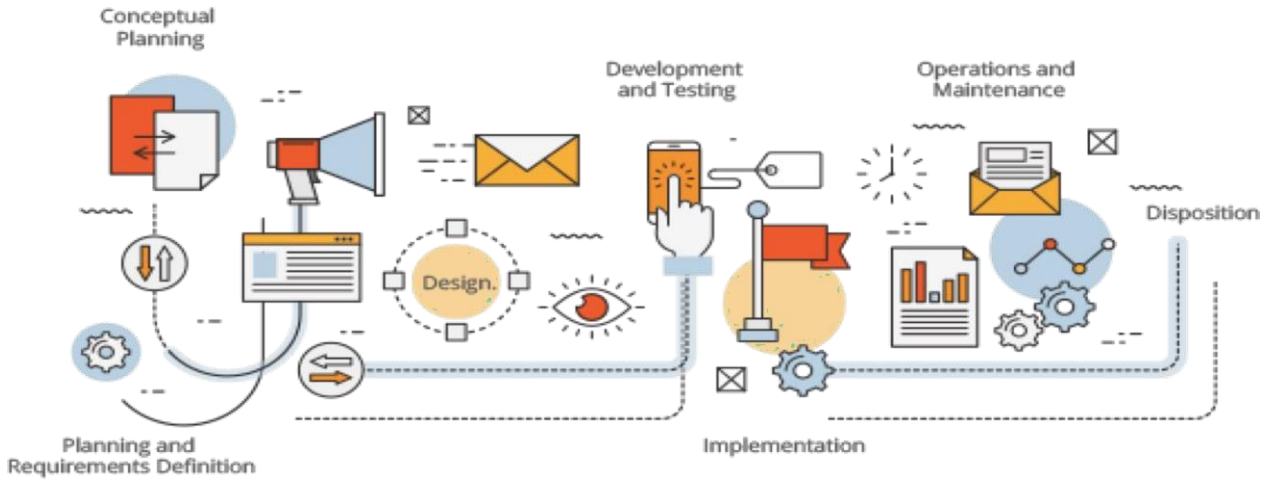
Software Engineering

Software Process Models
Lecture#05



Spiral Model



Spiral Model

- The **Spiral Model** is an evolutionary software development process that combines elements of the **Waterfall Model** and **Iterative Prototyping** while emphasizing **risk management**.

- Highlight **risk management and continuous improvement**, the Spiral Model helps to mitigate potential issues and deliver high-quality software.

- It has four stages:
    1. Objectives determination and identify alternative solutions
    2. Identify and resolve Risks
    3. Develop next version of the product
    4. Review and plan for the next phase

Key Features:

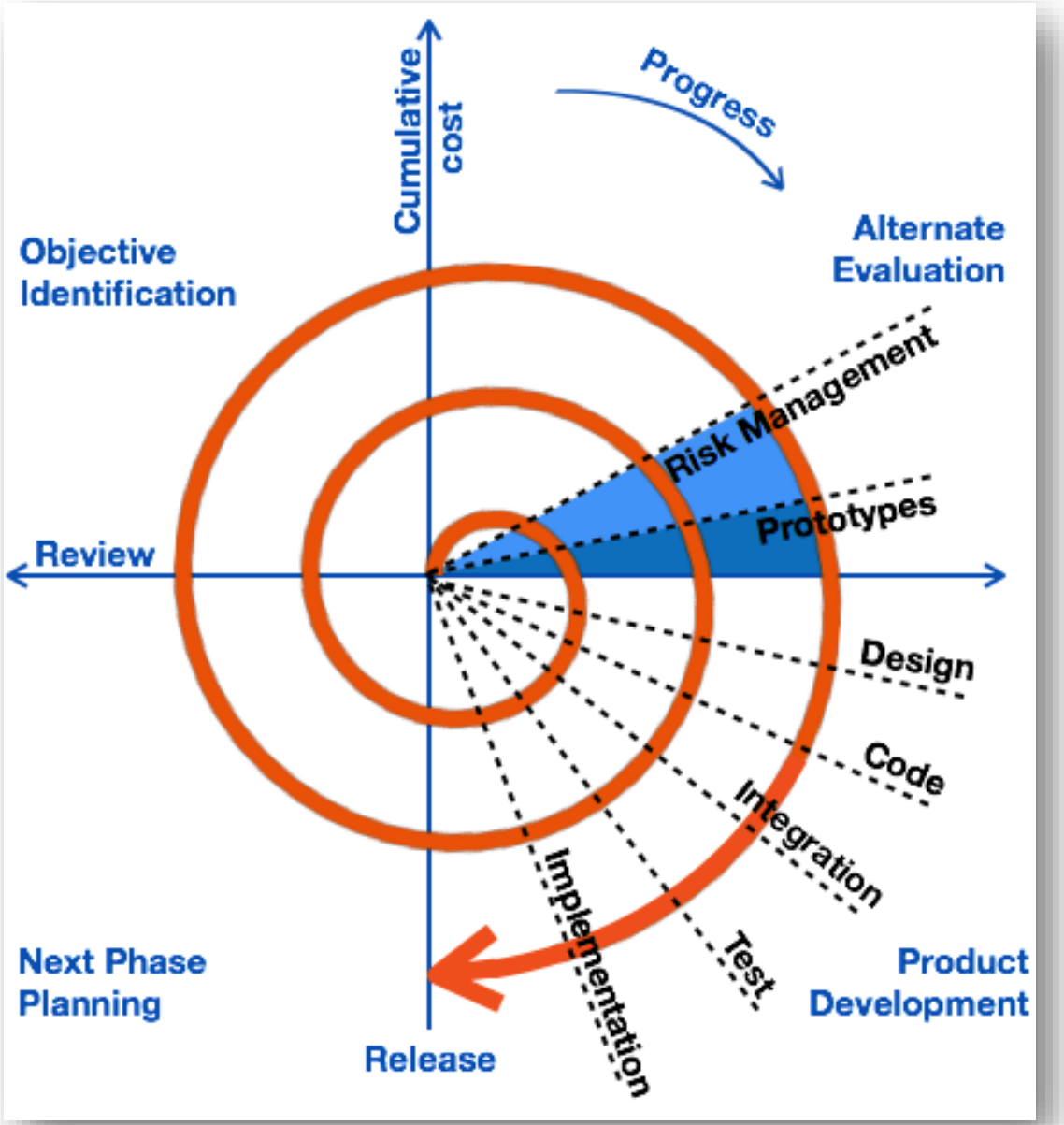
1. **Iterative Nature**: The model progresses in cycles (spirals), with each cycle representing a phase of the project.

2. **Risk Management**: Each iteration begins with identifying and analyzing risks, making it a risk-driven approach.

3. **Flexibility**: It allows for adjustments and refinements at each stage, making it adaptable to changing requirements.

4. **Combination of Models**: It integrates the systematic aspects of the waterfall model with the iterative nature of agile methodologies.

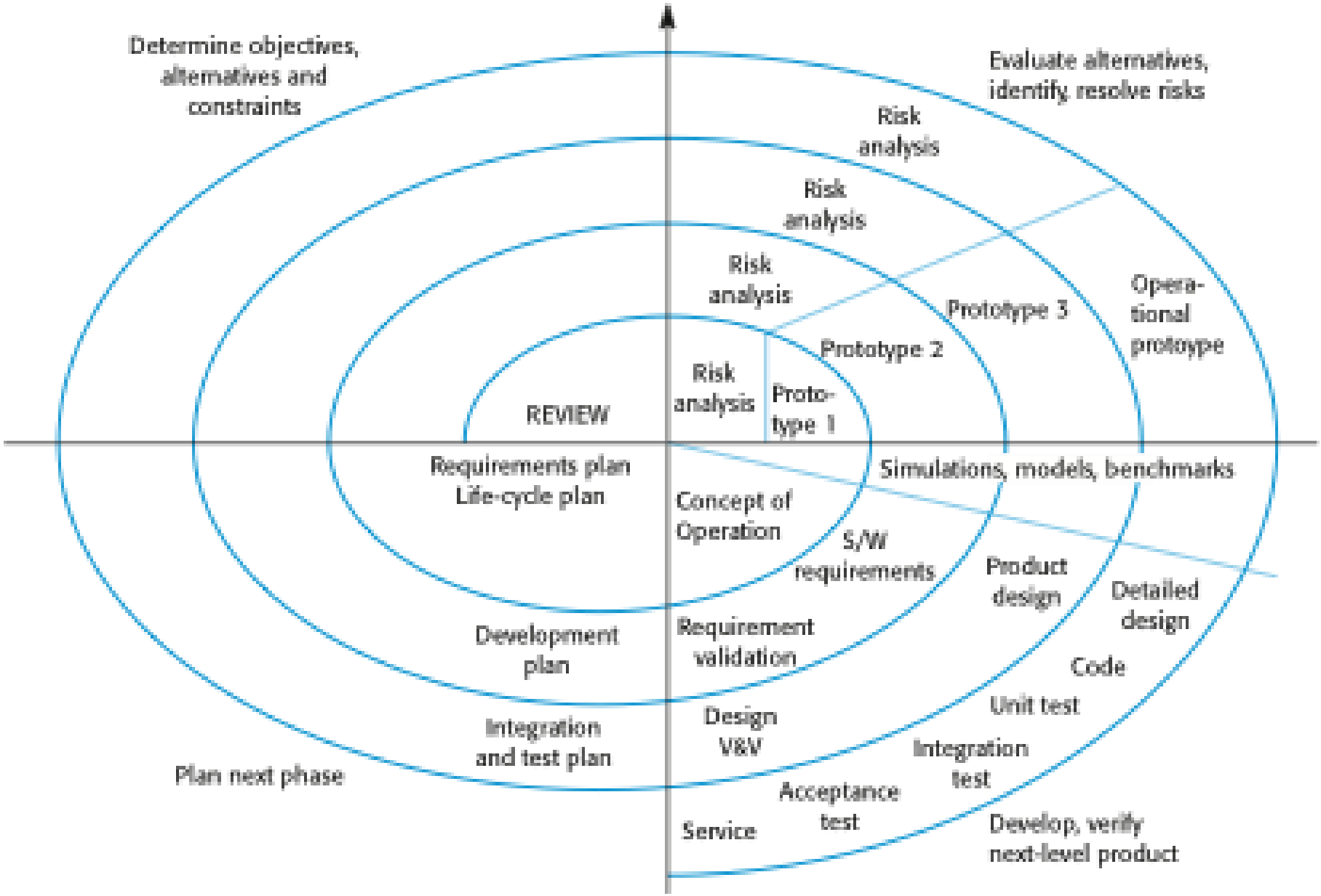
This phase **includes requirement gathering and analysis**. Based on the requirements, objectives are defined and different alternate solutions are proposed.



In this quadrant, all the proposed solutions are analyzed and any **potential risk is identified**, analyzed, and resolved.

The **software is evaluated by the customer**. It also includes risk identification and monitoring like cost overrun or schedule slippage and after that planning of the next phase is started.

This phase includes the **actual implementation** of the different features. All the implemented features are then verified with thorough testing.


Determine objectives, alternatives and constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Prototype 3
Opera-tional prototype
Prototype 2
Proto-type 1
REVIEW
Requirements plan
Life-cycle plan
Concept of Operation
Simulations, models, benchmarks
S/W requirements
Product design
Detailed design
Code
Unit test
Integration test
Acceptance test
Develop, verify next-level product
Requirement validation
Design V&V
Development plan
Integration and test plan
Plan next phase
Service

Project Scenario:

A bank wants to develop a secure online banking system where users can **check account balances, transfer funds, pay bills, and manage their accounts**. Since security is a major concern and requirements may change over time, the **Spiral Model** is a suitable approach.

Spiral 1: Planning & Requirement Analysis

- **Purpose**: Define the goals, constraints, and alternatives for the current iteration.
- **Activities**:
  - Identify the objectives of the phase (e.g., requirements gathering, prototyping, or testing).
  - Determine the scope of the iteration.
  - Plan resources, timelines, and deliverables.

2. Risk Analysis

- **Purpose**: Identify and mitigate potential risks that could impact the project.
- **Activities**:
  - Identify risks (e.g., technical, operational, or schedule-related).
  - Analyze the impact and probability of each risk.
  - Develop strategies to mitigate or resolve the risks.
- **Outcome**: A risk management plan is created to address the identified risks.

3. Engineering (Development and Testing)

- **Purpose**: Develop and validate the product for the current iteration.
- **Activities**:
  - Design and develop the product or a prototype based on the objectives.
  - Perform testing (e.g., unit testing, integration testing) to ensure quality.
  - Gather feedback from stakeholders.
- **Outcome**: A working product or prototype is delivered at the end of the iteration.

4. Evaluation and Planning for the Next Iteration

- **Purpose**: Review progress and plan the next steps.

- **Activities**:
  - Evaluate the results of the current iteration (e.g., whether objectives were met, risks were mitigated, and stakeholders are satisfied).
  - Decide whether to proceed to the next iteration or terminate the project if goals are achieved or risks are too high.
  - Plan the next iteration, including objectives, risks, and deliverables.

- **Outcome**: A decision is made to continue or end the project, and the next spiral begins.

Why is the Spiral Model Called the Meta Model?

The Spiral model is a Meta-Model that includes all other SDLC models. For instance, a single loop spiral represents the Iterative Waterfall Model. The spiral model combines the stepwise approach of the Classical Waterfall Model. It also uses the Prototyping Model by creating a prototype at the beginning of each phase to manage risks. Additionally, the spiral model supports the Evolutionary model as the iterations in the spiral can be seen as levels of evolution for the entire system.

When Should We Use the Spiral Model?

- Large-scale projects by breaking down work into manageable chunks.
- When frequent releases become vital.
- When there is a requirement to create a prototype.
- When risks and cost evaluations are necessary.
- When the risk of a project is moderate to high.
- When the requirements are complex.
- When modifications are required and possible anytime.

Spiral Model Disadvantages

1. Because of the prototype development and risk analysis in each phase, it is very **expensive and time taking**.

2. It is **not suitable for a simpler and smaller** project because of multiple phases.

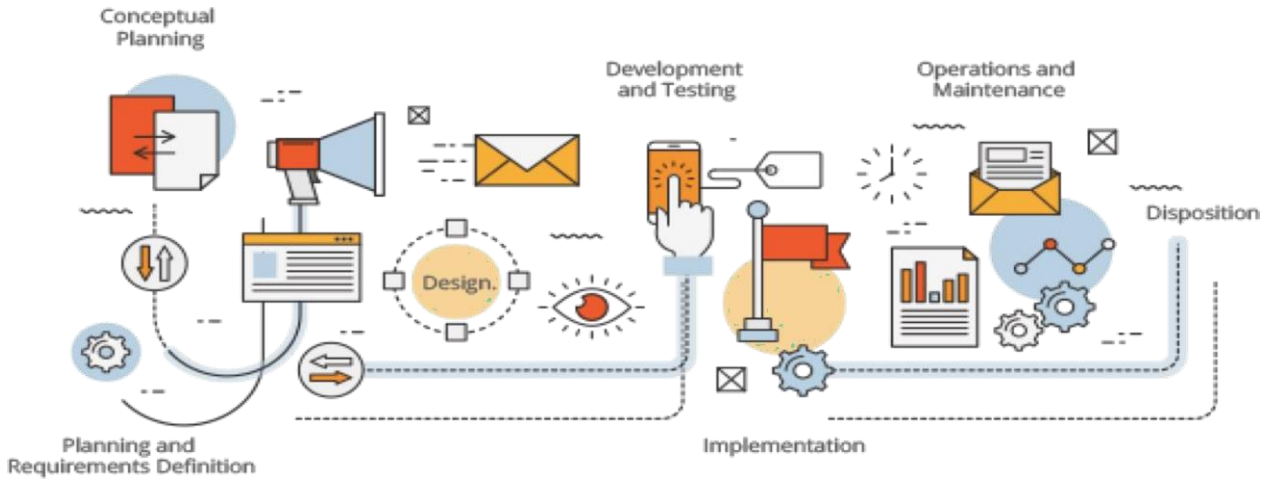
3. It requires **more documentation** as compared to other models.

4. Project **deadlines,** frequent prototyping and risk analysis can make things worse.

Summary

- Iterative as incremental waterfall
- Sequential as classical model
- Involves prototypes
- **Advantages**
  - Risk handling
  - Flexibility in requirements
  - Customer satisfaction
  - Good for large projects
- **Cons**
  - Complex, expensive, difficulty in time management, depends on risk analysis

Rapid Application Development Model



Rapid Application Development (RAD)

- The **RAD Model (Rapid Application Development)** is a software development methodology that focuses on **rapid prototyping** and **iterative delivery** to produce high-quality systems quickly. It emphasizes **user involvement**, **reusable components**, and **collaboration** to accelerate the development process. RAD is particularly suited for projects with **well-defined requirements** and a **short development timeline**.

- RAD is the foundation of many modern Agile & DevOps practices, making it a valuable approach in today's fast-paced tech industry.

Key characteristics of RAD include:

1. **Iterative Development**: RAD breaks the development process into small, manageable iterations or increments, typically called "rapid prototypes" or "cycles." Each cycle focuses on delivering a specific set of features or functionalities.

2. **Prototyping**: RAD relies heavily on prototyping to gather user feedback and refine requirements early in the development process. Prototypes are quickly developed and tested with end-users to validate functionality and gather feedback for further refinement.

3. **User Involvement**: RAD emphasizes active involvement of end-users and stakeholders throughout the development process. This involvement ensures that the final product meets user needs and requirements effectively.

4. **Incremental Delivery**: RAD emphasizes delivering working software incrementally and continuously throughout the development process. This allows users to start using and providing feedback on the software sooner, facilitating early validation and course corrections.

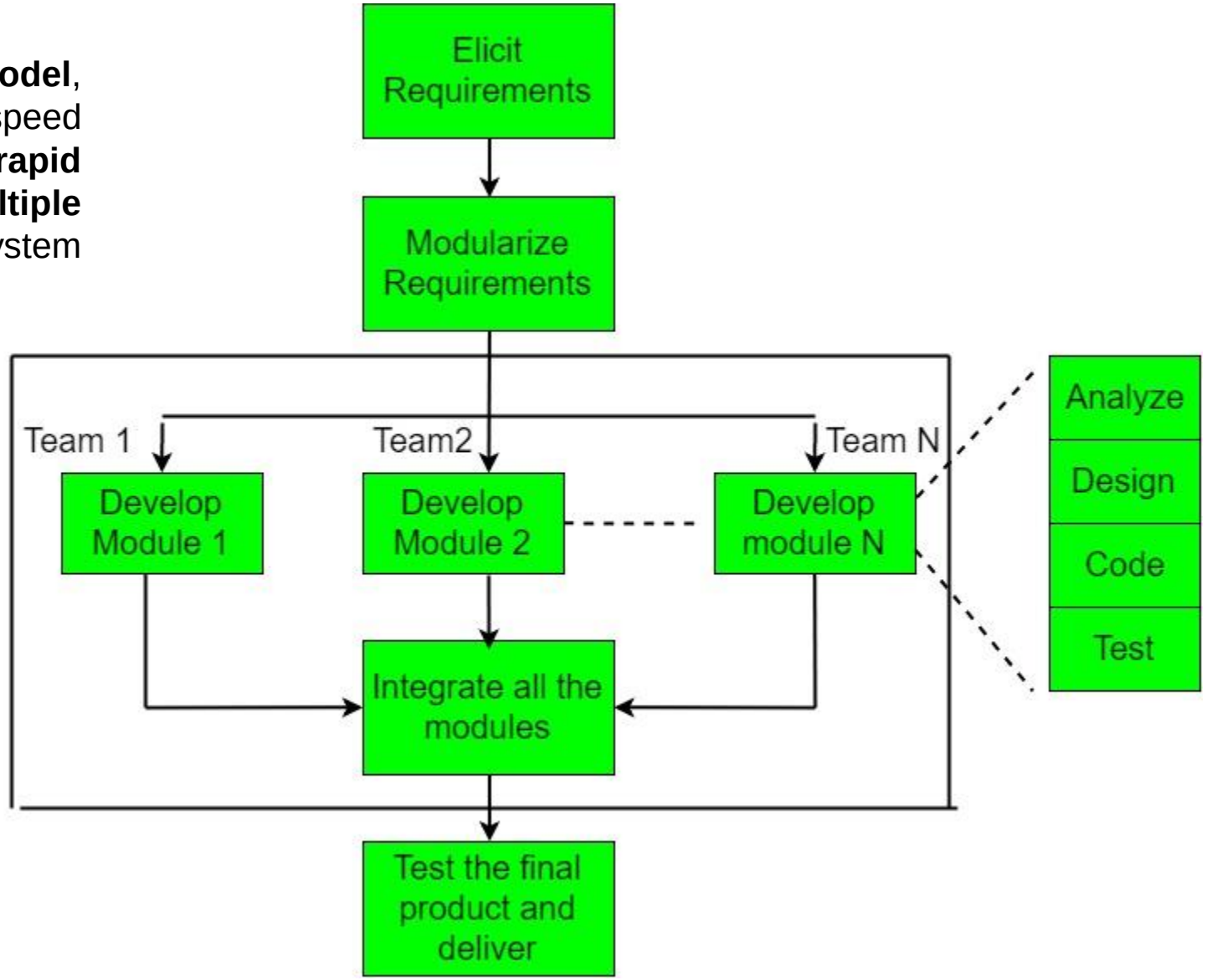
Key characteristics of RAD include:

5. **Timeboxing**: RAD projects are often time-constrained, with fixed deadlines for each iteration or cycle. Timeboxing ensures that development efforts remain focused and that the project stays on track to meet its objectives.

6. **Cross-functional Teams**: RAD encourages the formation of cross-functional teams that include developers, designers, testers, and stakeholders. These teams collaborate closely to rapidly iterate on software development and address any issues or feedback quickly.

7. **Reusability and Automation**: RAD promotes the reuse of existing components and tools to accelerate development and reduce time-to-market. Automation tools and techniques are also employed to streamline development processes and improve efficiency.

In the **Rapid Application Development (RAD) model**, teams **work in parallel on different modules** to speed up development. Since RAD emphasizes **rapid prototyping and iterative improvements**, **multiple teams** handle different parts of the system **simultaneously**.



Phases of RAD

1. **Requirements Planning**:
   - Define project goals, scope, and requirements with stakeholders.

2. **User Design**:
   - Create prototypes and refine designs based on user feedback.

3. **Rapid Construction**:
   - Build the system iteratively using reusable components.

4. **Cutover (Deployment)**:
   - Test, train users, and deploy the final system.

When to Use a RAD Model

RAD model is most effective when used in projects that require rapid development, high user involvement, well-defined requirements, prototype-driven development, and have a skilled and collaborative development team. It may not be suitable for all projects, so it's essential to evaluate the specific characteristics and requirements of each project before deciding whether to use the RAD model.

1. **E-commerce Website Development**: Developing an e-commerce website where quick iteration based on user feedback is crucial. RAD allows for rapid prototyping and frequent updates to improve user experience, add new features, and adapt to changing market trends.

2. **Internal Business Tools**: Creating internal business tools such as CRM systems, inventory management systems, or project management tools where requirements are well-understood and can be quickly validated by end-users. RAD enables fast development and customization of these tools to meet specific business needs.

3. **Mobile App Development**: Developing mobile applications where speed-to-market is essential, and user engagement is critical. RAD allows for rapid development and testing of mobile app prototypes across multiple platforms, enabling quick iteration based on user feedback.

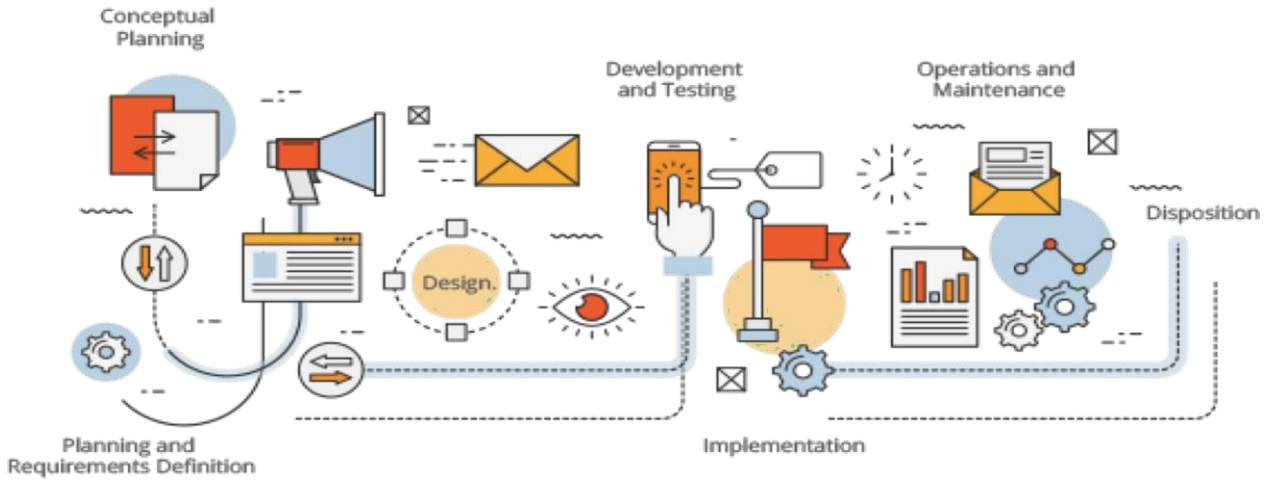
Drawbacks

- For large scalable projects RAD **requires sufficient human resources** to create right number of RAD teams

- RAD requires developers customers committed to complete a system in a short time frame, other wise if commitment is lacking from either side, RAD projects will fail.

- Rushed development iterations may lead to overlooked defects and lower overall software quality.
  - Quick iterations may **skip proper testing**, leading to hidden bugs.
  - Focus on speed over structure may result in **poor code maintainability**.

Rational Unified Process (RUP)



Rational Unified Process (RUP)

- The Rational Unified Process (RUP) is an iterative software development process framework developed by Rational Software Corporation, which was later acquired by IBM.
- RUP provides a disciplined approach to software development that emphasizes iterative development, continuous improvement, and collaboration among team members.
- It is based on a set of best practices drawn from industry standards and incorporates principles from object-oriented analysis and design.
- **RUP** is a more **structured and detailed approach** that focuses on managing risks and delivering **high-quality, scalable** systems, making it ideal for **large, complex projects**.

1. **Iterative and Incremental**: RUP is based on an iterative and incremental development approach, where the development cycle is divided into multiple iterations or phases. Each iteration produces a working increment of the software, allowing for continuous refinement and improvement based on feedback.

2. **Use-Case Driven**: RUP emphasizes the use of use cases to capture and model system requirements from the perspective of end-users. Use cases serve as a primary mechanism for defining the functionality of the system and guiding the development process.

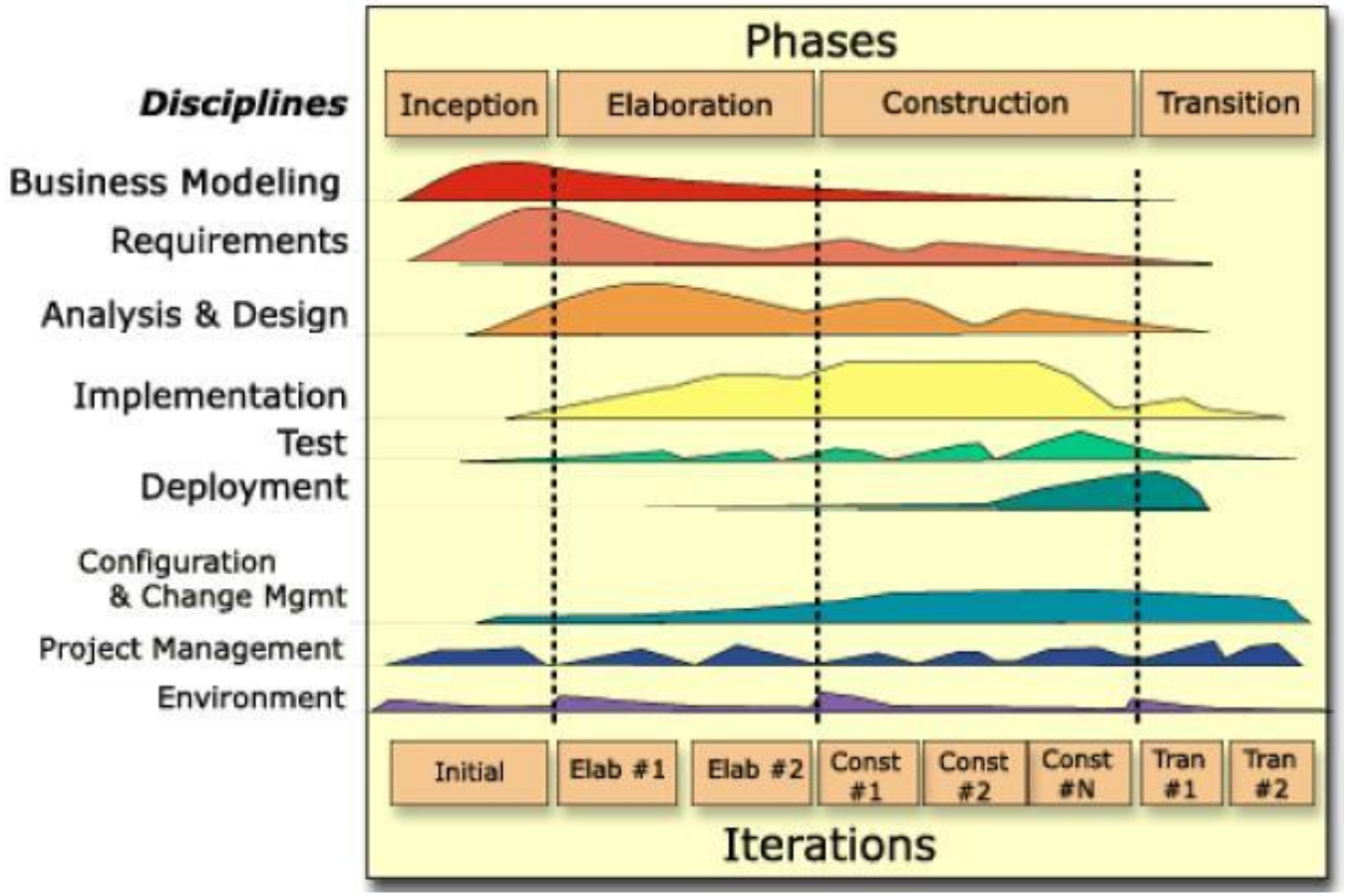
3. **Architecture-Centric**: RUP places a strong emphasis on defining and maintaining a robust software architecture throughout the development lifecycle. Architecture artifacts, such as architectural models, component diagrams, and design patterns, are used to guide the development process and ensure architectural integrity.

4. **Component-Based:** RUP promotes a component-based development approach, where software systems are decomposed into reusable components. Components encapsulate specific functionality and can be reused across different projects, improving productivity and maintainability.

5. **Risk-Driven:** RUP is driven by a focus on identifying and mitigating project risks early in the development process. Risk management activities, such as risk identification, analysis, and mitigation planning, are integrated into the development lifecycle to ensure that potential risks are addressed proactively.

6. **Roles and Responsibilities**: RUP defines specific roles and responsibilities for team members involved in the development process, including roles such as project manager, architect, analyst, designer, and developer. Each role has defined tasks and deliverables, promoting clear communication and collaboration among team members.

7. **Tailorable and Scalable**: RUP is designed to be tailorable and scalable to meet the specific needs of different projects and organizations. It provides a flexible framework that can be customized based on project size, complexity, and requirements, allowing organizations to adapt RUP to their unique circumstances.



These phases are **not strictly sequential**, and RUP encourages **iterative and overlapping activities** throughout the development process. Each phase is **divided into iterations**, with each iteration producing a working increment of the software. Iterative development allows for continuous feedback, adaptation, and improvement throughout the project lifecycle.

Phase-1: Inception Phase:

- **Objective**: The Inception phase aims to establish the business case and feasibility of the project. It involves identifying the project scope, business objectives, stakeholders, and high-level requirements.

- **Activities**: During this phase, activities include conducting a feasibility study, defining the project vision and scope, identifying risks, establishing the development team, and creating an initial project plan.

- **Deliverables**: The key deliverables of the Inception phase include the Vision document, Business Case, Initial Use-Case Model, Project Plan, and Risk Assessment.

Phase-2: Elaboration Phase:

- **Objective**: The Elaboration phase focuses on refining the project vision, defining the system architecture, and mitigating key risks. It involves detailed requirements analysis, architectural design, and prototyping.

- **Activities**: Activities in this phase include elaborating on use cases, refining the system architecture, identifying and mitigating high-priority risks, prototyping critical components, and creating a detailed project plan.

- **Deliverables**: Key deliverables of the Elaboration phase include the Software Architecture Document, Refined Use-Case Model, Supplementary Specifications, Executable Architecture Prototype, and updated Project Plan.

Phase-3: Construction Phase:

- **Objective**: The Construction phase is where the majority of the system development occurs. It involves implementing and testing the system components, integrating them into a cohesive whole, and preparing for deployment.

- **Activities**: Activities in this phase include iterative development of system components, coding, unit testing, integration testing, defect resolution, performance optimization, and user documentation.

- **Deliverables**: Key deliverables of the Construction phase include the Executable Software Build, User Manual, System Integration Test Plan, and Test Results.

Phase-4: Transition Phase:

- **Objective**: The Transition phase focuses on transitioning the system from development to production. It involves final testing, user training, deployment, and transitioning support and maintenance to operational teams.

- **Activities**: Activities in this phase include final system testing, user acceptance testing (UAT), deployment planning, user training, data migration, transitioning support responsibilities, and project closure.

- **Deliverables**: Key deliverables of the Transition phase include the System Acceptance Test Report, Deployment Plan, User Training Materials, System Documentation, and Lessons Learned Report.

RUP is best suited…

RUP is best suited for projects that require a disciplined, structured, and adaptable approach to software development. It is particularly effective for large-scale, long-term, and requirements-intensive projects where collaboration, flexibility, and quality assurance are critical to project success.

Projects that have utilized the Rational Unified Process (RUP) methodology over the years

1. **NASA's Space Shuttle Program**: NASA reportedly used RUP for software development and project management in various aspects of the Space Shuttle program, including mission control systems and software for on-board operations.

2. **Swiss Railway Ticketing System**: The Swiss Federal Railways (SBB) reportedly used RUP for the development of its ticketing system, which handles ticket sales, reservations, and travel planning for the Swiss rail network.

3. **IBM Rational Suite**: IBM, the company behind RUP, has utilized its own methodology in various software development projects, including the development of tools and software products such as the IBM Rational Suite, which includes tools for requirements management, design, testing, and project management.

4. **Banking and Financial Systems**: Various banks and financial institutions have reportedly employed RUP for the development of banking systems, financial trading platforms, and other software applications related to the finance industry.

5. **Telecommunications Systems**: Companies in the telecommunications industry, including providers of mobile networks, telecommunication infrastructure, and related services, have utilized RUP for the development of software systems supporting their operations.

Summary of Key Differences:

Feature	RUP (Rational Unified Process)	RAD (Rapid Application Development)
Development Approach	Structured, phased, risk-focused	Flexible, prototype-driven, speed-focused
Phases	Inception, Elaboration, Construction, Transition	Requirements, Design, Construction, Deployment
User Involvement	Involvement at specific phases	Constant user feedback throughout development
Risk Management	Emphasizes early risk mitigation	Minimal risk management focus
Suitable For	Large, complex systems with strong architecture	Smaller to medium projects requiring quick deployment
Documentation	Heavy documentation, structured process	Minimal documentation, informal process

Guess the Model

1. **A company is building self-driving car software that needs to be safe, reliable, and capable of real-time decision-making.**

Best Software Model: Spiral Model

Why?

- **High Risk** → Safety is a priority (collision detection, sensor failures).
- **Multiple Iterations Needed** → AI needs

2. A startup wants to develop a food delivery app quickly to enter the market before competitors.

Best Software Model: RAD Model

Why?

✓ **Fast Development:** A prototype can be created in weeks instead of months.
✓ **Customer Feedback Drives Changes:** UI/UX and features can be adjusted based on user testing.
✓ **Flexibility:** New restaurant partnerships and payment methods can be added in future iterations.