Software Process Models

By:

Ms. Ashima Tyagi

Outline

- Waterfall Model
- Incremental Model
- Iterative Model
- Evolutionary Model
- RAD Models
- Agile Model and its methodologies

Software Process Models

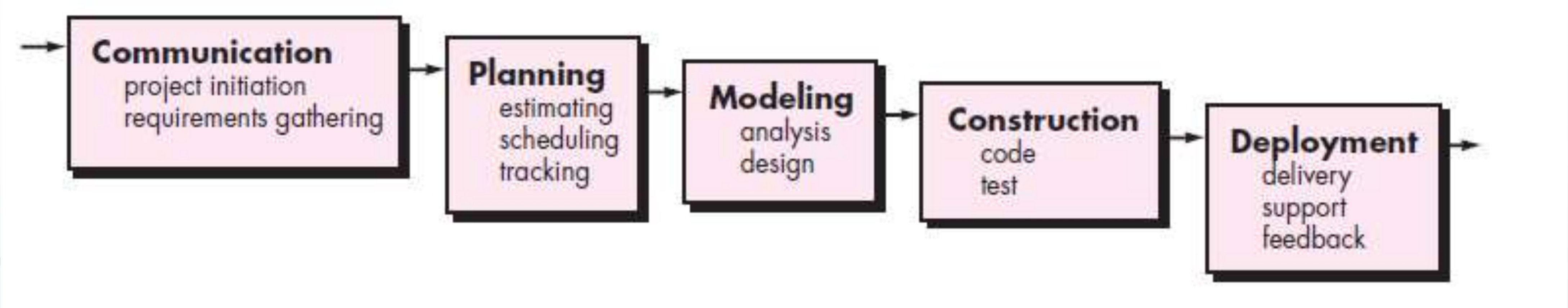
Based on the process flow, there are 4 types of software process models:

- Waterfall Model
- Incremental Model
- Iterative Model
- Evolutionary Model

Waterfall Model

- When work flows from communication through deployment in a reasonably linear fashion.
- The waterfall model, sometimes called the classic life cycle, suggests a systematic, *sequential approach to software development* that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software

Waterfall Model



Communication
project initiation
requirements gathering

Planning
estimating
scheduling
tracking

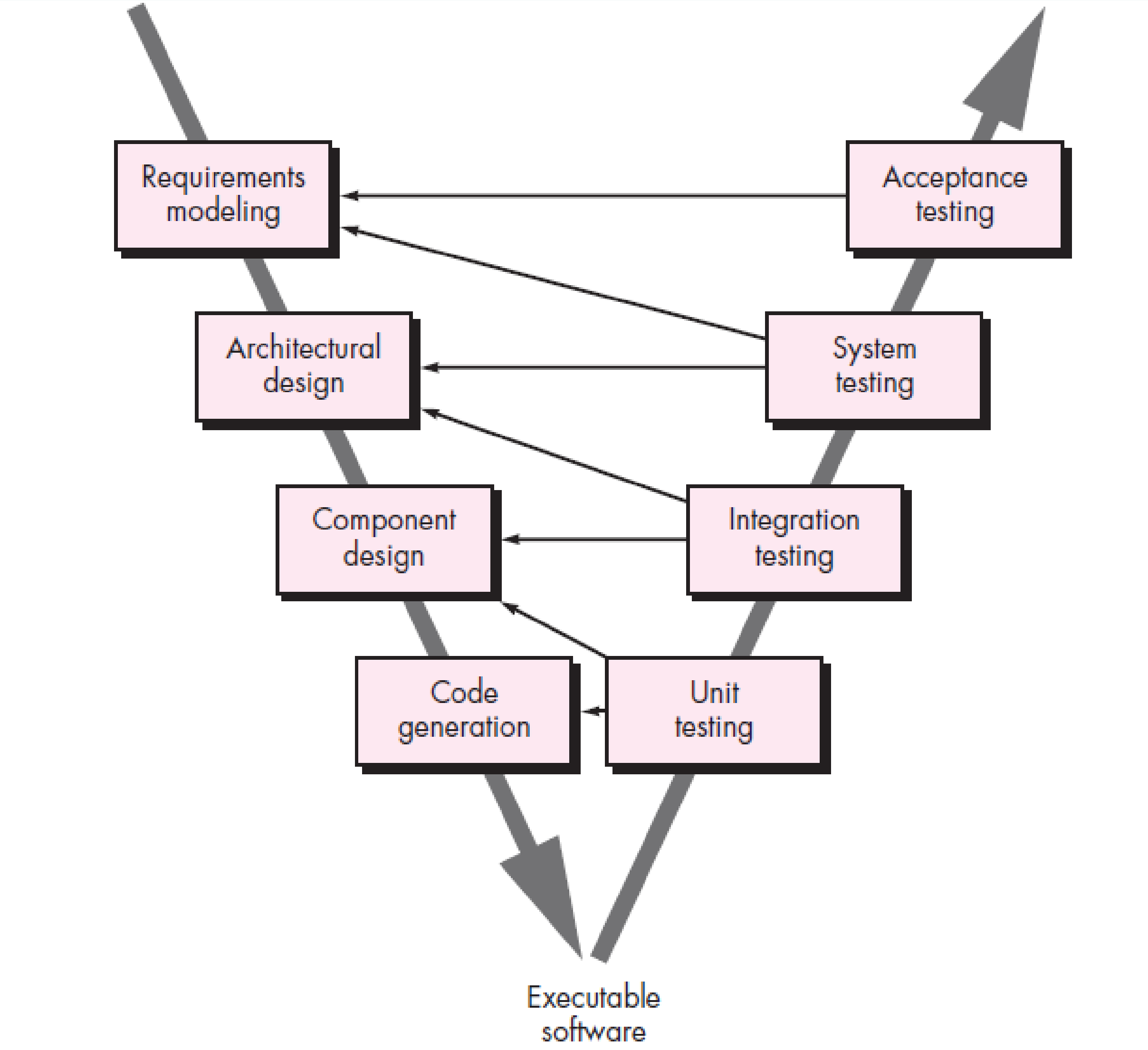
Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

V-model:

A variation in the representation of the waterfall model is called the V-model



V-model:

- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution. Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.
- In reality, there is no fundamental difference between the classic life cycle and the V-model. The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.

Advantages of Waterfall Model:

- **Simple** and Easy to Understand
- **Well-Structured and Organized:** Because the Waterfall model follows a sequential order, it ensures that there is a systematic approach to software development.
- **Emphasis on Documentation:** Each phase of the Waterfall model requires documentation before moving on to the next phase. This thorough documentation ensures that there is a formal record of the development process, which can be useful for future maintenance and enhancement.
- **Easy to Manage:** Due to its linear nature, the Waterfall model can be easier to manage, with clear milestones and deliverables. Project managers can easily monitor progress and resource allocation through the distinct phases.
- **Facilitates Early Identification of Issues:** With its emphasis on requirement analysis and design before coding begins, the Waterfall model can help in identifying potential issues early on. This can save time and resources by preventing significant changes in later stages.
- **Ideal for Stable Requirement Projects**: For projects with well-defined, unchanging requirements, the Waterfall model is particularly effective. It allows for a focused development effort without the need for constant revisions.
- **Clear Customer Expectations:** Since the Waterfall model requires detailed requirements at the beginning and has distinct phases with specific deliverables, it sets clear expectations for clients and stakeholders about what the product will do, thereby reducing the scope of misunderstanding or unexpected outcomes

Disadvantages of Waterfall Model:

- It is difficult to induce changes in the software.
- It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
- The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

Incremental Model

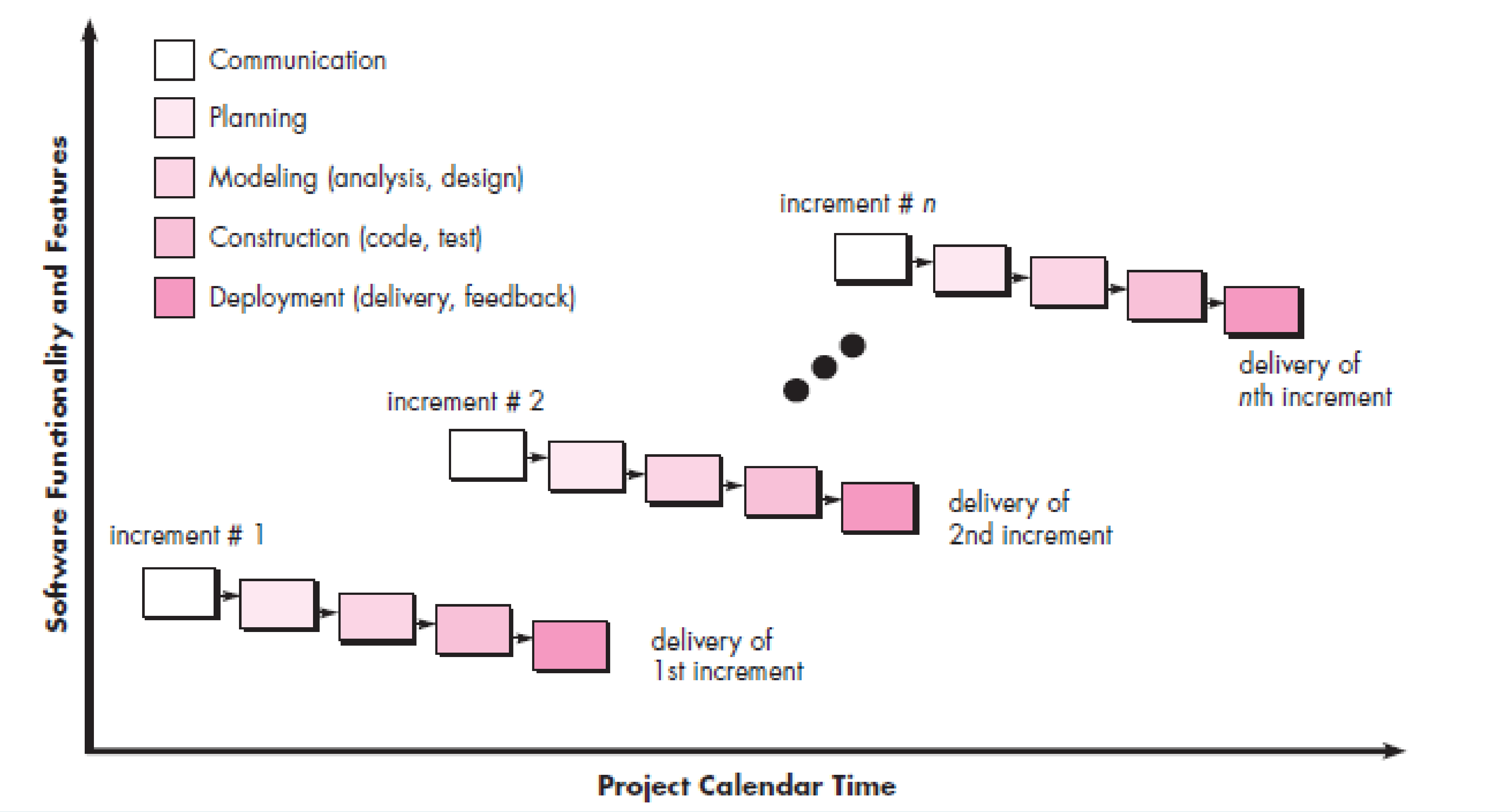
- There may be a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases.
- In such cases, you can choose a process model that is designed to produce the software in increments.
- *The incremental model combines elements of linear and parallel process flows.*
- Each linear sequence produces deliverable "increments" of the software.
- In the Incremental model of software development, increments or partial products are delivered to the customer, not just the development team. This approach divides the software development process into smaller, manageable segments, allowing each segment or increment to be developed, tested, and delivered sequentially.

For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.

It should be noted that the process flow for any increment can incorporate the prototyping paradigm.

In the **Incremental Model**, **parallel process flow** occurs because different increments (or modules) of the system can be developed **simultaneously** by separate teams. This parallelism accelerates development and enables incremental delivery of functional parts of the product.

Incremental Model



Incremental Model

- When an incremental model is used, the first increment is often a core product.
- That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation).
- As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.
- This process is repeated following the delivery of each increment, until the complete product is produced.

Advantages of Incremental Model:

- It is easy for breakdown of tasks because of divide and conquer approached used.
- It has lowers initial delivery cost.
- It has incremental resource deployment.
- It can deduct errors easily because core modules are used by the customer from the beginning of the phase and then 5. These are tested thoroughly.
- It is good to use when requirements are known up-front.
- It is good to use when projects having lengthy developments schedules.
- It generates working software quickly and early during the software life cycle.
- It is more flexible and less costly to change scope and requirements.
- It is easy to manage risk because of iterations.

Disadvantages of Incremental Model:

- It requires a good planning designing.
- It is costlier than waterfall model.
- Definition of system should be complete and clear.
- **Increased complexity**: As the project is developed incrementally, it can become more complex as each increment is added. This can make it harder to manage and maintain, as well as increase the risk of errors and bugs.
- **Higher costs:** Since each increment requires its own planning, design, coding, testing, and deployment, the overall cost of the project can be higher than with other development methodologies.
- **Difficulty in tracking progress:** With multiple increments being developed simultaneously, it can be challenging to track the progress of the project as a whole. This can make it harder to identify potential issues early on and take corrective action.
- **Increased communication overhead:** With each increment being developed by a different team or individual, there can be a significant increase in the communication overhead required to ensure that everyone is on the same page.
- **More time spent on testing:** With each increment requiring its own testing phase, more time may be spent on testing overall, which can delay the project's completion.

Iterative Model

- In this Model, you can start with some of the software specifications **and develop the first version of the software**.
- After the first version if there is a need to change the software, then a new version of the software is created with a new iteration.
- Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.
- The Iterative Model allows the accessing earlier phases, in which the variations made respectively.
- The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

Iterative Model

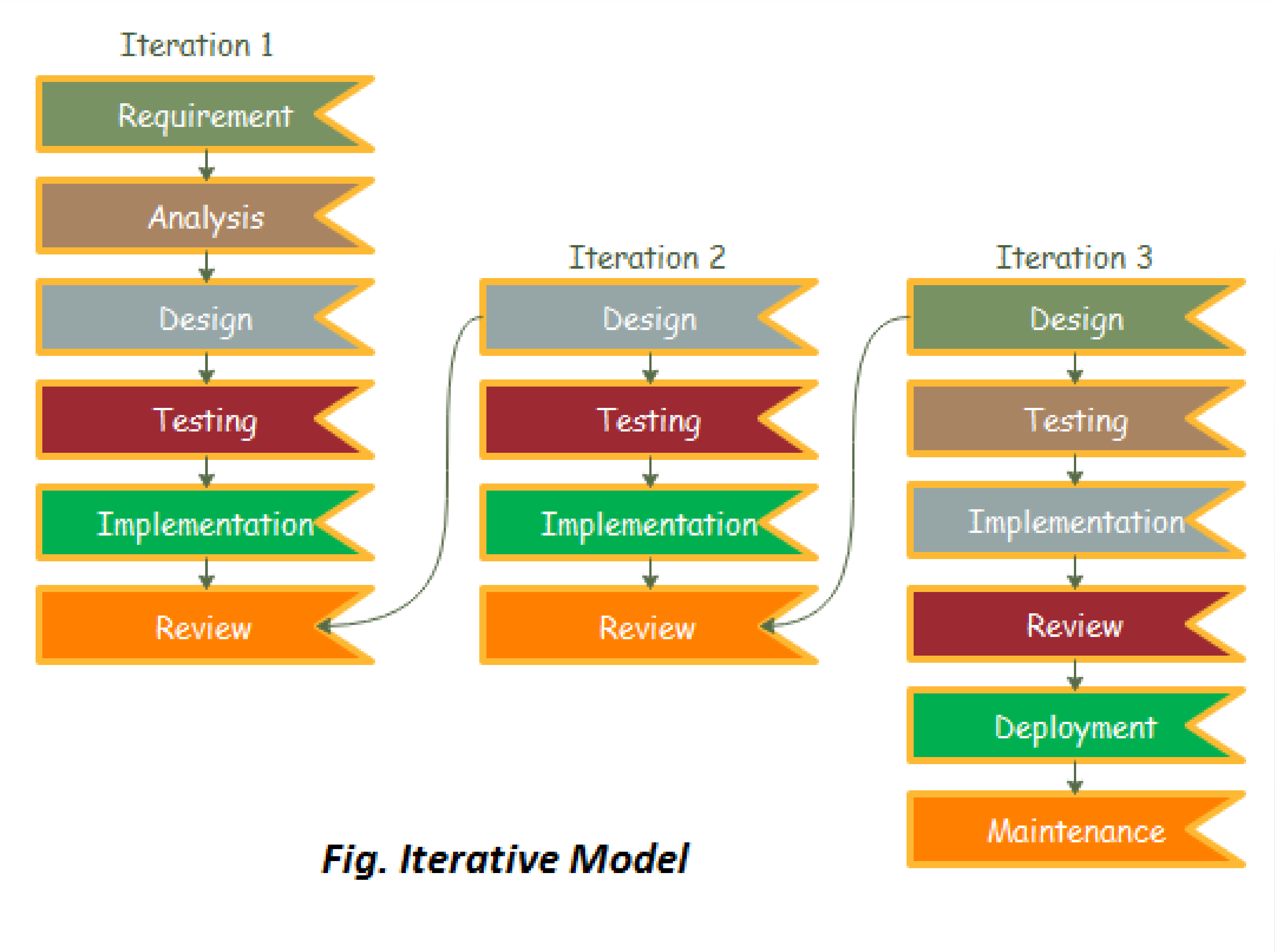


Fig. Iterative Model

Advantages of Iterative Model:

- Testing and debugging during smaller iteration is easy.
- It is easily acceptable to ever-changing needs of the project.
- Risks are identified and resolved during iteration.
- Limited time spent on documentation and extra time on designing.

Disadvantages of Iterative Model:

- It is not suitable for smaller projects.
- More Resources may be required.
- Design can be changed again and again because of imperfect requirements.
- Requirement changes can cause over budget.
- Project completion date not confirmed because of changing requirements.

Iterative model in Software Development
VS
Incremental model in Software Development

Aspect	Iterative Model	Incremental Model (Iterative Enhanced Model)
Development Approach	Repetitive cycles with continuous refinement	Delivering the software in functional parts
Flexibility	Highly flexible and adaptive to changes	Offers flexibility but less than iterative
Testing	Testing is integrated throughout the cycle	Testing is done for each increment
Delivery of Features	Complete features are developed in each iteration	Features are delivered incrementally
Risk Management	Risks are identified and addressed in each cycle	Risks are managed as increments are delivered
Client Feedback	Feedback is collected and incorporated regularly	Feedback is obtained after each increment
Project Visibility	Provides a clearer view of the project's progress	Offers visible progress with each delivered increment
Dependency Management	Dependencies can be identified and resolved in each iteration	Dependencies between increments need careful management
Completion Time	May take longer to deliver the complete product	Allows for partial delivery in shorter timeframes
Example Analogy	Writing a draft and refining it multiple times	Building a house floor by floor

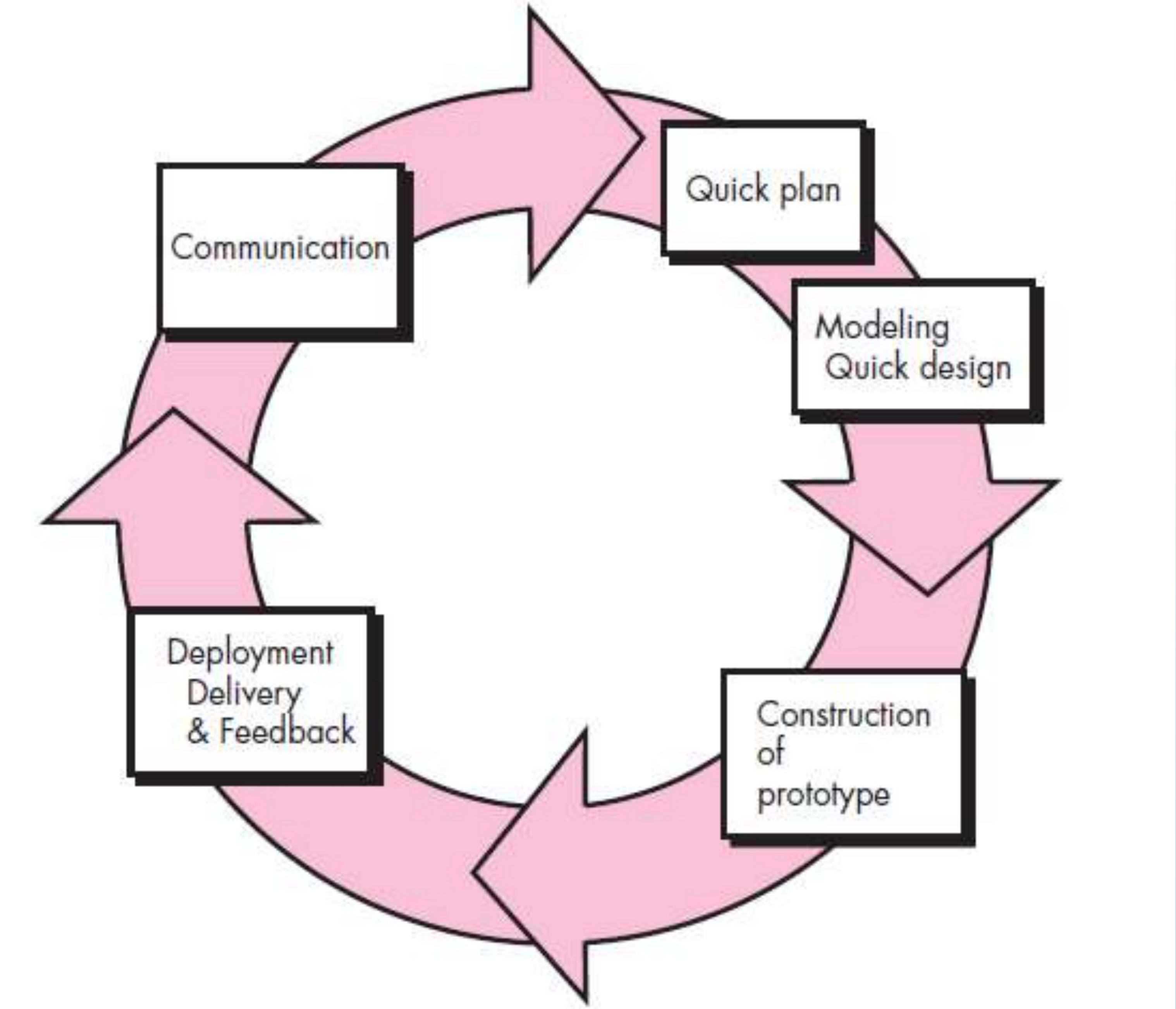
Evolutionary Process Models

- Software, like all complex systems, evolves over a period of time.
- Business and product requirements often change as development proceeds.
- You need a process model that has been explicitly designed to accommodate a product that evolves over time.
- *Evolutionary models are iterative. But in evolutionary it doesn’t require a useable product at the end of each cycle.*
- They are characterized in a manner that enables you to develop increasingly more complete versions of the software.
- There are two common evolutionary process models:
- Prototyping
- Spiral

1. Prototyping Model:

- The prototyping paradigm begins with communication.
- You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A prototyping iteration is planned quickly, and modeling (in the form of a "quick design") occurs.
- A quick design focuses on a representation of those aspects of the software that will be visible to end users (e.g., human interface layout or output display formats).
- The quick design leads to the construction of a prototype.
- The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- The prototype is developed quickly to test ideas or gather feedback, not necessarily intended for production.

Prototyping Model



Prototyping Model

- In most projects, the first system built is barely usable.
- It may be too slow, too big, awkward in use or all three.
- There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved.

- Both stakeholders and software engineers like the prototyping paradigm.
- Users get a feel for the actual system, and developers get to build something immediately.
- After the prototype is finalized, the system is typically developed from scratch (throwaway prototyping)

Advantages of Prototype Model:

- A prototype of the working model is delivered early in the project timeline.
- Changes can made at any point.
- Error found and fixed easily.

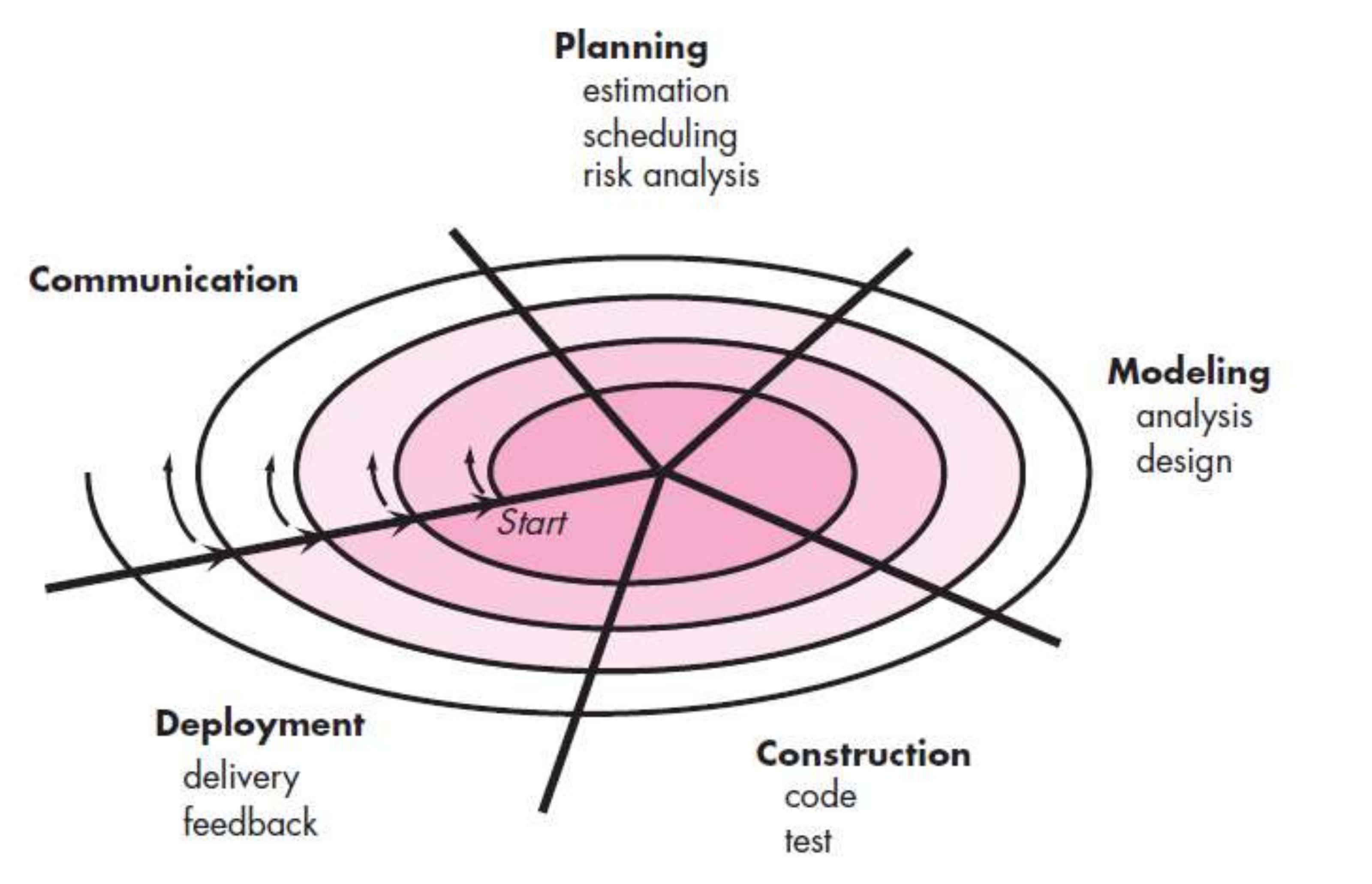
Disadvantages of Prototype Model:

- Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability. When informed that the product must be rebuilt so that high levels of quality can be maintained, stakeholders cry foul and demand that "a few fixes" be applied to make the prototype a working product. Too often, software development management relents.
- As a software engineer, you often make implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability. After a time, you may become comfortable with these choices and forget all the reasons why they were inappropriate. The less-than-ideal choice has now become an integral part of the system.

2. Spiral Model:

- The spiral model is an evolutionary software process model that couples the iterative nature of prototyping.
- Using the spiral model, software is developed in a series of evolutionary releases.
- During early iterations, the release might be a model or prototype.
- During later iterations, increasingly more complete versions of the engineered system are produced.
- The spiral model demands a **direct consideration of technical risks at all stages of the project** and, if properly applied, should reduce risks before they become problematic.
- To combine iterative development with a systematic focus on risk management and continuous refinement.

Spiral Model



- As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center.
- Risk is considered as each revolution is made.
- Anchor point milestones—a combination of work products and conditions that are attained along the path of the spiral—are noted for each evolutionary pass.
- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.

Advantages of Spiral Model:

- A prototype of the working model is delivered early in the project timeline.
- Changes can made at any point.
- Error found and fixed easily.
- Risk is handled at any time.
- Customer satisfaction
- Suitable for high risk projects

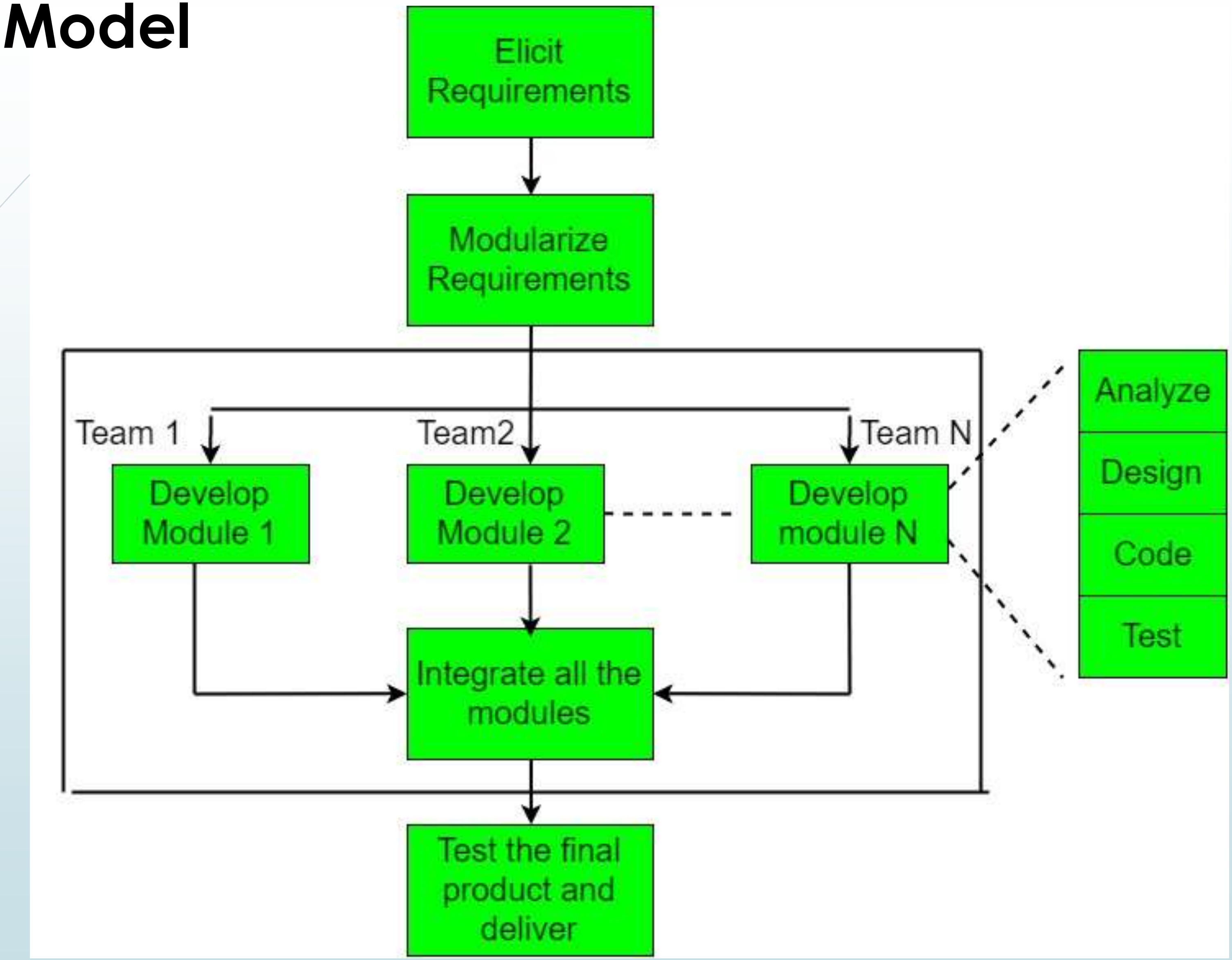
Disadvantages of Spiral Model:

- Not suitable for small projects
- Complex
- Spiral may go on indefinitely
- Not suitable for low risk projects.

RAD Model

- The **Rapid Application Development (RAD)** Model is a type of software development methodology that emphasizes quick development and delivery of high-quality software.
- It focuses on iterative development and active user involvement, enabling rapid production of prototypes or working systems.
- It focusses on minimal planning in favor of rapid prototyping.

RAD Model


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

Advantages of RAD Model:

- Faster Development Cycle: The RAD model emphasizes rapid prototyping, leading to quicker delivery of functional software.
- High Flexibility: Changes in requirements can be easily accommodated at any stage of development.
- Increased User Involvement: Regular feedback from users ensures the final product aligns closely with user needs.
- Reduced Risk: Issues and bugs can be identified and addressed early due to iterative development and frequent testing.
- Encourages Reusability: Reusable components and prototypes save time and effort in development.
- Improved Quality: Continuous testing and user feedback enhance the software quality.

Disadvantages of RAD Model:

- Dependence on User Availability: Requires consistent and active involvement from users, which may not always be feasible.
- Not Suitable for All Projects: Works best for projects with clearly defined requirements and small to medium-sized teams. It may not handle large, complex projects well.
- High Cost: Requires skilled developers, tools for rapid prototyping, and frequent user involvement, increasing the cost.
- Limited Documentation: Focus on rapid prototyping may lead to insufficient documentation, complicating future maintenance.
- Team Dependency: Success depends heavily on the skills and collaboration of the team.
- Poor Performance with Large Teams: Coordination and communication challenges arise in large, distributed teams..

Agile Model

- The main difficulties developers faced were handling customer change requests during project development and the high cost and time required to incorporate these changes.
- To overcome these drawbacks of the Waterfall Model, in the mid-1990s the Agile Software Development model was proposed.

- Agile model is iterative by design
- The Agile Model was primarily designed to help a project adapt quickly to change requests.
- So, the main aim of the Agile model is to facilitate quick project completion.
- To accomplish this task, agility is required. Agility is achieved by fitting the process to the project and removing activities that may not be essential for a specific project.
- Also, anything that is a waste of time and effort is avoided.

Agile Model

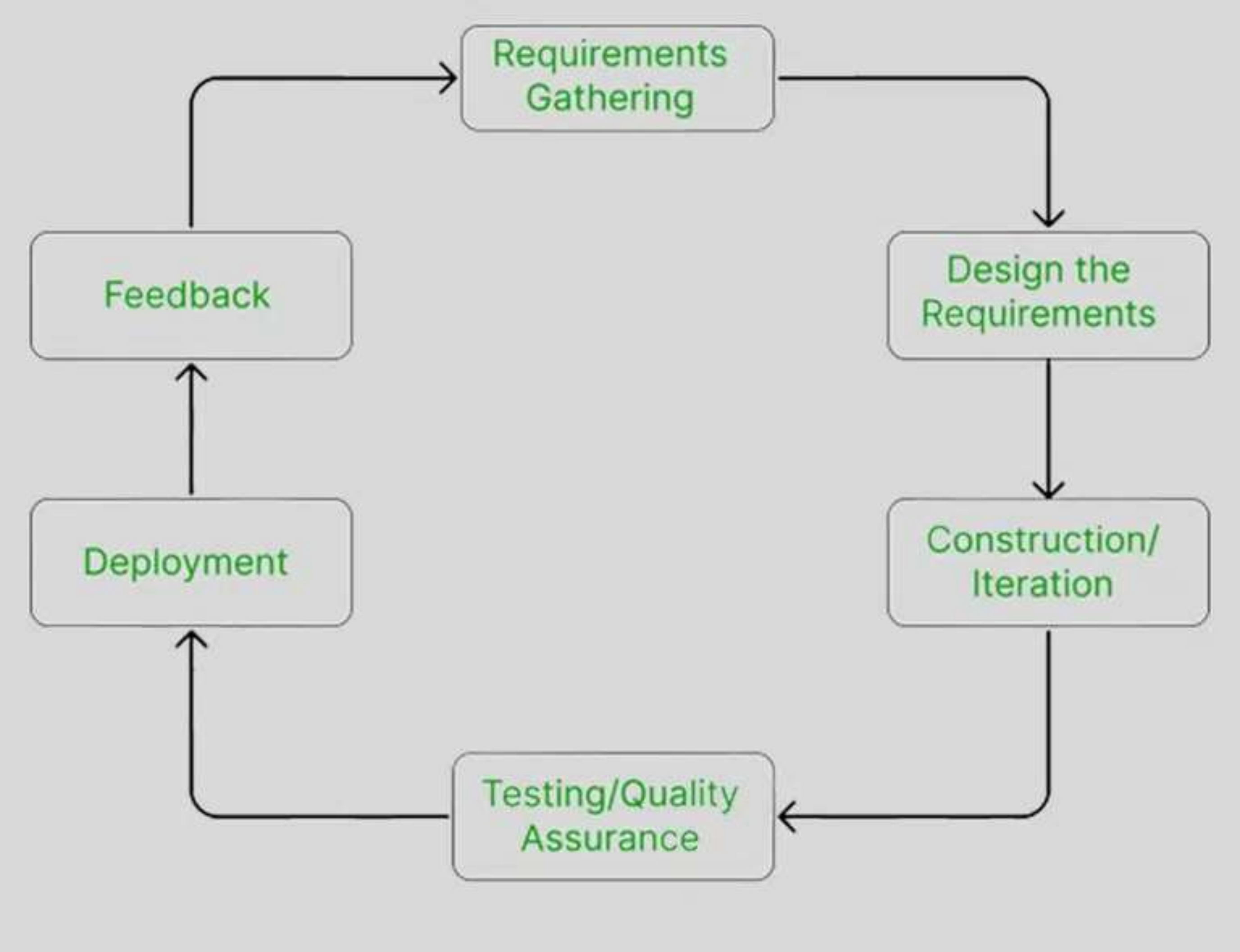


Fig:- Steps in Agile SDLC Model

Agile Model

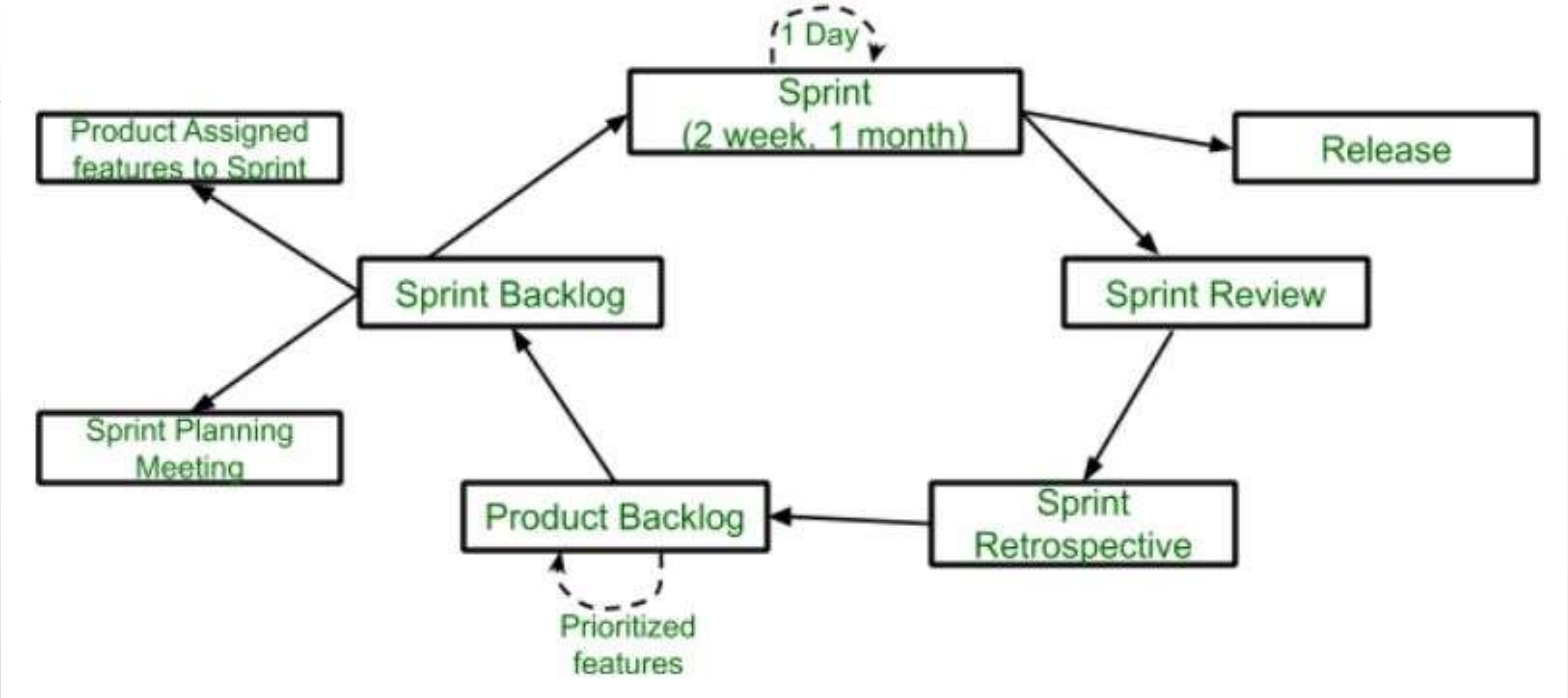
The Agile Model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves.

1. Scrum
2. XP
3. Lean
4. Kanban

1. Scrum Methodology:

- The word SCRUM is inspired by a scrum in the sport of rugby. In rugby, the team comes together in what they call a scrum to work together to move the ball forward. In this context, Scrum is where the team comes together to move the product forward.
- Think of Scrum as a way to get work done as a team in small pieces at a time, with continuous experimentation and feedback loops along the way to learn and improve as you go.
- Scrum helps people and teams deliver value incrementally in a collaborative way. As an agile framework, Scrum provides just enough structure for people and teams to integrate into how they work, while adding the right practices to optimize for their specific needs.
- The Scrum framework is fairly simple being made up of a Scrum Team consisting of a Product Owner, a Scrum Master and Developers, each of which have specific accountabilities.

Lifecycle of Scrum



Sprint: A Sprint is a time box of one month or less. A new Sprint starts immediately after the completion of the previous Sprint.
After every sprint, the product increment is typically shown to the customer or stakeholders during the Sprint Review. This is a key part of the Scrum process in Agile methodology.
Release: When the product is completed, it goes to the Release stage.

Lifecycle of Scrum

- **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.
- **Product-Assigned Features:** The Product Owner collaborates with the team to prioritize features or user stories from the Product Backlog that are to be included in the Sprint Backlog.
- **Sprint Backlog:** is a subset of the Product Backlog. It contains the list of tasks or user stories that the development team commits to completing during the sprint. It is developed during the Sprint Planning Meeting.

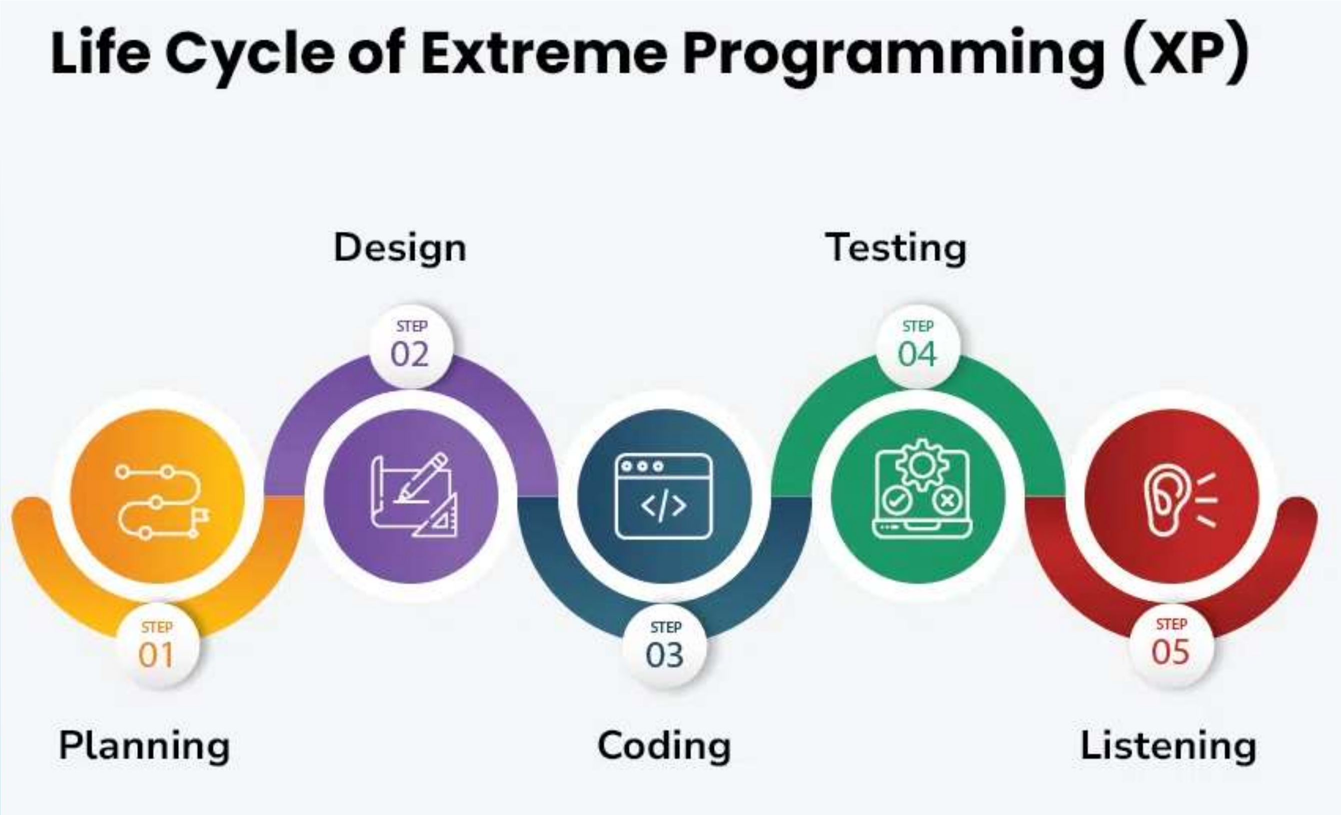
2. XP Methodology:

- Extreme Programming (XP) is an Agile software development methodology designed to improve software quality and responsiveness to changing customer requirements.
- XP is built on values like communication, simplicity, feedback, courage, and respect, with a strong focus on engineering practices and customer collaboration.

Core Values of XP:

- **Communication**: Promotes constant interaction between team members and the customer.
- **Simplicity:** Encourages building only what is needed and avoiding unnecessary complexity.
- **Feedback:** Gathers frequent feedback from customers and tests to adapt to changes quickly.
- **Courage:** Refactoring involves restructuring or improving code without changing its external behavior, which can sometimes be risky or challenging. Courage promotes boldness in refactoring, discarding outdated code, and responding to changes.
- **Respect:** Ensures a positive team environment where contributions are valued.

Lifecycle of XP



The Extreme Programming Life Cycle consist of five phases:

- Planning: The first stage of Extreme Programming is planning. During this phase, clients define their needs in concise descriptions known as user stories. The team calculates the effort required for each story and schedules releases according to priority and effort.
- Design: The team creates only the essential design needed for current user stories, using a common analogy or story to help everyone understand the overall system architecture and keep the design straightforward and clear.
- Coding: Extreme Programming (XP) promotes pair programming and developers work together at one workstation, enhancing code quality and knowledge sharing. They write tests before coding to ensure functionality from the start (TDD), and frequently integrate their code into a shared repository with automated tests to catch issues early.
- Testing: Extreme Programming (XP) gives more importance to testing that consist of both unit tests and acceptance test. Unit tests, which are automated, check if specific features work correctly. Acceptance tests, conducted by customers, ensure that the overall system meets initial requirements. This continuous testing ensures the software's quality and alignment with customer needs.
- Listening: In the listening phase regular feedback from customers to ensure the product meets their needs and to adapt to any changes.

3. Lean:

- Lean methodology in Agile is an approach focused on maximizing value while minimizing waste in the software development process.

Lean methodology or Lean Software Development(LSD) is based on 7 key principles:

1. Eliminating the Waste
2. Fast delivery
3. Amplify Learning
4. Build Quality
5. Respect Teamwork
6. Delay the Commitment
7. Optimizing the Whole System

Principles of Lean:

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 Minimum Viable Product (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: Delay decisions until you have enough information to make informed choices. This flexibility allows better handling of changing requirements. 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. Look at the entire system rather than optimizing individual parts. Ensure that all steps in the process contribute to delivering value efficiently.

LSD Process

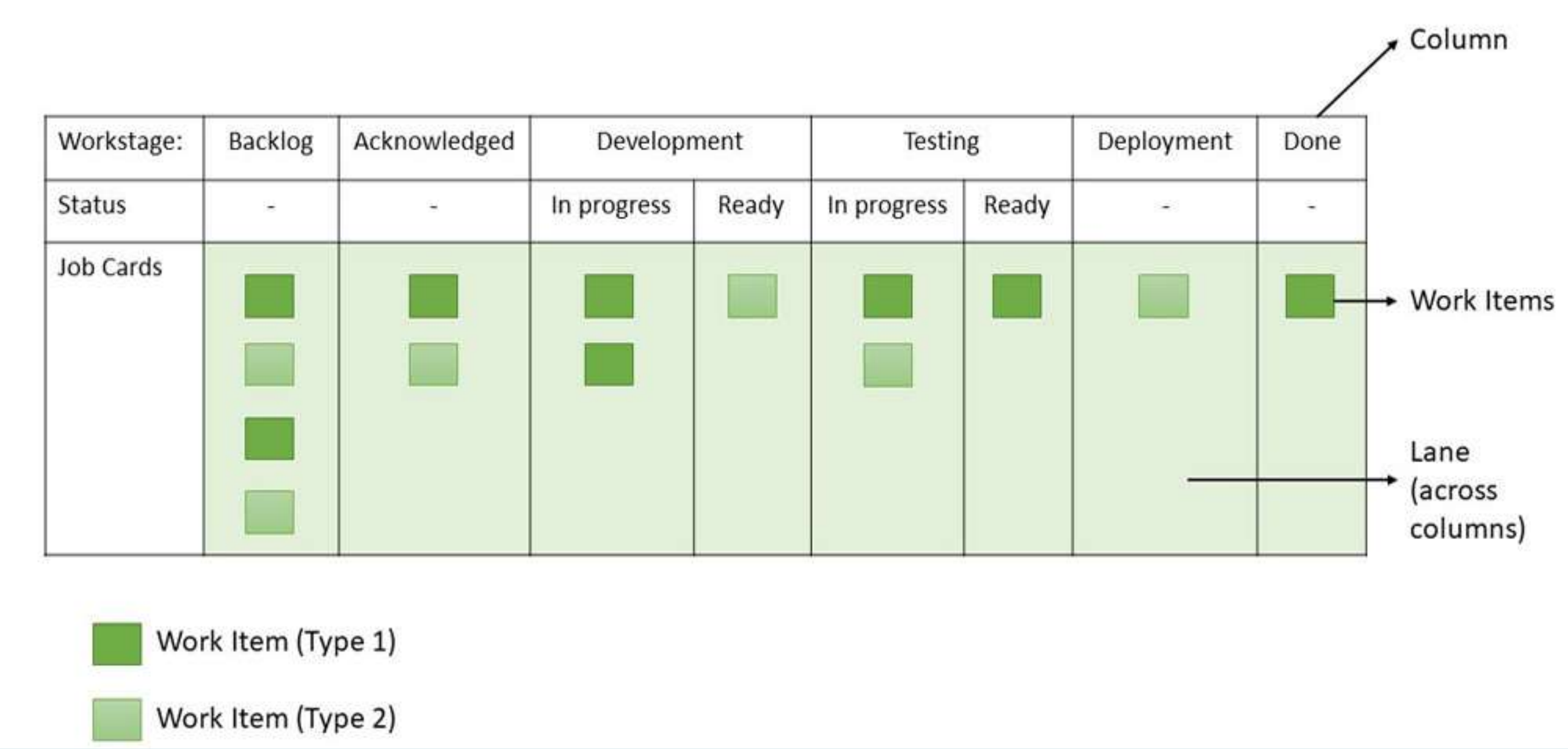
Here is the overview of the lean software development process:

1. **Identify Value:** Understand the customer values and focus on delivering features that meet these needs.
2. **Map the Value Stream:** This involves mapping out the entire software development process to identify and eliminate wasteful activities that do not add value.
3. **Create Flow:** Ensure a smooth and continuous flow of work by minimizing delays and interruptions.
4. **Establish Pull:** Develop features based on customer demand rather than pushing features through the process.
5. **Seek Perfection:** Regularly review and refine the development process. Always encourage the team members to identify the areas of improvement and implement changes iteratively.
6. **Build Quality In:** Use practices such as test-driven development (TDD) and continuous integration to integrate quality assurance throughout the development process.
7. **Empower Teams:** Empower development teams by providing them with the necessary tools, resources, and autonomy to make decisions.

4. Kanban:

- Kanban is a popular framework in the Agile methodology that emphasizes visualizing workflow, limiting work in progress (WIP), and improving efficiency through continuous delivery.
- It is highly adaptable and focuses on managing and optimizing the flow of work.

Sample Kanban Board



Workstage:	Backlog	Acknowledged	Development		Testing		Deployment	Done
Status	-	-	In progress	Ready	In progress	Ready	-	-
Job Cards	🟩 ▨ 🟩 ▨	🟩 ▨	🟩 🟩	▨	🟩 ▨	🟩	▨	🟩

🟩 Work Item (Type 1)

▨ Work Item (Type 2)

From the sample kanban board, these can be inferred about Kanban Boards or Cards:

- **Workflow**: Backlog -> Acknowledged -> Development -> Testing -> Deployment/UAT -> Done
- **Work stages:** Acknowledged, Development (In Progress), Development (Ready), Testing (In Progress), Testing (Ready), Deployment/UAT, Backlog (Arrival Queue), Done (Finished Stage)
- **Work Items:** Two types of tasks (Type 1 and Type 2) are represented by work items

Principles of Kanban:

1. **Visualize Workflow:** Use a Kanban board to display tasks as they move through stages, typically represented as columns (e.g., To Do, In Progress, Done).Helps identify bottlenecks and provides a clear view of the current workload.
2. **Limit Work in Progress (WIP):** Restrict the number of tasks being worked on simultaneously. Prevents overloading the team, ensuring focus and faster completion of tasks.
3. **Focus on Flow:** Optimize the movement of tasks through the system to achieve a smooth and predictable flow. Monitor key metrics like cycle time (time to complete a task).
4. **Make Process Policies Explicit:** Clearly define rules and guidelines for how tasks move between stages. Ensures alignment and understanding across the team.
5. **Implement Feedback Loops**: Regularly review the process and performance through meetings like daily stand-ups and retrospectives. Encourage continuous learning and improvement.
6. **Improve Collaboratively and Evolve Experimentally**: Use feedback, metrics, and experimentation to refine processes incrementally.

Thank You