Chapter 2

Agile Software Development

Methodologies, Practices & Implementation

Dr. Hitesh Mohapatra

Chapter Overview

- Introduction to Agile methodology
- Core principles and values
- Agile frameworks and approaches
- Planning and requirements gathering
- Implementation and best practices

What is Agile?

- Iterative and incremental approach to software development
- Responsive to change and customer feedback
- Emphasizes teamwork and continuous delivery
- Flexible alternative to traditional waterfall methods

Core Agile Principles

- Individuals and interactions over processes
- Working software over documentation
- Customer collaboration over contracts
- Responding to change over rigid plans

Extreme Programming (XP)

- Most famous agile methodology
- Set of simple yet interdependent practices
- Practices work together to form a cohesive whole
- Emphasizes technical excellence and continuous improvement

XP Core Practices (Part 1)

- Pair Programming: Two programmers per workstation
- Test-Driven Development (TDD)
- Continuous Integration: Code checked in daily
- Coding Standards: Uniform code style

XP Core Practices (Part 2)

- Refactoring: Continuous code improvement
- Collective Code Ownership
- Simple Design: Minimal yet complete solutions
- Metaphor: Shared vision of system architecture

Agile Planning Approach

Planning game divides responsibility between business and development:

- Developers know technical feasibility and risks
- Business prioritizes stories based on value
- Iterations typically 1-2 weeks in length

Requirements Gathering

- User stories capture functional requirements
- Estimates based on complexity not time
- Specifications emerge during development
- Close collaboration with stakeholders

Release and Iteration Planning

Release planning creates a release plan spanning 3-6 months:

- Map next iterations
- Collect user stories and prioritize

Testing Strategy

- Unit tests written before code implementation
- Acceptance tests created by customers
- Continuous testing throughout development
- All tests run multiple times daily

Pair Programming

Two programmers work on same code:

- One writes code (driver)
- One reviews and provides feedback (navigator)
- Improves code quality and knowledge sharing

Code Quality Practices

- Refactoring: Continuous improvement without changing functionality
- Simple design: Minimal yet complete solutions
- Coding standards: Consistent style and conventions
- Technical debt reduction

Continuous Integration

- Code integrated multiple times daily
- Automated builds and tests
- Early detection of integration issues
- Rapid feedback to developers

Agile Team Structure

- Cross-functional teams with diverse skills
- Daily stand-up meetings
- Shared responsibility for outcomes
- Product owner defines priorities

Customer and Stakeholder Engagement

- Customers deeply involved in development
- Regular feedback sessions and reviews
- Acceptance criteria defined collaboratively
- Adaptive to changing requirements

Documentation Approach

- Minimal but sufficient documentation
- Focus on working software over documentation
- Self-documenting code through clarity
- Just enough design upfront

Metrics and Project Tracking

- Velocity tracking for estimation accuracy
- Burndown charts monitor progress
- Defect tracking and resolution
- Team productivity and quality metrics

Agile Challenges

- Scaling agile to large organizations
- Managing distributed teams effectively
- Balancing flexibility with predictability
- Technical debt management

Key Takeaways

Agile enables teams to deliver quality software through iterative development, continuous feedback, and adaptive planning while maintaining focus on customer needs and technical excellence.