UNIT II

MU – SPM – SEM 5

Selection of an Appropriate Project Approach

- Selecting an appropriate project approach is crucial to ensure project success. It involves choosing the right methodology and strategies to plan, execute, and complete the project effectively. Here's a guide on how to select the best project approach:
- **Understand Project Requirements**
  - **Assess Objectives and Scope**: Clearly define what the project aims to achieve and the extent of work involved.
  - **Identify Stakeholder Needs**: Gather requirements from all stakeholders to ensure their expectations are met.

- **Evaluate Project Characteristics**
  - **Project Size and Complexity**: Determine the project's scale and intricacy, which will influence the choice of approach.
  - **Duration and Timeline**: Consider the project’s time constraints and deadlines.
  - **Technical Requirements**: Assess the technological needs and innovations required.
- **Consider Resource Availability**
  - **Team Skills and Experience**: Evaluate the competencies and expertise of the project team.
  - **Budget Constraints**: Understand the financial resources available.
  - **Tools and Infrastructure**: Identify the tools and infrastructure necessary for the project.

- **Analyze Risks and Uncertainties**
  - **Identify Potential Risks**: List possible risks and uncertainties that could impact the project.
  - **Risk Management Strategies**: Choose an approach that includes effective risk mitigation and management strategies.
- **Determine Flexibility and Adaptability Needs**
  - **Scope for Changes**: Assess how frequently requirements may change.
  - **Stakeholder Involvement**: Consider the level of stakeholder engagement needed throughout the project.

- **Review Methodologies**
  - **Traditional (Waterfall)**: Suitable for projects with well-defined requirements and minimal changes expected. Follows a linear, sequential design.
  - **Agile**: Best for projects with dynamic requirements and a need for flexibility. Involves iterative development and frequent stakeholder feedback.
  - **Hybrid**: Combines elements of both Waterfall and Agile methodologies to balance structure and flexibility.
- **Align with Organizational Practices**
  - **Existing Processes**: Consider the organization's existing project management practices and frameworks.
  - **Cultural Fit**: Ensure the approach aligns with the organizational culture and values.

- **Decision-Making and Approval**
  - **Consult Stakeholders**: Involve key stakeholders in the decision-making process to gather input and gain buy-in.
  - **Document the Decision**: Clearly document the chosen approach, including the rationale behind the decision.

Choosing Methodologies and Technologies

- **Choosing Methodologies**
- **Assess Project Requirements**:
  - **Scope and Objectives**: Define what needs to be achieved and the extent of the work.
  - **Stakeholder Needs**: Ensure that stakeholder expectations and requirements are clearly understood.
- **Evaluate Project Characteristics**:
  - **Size and Complexity**: Larger, more complex projects may require more structured methodologies.
  - **Duration and Timeline**: Short timelines may benefit from more flexible approaches.

- **Consider Methodologies**:
  - **Waterfall**: Ideal for projects with clear, fixed requirements and well-defined stages. It follows a linear, sequential process.
  - **Agile**: Suitable for projects with evolving requirements and a need for flexibility. It involves iterative development and frequent stakeholder feedback.
  - **Scrum**: A form of Agile that focuses on short, fixed iterations (sprints) and regular reviews.
  - **Hybrid**: Combines elements of Waterfall and Agile to balance structure with flexibility.

- **Match Methodology to Project Needs**:
  - **Fixed Requirements**: Use Waterfall or similar methodologies.
  - **Evolving Requirements**: Use Agile, Scrum, or Kanban.
  - **Balance Needed**: Use a Hybrid approach.

Choosing Technologies

- **Identify Project Needs:**
  - Technical Requirements: Determine the specific technical needs based on project goals.
  - Integration: Ensure the technology can integrate with existing systems and tools.
- **Evaluate Available Technologies:**
  - Tools and Platforms: Consider tools that support the chosen methodology (e. g., Jira for Agile, Microsoft Project for Waterfall).Development
  - Frameworks: Choose frameworks that fit the project’s technical requirements (e.g., React for web development, TensorFlow for machine learning).

- **Consider Scalability and Flexibility**:
  - **Scalability**: Ensure the technology can handle the projected growth and increased workload.
  - **Flexibility**: Choose technologies that can adapt to changes in project scope or requirements.
- **Assess Resource Availability**:
  - **Skills and Expertise**: Ensure the team has the necessary skills or can quickly learn the new technology.
  - **Support and Community**: Choose technologies with strong support and active communities for troubleshooting and guidance.

- **Budget and Cost Constraints:**
  - Evaluate the costs associated with acquiring, implementing, and maintaining the technology within the project budget.
- **Security and Compliance:**
  - Ensure the chosen technologies comply with relevant security standards and regulatory requirements.
- **Test and Evaluate**:
  - Conduct pilot tests to evaluate the technology’s performance and suitability for the project before full-scale implementation.

Software Processes and Process Models

- **Software Processes**
- A software process is a set of related activities that lead to the production of a software product. These processes provide a structured approach to software development, ensuring the product is built efficiently and meets quality standards.

Key Activities in Software Processes

- **Specification**:
  - **Description**: Define the software's functionalities and constraints.
  - **Activities**: Requirement gathering, analysis, and documentation.
  - **Outcome**: A comprehensive requirement specification document.
- **Design**:
  - **Description**: Create the architecture and detailed design of the software.
  - **Activities**: System design, data design, interface design.
  - **Outcome**: Design documents, diagrams, and prototypes.

- **Implementation**:
  - **Description**: Convert the design into executable code.
  - **Activities**: Coding, unit testing, debugging.
  - **Outcome**: Source code and tested components.
- **Testing**:
  - **Description**: Verify the software against requirements to ensure it is free of defects.
  - **Activities**: Integration testing, system testing, acceptance testing.
  - **Outcome**: Test plans, test cases, and test reports.
- **Maintenance**:
  - **Description**: Modify the software to correct issues, improve performance, or adapt to changes.
  - **Activities**: Bug fixing, performance enhancement, updating documentation.
  - **Outcome**: Updated software and documentation.

Software Process Models

- **Common Software Process Models**
- **Waterfall Model**:
  - **Description**: A linear and sequential approach where each phase must be completed before the next begins.
  - **Advantages**: Simple to manage; well-suited for projects with clear requirements.
  - **Disadvantages**: Inflexible to changes; late detection of defects.
- **Incremental Model**:
  - **Description**: Develops software in increments or iterations, each adding functionality.
  - **Advantages**: Delivers functional software early; easier to manage changes.
  - **Disadvantages**: Requires careful planning; complex integration.

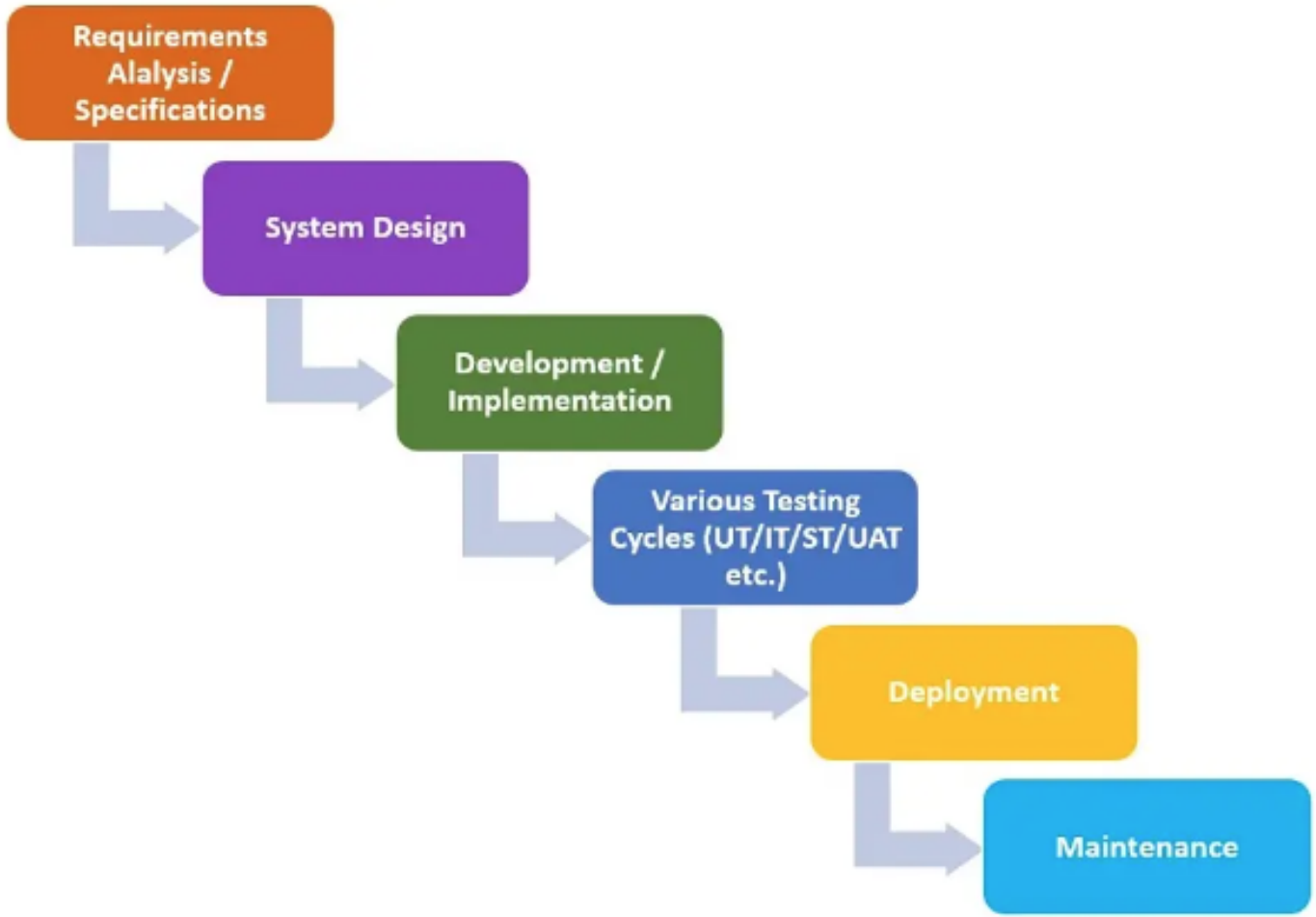
- **Spiral Model**:
  - **Description**: Combines iterative development with systematic risk analysis. Each iteration is a spiral of activities.
  - **Advantages**: Focuses on risk management; iterative refinement.
  - **Disadvantages**: Complex to manage; requires expertise in risk analysis.
- **V-Model (Verification and Validation Model)**:
  - **Description**: An extension of the Waterfall model that emphasizes verification and validation at each development stage.
  - **Advantages**: Early defect detection; clear testing phases.
  - **Disadvantages**: Inflexible; challenging to accommodate changes

- **Agile Model**:
  - **Description**: Emphasizes iterative development, customer collaboration, and flexibility to change.
  - **Advantages**: Adaptable to changes; continuous customer feedback.
  - **Disadvantages**: Requires high customer involvement; less predictable.
- **Scrum**:
  - **Description**: An Agile framework focusing on iterative progress through small, cross-functional teams. Work is divided into sprints.
  - **Advantages**: Flexibility; team collaboration; rapid delivery.
  - **Disadvantages**: Requires disciplined teams; challenging for large projects.

Choosing the Right Model

- **Project Requirements**: Well-defined and stable requirements are suited to Waterfall or V-Model. Evolving requirements fit Agile or Incremental models.
- **Risk Management**: The Spiral model is ideal for high-risk projects.
- **Flexibility**: Agile methodologies like Scrum and Kanban provide high adaptability.
- **Customer Involvement**: Agile models necessitate active customer participation, whereas Waterfall and V-Model involve customers mainly at the beginning and end.

The Waterfall Model



The Waterfall Model

- The Waterfall Model is one of the earliest and most traditional software development methodologies. It follows a linear and sequential approach, where each phase must be completed before the next one begins.
- This model is named "Waterfall" because the progress flows downwards like a waterfall through the various phases of the software development lifecycle (SDLC).

Key Phases of the Waterfall Model

- **Requirements Analysis**
- **Objective**: Gather and document all software requirements.
- **Activities**:
  - Conduct stakeholder interviews and meetings.
  - Gather detailed functional and non-functional requirements.
  - Document requirements in a Software Requirements Specification (SRS) document.
- **Outcome**: SRS document which serves as a guideline for the subsequent phases.

- **System Design**
- **Objective**: Create the system architecture and detailed design based on the requirements.
- **Activities**:
  - Develop system architecture and high-level design.
  - Create detailed design for each system component.
  - Document design in Design Specification Document (DSD).
- **Outcome**: DSD which includes architecture diagrams, data models, and interface designs.

- **Implementation (Coding)**
- **Objective**: Translate the system design into executable code.
- **Activities**:
  - Write code for each component based on the design specifications.
  - Conduct unit testing to ensure individual components work correctly.
- **Outcome**: Source code and unit-tested components.

- **Integration and Testing**
- **Objective**: Integrate all components and verify the entire system.
- **Activities**:
  - Integrate coded components to form the complete system.
  - Conduct various levels of testing (integration testing, system testing).
  - Fix bugs and issues identified during testing.
- **Outcome**: A fully integrated and tested system ready for deployment.

- **Deployment**
- **Objective**: Deploy the software to the production environment.
- **Activities**:
  - Prepare deployment plan and environment.
  - Install and configure the software in the production environment.
  - Conduct user training and documentation.
- **Outcome**: Software is deployed and available for use by end-users.

- **Maintenance**
  - **Objective**: Maintain and update the software post-deployment.
  - **Activities**:
    - Monitor the software for issues and bugs.
    - Provide patches and updates as needed.
    - Enhance functionality based on user feedback.
  - **Outcome**: Updated and improved software over its lifecycle.

Advantages of the Waterfall Model

1. **Simplicity and Ease of Use:**
    - Easy to understand and implement due to its straightforward, linear approach.
2. **Structured Approach:**
    - Each phase has specific deliverables and a review process, ensuring thorough documentation and accountability.
3. **Clear Milestones:**
    - Defined stages with clear milestones make project tracking straightforward.
4. **Ease of Management:**
    - The sequential nature makes it easier to manage and control project progress.

Disadvantages of the Waterfall Model

Inflexibility to Changes:

- Difficulty accommodating changes once a phase is completed. Changes require going back to earlier phases, which is costly and time-consuming.

Late Testing:

- Testing is conducted after the implementation phase, leading to late discovery of defects, which can be expensive to fix.

Limited Customer Involvement:

- Customer feedback is primarily gathered at the beginning (requirements phase) and end (testing phase), limiting the opportunity for ongoing input and adjustments.

Constructive Cost Model (COCOMO)

- The Constructive Cost Model (COCOMO) is a software cost estimation model that helps predict the effort, cost, and schedule required for a software development project. Developed by Barry Boehm in 1981, COCOMO uses a mathematical formula based on the size of the software project, typically measured in lines of code (LOC).
- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** This simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

Types of Projects in the COCOMO Model

- **Organic:** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
- **Semi-detached**: A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.

- **Embedded:** A software project requiring the highest level of complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Importance of the COCOMO Model

- **Cost Estimation:** To help with resource planning and project budgeting, COCOMO offers a methodical approach to software development cost estimation.
- **Resource Management:** By taking team experience, project size, and complexity into account, the model helps with efficient resource allocation.
- **Project Planning**: COCOMO assists in developing practical project plans that include attainable objectives, due dates, and benchmarks.
- **Risk management**: Early in the development process, COCOMO assists in identifying and mitigating potential hazards by including risk elements.

- **Support for Decisions**: During project planning, the model provides a quantitative foundation for choices about scope, priorities, and resource allocation.
- **Benchmarking**: To compare and assess various software development projects to industry standards, COCOMO offers a benchmark.
- **Resource Optimization:** The model helps to maximize the use of resources,

Types of COCOMO Model

- Basic COCOMO Model
- Intermediate COCOMO Model
- Detailed COCOMO Model

- **Basic COCOMO Model**
  - The Basic COCOMO model is a straightforward way to estimate the effort needed for a software development project. It uses a simple mathematical formula to predict how many person-months of work are required based on the size of the project, measured in thousands of lines of code (KLOC).
  - $E = a*(KLOC)^b\ PM$
- $Tdev = c*(E)^d$
- *Person required = Effort/ Time*
- *Where,*
- *E is effort applied in Person-Months*
- *KLOC is the estimated size of the software product indicate in Kilo Lines of Code*
- *Tdev is the development time in months*
- *a, b, c are constants determined by the category of software project given in below table.*

Software Projects	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

- **Intermediate COCOMO Model**
  - The basic COCOMO model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software systems.
  - However, in reality, no system's effort and schedule can be solely calculated based on Lines of Code.
  - For that, various other factors such as reliability, experience, and Capability. These factors are known as **Cost Drivers (multipliers)** and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their Attributes:

- **Product attributes:**
  - Required software reliability extent
  - Size of the application database
  - The complexity of the product
- **Hardware attributes**
  - Run-time performance constraints
  - Memory constraints
  - The volatility of the virtual machine environment
  - Required turnabout time

- **Personal attributes**
  - Analyst capability
  - Software engineering capability
  - Application experience
  - Virtual machine experience
  - Programming language experience
- **Project attributes**
  - Use of software tools
  - Application of software engineering methods
  - Required development schedule

- *The **Effort Adjustment Factor (EAF)** is determined by multiplying the effort multipliers associated with each of the 15 attributes.*
- $E = a*(KLOC)^b * EAF\ PM$
- $Tdev = c*(E)^d$
- *Where,*
- *E is effort applied in Person-Months*
- *KLOC is the estimated size of the software product indicate in Kilo Lines of Code*
- *EAF is the Effort Adjustment Factor (EAF) is a multiplier used to refine the effort estimate obtained from the basic COCOMO model.*
- *Tdev is the development time in months*
- *a, b, c are constants determined by the category of software project given in below table.*

Software Projects	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Detailed COCOMO Model

- Detailed COCOMO goes beyond Basic and Intermediate COCOMO by diving deeper into project-specific factors. It considers a wider range of parameters, like team experience, development practices, and software complexity.
- By analyzing these factors in more detail, Detailed COCOMO provides a highly accurate estimation of effort, time, and cost for software projects. It’s like zooming in on a project’s unique characteristics to get a clearer picture of what it will take to complete it successfully.

COCOMO Model

Basic COCOMO Model: Example

E.g.1: Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes I.e., organic, semidetached and embedded

The basic COCOMO equation take the form:

$$E = a_b(KLOC)^{b_b}$$

$$D = c_b(E)^{d_b}$$

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi detected	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example :

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e. organic , semidetached and embedded.

Solution The basic COCOMO equations take the form:

$$E = a_b (KLOC)^{bb}$$

$$D = c_b (E)^{db}$$

Estimated size of the project = 400 KLOC

1. **Organic Mode**

$$E = 2.4 (400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5 (1295.31)^{0.38} = 38.07 \text{ M}$$

2. **Semi detached Mode**

$$E = 3.0 (400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5 (2462.79)^{0.35} = 38.45 \text{ M}$$

3. **Embedded Mode**

$$E = 3.6 (400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5 (4772.81)^{0.32} = 37.59 \text{ M}$$

- A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM

D=2.5(1133.12)0.35=29.3PM

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

Practice Question

- **Example 1:** For a given project was estimated with a size of 310 KLOC. Calculate the Effort, Scheduled time for development by considering the developer having high application experience and very low experience in programming.
- **Example 2:** For a given project was estimated with a size of 420 KLOC. Calculate the Effort, Scheduled time for development by considering the developer having very high product complexity and low application experience in programming.

Project Characteristics Table

Cost adjustments for computing the EAF (Effort Adjustment Factor)

	v.low	low	nominal	high	v.high	ex.high
product attributes						
required software reliability	0.75	0.88	1.00	1.15	1.40	
database size		0.94	1.00	1.08	1.16	
product complexity	0.70	0.85	1.00	1.15	1.30	1.65
computer attributes						
execution time constraints			1.00	1.11	1.30	1.66
main storage constraints			1.00	1.06	1.21	1.56
virtual machine volitility	0.87	1.00	1.15	1.30		
computer turnaround time		0.87	1.00	1.07	1.15	
personnel attributes						
analyst capability	1.46	1.19	1.00	0.86	0.71	
applications experience	1.29	1.13	1.00	0.91	0.82	
programmer capability	1.42	1.17	1.00	0.86	0.70	
virtual machine experience	1.21	1.10	1.00	0.90		
programming language experience	1.14	1.07	1.00	0.95		
project attributes						
use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
use of software tools	1.24	1.10	1.00	0.91	0.83	
required development schedule	1.23	1.08	1.00	1.04	1.10	

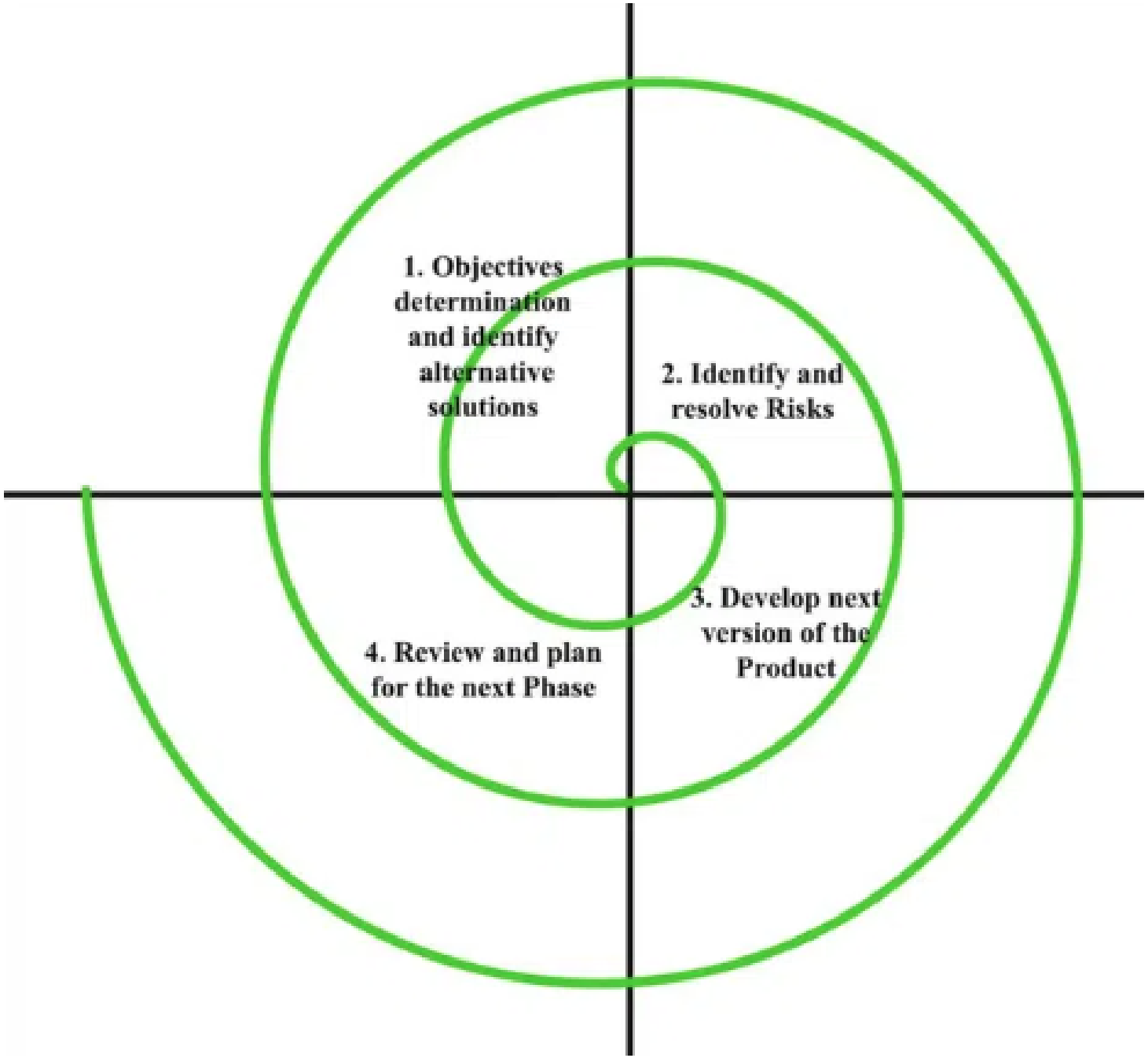
The Spiral Model

- **The Spiral Model** is one of the most important Software Development Life Cycle models. The Spiral Model is a combination of the waterfall model and the iterative model. It provides support for **Risk Handling**. The Spiral Model was first proposed by **Barry Boehm**. This article focuses on discussing the Spiral Model in detail.

- The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.
- As the project manager dynamically determines the number of phases, the project manager has an important role in developing a product using the spiral model.
- It is based on the idea of a spiral, with each iteration of the spiral representing a complete software development cycle, from requirements gathering and analysis to design, implementation, testing, and maintenance.

What Are the Phases of the Spiral Model?

- **Planning:** The first phase of the Spiral Model is the planning phase, where the scope of the project is determined and a plan is created for the next iteration of the spiral.
- **Risk Analysis:** In the risk analysis phase, the risks associated with the project are identified and evaluated.
- **Engineering:** In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.
- **Evaluation:** In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.
- **Planning:** The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.


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

Advantages of the Spiral Model

- **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
- **Flexibility in Requirements:** Change requests in the Requirements at a later phase can be incorporated accurately by using this model.
- **Customer Satisfaction:** Customers can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

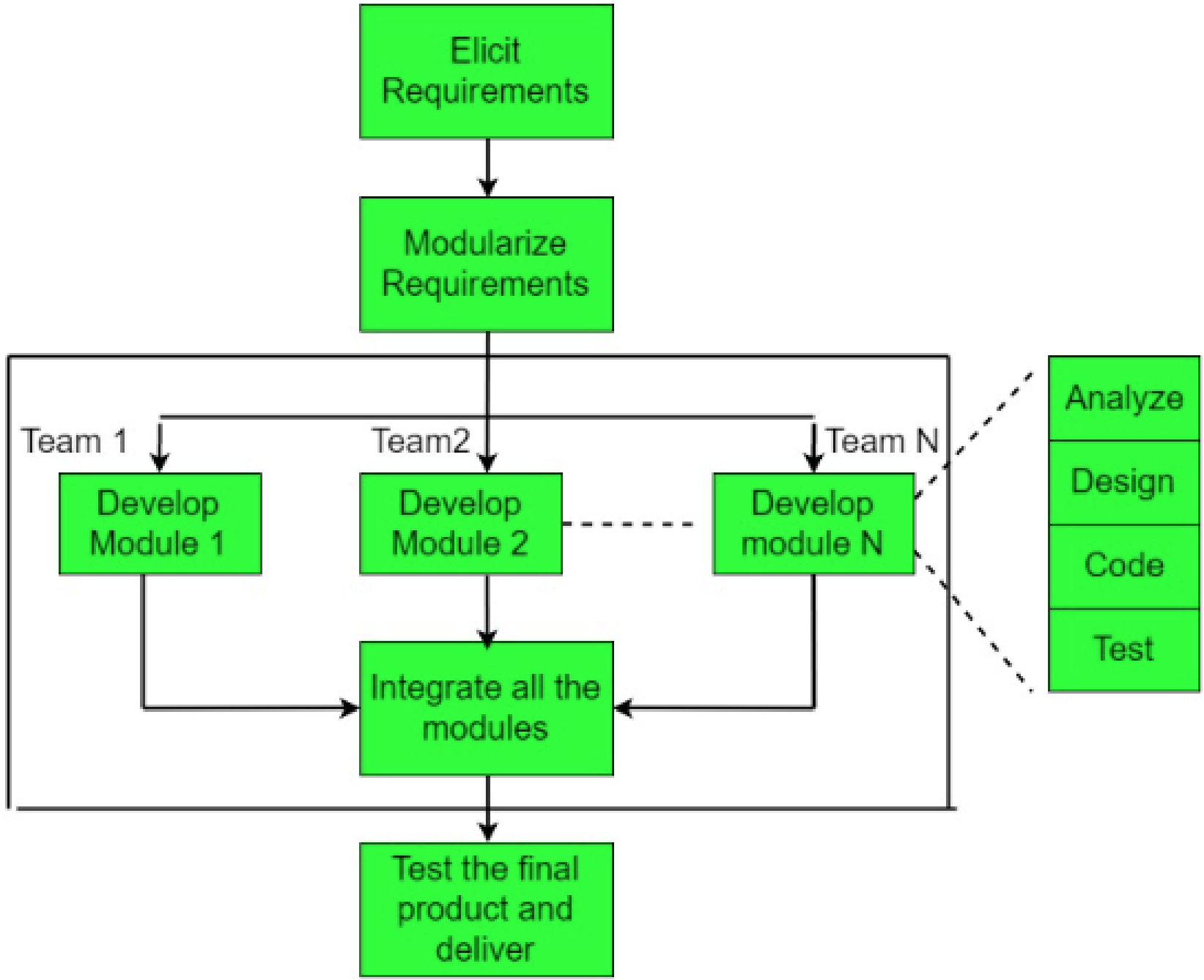
Disadvantages of the Spiral Model

- **Complex:** The Spiral Model is much more complex than other SDLC models.
- **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
- **Difficulty in time management:** As the number of phases is unknown at the start of the project, time estimation is very difficult.
- **Complexity:** The Spiral Model can be complex, as it involves multiple iterations of the software development process.
- **Time-Consuming:** The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.

Rapid Application Development

- The Rapid Application Development Model was first proposed by IBM in the 1980s. The RAD model is a type of incremental process model in which there is an extremely short development cycle.
- When the requirements are fully understood and the component-based construction approach is adopted then the RAD model is used. Various phases in RAD are Requirements Gathering, Analysis and Planning, Design, Build or Construction, and finally Deployment.

- A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams.
- These modules can finally be combined to form the final product. Development of each module involves the various basic steps as in the waterfall model i.e. analysing, designing, coding, and then testing, etc. as shown in the figure.
- Another striking feature of this model is a short period i.e. the time frame for delivery(time-box) is generally 60-90 days.
- Multiple teams work on developing the software system using the RAD model parallelly.


Elicit Requirements
Modularize Requirements
Team 1
Team2
Team N
Develop Module 1
Develop Module 2
Develop module N
Integrate all the modules
Test the final product and deliver
Analyze
Design
Code
Test

- **Requirements Planning –** This involves the use of various techniques used in requirements elicitation like brainstorming, task analysis, form analysis, user scenarios, FAST (Facilitated Application Development Technique), etc. It also consists of the entire structured plan describing the critical data, methods to obtain it, and then processing it to form a final refined model.
- **User Description –** This phase consists of taking user feedback and building the prototype using developer tools. In other words, it includes re-examination and validation of the data collected in the first phase. The dataset attributes are also identified and included in this phase.

- **Construction –** In this phase, refinement of the prototype and delivery takes place. It includes the actual use of powerful automated tools to transform processes and data models into the final working product. All the required modifications and enhancements are to be done in this phase.
- **Cutover –** All the interfaces between the independent modules developed by separate teams have to be tested properly. The use of powerfully automated tools and subparts makes testing easier. This is followed by acceptance testing by the user.

When to use the RAD Model?

- **Well-understood Requirements:** When project requirements are stable and transparent, RAD is appropriate.
- **Time-sensitive Projects:** Suitable for projects that need to be developed and delivered quickly due to tight deadlines.
- **Small to Medium-Sized Projects:** Better suited for smaller initiatives requiring a controllable number of team members.
- **High User Involvement:** Fits where ongoing input and interaction from users are essential.
- **Innovation and Creativity**: Helpful for tasks requiring creative inquiry and innovation.
- **Prototyping:** It is necessary when developing and improving prototypes is a key component of the development process.
- **Low technological Complexity:** Suitable for tasks using comparatively straightforward technological specifications.

Advantages of Rapid Application Development Model (RAD)

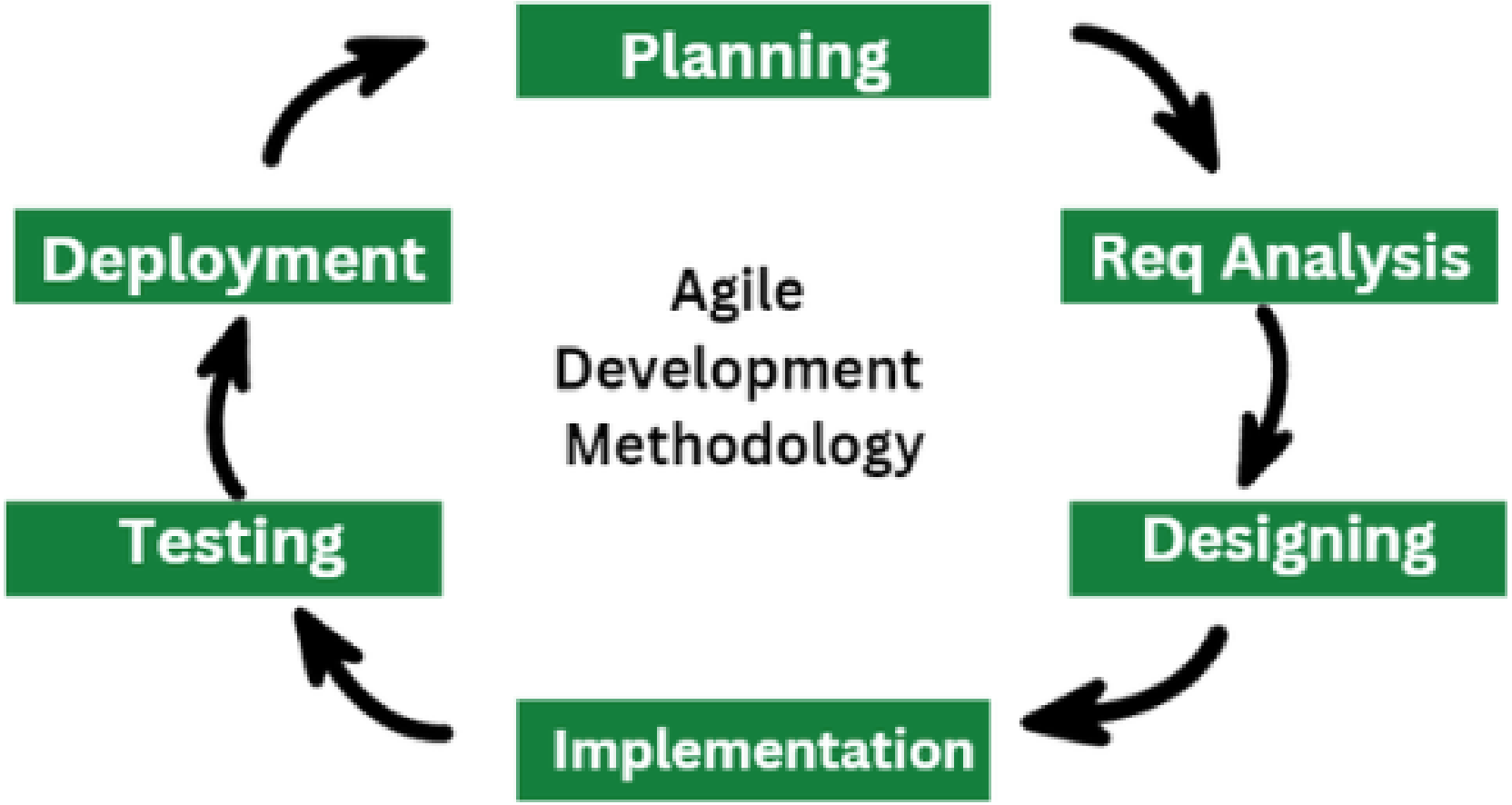
- The use of reusable components helps to reduce the cycle time of the project.
- Feedback from the customer is available at the initial stages.
- Reduced costs as fewer developers are required.

Disadvantages of Rapid application development model (RAD)

- The use of powerful and efficient tools requires highly skilled professionals.
- The absence of reusable components can lead to the failure of the project.
- The team leader must work closely with the developers and customers to close the project on time.
- Customer involvement is required throughout the life cycle.

Agile Methodology

- The Agile methodology is a project management and software development approach that emphasizes flexibility, collaboration, and customer-centricity. It is the latest model used by major companies today like Facebook, google, amazon, etc.
- It focuses on delivering smaller pieces of work regularly instead of one big launch.
- This allows teams to adapt to changes quickly and provide customer value faster


Planning
Req Analysis
Designing
Implementation
Testing
Deployment
Agile
Development
Methodology

- **Requirement Gathering**
- In this stage, the project team identifies and documents the needs and expectations of various stakeholders, including clients, users, and subject matter experts.
- It involves defining the project’s scope, objectives, and requirements.
- Establishing a budget and schedule.
- Creating a project plan and allocating resources.

- **Design**
- Developing a high-level system architecture.
- Creating detailed specifications, which include data structures, algorithms, and interfaces.
- Planning for the software’s user interface.

- **Development (Coding)**
- Writing the actual code for the software. Conducting unit testing to verify the functionality of individual components.

- **Testing**
- This phase involves several types of testing:
- **Integration Testing:** Ensuring that different components work together.
- **System Testing:** Testing the entire system as a whole.
- **User Acceptance Testing:** Confirming that the software meets user requirements.
- **Performance Testing:** Assessing the system’s speed, scalability, and stability.

- **Deployment**
- Deploying the software to a production environment.
- Put the software into the real world where people can use it.
- Make sure it works smoothly in the real world.
- Providing training and support for end-users.

- **Review (Maintenance)**
- Addressing and resolving any issues that may arise after deployment.
- Releasing updates and patches to enhance the software and address problems.

Benefits of Agile Methodology

- **Immediate Feedback:** It allows immediate feedback, which aids software improvement in the next increment.
- **Adapts to Changing Requirements:** It is a highly adaptable methodology in which rapidly changing requirements, allowing responsive adjustments.
- **Face-to-Face Communication:** Agile methodology encourages effective face-to-face communication.
- **Time-Efficient:** It is well-suited for its time-efficient practices, which help in delivering software quickly and reducing time-to-market.
- **Frequent Changes:** It effectively manages and accommodates frequent changes in project requirements according to stakeholder convenience.
- **Customer Satisfaction:** It prioritizes customer satisfaction.
- **Flexibility and Adaptability:** Agile methodologies are known for their flexibility and adaptability.

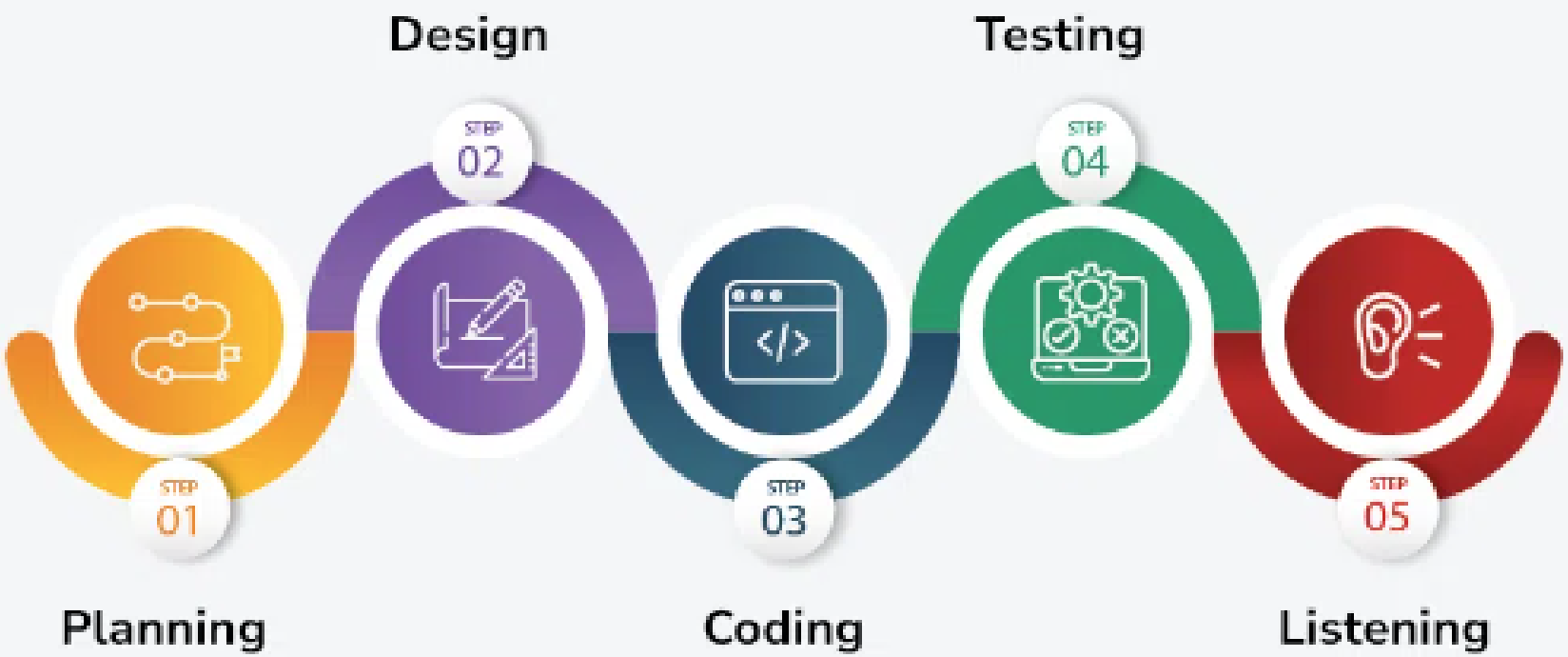
Limitations of Agile Methodology

- **Less Documentation:** Agile methodologies focus on less documentation; it prioritizes working on projects rather than paperwork.
- **Challenges in Large Organizations:** Busy schedule of clients can make daily meetup and face-to-face communication difficult.
- **Need for Senior Programmers:** It may require experienced programmers to make critical decisions during the development of software.

Extreme Programming (XP)

- Extreme Programming (XP) is an Agile software development methodology that focuses on delivering high-quality software through frequent and continuous feedback, collaboration, and adaptation.
- XP emphasizes a close working relationship between the development team, the customer, and stakeholders, with an emphasis on rapid, iterative development and deployment.

Life Cycle of Extreme Programming (XP)



- Extreme programming is one of the most popular and well-known approaches in the family of agile methods. an XP project starts with user stories which are short descriptions of what scenarios the customers and users would like the system to support.
- Each story is written on a separate card, so they can be flexibly grouped.
- XP is based on the frequent iteration through which the developers implement User Stories. User stories are simple and informal statements of the customer about the functionalities needed.
- A User Story is a conventional description by the user of a feature of the required system.

- Based on User stories, the project team proposes Metaphors. Metaphors are a common vision of how the system would work.
- The development team may decide to build a Spike for some features. A Spike is a very simple program that is constructed to explore the suitability of a solution being proposed. It can be considered similar to a prototype.

Basic activities that are followed during software development by using the XP model

- **Coding:** The concept of coding which is used in the XP model is slightly different from traditional coding. Here, the coding activity includes drawing diagrams (modeling) that will be transformed into code, scripting a web-based system, and choosing among several alternative solutions.
- **Testing:** The XP model gives high importance to testing and considers it to be the primary factor in developing fault-free software.

- **Listening:** The developers need to carefully listen to the customers if they have to develop good quality software. Sometimes programmers may not have the depth knowledge of the system to be developed. So, the programmers should understand properly the functionality of the system and they have to listen to the customers.
- **Designing:** Without a proper design, a system implementation becomes too complex, and very difficult to understand the solution, thus making maintenance expensive. A good design results elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.

- **Feedback:** One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs. Frequent contact with the customer makes the development effective.
- **Simplicity:** The main principle of the XP model is to develop a simple system that will work efficiently in the present time, rather than trying to build something that would take time and may never be used. It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.

- **Pair Programming:** XP encourages pair programming where two developers work together at the same workstation. This approach helps in knowledge sharing, reduces errors, and improves code quality.
- **Continuous Integration:** In XP, developers integrate their code into a shared repository several times a day. This helps to detect and resolve integration issues early on in the development process.
- **Refactoring:** XP encourages refactoring, which is the process of restructuring existing code to make it more efficient and maintainable. Refactoring helps to keep the codebase clean, organized, and easy to understand.

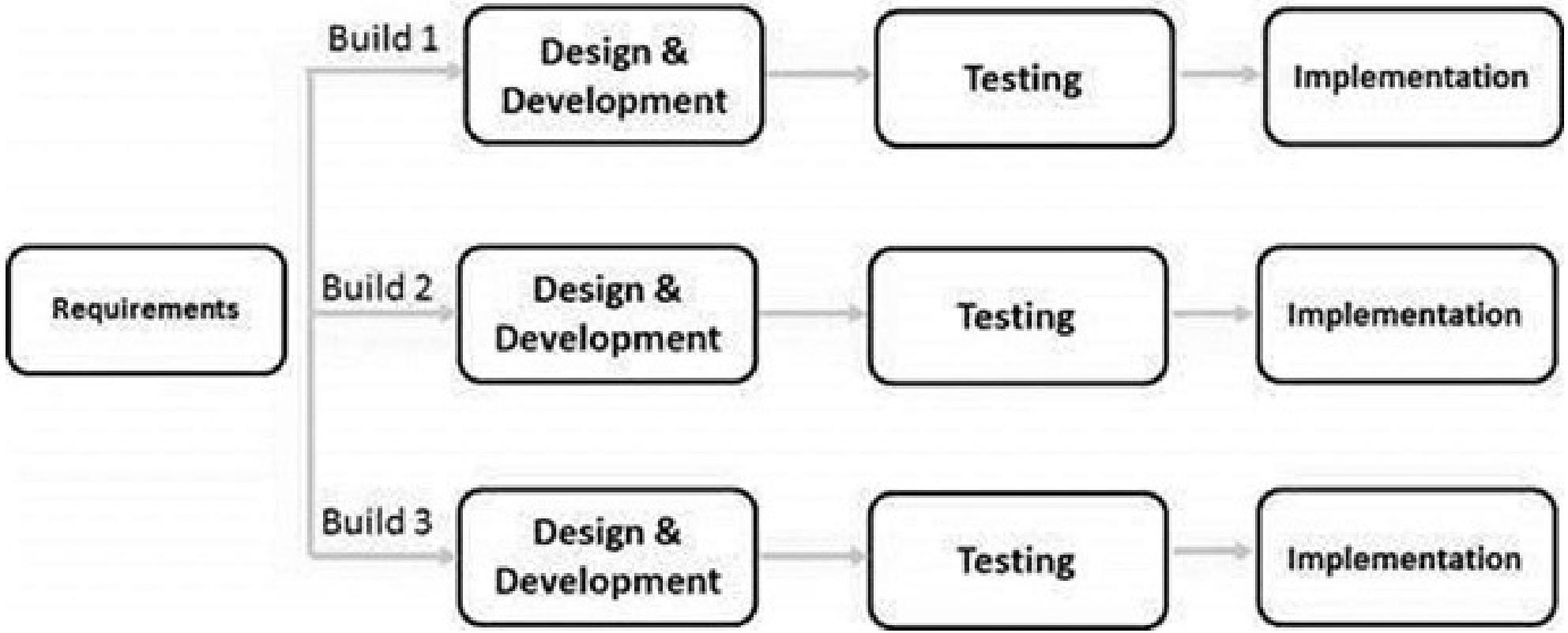
- **On-site Customer:** XP requires an on-site customer who works closely with the development team throughout the project. This approach helps to ensure that the customer’s needs are understood and met, and also facilitates communication and feedback.

Applications of Extreme Programming (XP)

- Small projects
- Projects involving new technology or Research projects
- Web development project
- Collaborative projects
- Projects with tight deadlines
- Projects with rapidly changing requirements
- Projects where quality is a high priority

Iterative Model

- Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented.
- At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).


Requirements
Build 1
Design & Development
Testing
Implementation
Build 2
Design & Development
Testing
Implementation
Build 3
Design & Development
Testing
Implementation

- 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.
- In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases.
- The process continues till the complete system is ready as per the requirement.

Advantages of the Iterative and Incremental

- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Easier to manage risk - High risk part is done first.
- Risk analysis is better.

Disadvantages of the Iterative and Incremental

- Not suitable for smaller projects.
- Management complexity is more.
- Highly skilled resources are required for risk analysis.

Bottom-up estimating

- Bottom-up estimating is a project management technique used to create an accurate estimation of project costs and durations by breaking down the project into smaller, more manageable components or tasks.
- Each component is estimated individually, and then these estimates are aggregated to form the overall project estimate. This method is often used in Software Project Management (SPM) to ensure detailed and precise project planning.

Key Steps in Bottom-Up Estimating

- **Decomposition of Project Work**:
  - Break down the project into smaller, manageable tasks or work packages. This is often done using a Work Breakdown Structure (WBS).
  - Each task should be well-defined, with clear deliverables and requirements.
- **Estimation of Individual Tasks**:
  - Estimate the resources, time, and costs required for each task. This involves:
    - **Resource Estimation**: Determine the personnel, equipment, and materials needed.
    - **Duration Estimation**: Estimate the time required to complete each task.
    - **Cost Estimation**: Calculate the cost of resources, including labor, materials, and overheads.

- **Review and Validation**:
  - Review the aggregated estimates with stakeholders to ensure accuracy and completeness.
  - Validate estimates by comparing them with historical data from similar projects.
- **Adjustment and Refinement**:
  - Make necessary adjustments based on feedback and additional information.
  - Refine estimates as more details become available throughout the project lifecycle.

Lean Software Development

- Lean Software Development (LSD) is an approach derived from lean manufacturing principles aimed at optimizing efficiency and **minimizing waste** in the software development process.
- **Prevent Defects:** It integrates quality assurance throughout the development process to prevent defects.
- **Eliminate Waste:** It focuses on activities that add value to the customer and eliminates those activities that do not add value.
- **Fast Delivery:** Reduces cycle time to deliver software quickly and respond to feedback and changing requirements rapidly.
- **Delay Decisions:** Delay decisions until they can be made based on facts.

Seven Principles of LSD

- **1. Eliminating the Waste**
- To identify and eliminate wastes e.g. unnecessary code, delay in processes, inefficient communication, issues with quality, data duplication, more tasks in the log than completed, etc. regular meetings are held by Project Managers. This allows team members to point out faults and suggest changes in the next turn.
- **2. Fast Delivery**
- Previously long-time planning used to be the key to success in business, but with time, it has been found that engineers spend too much time on building complex systems with unwanted features. So they came up with an MVP strategy which resulted in building products quickly that included a little functionality and launching the product to market and seeing the reaction. Such an approach allows them to enhance the product based on customer feedback.

- **3. Amplify Learning**
- Learning is improved through ample code reviewing and meetings that are cross-team applicable. It is also ensured that particular knowledge isn’t accumulated by one engineer who’s writing a particular piece of code so paired programming is used.
- **4. Builds Quality**
- LSD is all about preventing waste and keeping an eye on not sacrificing quality. Developers often apply test-driven programming to examine the code before it is written. Quality can also be gained by getting constant feedback from team members and project managers.

- **5. Respect Teamwork**
- LSD focuses on empowering team members, rather than controlling them. Setting up a collaborative atmosphere, keeping perfect balance when there are short deadlines and immense workload. This method becomes very important when new members join a well-established team.
- **6. Delay the Commitment**
- In traditional project management, it often happens when you make your application and it turns out to be completely unfit for the market. LSD method recognizes this threat and makes room for improvement by postponing irreversible decisions until all experiment is done. This methodology always constructs software as flexible, so new knowledge is available and engineers can make improvements.

- **7. Optimizing the Whole System**
- Lean’s principle allows managers to break an issue into small constituent parts to optimize the team’s workflow, create unity among members, and inspire a sense of shared responsibility which results in enhancing the team’s performance

Benefits of LSD

- Increased Efficiency
- Higher Quality
- Faster Delivery
- Adaptability
- Enhanced Collaboration

Limitations of LSD

- Cultural Resistance
- Requires Strong Leadership
- Difficulty in Measuring Waste

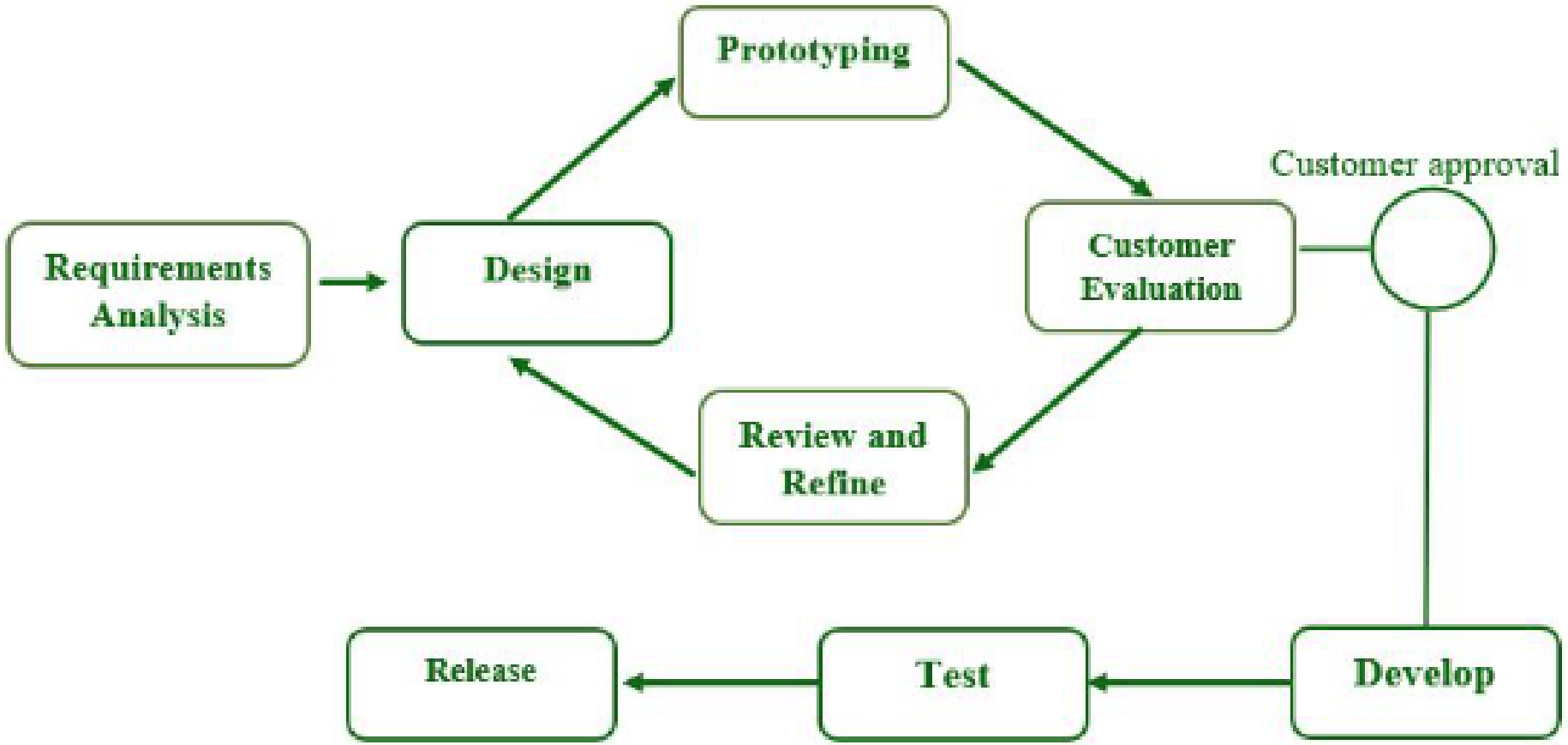
Software Prototyping

- A prototype can be a limited-functionality software performance model.
- The prototype does not always contain the exact logic used in the specific software application and is an additional effort to be considered when estimating effort.

- **Step-1: Requirements gathering and analysis :**
- Requirement analysis is the first step in developing a prototyping model. During this phase, the system’s desires are precisely defined. During the method, system users are interviewed to determine what they expect from the system.
- **Step-2: Quick design :**
  The 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. It provides the user with a quick overview of the system. The rapid design aids in the development of the prototype.

- **Step-3: Build a Prototype :**
  During this stage, an actual prototype is intended to support the knowledge gained from quick design. It is a small low-level working model of the desired system.
- **Step-4: Initial user evaluation :**
  The proposed system is presented to the client for preliminary testing at this stage. It is beneficial to investigate the performance model’s strengths and weaknesses. Customer feedback and suggestions are gathered and forwarded to the developer.

- **Step-5: Refining prototype :**
  If the user is dissatisfied with the current model, you may want to improve the type that responds to user feedback and suggestions. When the user is satisfied with the upgraded model, a final system based on the approved final type is created.
- **Step-6: Implement Product and Maintain :**
  The final system was fully tested and distributed to production after it was developed to support the original version. To reduce downtime and prevent major failures, the programmer is run on a regular basis.


Prototyping
Customer approval
Requirements Analysis
Design
Customer Evaluation
Review and Refine
Release
Test
Develop

Advantages of Software Prototyping

- Assists team members in effectively communicating.
- Customer satisfaction exists, and he can feel the product from the start.
- There will be no risk of software loss.
- Quick user feedback aids in the development of better software solutions.

Structure in SPM

- Project Planning: Defining project scope, objectives, deliverables, timelines, and resources.
- Process Frameworks: Using established methodologies like Waterfall, Agile, Scrum, or Kanban to guide project activities.
- Documentation: Creating comprehensive documentation for requirements, design, testing, and user manuals.
- Risk Management: Identifying, analyzing, and mitigating potential project risks.
- Quality Assurance: Ensuring that the project meets predefined quality standards through testing and reviews.

- Time-to-Market: Reducing the duration from project initiation to product launch.
- Iterative Development: Delivering work in small, functional increments, often seen in Agile and Scrum methodologies.
- Minimal Viable Product (MVP): Developing a version of the product with just enough features to satisfy early users and provide feedback for future development.
- Continuous Delivery/Deployment: Automating the software release process to enable frequent and reliable deployment of updates and new features.

Disadvantages of Software Prototyping

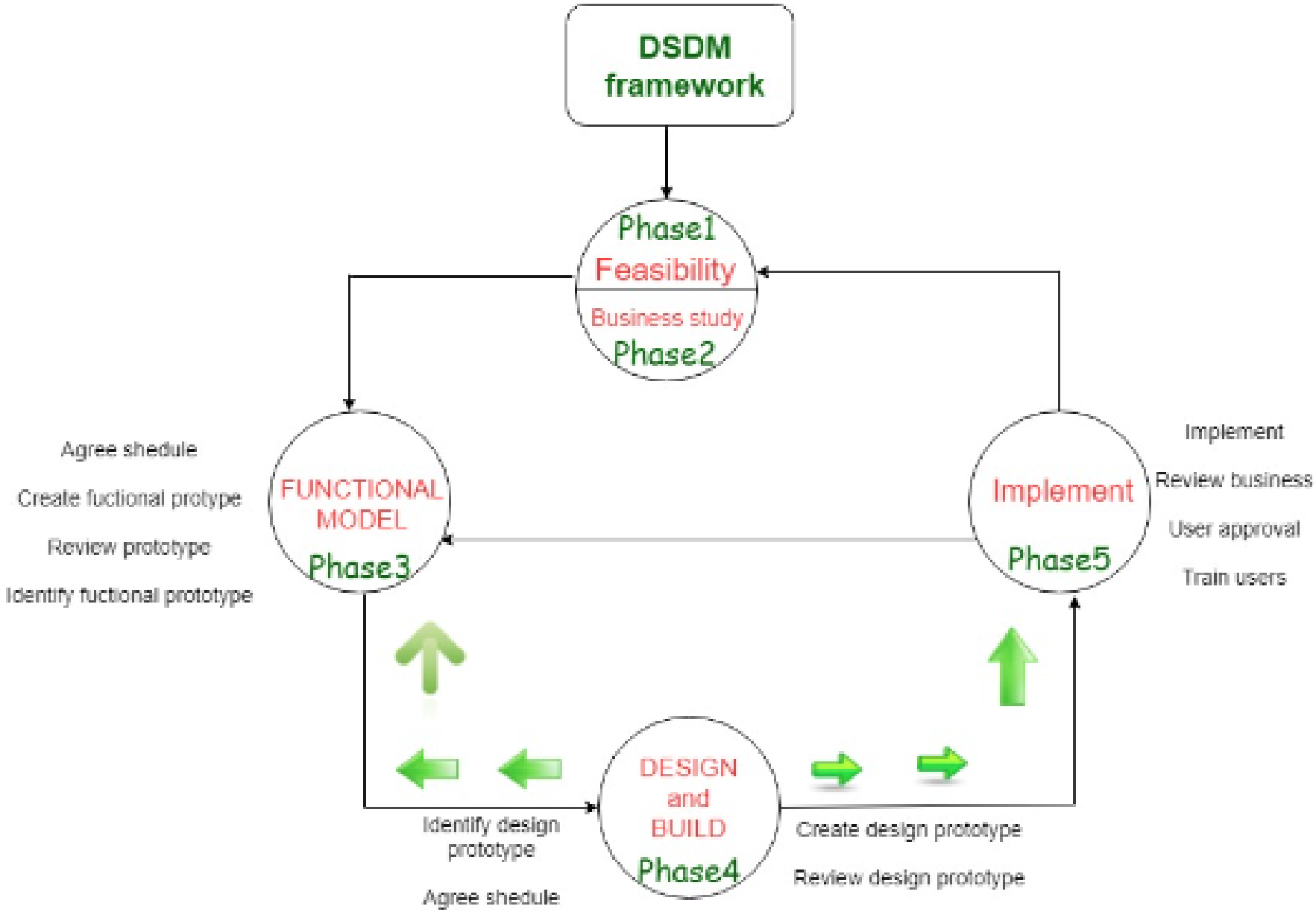
- Prototyping is a time-consuming and labour-intensive process.
- The cost of creating a specific type of waste is completely wasted because the prototype is eventually discarded.
- Poor documentation as a result of changing customer needs.

Atern/Dynamic Systems Development Method

- DSDM is An iterative code method within which every iteration follows the 80% rule that simply enough work is needed for every increment to facilitate movement to the following increment. The remaining detail is often completed later once a lot of business necessities are noted or changes are requested and accommodated.

- **Feasibility Study:**
  It establishes the essential business necessities and constraints related to the applying to be designed then assesses whether or not the application could be a viable candidate for the DSDM method.
- **Business Study:**
  It establishes the use and knowledge necessities that may permit the applying to supply business value; additionally, it is the essential application design and identifies the maintainability necessities for the applying.

- **Functional Model Iteration:**
  It produces a collection of progressive prototypes that demonstrate practicality for the client.
- **Design and Build Iteration:**
  It revisits prototypes designed throughout useful model iteration to make sure that everyone has been designed during a manner that may alter it to supply operational business price for finish users. In some cases, useful model iteration and style and build iteration occur at the same time.
-


DSDM framework
Phase1
Feasibility
Business study
Phase2
Agree shedule
Create fuctional protype
Review prototype
Identify fuctional prototype
FUNCTIONAL MODEL
Phase3
Implement
Review business
User approval
Train users
Implement
Phase5
Identify design prototype
Agree shedule
DESIGN and BUILD
Phase4
Create design prototype
Review design prototype

Selecting the Most Appropriate Process Model

- Selecting the most appropriate process model for a software development project depends on various factors such as project size, complexity, requirements clarity, team size, customer involvement, and project timeline. Here are some commonly used process models and the scenarios where they are most suitable:

- **Waterfall Model:**
  - **Suitable for:** Projects where requirements are well-understood and fixed, and changes are unlikely during the development process. Also, for small to medium-sized projects with a clear sequential flow.
  - **Advantages:** Simple and easy to manage. Well-suited for projects with stable requirements.
- **Iterative and Incremental Models (e.g., Agile):**
  - **Suitable for:** Projects where requirements are likely to evolve or change, and flexibility is needed. Also, for projects requiring frequent releases and continuous customer feedback.
  - **Advantages:** Allows for early delivery of a working product, accommodates changes during the development process, and promotes customer collaboration.

- **Spiral Model:**
  - **Suitable for:** Projects with high risk and uncertainty, where continuous risk assessment and mitigation are necessary. Also, for large-scale projects where each iteration involves planning, risk analysis, engineering, and evaluation phases.
  - **Advantages:** Provides flexibility in terms of accommodating changes and addressing risks early in the development process.
- **RAD (Rapid Application Development) Model:**
  - **Suitable for:** Projects requiring a quick turnaround time with a focus on user involvement and feedback. Also, for small to medium-sized projects where reusable components can be utilized.
  - **Advantages:** Accelerates the development process through the use of pre-existing templates and tools, and promotes iterative development.

The Top-down Approach

- In software project management (SPM), the top-down approach refers to a methodology where the project planning and implementation start with a broad overview and then progressively delve into finer details.
- This approach is commonly used in various stages of project management, including planning, design, development, and testing. Here's how the top-down approach is typically applied in SPM:

- **Planning Phase:**
- **Project Initiation:** Identify the high-level goals, objectives, and scope of the project. This involves understanding the overall purpose of the software project and its alignment with business objectives.
- **Scope Definition:** Define the boundaries of the project and outline what is included (in-scope) and what is not included (out-of-scope). This helps in setting realistic expectations and managing stakeholders' requirements.
- **Work Breakdown Structure (WBS):** Create a hierarchical decomposition of the project deliverables into smaller, manageable tasks. Initially, the WBS focuses on major deliverables and gradually breaks them down into more detailed activities.

- **Design Phase:**
- **Architecture Design:** Develop a high-level architecture or system design that outlines the overall structure of the software solution. This includes defining major components, interfaces, and data flows.
- **Module Design:** Break down the system into modules or components based on the architectural design. Each module is designed to handle specific functionalities or features of the software.

- **Development Phase:**
- **Coding:** Begin coding based on the design specifications. Initially, focus on implementing the core functionalities and major components of the software solution.
- **Integration:** Integrate the modules and components developed separately into a unified system. This involves testing the interactions between different modules and ensuring that they work together as intended.

- **Testing Phase:**
- **System Testing:** Conduct comprehensive testing to verify the overall functionality, performance, and reliability of the software system as a whole. Initially, focus on testing critical functionalities and gradually expand to cover all aspects of the system.
- **Acceptance Testing:** Involve stakeholders and end-users in testing the software to ensure it meets their requirements and expectations. Start with high-level acceptance criteria and progressively refine them based on feedback and test results.

Parametric Models

- In software project management (SPM), parametric models refer to estimation techniques that use statistical relationships between historical data and project parameters to predict effort, cost, and other project attributes.

Key Aspects of Parametric Models:

- **Statistical Basis:** Parametric models rely on statistical analysis of historical project data to establish mathematical formulas or algorithms that relate project attributes to effort or cost.
- **Project Parameters:** They consider various project parameters such as:
  - **Size:** Usually measured in lines of code (LOC), function points, or other metrics.
  - **Complexity:** Factors that influence the complexity of the software solution.
  - **Team Experience:** Skill levels and experience of the development team.
  - **Technology Used:** Tools, platforms, and methodologies employed in the project.

- **Estimation Outputs:** Parametric models typically provide estimates for:
  - **Effort:** The amount of person-hours or person-days required to complete the project.
  - **Cost:** Total financial cost associated with the project.
  - **Duration:** Time required to complete the project.

Examples of Parametric Models:

- **COCOMO (Constructive Cost Model):**
- **Overview:** COCOMO is one of the most well-known parametric models used for estimating effort, cost, and duration of software projects.
- **Types:** It comes in different versions:
  - **Basic COCOMO:** Suitable for early stages of project planning based on LOC.
  - **Intermediate COCOMO:** Incorporates more project-specific details.
  - **Detailed COCOMO:** Provides a more detailed estimation using a set of cost drivers.

The Basis for Software Estimating

- Software estimating in software project management (SPM) is a critical process that involves predicting the effort, time, and resources required to complete a software development project.
- The accuracy of software estimates significantly impacts project planning, budgeting, scheduling, and overall project success. Here are the key bases and factors considered in software estimating:

- **Size Estimation:**
  - **Definition:** Size estimation involves quantifying the scope of the software project in terms of lines of code (LOC), function points, or other relevant metrics.
  - **Basis:** Estimating the size of the software system is foundational to all other estimation activities. It helps in understanding the scale and complexity of the project, which directly impacts effort and cost estimates.
- **Effort Estimation:**
  - **Definition:** Effort estimation involves predicting the amount of person-hours or person-days required to complete the project tasks.
  - **Basis:** Effort estimation is based on historical data, experience, and industry standards. Parametric models like COCOMO (Constructive Cost Model) and function points analysis are commonly used to quantify effort based on project size, complexity, team experience, and other factors.

- **Cost Estimation:**
  - **Definition:** Cost estimation involves predicting the financial resources required to complete the project, including labor costs, equipment costs, software and hardware costs, etc.
  - **Basis:** Cost estimation is derived from effort estimation, where effort estimates are converted into monetary terms by applying labor rates and other cost factors. It also considers overhead costs, contingency reserves, and other project-specific expenses.
- **Duration Estimation:**
  - **Definition:** Duration estimation involves predicting the total calendar time required to complete the project from start to finish.
  - **Basis:** Duration estimation considers the estimated effort, resource availability, project dependencies, and any constraints or milestones that affect the project timeline. It is crucial for project scheduling and resource planning.

Factors Affecting Software Estimating:

- **Project Scope and Requirements:**
  - **Clarity:** Clear and well-defined requirements lead to more accurate estimates.
  - **Scope Changes:** Frequent changes in scope can impact estimates, requiring adjustments throughout the project.
- **Development Methodology:**
  - **Agile vs. Waterfall:** Agile methodologies allow for iterative development and adaptive planning, while waterfall methodologies follow a sequential approach with detailed upfront planning.
- **Team Experience and Skills:**
  - **Expertise:** Experienced teams with relevant skills tend to produce more accurate estimates based on their historical performance.

- **Technology and Tools:**
  - **Complexity:** The complexity of technology and tools used in development can influence effort and duration estimates.
- **Risk and Uncertainty:**
  - **Risk Management:** Assessing and mitigating project risks is critical for managing uncertainties that can affect estimates.

Software Effort Estimation Techniques

- Effort estimation techniques in software project management are crucial for planning and resource allocation. Here are some commonly used techniques:
  - **Expert Judgment:** Relies on the expertise of experienced individuals or teams to estimate effort based on their knowledge and insights.
  - **Analogous Estimation:** Uses historical data from similar past projects as a basis for estimating effort for current projects.
  - **Parametric Models:** Utilizes mathematical models that correlate project characteristics (like size, complexity) to effort estimates. Examples include COCOMO (Constructive Cost Model).

- **Algorithmic Models:** Applies complex algorithms or statistical methods to estimate effort based on various project parameters and historical data.
- **Function Points:** Measures the size of a software project based on its functionality, which is then used to estimate effort.
- **Use Case Points:** Estimates effort based on the number and complexity of use cases identified for the software.

Albrecht's Function Point Method

- Albrecht's Function Point Method is a technique used to measure the size and complexity of software systems, which was developed by Allan Albrecht at IBM in the late 1970s. This method is widely used in software engineering for estimating the amount of effort required to develop and maintain software applications.

Key Components of Function Point Analysis (FPA):

- **External Inputs (EI):** These are the processes or transactions where data is received by the system from external sources. For example, user inputs, file imports, or data received from other systems.
- **External Outputs (EO):** These processes involve data that is sent out from the system to external destinations. For instance, reports, messages, or file exports.
- **External Inquiries (EQ):** These are interactive inputs that result in the immediate output of data. It includes both input and output operations, typically without altering the internal data.

- **Internal Logical Files (ILF):** These are the files or databases that the system maintains. They are used internally by the system to store data.
- **External Interface Files (EIF):** These files are similar to ILFs but are used to store data that the system uses but does not maintain. They are maintained by other systems.

Steps in Function Point Analysis:

- **Identify and Count Functions:** Determine and classify each of the functions (EI, EO, EQ, ILF, EIF) in the software system.
- **Determine Complexity:** Assign a complexity level (low, average, high) to each function based on specific criteria, such as the number of data elements or file types involved.
- **Assign Function Points:** Use standard tables to assign a certain number of function points to each function based on its type and complexity.

- **Calculate the Total Function Points:** Sum the function points for all functions to get the total function points for the system.
- **Adjust for Environmental Factors:** Adjust the total function points based on various factors like system performance, user interaction, and programming environment. This adjustment is done using a Value Adjustment Factor (VAF).

Function Points Mark II

- Function Points Mark II (also known as Mk II Function Point Analysis) is an evolution of the original Function Point Analysis (FPA) developed by Charles Symons in the late 1980s.
- This method was designed to address some of the perceived limitations of the original FPA and to provide a more refined approach for measuring software size and complexity.

Key Differences from the Original Function Points:

- **Focus on Data Movement:** While the original FPA focuses on transactions and data management, Mk II emphasizes the movement of data through the system.
- **Simplified Complexity Adjustment:** Mk II uses a simpler approach to adjusting for complexity, making it easier to apply consistently.

- **New Categories:** Mk II introduces new categories for classifying data movements:
  - **Input (E1):** Data entering the system from outside.
  - **Output (E2):** Data leaving the system to go outside.
  - **Read (E3):** Data read from the system.
  - **Write (E4):** Data written to the system.
  - **Inquiry (E5):** Interactive transactions involving both read and write operations.

Steps in Mk II Function Point Analysis:

- **Identify Data Movements**: Classify all the data movements in the system into the appropriate categories (E1, E2, E3, E4, E5).
- **Assign Complexity:** Each data movement is evaluated for complexity based on the number of data elements involved and the number of files or data structures referenced.
- **Calculate Raw Function Points:** Use standard tables to assign a specific number of function points to each data movement based on its category and complexity.
- **Apply Complexity Adjustment:** Adjust the raw function points based on a set of predefined complexity factors to obtain the final function point count.

Example System Description

Example System Description

Assume we have a simple inventory management system

1. Adding new items to the inventory.
2. Generating inventory reports.
3. Querying inventory status.
4. Updating item quantities.
5. Deleting items from the inventory.

Step 1: Identify Data Movements

We need to classify the data movements for the functionalities described:

1. **Add New Item**
   - Input (E1): Adding item details to the system.
2. **Generate Inventory Report**
   - Output (E2): Producing a report of all items in the inventory.
3. **Query Inventory Status**
   - Inquiry (E5): Checking the status of a specific item.
4. **Update Item Quantities**
   - Input (E1): Updating the quantity of an existing item.
5. **Delete Item**
   - Input (E1): Removing an item from the inventory.

Step 2: Assign Complexity

Let's assign complexity levels based on the number of data elements and file types involved. For simplicity, we'll assume all data movements are of average complexity.

1. **Add New Item** - Input (E1): Average complexity
2. **Generate Inventory Report** - Output (E2): Average complexity
3. **Query Inventory Status** - Inquiry (E5): Average complexity
4. **Update Item Quantities** - Input (E1): Average complexity
5. **Delete Item** - Input (E1): Average complexity

Calculate the raw function points for each function:

1. **Add New Item** - Input (E1): 4 points
2. **Generate Inventory Report** - Output (E2): 5 points
3. **Query Inventory Status** - Inquiry (E5): 4 points
4. **Update Item Quantities** - Input (E1): 4 points
5. **Delete Item** - Input (E1): 4 points

Step 3: Calculate Raw Function Points

Using standard tables, we assign function points based on the type and complexity of each data movement. For average complexity, let's assume the following points:

- Input (E1): 4 points
- Output (E2): 5 points
- Inquiry (E5): 4 points

Step 4: Apply Complexity Adjustment

In Mk II, complexity adjustments are based on a set of predefined factors. For simplicity, let's assume there are no additional complexity adjustments required. Therefore, the raw function points remain the same.

Final Function Point Count

Summing up the raw function points for all data movements:

- Add New Item: 4 points
- Generate Inventory Report: 5 points
- Query Inventory Status: 4 points
- Update Item Quantities: 4 points
- Delete Item: 4 points

Total Function Points: 4 + 5 + 4 + 4 + 4 = 21 points

COSMIC Full Function Points

- COSMIC (Common Software Measurement International Consortium) Full Function Points is a modern and standardized method for measuring software size, particularly designed to address the limitations of traditional Function Point Analysis methods.

Key Concepts of COSMIC

1. **Functional User Requirements:** The functional processes that the software must execute in response to events from the environment.
2. **Data Movement:** The key concept in COSMIC, which involves movements of data in and out of the software. These are classified into four types:
    - **Entry (E):** Data that enters the software from the outside.
    - **Exit (X):** Data that leaves the software to the outside.
    - **Read (R):** Data that is read from a persistent storage within the software.
    - **Write (W):** Data that is written to a persistent storage within the software.

Steps in COSMIC Function Points Calculation

1. **Identify Functional Processes:** Break down the software into functional processes based on the functional user requirements.
2. **Identify Data Movements:** For each functional process, identify all data movements (Entry, Exit, Read, Write).
3. **Measure Data Movements:** Assign a size of 1 COSMIC Function Point (CFP) to each data movement.
4. **Calculate Total COSMIC Function Points:** Sum the CFPs for all data movements across all functional processes.

Example of COSMIC Full Function Points Calculation

Let's consider a simple example of an e-commerce order management system with the following functionalities:

1. Add a new order.
2. Update an existing order.
3. Generate an order report.
4. Query order status.

Step 1: Identify Functional Processes

We have four functional processes:

1. Add Order
2. Update Order
3. Generate Order Report
4. Query Order Status

Step 2: Identify Data Movements

For each functional process, identify the data movements:

1. **Add Order:**
   - Entry (E): Order data (from user input)
   - Write (W): Store order data in the database
2. **Update Order:**
   - Entry (E): Updated order data (from user input)
   - Read (R): Retrieve existing order data from the database
   - Write (W): Update order data in the database
3. **Generate Order Report:**
   - Entry (E): Report request (from user input)
   - Read (R): Retrieve order data from the database
   - Exit (X): Output report data to the user
4. **Query Order Status:**
   - Entry (E): Query request (from user input)
   - Read (R): Retrieve order status from the database
   - Exit (X): Output order status to the user

Step 3: Measure Data Movements

Assign 1 CFP to each data movement:

1. **Add Order:**
    - Entry (E): 1 CFP
    - Write (W): 1 CFP
2. **Update Order:**
    - Entry (E): 1 CFP
    - Read (R): 1 CFP
    - Write (W): 1 CFP
3. **Generate Order Report:**
    - Entry (E): 1 CFP
    - Read (R): 1 CFP
    - Exit (X): 1 CFP
4. **Query Order Status:**
    - Entry (E): 1 CFP
    - Read (R): 1 CFP
    - Exit (X): 1 CFP

Step 4: Calculate Total COSMIC Function Points

Sum the CFPs for all data movements:

1. **Add Order: 1 + 1 = 2 CFP**
2. **Update Order: 1 + 1 + 1 = 3 CFP**
3. **Generate Order Report: 1 + 1 + 1 = 3 CFP**
4. **Query Order Status: 1 + 1 + 1 = 3 CFP**

Total COSMIC Function Points: 2 + 3 + 3 + 3 = 11 CFP

Parametric Productivity Model

- A Parametric Productivity Model is a quantitative method used in software engineering and project management to estimate the productivity and effort required for a software development project. These models use mathematical equations that relate various parameters (such as software size, complexity, and team experience) to the effort, cost, and duration of the project.
- The most well-known parametric productivity models in software engineering are COCOMO (Constructive Cost Model) and SLIM (Software Lifecycle Management).

Key Components of Parametric Productivity Models

1. **Size Estimation:** This often involves using software metrics such as Function Points, Lines of Code (LOC), or COSMIC Function Points to estimate the size of the software.
2. **Effort Estimation:** The model estimates the effort required to complete the project, usually measured in person-months or person-hours.
3. **Cost Estimation:** Based on the effort, the model can estimate the cost of the project by considering factors like labor rates.
4. **Duration Estimation:** The model also estimates the project duration based on the effort and available resources.
5. **Productivity Factors:** Various factors such as team experience, technology, and complexity are taken into account to adjust the estimates.

Example: COCOMO II Model

Cost Estimation

- Cost estimation in software engineering is a critical process for determining the resources needed to complete a software project. It involves predicting the effort, time, and financial investment required. Accurate cost estimation helps in budgeting, planning, and managing resources efficiently.
- Various methods and models are used for cost estimation, including expert judgment, analogy-based estimation, parametric models, and algorithmic models like COCOMO.

Key Elements of Cost Estimation

1. **Effort Estimation:** The amount of work needed, typically measured in person-months or person-hours.
2. **Time Estimation:** The duration required to complete the project.
3. **Resource Estimation:** The types and quantities of resources needed, including personnel, hardware, and software.
4. **Cost Estimation:** The financial cost associated with the effort, resources, and duration.

Methods of Cost Estimation

1. Expert Judgment

- Involves consulting with experienced professionals who use their knowledge to estimate costs.
- Advantages: Quick and leverages expertise.
- Disadvantages: Subjective and may lack consistency.

2. Analogous Estimation

- Uses data from similar past projects to estimate the cost of the current project.
- Advantages: Utilizes historical data, relatively quick.
- Disadvantages: May not be accurate if past projects differ significantly from the current one.

3. Parametric Models

- Uses mathematical models to estimate costs based on project parameters.
- Example: COCOMO II, Function Points Analysis.
- Advantages: More systematic and can handle large datasets.
- Disadvantages: Requires accurate parameter estimation and may be complex.

4. Algorithmic Models

- Employs algorithms to estimate costs based on various factors such as size, complexity, and team capability.
- Example: COCOMO (Constructive Cost Model).
- Advantages: Objective and reproducible.
- Disadvantages: Requires detailed input data and can be complex to implement.

Staffing Pattern

- A staffing pattern in project management, especially in software development, refers to the plan and allocation of human resources over the project's lifecycle. It outlines the number of people needed, their roles, and when they are required.
- Proper staffing patterns ensure that the project has the right mix of skills and manpower at each stage, optimizing productivity and project success.

Key Elements of a Staffing Pattern

- **Project Phases**: Different stages of the project, such as initiation, planning, design, development, testing, deployment, and maintenance.
- **Roles and Responsibilities**: Specific roles required at each phase, such as project managers, developers, designers, testers, and analysts.
- **Resource Allocation**: Number of personnel needed for each role at different times.
- **Timeline**: Duration each role is required for the project.

Typical Roles in a Software Development Project

1. **Project Manager**: Oversees the project, ensures timelines and budgets are met, manages risks, and communicates with stakeholders.
2. **Business Analyst**: Gathers and analyzes requirements, acts as a bridge between stakeholders and the development team.
3. **System Architect**: Designs the system architecture, ensures it meets technical and business requirements.
4. **Developers**: Write code, develop features, fix bugs.
5. **UI/UX Designers**: Design the user interface and experience, ensuring it is user-friendly.
6. **Testers/QA Engineers**: Test the software to find and fix defects, ensure it meets quality standards.
7. **DevOps Engineers**: Manage infrastructure, deployment processes, ensure continuous integration and delivery.

Capers Jones Estimating Rules of Thumb.

- Rule 1: Equivalence of SLOC Function Points for C programs, one function point equals 125 SLOC.
- Rule 2: Calculate the Project Duration. The development period is roughly predicted by Function Points raised to the power of 0.4 in calendar months.
- Rule 3: From the design to the coding phases, users need to creep in at a pace of about 2% per month on average.
- Rule 4: Defect Removal Efficiency 30% of all bugs will be found and fixed during each program review, inspection, or test stage.

- Rule 5: Calculate the projected workforce. The number of employees needed to construct the program is roughly predicted by dividing the size of the software by 150.
- Rule 6: Calculate the effort involved in developing software. The software development time multiplied by the required staff headcount yields the approximate number of staff months needed to produce software.
- Rule 7: The number of employees needed to do routine maintenance tasks is roughly predicted by dividing the function points by 500

Schedule compression

- Definition: Schedule compression is a project management technique used to shorten the duration of a project without sacrificing its scope or quality

Why do Schedule Compression

- A Progam Manager might want to utilize a schedule compression for various reasons, but the main one is usually to get back on schedule. Below are a few reasons:
- The initial schedule proved to be unrealistic
- Make up for a Delay in resources
- Reduce risk
- Change in the delivery schedule
- Take advantage of an opportunity
- Make up for a mistake

Schedule Compression Techniques

- Two techniques are commonly used in schedule compression. These are:
  - Crashing
  - Fast Tracking
  - Resource re-allocation

- (1**) Crashing** Crashing assigns more resources to an activity to decrease the overall completion time. The cost benefits of this activity have to be explored in order to make it a useful technique.
- (2) **Fast Tracking** Fast Tracking is the process of executing activities or phases that were originally scheduled sequentially in parallel. Activities can be overlapped, started earlier than proposed, started activities that require different resources and may be combined activities in the schedule. This process adds risk to the schedule and program and must be executed carefully.

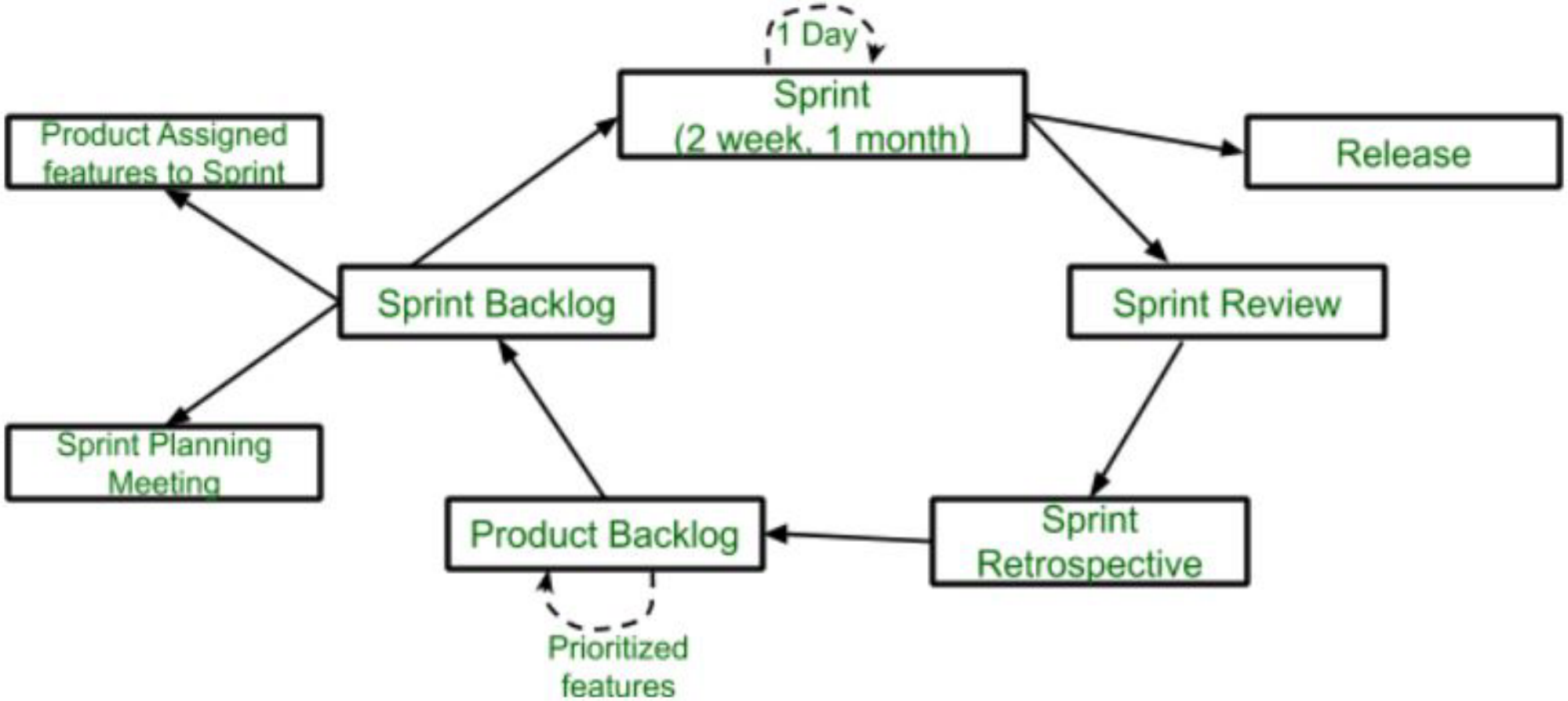
- (3) **Resource re-allocation** Through re-allocation, resources are moved from jobs that are not on the critical path to those that are. It makes better use of the tools we already have without increasing costs or changing the way we schedule things. But it only works if a resource is moved from a job that isn’t on the critical path to one that is.

Scrum

- Scrum is a management framework that teams use to self-organize tasks and work towards a common goal. It is a framework within which people can address complex adaptive problems while the productivity and creativity of delivering products are at the highest possible value.
- Scrum allows us to develop products of the highest value while making sure that we maintain creativity and productivity.The iterative and incremental approach used in scrum allows the teams to adapt to the changing requirements.

Silent features of Scrum

- Scrum is a light-weighted framework.
- Scrum is simple to understand
- Scrum framework helps the team to work together


1 Day
Sprint
(2 week, 1 month)
Release
Product Assigned
features to Sprint
Sprint Backlog
Sprint Review
Sprint Planning
Meeting
Product Backlog
Sprint
Retrospective
Prioritized
features

- **Sprint**: A Sprint is a time box of one month or less. A new Sprint starts immediately after the completion of the previous Sprint. Release: When the product is completed, it goes to the Release stage.
- **Sprint Review**: If the product still has some non-achievable features, it will be checked in this stage and then passed to the Sprint Retrospective stage.
- **Sprint Retrospective**: In this stage quality or status of the product is checked. Product Backlog: According to the prioritize features the product is organized.
- **Sprint Backlog**: Sprint Backlog is divided into two parts Product assigned features to sprint and Sprint planning meeting

Problems with Over- and Under-Estimates

- “Overestimate” means to judge or estimate something to be greater than it actually is.
- “Underestimate” means to judge or estimate something to be less than it actually is.
- “Overestimate” typically leads to mistakes, overspending, or over-committing.
- “Underestimate” frequently leads to delays, inefficiencies, or failures to meet expectations.
- “Overestimate” is corrected by adjusting expectations or taking a more realistic approach.
- “Underestimate” is corrected by re-evaluating assumptions or gathering more information.

	Overestimate	Underestimate
Definition	Oxford Dictionary defines it as a verb meaning to "estimate something to be better, larger, or more important than it really is." It means to "overvalue or exaggerate the worth or importance of someone or something."	The word "***underestimate***" is a verb that means to estimate, judge, or evaluate something as being less than it actually is in reality or potential.
Contexts	The word "***overestimate***" means to assess or evaluate it as being greater in value, size, importance, or ability than it actually is in reality. It results in unrealistic expectations, misplaced priorities, or incorrect assumptions about a situation or person.	The word "***underestimate***" implies a lack of proper understanding or awareness of the true nature or value of something, which leads to mistakes, misjudgments, or missed opportunities.
Example Sentences	"The company ***overestimated*** the demand for their new product, and as a result, they had to cut back on production.""The politician's campaign promises were clearly ***overestimated***, as they failed to deliver on most of them once elected.""Don't ***overestimate*** your ability to finish the race – it's much longer and more difficult than you think.""We ***overestimated*** how much money we would need for the trip, and ended up having to cut back on activities."	"The potential of the new employee was ***underestimated*** during the interview process, and they quickly proved to be an invaluable asset to the team.""The opposition party ***underestimated*** the level of support for the incumbent, and were surprised when they lost the election.""The hiker ***underestimated*** the difficulty of the trail and didn't bring enough water, which led to dehydration and exhaustion.""Don't ***underestimate*** the importance of regular exercise. It can have a huge impact on your overall health."

Build vs. Buy Decision in Software Development

- When making decisions about information technology (IT) investments, organizations have two fundamental choices: build or buy. With the build option, the organization constructs the IT capability needed from scratch. The organization acquires an already-existing IT capability from a software vendor or other external sources with the buy option.

Key Factors to Consider Whether to Build or Buy Software

- **Cost**
  - The cost of building software includes the cost of developing, maintaining, and upgrading the software over time. The cost of buying software consists of the purchase price and the cost of licensing and maintaining the software. By comparing the price gap between these two approaches, you can get a sense of which option is more cost-effective.
- **Time**
  - The time required to build software in-house can be significantly longer than the time required to buy software from a third party. If the organization needs a quick solution, in this case, purchasing an existing solution may be the better option.

- **Quality**
  - It is often difficult to achieve the same level of quality with in-house software development as with commercial software. This is because commercial software is typically developed by experienced professionals who have access to better tools and resources.
- **Flexibility**
  - In-house software can be customized to meet the specific needs of an organization, but it is not always possible to find a third-party solution that meets all of an organization’s requirements.

- **Risks**
  - Developing software in-house can be a risky proposition, as there is no guarantee that the final product will meet all of the organization’s needs. Buying software from a third party carries its own risks, such as vendor dependence and the possibility of being locked into a particular platform or solution.

Thank You