Class-1 (Home Work)

Agile Development & Extreme Programming.

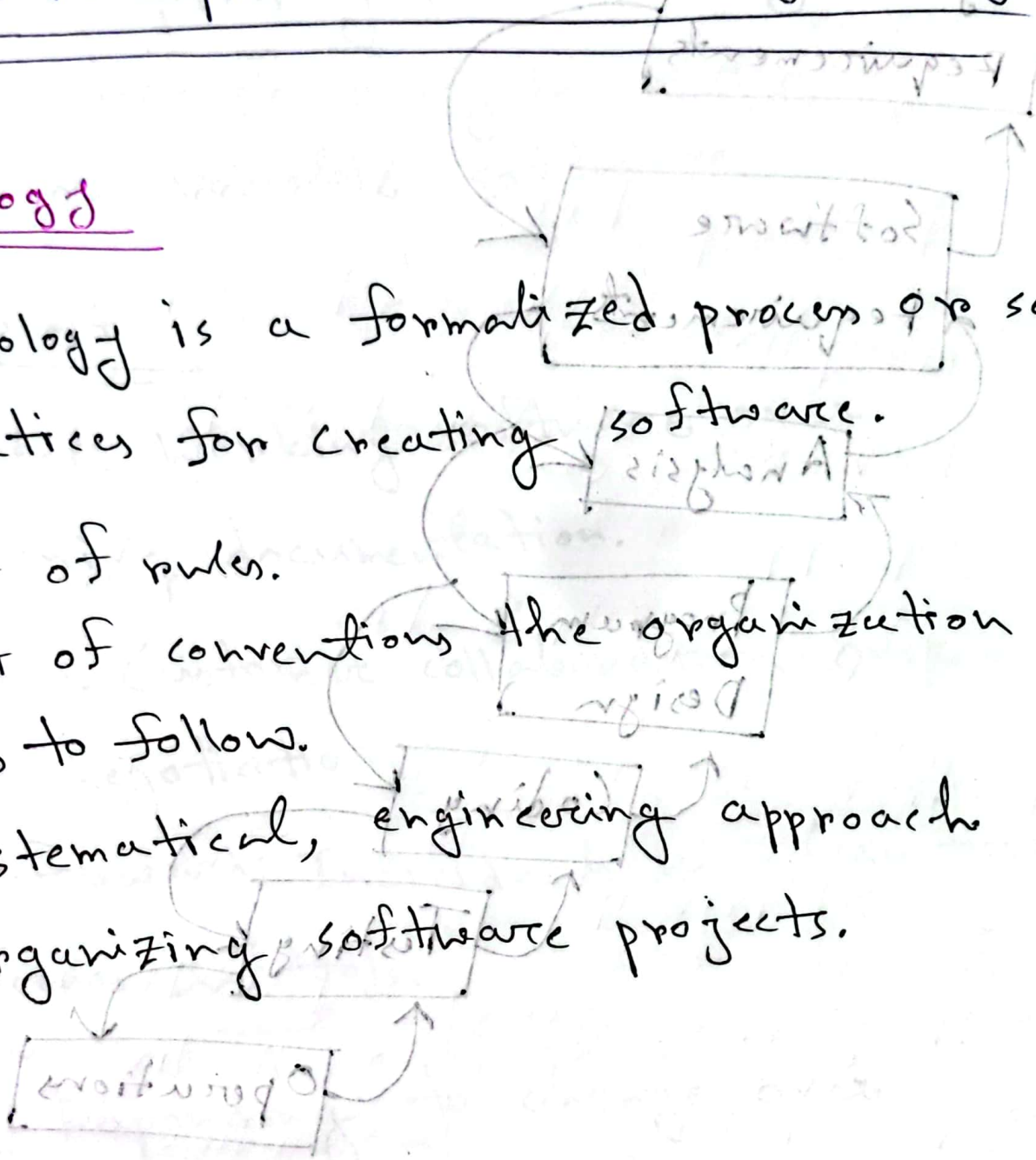
Methodology

A methodology is a formalized process or set of practices for creating software.

→ A set of rules.

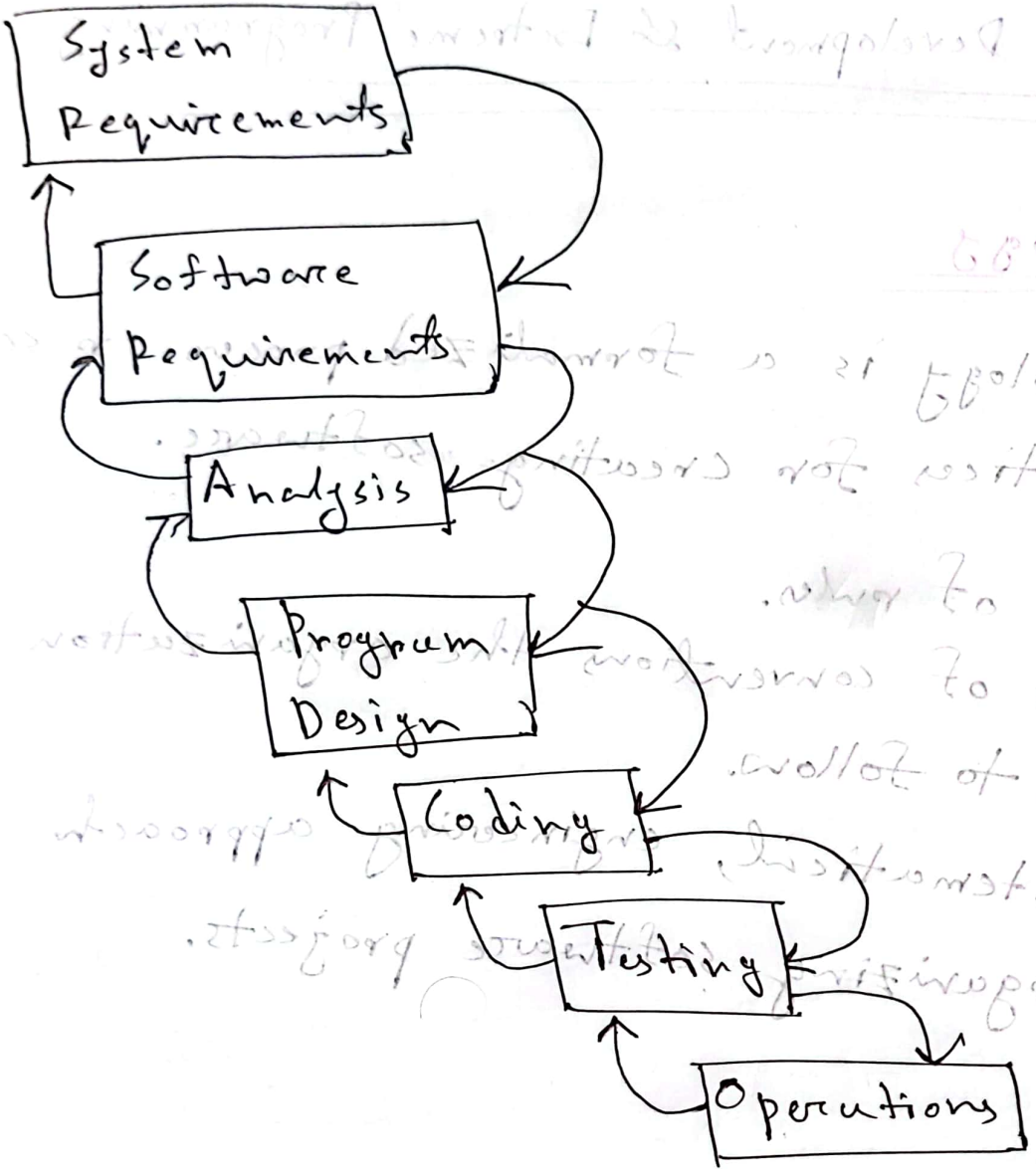
→ A set of conventions the organization decides to follow.

→ A systematical, engineering approach for organizing software projects.



P.T.O

The Waterfall Process



P.T.O

Agile Development

Our highest priority is to satisfy the customer through early & continuous delivery of valuable software.

Agile Spirit

Incremental: Working software over comprehensive documentation.

Cooperation: Customer collaboration over contract negotiation.

Straightforward: Individuals & interactions over processes & tools.

Adaptive: Responding to change over following plan.

P.T.O

Agile Methodologies

- → extreme Programming (XP)
- → Scrum.
- → Crystal family of methodologies.
- → Feature-Driven Development (FDD)
- → Adaptive-Software-Development (ASD)
- → Dynamic System Development Model (DSDM)
- → Agile Unified Process (AUP)

Extreme Programming

12 XP Principles

- → The planning game
- → Small releases
- → Metaphors
- → Simple Design
- → Test-Driven Development
- → Refactoring
- → pair Programming.
- → Collective Ownership.
- → Continuous Integration
- → 40-hour Workweek.
- → On-site Customer
- → Coding Standards.

1. Metaphor

Guide all development & conversations with a simple shared story of how the whole system works.

→ Gives the team a whole picture of describing the system, where new parts fit, etc.

→ Identify technical entities.

→ Business domain.

2. Release Planning [User stories]

→ Requirements via User stories.

→ Short cards with natural language description of what a customer wants.

→ Prioritized by customer.

→ Resource & risk estimated by developers [Planning Game]

→ Playing the planning game after each increment.

Pros/Cons [For me]

→ SRS is better than user stories.
→ Written documentation works well for large projects.
→ Prototyping the user interface as source of documentation.
→ Sometimes its hard to estimate their required resources.
→ Small releases have less risk.

3. Testing

*Test-Driven Development (TDD)

→ Write test before code.
→ Tests are automated.
→ Must run at 100% before proceeding.

*Acceptance Tests

→ Written with customer
→ Acts as "contract"
→ Measure of progress

* developers write unit tests before coding.

* Motivates coding.

→ Improves design: cohesion & coupling.

→ Proves regression tests.

→ Provides specification by example.

Pros/Cons [For me]

→ TDD is good for most projects; not for all

→ I always need the functionality "for tomorrow"!

→ Unit testing for complex logic only.

→ Testing simple logic is overhead.

4. Pair Programming

→ Two software engineers work on one task at one computer.

→ The driver has control of the keyboard and mouse and creates the implementation.

P.T.O

→ The navigator watches the driver's implementations.

→ Identifies defects & participates in on-demanding brainstorming.

→ The roles of driver & observer are periodically rotated.

Pros [For me]

→ produce higher quality code.
→ complete tasks faster.
→ enjoy work more.
→ feel more confident in work.
→ good for complex & critical logic
→ When developers need good concentration
→ Reduces time wasting.

* Trivial tasks can be done alone.
* peer reviews is often alternative.

5. Refactoring

→ Improve the design of existing code without changing its functionality.

→ relies on unit testing to ensure the code is not broken.

* Bad smells in code:

→ long method / class

→ Duplicate code.

→ method does several different things [bad cohesion].

→ Too much dependencies [bad coupling]

→ complex / unreadable code.

pros / cons [For me]

→ Delivering working software is faster is important!

→ write code to run somehow [with simple design; with less effort]

P.T.O

Class - 2 (H.W)

XP steps [continue].

6. Simple Design

* No big design up front. [BDUF]
→ Reduce the overhead.
→ Ship working functionality faster and get feedback early.
* Do the simplest thing that could possibly work.
→ Later use refactoring to change it.
* Not too much formal documentation.

For me:

* It is about establishing priorities.
* It's a set of trade offs you make.
* important for release; designed well.
* Don't lose time to design something you will not use soon!

P.T.O

7. Collective code ownership

* Code to belongs to the project; not to an individual engineer.
* Any engineer can modify any code.
* Better quality of the code.
* Faster progress.
* No need to wait for someone else to fix something.

How it works:

* Collective code ownership is absolutely indispensable.
* Fight the people who don't agree with this!
* Fire people writing unreadble and unmaintainable code.
* Don't allow somebody to own some module and be irreplaceable.

8. Continuous Integration

* Pair writes up unit test cases and code for a task.
* Pair unit test code to 100%.
* Pair integrates.
* Pair runs all unit test cases to 100%.
* Pair moves on to next task with clean state and clear mind.
* Should happens once or twice a day.

For me

* Sometimes you can't finish a task for one day to integrate it.
* For small projects with small teams integration is not an issue.
* For large and complese projects it's crucial.

9. Onsite Customer

* Clearly identify user stories.
* Make critical business decision.
* Developers don't make assumption.
* Developers don't have to wait for decisions.
* minimizes the chances of misunderstanding.

For me

* actually doesn't work!
* Customer are busy.
* Customer always say "Yes, this is what I want" & later says opposite.
* You need to think instead of them.
* Use prototyping.

P.T.O

10. Small Releases

* Timeboxed.
* As small as possible; but still delivering business value.
* No releases to "implement the database".
* Get customer feedback.
* Do the planning game after each iteration.

For me

* Manage the risk of delivering something wrong.
* Helps the customer to define better requirements.
* Release every few weeks.
* Large projects; try to release something even you know that it will be changed.

P.T.O

11. 40 Hours Work Week

* Fresh & eager every morning and tired and statisfied every night.
* Burning the midnight oil kills performance.
* Tired developers make more mistakes.
* Mess over people's personal lives, the project will pay the consequences.

For me:

* Overtime is not recommendable but often can't be avoided.
* Better planning can help.
* Highly skilled senior engineers always suffer of overtime and high pressure.

P.T.O

12. Coding Standards

* Rules for naming, formatting;
* Write readable & maintainable code.
* method commenting.
* Self-documenting code.
* Refactor to improve design.
* Re-write bad code.

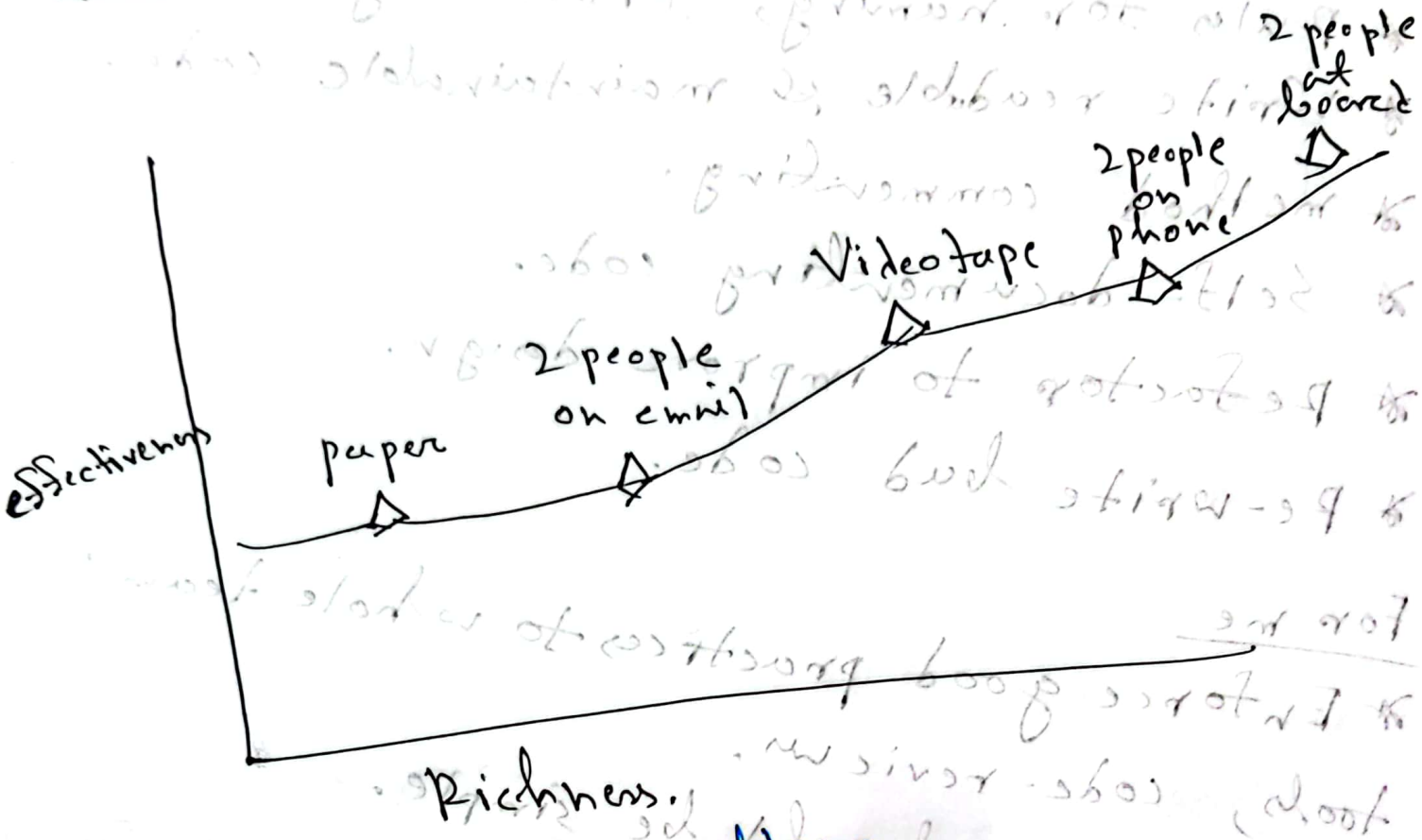
For me

* Enforce good practices to whole team; tools, code-reviews.
* Standards should be simple.
* Standards should be more strict for larger team.

13. Stand up meeting [optional]

→ Start the day with 15 minute meeting.
→ Stands up in circle.
→ Going around the rooms everyone says [did day before, plan to do today, any obstacles.]
→ Can be the way pairs are formed.

Effective communication graph



XP solve some SE problems

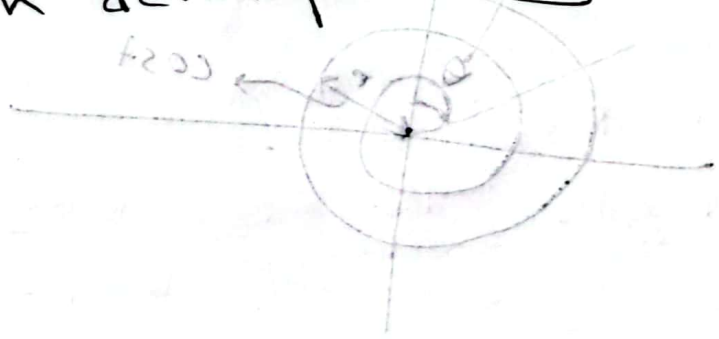
Problem	Solution
Stipped schedule	short development cycles.
Cancelled project	Intensive customer presence.
Cost of changes	on going testing.
Defect rates	Unit tests; customer tests
Misunderstand business	Customer part of the team
staff turnover	intensive team work.

XP Appl'g

* Domains with changing requirements.
* High-risk projects.
* Small projects [2-12 programmers]
* Can't be used to a large team.
* Extended development team.
* Developers; managers; customers.
* Automated testability.

Own Development Process [Mix & match].

* different agile method can be extracted & combined.
* Built it step-by-step.
* Adapt good practices one by one.
* Mix [Pair programing; 15 minute meeting; Test-driven development].



* Risk Analysis → Prototyping
  [Large scale, complex project]
* Waterfall [iterative] ✓
* Better quality
* Time consuming, costly, complicated.

[22.3.25] C.T-1 [Waterfall, XP, Scrum, Spiral] 20 marks

Scrum

Scrum is an empirical process, where decisions are based on observation, experience & experimentation.

3 pillars:

→ transparency
→ inspection
→ adaptation.

Scrum values:

→ Courage
→ Focus
→ Commitment
→ Respect
→ Openness.

Sprints

Increments of valuable work are delivered in short cycle of one month or less, which are called sprints.

It allows:
→ inspection.
→ adaptation.

Scrum team has

→ Scrum Master
→ Product Owner
→ Developers

Stakeholders

→ Organization, business, user or customer-base.
→ inspect the result of the spirints.

Scrum team

→ Cross functional [have all the necessary skills to create value each sprints].
→ self-managing.
→ small team [10 or fewer] enough to complete significant work within a sprint.

→ responsible for all product related activities [stakeholder collaboration, verification, maintenance, operation, experimentation, research & development].

The Scrum Master

helps the scrum team:

Guide the team in working independently, collaborating across roles, delivering valuable work, solving obstacles & keeping scrum meetings effective and on-time.

helps the product owner:

→ define clear goal.
→ manage the product backlog.
→ plan effectively in complex situations.
→ support team work & stakeholders.

P.T.O

Supports the organization

→ Guide, train, and support Scrum adoption

→ promote an empirical approach.

→ improve collaboration between stakeholders and Scrum team.

Scrum Master stances

→ Servant leader
→ Facilitator
→ Coach manager
→ Mentor
→ Teacher
→ Impediment Remover.
→ Change Agent.

Characteristics of A Scrum Master

→ Change Agent
→ Works accross the organization.
→ Creates transparency
→ empowers self-emplogment
→ Empirically focused.
→ Knows more than scrum.
→ Accountable for scrum team effectiveness.

→ Team builder
→ Leader
→ Takes appropriate stances.

The Accountabilities of the Product Owner

→ ensures the product delivers maximum value by setting clear goals, prioritizing work, guiding the team throughout its life cycle.

Product Owner do

→ Define & share the product goal.
→ Create & organize backlog items.
→ keep the backlog clear & accessible to everyone.

Product Owner Stances

→ Visionary.
→ Collaborator
→ Decision Maker
→ Customer Representative
→ Influencer
→ Experimenter.

The Accountabilities of a Developer

Developers are committed to creating any aspect of a usable Increment each sprint.

Developer do

→ Plan the sprint.
→ ensure quality.
→ adjust daily toward the goal.
→ hold each other accountable.
→ manage daily scrum.
→ Choose a facilitator and share skills through mentoring or pair programming to learn from each other. [Facilitation, mentoring, teaching, coaching].

P.T.O

The Scrum Events

Events that create regularity and minimize other meetings

The Sprints hold all scrum events, ensuring transparency, regularity and opportunities to inspect and adapt while reducing extra meetings & complexity.

What is Sprint

Sprints are fixed-length work cycles in scrum that turn ideas into value, ensure frequent feedback, include all scrum events for continuous improvement.

During Sprint

→ the goal stays unchanged
→ quality remains high.
→ backlog is refined.
→ scope can be adjusted with the product owner.

Product Goal

It defines the future state of the product, guiding the scrum team's work & shaping the product backlog until it achieved or changed.

Sprint Goal

The sprint goal is a shared objective that guides the sprint, providing focus, flexibility and team work for scrum team.

Empiricism

It helps the scrum team learn by doing, make decisions based on past work, and improve continuously in complex environments.

What is Sprint Planning

It starts the sprint by defining the work, with the product owner, ensuring key backlog items align with the product goal; scrum team may invite others for advice.

1. Why is the sprint valuable?

The product owner proposes how to increase value and the scrum team defines the sprint goal explaining its value to stakeholders.

2. What can be done this sprint?

Developers select and refine product Backlog items with the product owner, using past performance and capacity to forecast what can be done.

P.T.O.

3. How will the chosen work get done?

Developers plan the work to meet the Definition of Done, breaking items into smaller tasks. This along with the sprint Goal, forms the sprint backlog.

Sprint Planning is timeboxed to 8 hours for a one-month Sprint, shorter for shorter sprints.

What is a Daily Scrum?

It is a 15-minute meeting where Developers review progress, adjust the plan for the sprint goal, and improve focus, communication & decision-making.

P.T.O

Inspection & Adaption at the heart of Scrum:

The daily scrum helps developers inspect progress towards the sprint Goal, adapt the plan, improve communication, remove impediments and make quick decisions, ensuring the team stays on track to meet the goal.

The Role of Scrum Master During the Daily Scrum:

The scrum master ensures the daily scrum happens and stays within the 15-minute time-box; while developers lead the meeting, and the scrum master prevents disruptions from others.

What is a Sprint Review

It is a meeting to inspect outcome, discuss progress & collaborate on future actions, with possible product backlog adjustments. It's timeboxed to 4 hours for a one-month sprint.

It includes:

→ Scrum teams & stakeholders discussing completed & incomplete product backlog.
→ sprints successes & challenges.
→ demonstrating the increments.
→ reviewing the product backlog & potential delivery dates.
→ collaborating on next step.
→ adjusting the product backlog to reflect new opportunities, market changes, anticipated releases.

What is a Sprint Retrospective?

It is a meeting where the scrum team reflects on the sprint to identify improvements for quality & effectiveness with actionable changes added to the next sprint's plan.

It is timeboxed to a maximum of 3 hours for a one-month sprint. For shorter, shorter.

The team discusses:

→ what went well in the sprint.

→ what could be improved.

→ what will we commit to improve in the next sprint.

The scrum master encourages the team to identify & implement process improvements in the sprint retrospective to enhance effectiveness, product quality; definition of Done.

The Scrum Artifacts

They are transparent, allowing inspection and future adaption, with each artifact having a commitment to help the team track progress.

Each artifact contains a commitment:

→ For the product backlog it is the product goal.

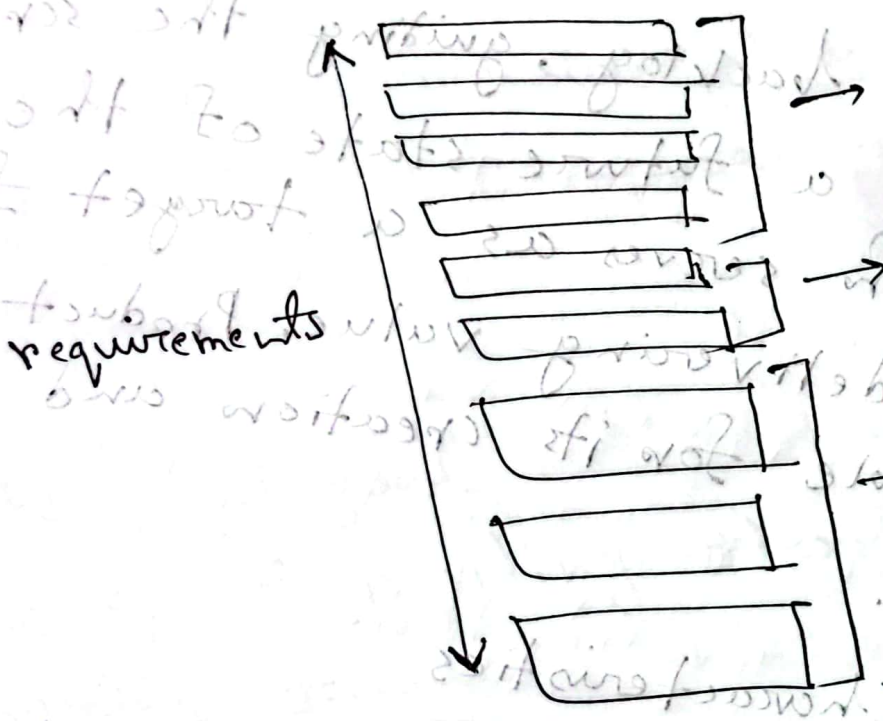
→ For sprint backlog; it is the sprint goal.

→ For the increment it is the definition of Done.

P.T.O

What is a Product backlog?

The product backlog is a dynamic, ordered list of product improvements, refined continuously to enhance transparency, with developers responsible for sizing and a single backlog used across multiple Scrum teams.



P.I.°

2.T.1

Product Backlog Refinement

It breaks down items into smaller, precise tasks, occuring anytime during a sprint to enhance transparency, though it is not mandatory, but a good practice.

Commitment : Product Goal

The product goal is the long-term objective in the product backlog, guiding the scrum team toward a future state of the product, which serves as a target for planning and delivering value. Product Owner responsible for its creation and communication.

Product Goal Characteristics

→ single, evolving objective in the product backlog. [Clear & concise]

→ guiding the scrum teams works.

P.T.O

→ influencing sprint planning.
→ ensuring transparency through continuous inspection & adaptation.

Sprint Backlog

The sprint backlog is a dynamic plan created by developers, consisting of the sprint goal, selected product Backlog items, and an actionable plan, continuously updated for transparency and progress tracking.

Commitment: Sprint Goal

The sprint goal is the single objective for the sprint, providing flexibility while ensuring focus and teamwork; it is created during sprint planning, added to the Sprint backlog and guides developers in adapting work as needed without changing the goal.

Increment

An Increment is a concrete step toward the product goal, additive to prior increments, thoroughly verified and usable; multiple Increments can be created in a sprint, delivered anytime, and must meet the definition of Done.

Sprint Goal

It is the commitment of the Sprint backlog providing a single objectives that guides the developers, remains unchanged during the sprint, and may lead to sprint cancellation if invalidated.

P.T.O

Sprint Goal Characteristics

→ clear, attainable.
→ focused on delivering value incrementally
→ visible to maintain focus, with progress inspected during the Daily Scrum.

The Definition of Done

The Definition of Done is the commitment by Developers to meet quality standards for an increment, ensuring it is complete and releasable; it creates transparency and shared understanding across the team & may include organizational standards.

P.I.°

Some examples of the Items in a DOD for a written Marketing Case study:

→ Meets feature client branding guidelines.
→ Written in AP style. [Associated press]
→ Reviewed by the client featured and feedback received.
→ Feedback implemented.
→ Final draft approved by client.

health-focused Software Application:

DOD:

→ All testing completed.
→ No known defects.
→ Code review completed & passed.
→ Meets HIPAA Compliance standards.
→ In general security requirements.

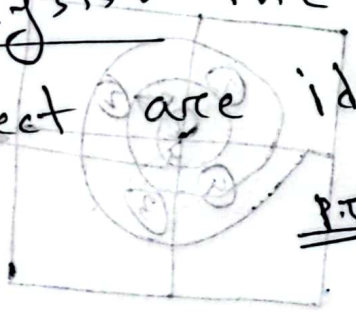
Spiral Model

The Spiral model is an iterative SDLC approach with multiple loops, each representing a development phase, allowing flexibility based on project needs. It allows project manager to determine the number of phases dynamically with each iteration representing a full software development cycle, ensuring risk management and flexibility.

Phases of the Spiral Model:

Objective Defined: clarify what the project aims to achieve, including functional & non-functional requirements.

Risk Analysis: The risks associated with the project are identified & evaluated.



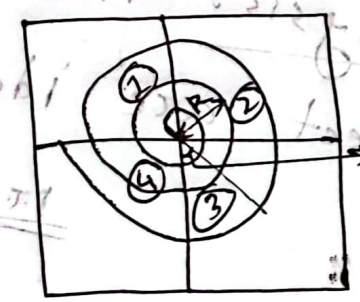
P.T.O

Engineering: the software is developed based on the requirements gathered in the previous iteration.

Evaluation: the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.

Planning: next iteration, begins with planning phase, based on the results of the evaluations.

* The radius (r) of the spiral at any point represents the expenses (cost) of the project, and the angular (θ) dimension represents the progress made so far in the current phase.



1. Objectives determination & identify alternative solutions: Requirements are gathered, analyzed and alternative solutions are proposed at the start of each phase.

2. Identify & resolve risks: solutions are evaluated; risks are addressed and a prototype is built for the best option.

3. Develop the next version of the product: features are developed, tested and next software version is released.

4. Review and plan for the next phase: customers evaluate the software; and planning for the next phase begins.

P.T.O

Risk Handeling in Spiral Model

→ The spiral model manages risks by allowing prototypes at every phase.

→ Prototyping model requires risks to be identified before development starts.

→ In real projects, risk may arise after development begins, making prototyping model unsuitable.

→ Each phase analyzes product features & identifies risks.

→ more flexible than other SDLC models.

P.T.O

Spiral Model as Meta-Model

→ A single loop represents the Iterative waterfall Model.
→ It follows the stepwise approach of waterfall model.
→ It use prototyping for risk handling like the prototyping model.
→ Iterations resemble the Evolutionary Model, building the system progressively.

Advantages

→ Risk Handeling
→ Good for Large project
→ Flexibility in requirements.
→ Customer satisfaction.
→ Iterative & incremental Approach.
→ Emphasis on risk management.
→ Improved communication.
→ Improved Quality.

Disadvantages

→ Complex
→ Expensive
→ Too much dependability on risk analysis.
→ Difficulty in time management
→ Complexity
→ Time-Consuming.
→ Resource Intensive.

When to Use Spiral Model?

→ For Large, complex project.
→ When frequent releases are needed.
→ If prototyping is required
→ When risk & cost evaluation is crucial.
→ Suitable for moderate to high-risk projects.
→ When requirement are unclear or may changes.
→ If modification are needed at any stage.
→ When long-term commitment is uncertain.