[1] Extreme Programming (XP)

An agile methodology that focuses on:
1. High Software quality
2. Fast feedback
3. Continuous Testing
4. Strong Collaboration between Team members and Customers

→ Called extreme because it pushes good practices to the extreme → Testing all the time, Integrating Code frequently, Communicating Continuously

[2] Core Values of XP

1. Communication
   - Face to Face Communication is preferred
   - Developers, Testers and Customers Communicate daily
   - Reduces misunderstanding and rework
2. Simplicity
   1. Do the Simplest thing that works
   2. Prevent over-engineering
   3. Build only what is needed now
3. Feedback
   1. Fast Feedback from Tests, Customers, Team members
   2. Helps detect problems early
4. Courage
   To: 1. change Code, 2. Refactor, 3. Admit mistakes, 4. Improve Continuously
5. Respect
   1. Team members respects each others work
   2. Everyone Contributes to quality
   3. No Blaming Culture.

[3] XP lifetime Overview → Follows a short, Continuous Cycle.

1. Planning:
   - Customers write user Stories
   - Team estimates and Prioritizes
2. Development:
   - Pair Programming
   - Simple design
   - Continuous refactoring
3. Testing:
   - Tests written before Code (TDD)
   - Automated Testing.
4. Integration:
   - Code integrated multiple times daily
   - Tests run automatically

[4] Key XP Practices

1. Pair Programming:
   - 2 developers work on one Computer
     - Driver → Writes the Code
     - Navigator → Reviews, Thinks ahead
   - Roles are Switched frequently

   Benefits:
   1. Better Code quality
   2. Fewer Bugs
   3. Knowledge Sharing
   4. Faster Problem Solving.

2. Simple Design:
   - Means: 1. Code passes all tests, 2. Code is easy to understand, 3. Minimum number of classes and Methods.
   - Goal → Maintainability and clarity

3. Refactoring:
   - Improving Code Structure without changing behavior

   Benefits:
   1. Reduces technical dept
   2. Keeps code clean
   3. Makes future changes easier.

4. Continuous Integration [CI]:
   - Code is integrated many times per day
   - Automated build and test run after every integration
   - Detects Conflicts and bugs early

5. Small Releases:
   - Deliver Software in small, frequent releases
   - Reduces risk.
   - Gets early Customer feedback.

[5] Planning and Requirements in XP

1. User Stories:
   - [4] Describe requirements from user prespective
   - As a [user], I want [Feature] So that [benefit]
2. INVEST Criteria:
   Independent, Negotiable, Valuable, Estimable, Small, Testable.
3. Story Splitting:
   - Using: 1. Workflow Steps, 2. Business rules, 3. Data variations, 4. Acceptance Criteria.

[6] Release and Iteration Testing

Decide which Stories go into release(s) Based on: 1. Business value, 2. Story Points, 3. Dependencies

Sprint:
- Select Stories for the iteration
- Break Stories into technical tasks
- Commit to test first development

[7] Testing in extreme Programming

1. Test driven development:
   - Tests are written before Code
   - TDD Cycle: Red → Writing/Posting test Failing → Write minimal Code to pass the Tests → Improve Code Structure
   - Benefits: 1. Fewer bugs, 2. Better design, 3. Built-in regression Testing, 4. Safer refactoring
2. Unit Testing:
   - Tests individual Components
   - Goes: 1. Isolated, 2. Repeatable, 3. Automated, 4. Self-validating
   - Common frameworks: 1. JUnit, 2. NUnit, 3. PyTest, 4. Jest
3. Acceptance Testing:
   - Defined by Customers
   - Validates business requirements
   - Often automated

   Acceptance tests act as executable Requirements

[8] Agile Testing Principles

1. Whole Team approach:
   - Quality is everyone's responsibility
   - Developers, Testers, Customers work together
2. Collective Code Ownership:
   - Anyone can modify any Code
   - No ownership silos
   - Encourages Shared responsibility
3. Continuous Customer Involvement:
   - Customer give Continuous Feedback
   - Clarify requirements early

[9] Agile Testing Quadrants

1. Unit Testing → Code level
2. Integration Testing → Component interaction
3. Acceptance Testing → Business validation
4. System Testing → End-to-end validation

[10] XP Implementation Challenges

1. Cultural Resistance:
   - Fear of change
   - Managers losing Control
   - Developers afraid of exposing mistakes
2. Skill Gaps:
   - Pair Programming requires Communication Skills
   - TDD needs testing Knowledge
3. Organizational Conflicts:
   - XP Clashes with Siloed departments
   - Needs Cross functional teams

[11] Technical Debt in XP → Short-term Shortcuts that Cause long-term Problems

- XP reduces debt through:
  1. Pair Programming
  2. TDD
  3. Continuous integration
  4. Refactoring

[12] Scaling XP → For large or distributed teams

- Break into Small Sub-teams
- Virtual Pair Programming
- Use Automation heavily
- Strong Continuous integration Pipelines

[13] Tester engagement Strategy

→ A plan to involve testers throughout the entire development lifecycle not just at the end

- Shift left and Shift Right:
  - Testing early → Requirements and planning
  - Testing after release → Monitoring and feedback.
- Problems without Strategy: Testers Seen as [Quality police], Burnout, Less respect on decisions

[14] The 4 Pillars of Tester engagement

1. Cultural Shift:
   - Quality is a team responsibility
   - Measure quality as a team metric
2. Process Integration:
   Testers Participates in:
   1. Backlog Refinement
   2. Sprint planning
   3. Daily Scrum
   4. Sprint review
   5. Retrospectives
3. Technical Collaboration:
   - Tester + Developer Pairing
   - Shared test automation
   - Mob Testing
4. Skill Development:
   - Cross Training
   - Communities of Practice
   - Clear Career paths for testers