Bachelor of Science in Computer Science (BSc in CS)

SCS1303: Software Engineering

Agile Processes
Agile Software Development

Prof. K. P. Hewagamage

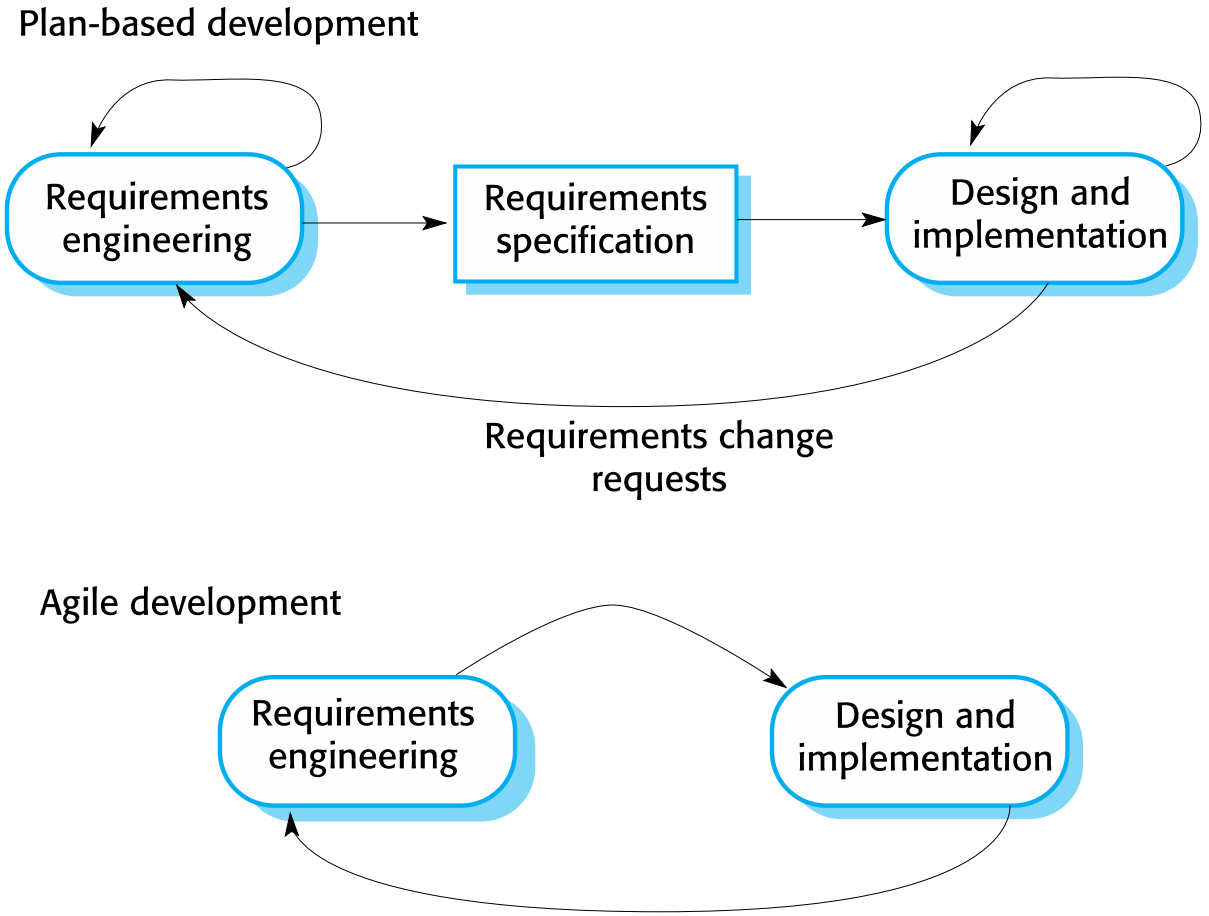
Rapid software development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems

Agile development

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

Plan-driven and agile development



Plan-driven and agile development

- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

Agile ……..

A kind of buzzword ? To describe the modern software development

Is Agile Fragile ?

Pervasiveness of Change?

Changes in Requirements

Changes in Design, Implementation, ….

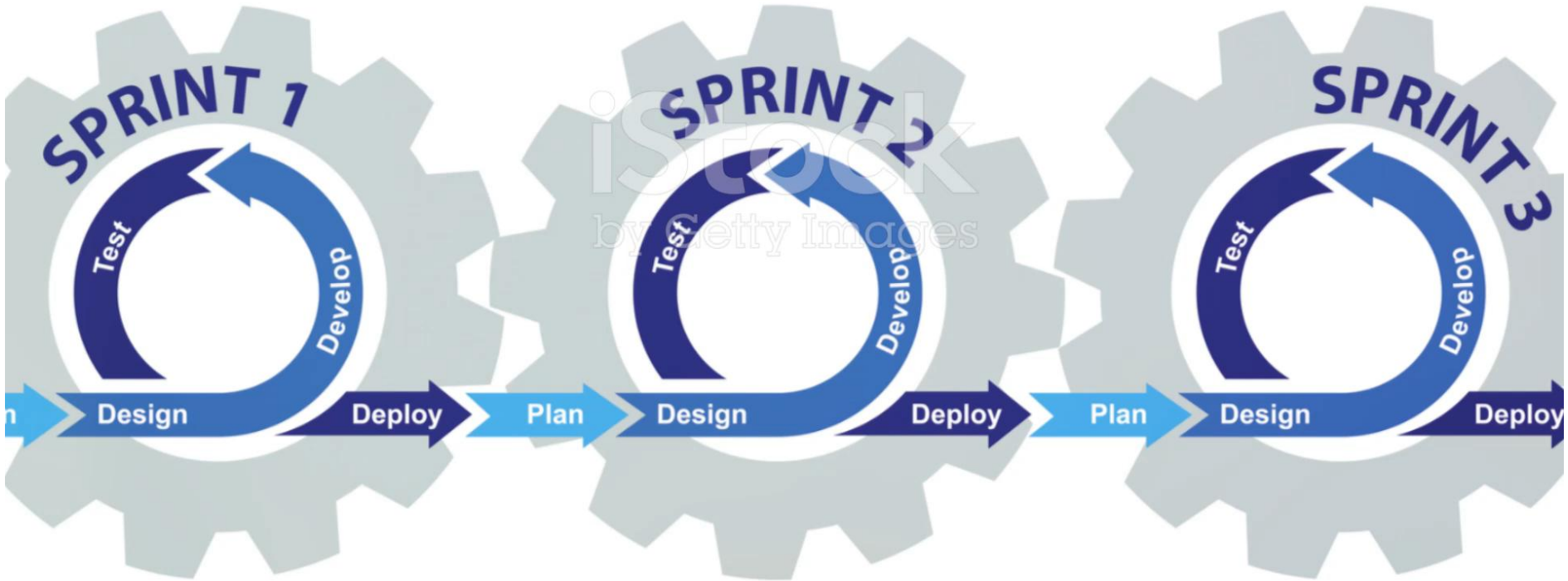
Changes in Technology

Changes in Team

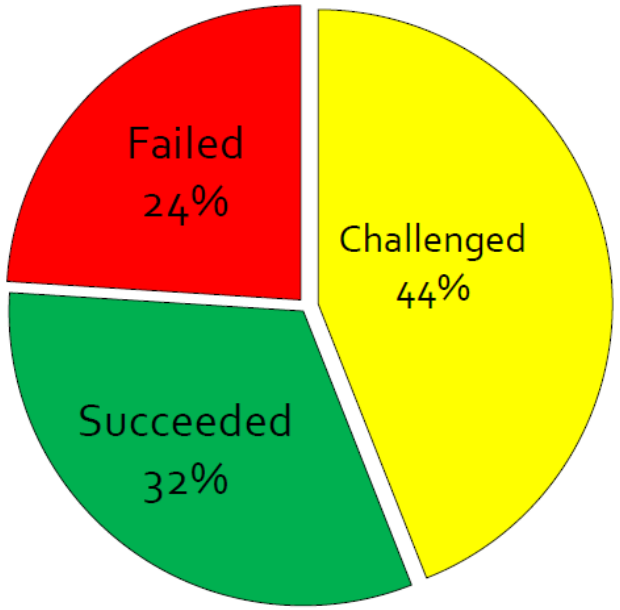
Changes in users/client contacts

S/W Eng. must be quick to accommodate the rapid changes

AGILE METHODOLOGY



Status of Software Projects in 10 Years ago…

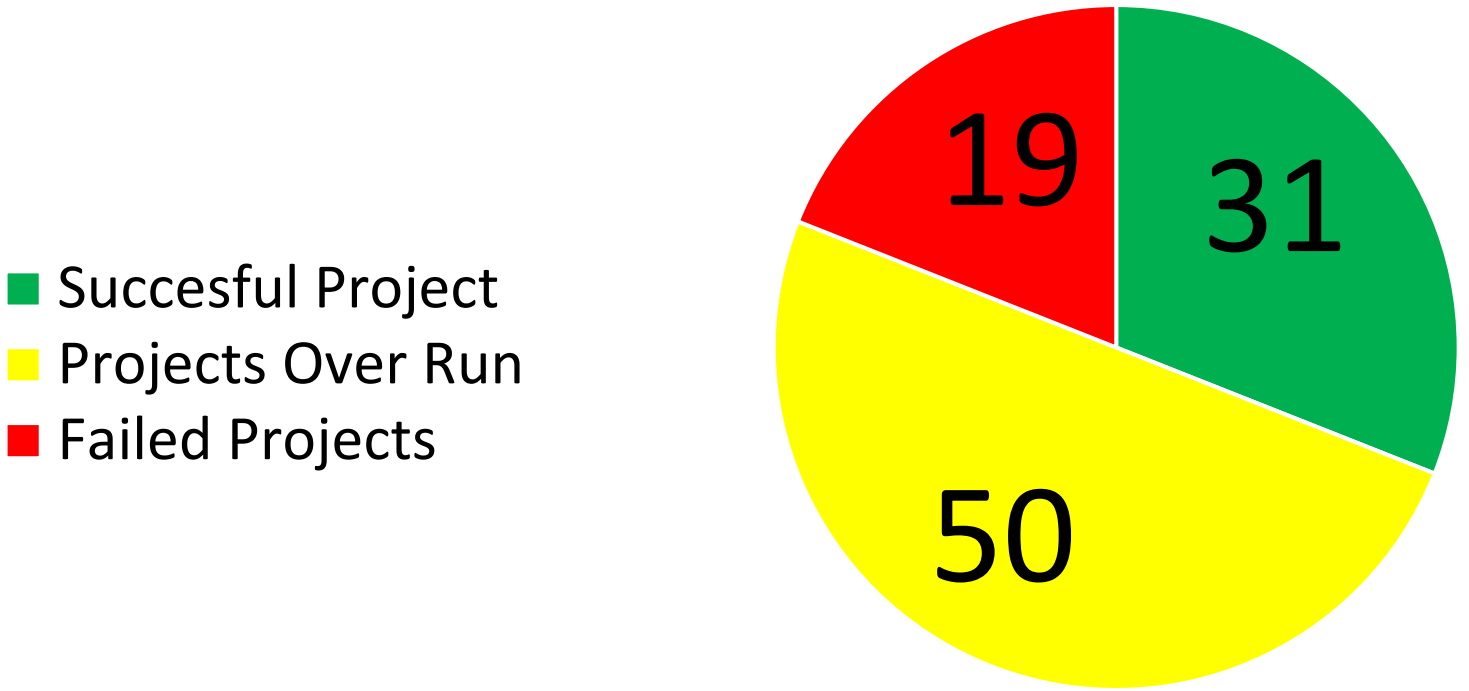


- 32% of all projects succeeded (on time, on budget, with required features)
- 44% are challenged (late, over budget and/or with less than the required features)
- 24% have failed (cancelled prior to completion and/or never used)

Source: Standish CHAOS Report 2009
Boston, Massachusetts, April 23, 2009

Status of Software Projects after 10 Years

Sandish CHOS Report 2024



The top five factors found in successful projects are:

1. User involvement (Engaged users + continuous feedback)
2. Executive management support/Project Management
3. Clear Statement of Requirements/Objectives
4. Proper planning/Strong Teams
5. Realistic Expectations/Adaptability and Risk Management

The top five indicators found in challenged projects are:

1. Lack of user input
2. Incomplete/changing Requirements & Specifications
3. Scope Creep/Changing Objective/Goals
4. Lack of executive support/Poor Management
5. Technical incompetence/Weak Teams

All of the top factors found in failed projects include

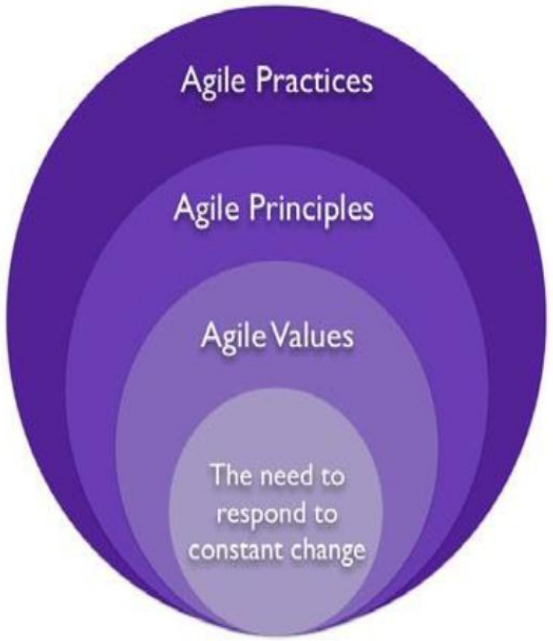
- Incomplete Requirements
- Lack of user involvement
- Lack of resources
- Unrealistic expectations
- Lack of executive support
- Changing Requirements & Specifications
- Lack of planning
- Didn’t need it any longer
- Lack of IT management
- Technical illiteracy

What is agile?

Philosophy + a set of Guidelines for software development

"Agile is a mind set defined by **Values**, guided by **Principles** and manifested through many different **Practices**"

Relationship between Agile Values, Principles and Practices



- 4 Values
- 12 Principles
- Many practices, grouped with as methodologies
  - E.g. XP, SCRUM, ….

Communication and collaboration

Adopts the customer/user as a part of team

to eliminate the attitude "*us and them*" Flexible project planning in uncertain world

Don't make mistakes of assuming that agility gives you license to do it

A process is required and <u>discipline is essential</u>

Cooperation and Collaboration
Chickens and Pigs

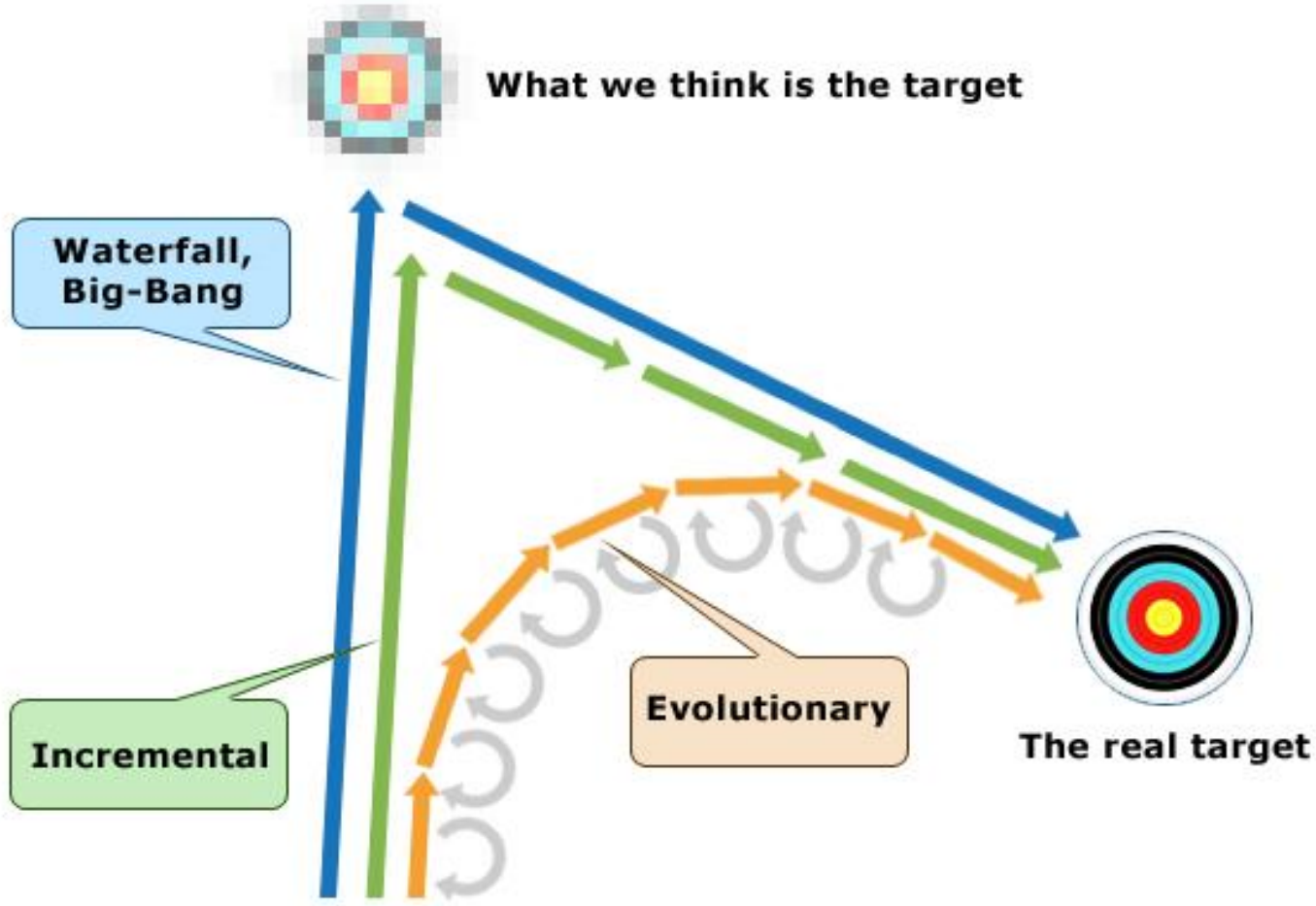
Pigs: Product Owner, Team, Scrum Master

Chickens: Everyone else who is involved, consulted or interested in the project

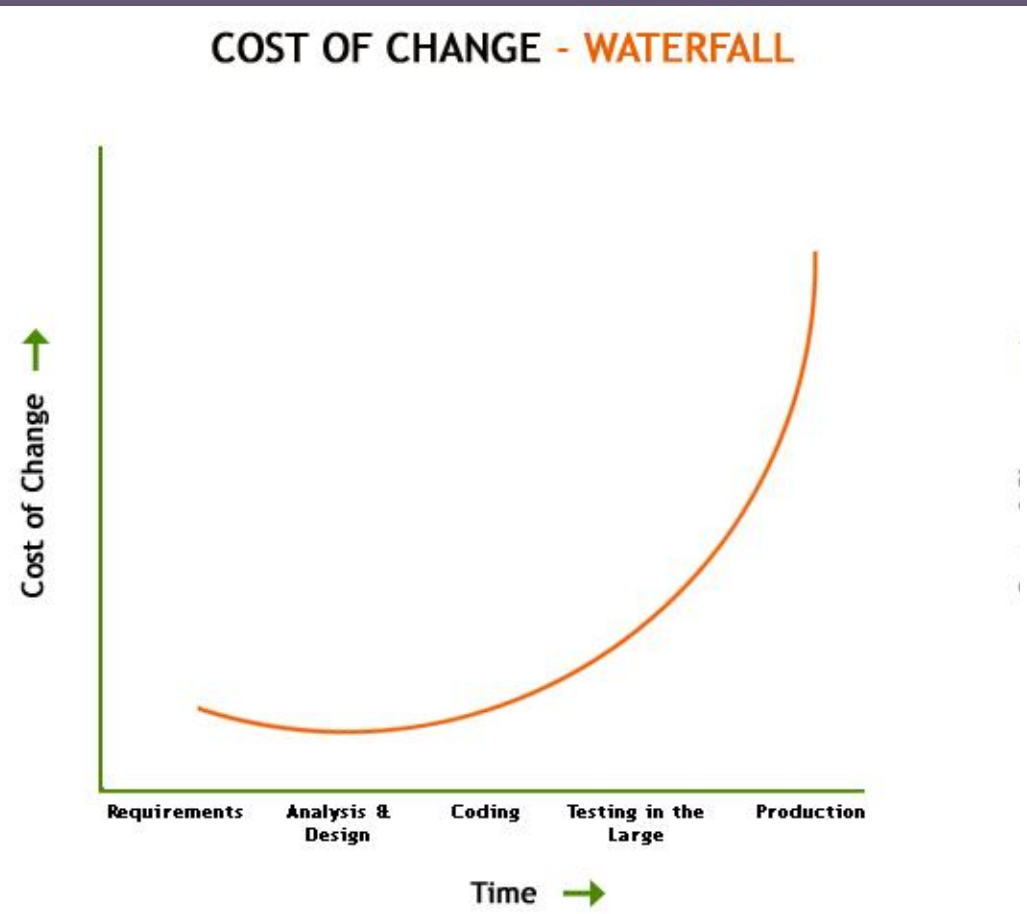
Agile Software Development:
Learn ⇢ Adapt ⇢ Deliver

- Ability to adapt
  - To changing requirements, To changing circumstances
- Agile Software development?
  - based on Learn & Adapt rather than rigid processes
  - Evolving systems in short iterations
    - Each release is a working system
    - Focus on business value
    - Design for change
  - Communicating efficiently & real time visibility to progress
  - Leveraging human strengths
    - Engage, align, and empower the team
    - Get power from each member
  - Agile is not an excuse for “no discipline”

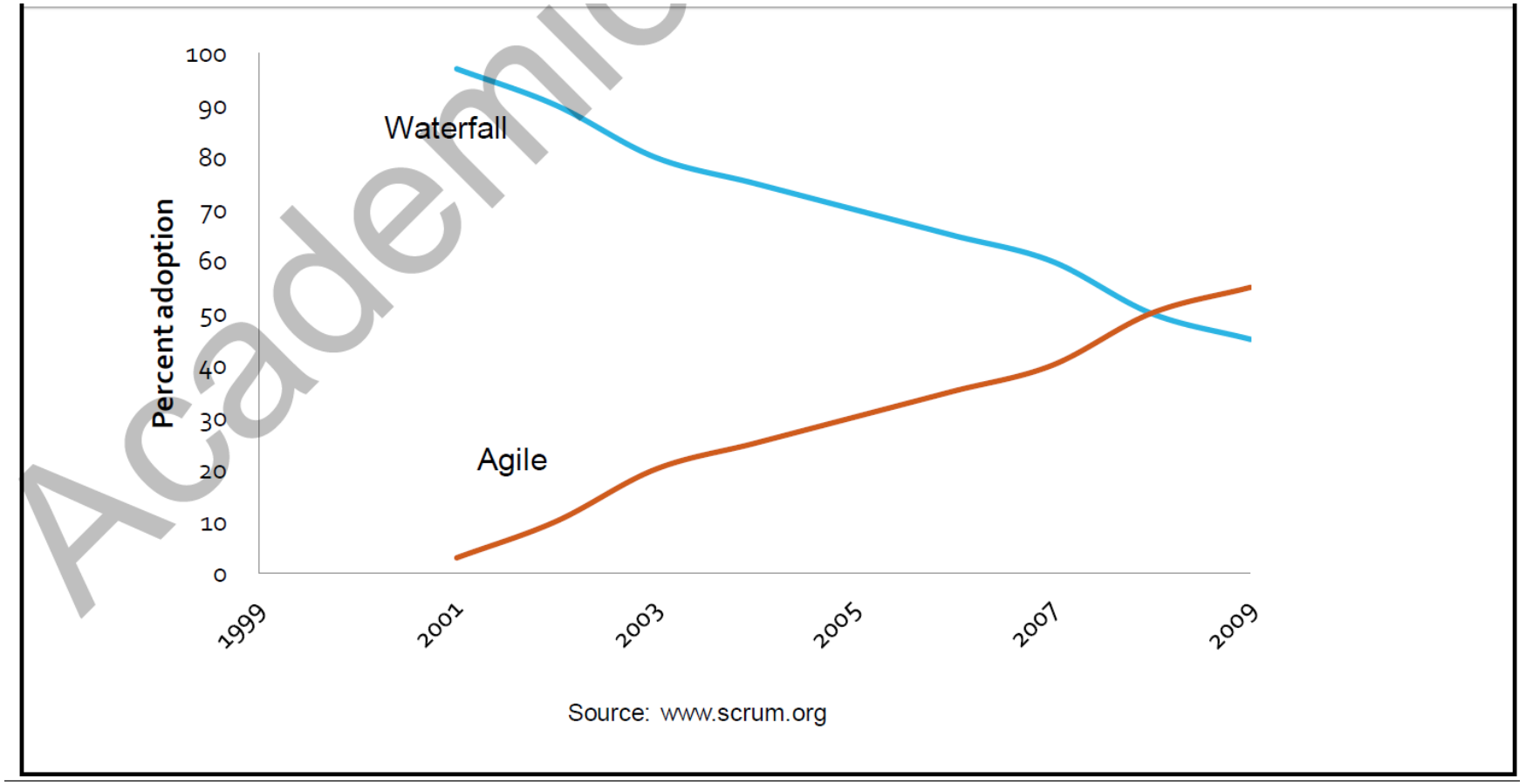
Reaching the Goal

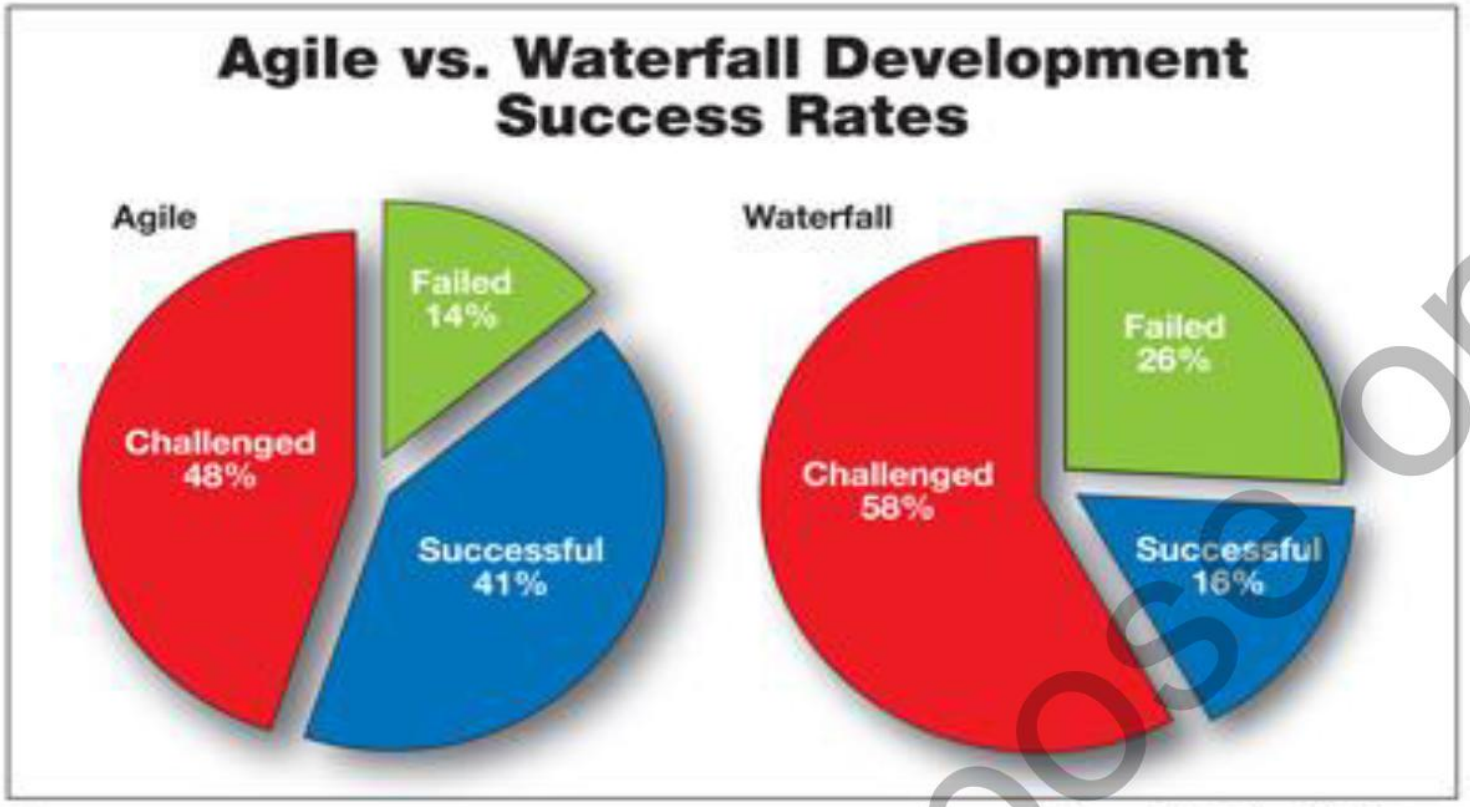


Why Agile? ……


COST OF CHANGE - WATERFALL
Cost of Change
Requirements
Analysis & Design
Coding
Testing in the Large
Production
Time

No. of Projects Success

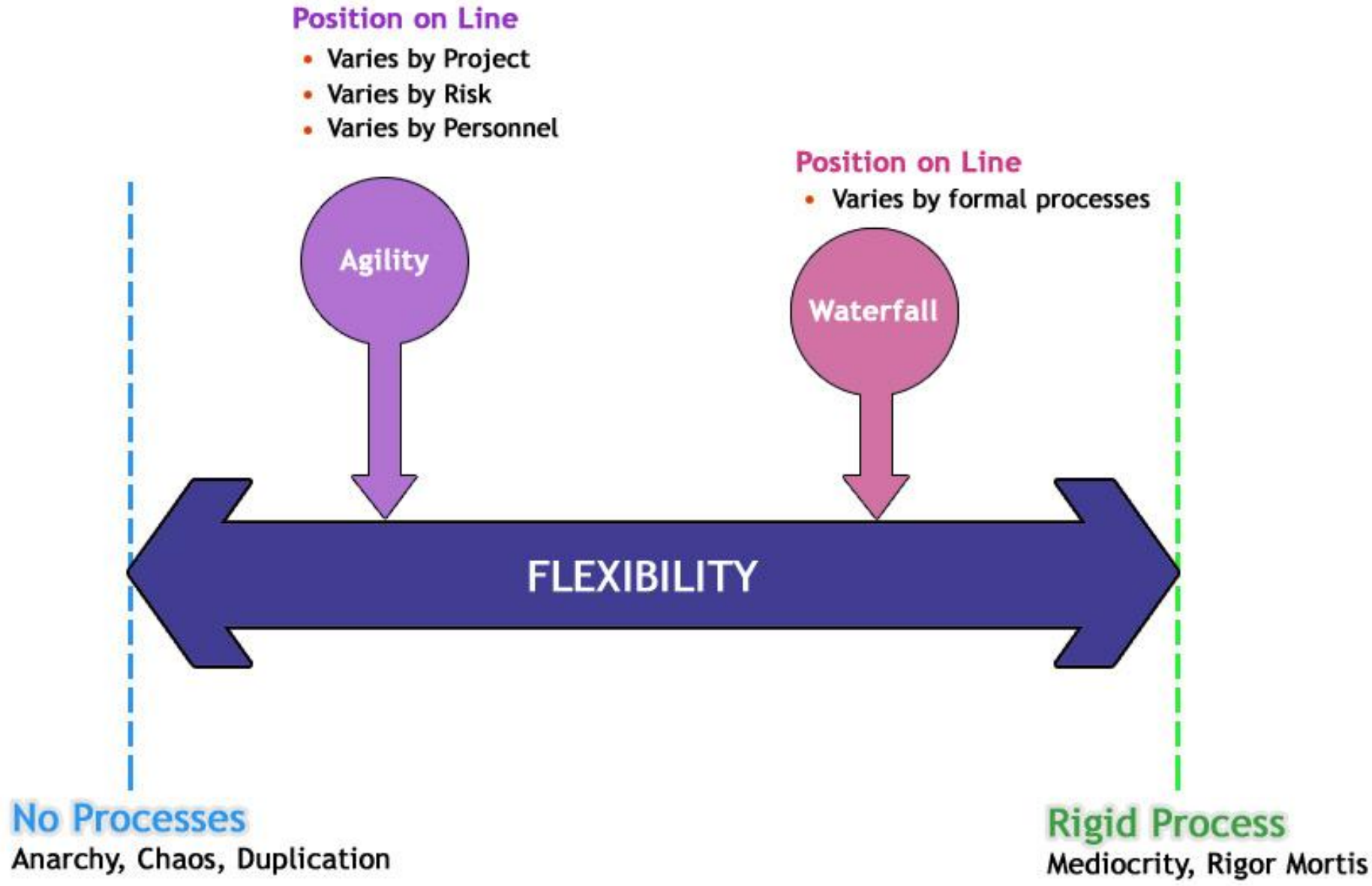




Agile vs Waterfall

	Agile Methods	Waterfall Methods
Approach	Adaptive	Predictive
Success Measurement	Business Value	Conformation to plan
Management Style	Decentralized	Autocratic
Perspective to Change	Change Adaptability	Change Sustainability
Culture	Leadership-Collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Upfront Planning	Minimal	Comprehensive
ROI	Early in Project	End of Project
Team Size	Small/Creative	Large

Adaptability Scale



I want to run agile project

- http://www.youtube.com/watch?v=4u5N00ApR_k

Leadership Resistance: A Major Hurdle in Agile Adoption

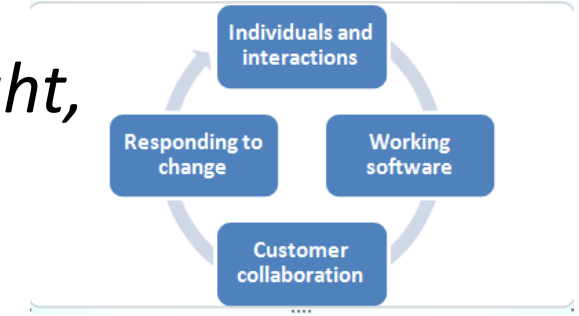
- Traditional Mindset: Leaders accustomed to plan-driven approaches may struggle with Agile's iterative nature.
- Fear of Loss of Control: Agile's emphasis on team autonomy can be perceived as a threat to managerial authority.
- Lack of Understanding: Misconceptions about Agile principles can lead to skepticism and resistance.
- Cultural Misalignment: Existing organizational cultures may conflict with Agile values, hindering adoption
- https://www.youtube.com/shorts/c_YpIwdEEco

Introduction to Agile Manifesto – Values

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

"While there is value in the items on the right,

We value the items on the left more."



Agile Principles

1. The highest priority is to satisfy the costumer through early and continuous delivery of valuable software
2. Welcome changing requirements, even late in development. Agile process harness change for the customer´s competitive advantage
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
4. Business people and developers must work together daily throughout the project
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

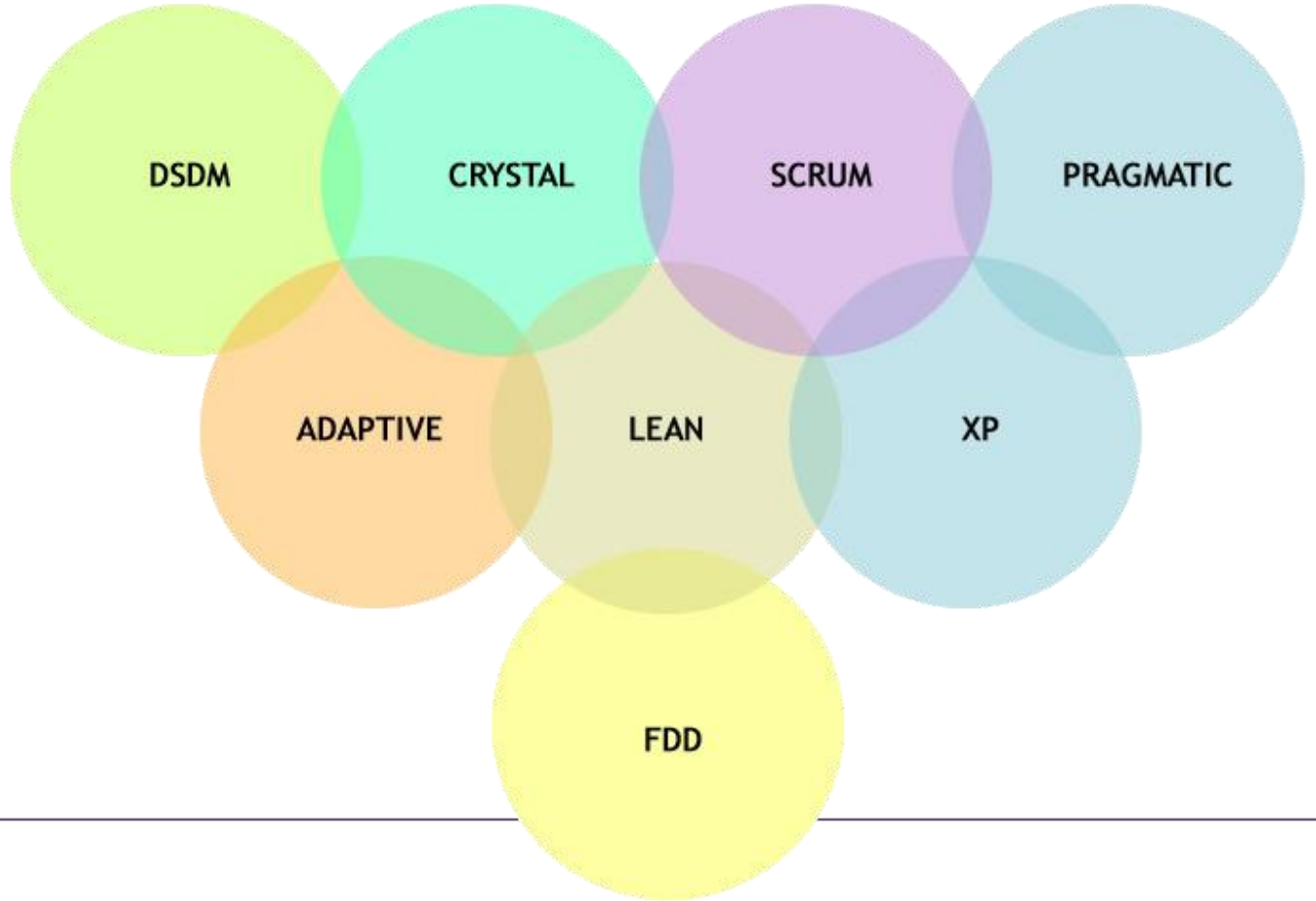
Agile Principles …

7. Working software is the primary measure of progress
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
9. Continuous attention to technical excellence and good design enhances agility
10. Simplicity – the art of maximizing the amount of work not done – is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly


What is Agile?

Agile Methodologies

- There is a significant number of agile methodologies
- Each has roots in a specific focus with the overall theme of agility



I want to run Agile Project – Part 2

- http://youtu.be/lAf3q13uUpE

The most widely-used Agile methodologies

A. Scrum
B. Kanban
C. Extreme Programming (XP)
D. Lean Software Development
E. Crystal
F. Feature-Driven Development (FDD)
G. Dynamic Systems Development Method (DSDM)
H. Adaptive Software Development (FDD)

SCRUM

Scrum: Most Popular Agile Methodology

- Popularity/Market share 60+% Compared to other methodologies
- **lightweight**, simple to understand, but difficult to master.
- formalized by **Ken Schwaber and Jeff Sutherland** in the early 1990s.
- particularly suitable for **complex, adaptive problems**, delivering high-value products through collaboration.
- Scrum is **not a process or technique**, but a framework within which various processes and techniques can be employed.

Scrum Metaphor – Rugby

- the team **huddles together, pushes forward as a unit**, and works collaboratively to gain control
- **high-performing product development teams (like rugby teams)**
  - **cross-functional, self-organizing, and working together toward a common goal** of the ball

Scrum Core Values & Philosophy

- **Commitment** – Team members personally commit to achieving the team goals.
- **Courage** – Team members have the courage to do the right thing and work through tough problems.
- **Focus** – Everyone focuses on the work of the Sprint and the goals of the Scrum Team.
- **Openness** – The team and stakeholders agree to be open about all the work and the challenges.
- **Respect** – Scrum Team members respect each other to be capable, independent people.

Key Roles & Responsibilities

Scrum Master

- Ensures Scrum is understood and enacted.
- Facilitates Scrum events.
- Removes impediments, coaches the team.
- *Not a manager; not responsible for delivery*

Product Owner

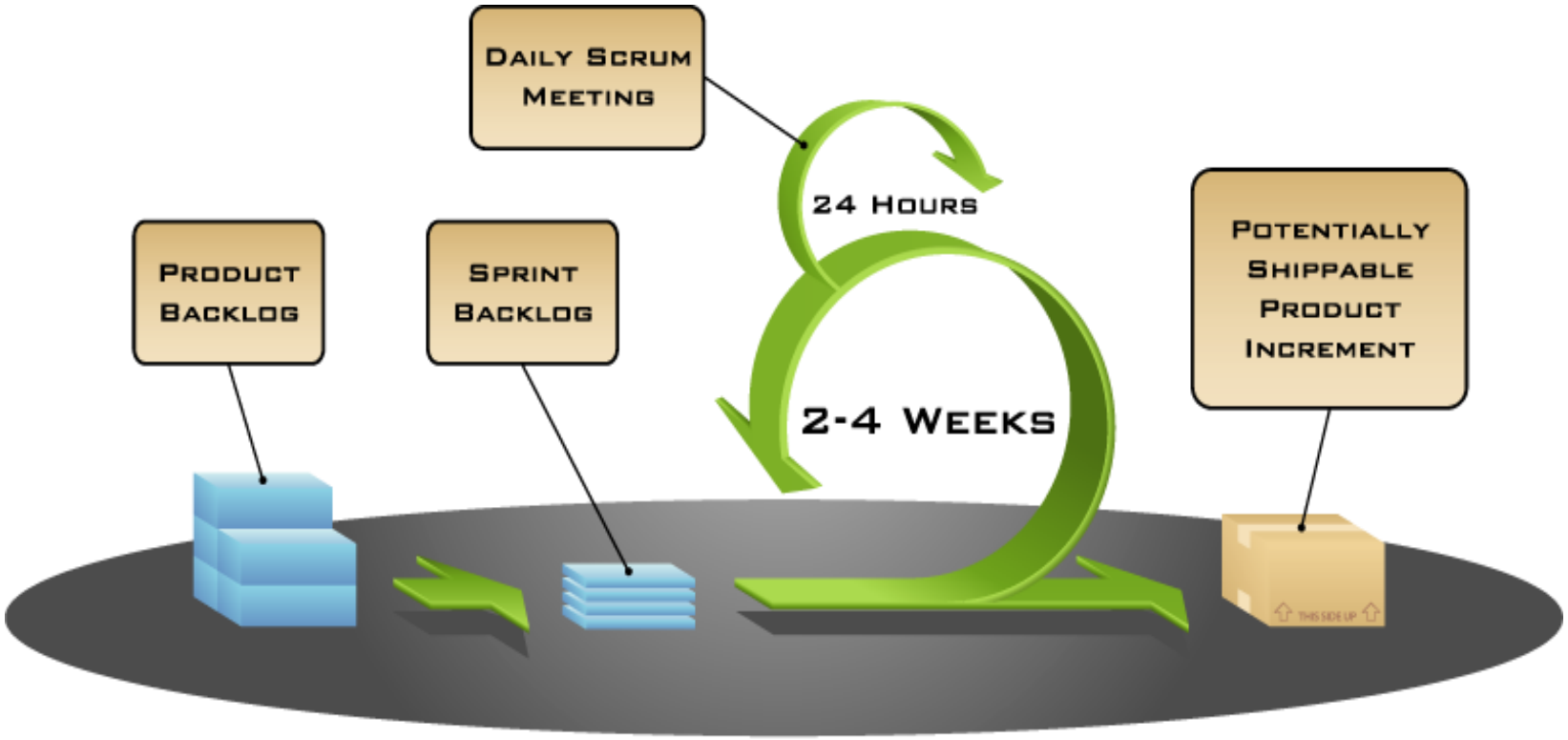
- Manages the **Product Backlog**.
- Prioritizes work based on business value.
- Represents stakeholders and customers.
- Accountable for maximizing product value.

Developers (Scrum Team)

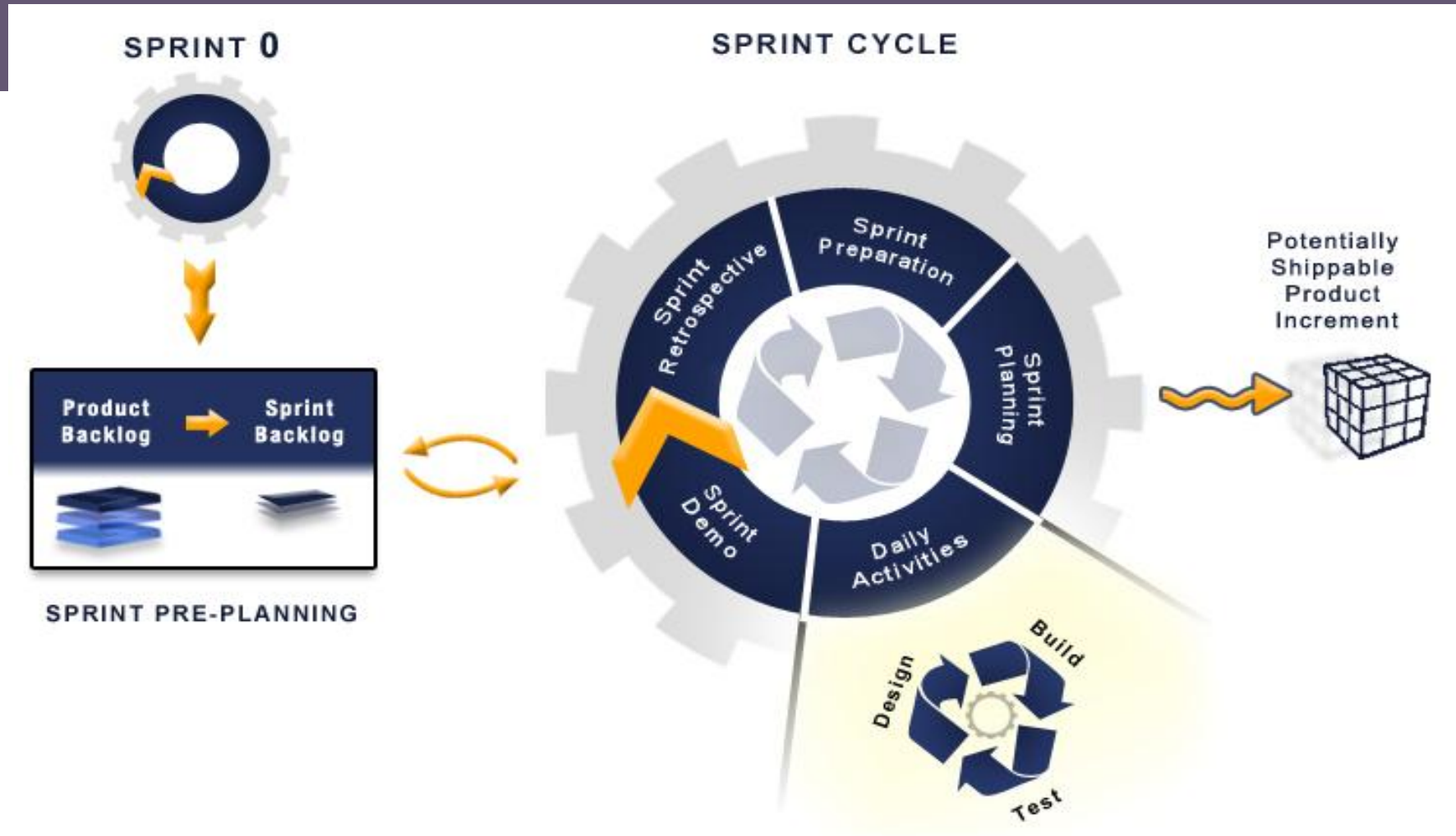
- Cross-functional and self-managing.
- Responsible for delivering a **potentially shippable increment** every Sprint.
- Estimate, design, code, test, and deliver features.

Scrum – Structure

Scrum is organized into **Sprints**, each typically lasting 2–4 weeks. The workflow includes the following events:



Scrum – Process



Sprint 0 – could be envisioned as Pre-game; which includes all the initial activities of req. gathering, planning, etc

Scrum : Process and Workflow

Scrum Events (Ceremonies)

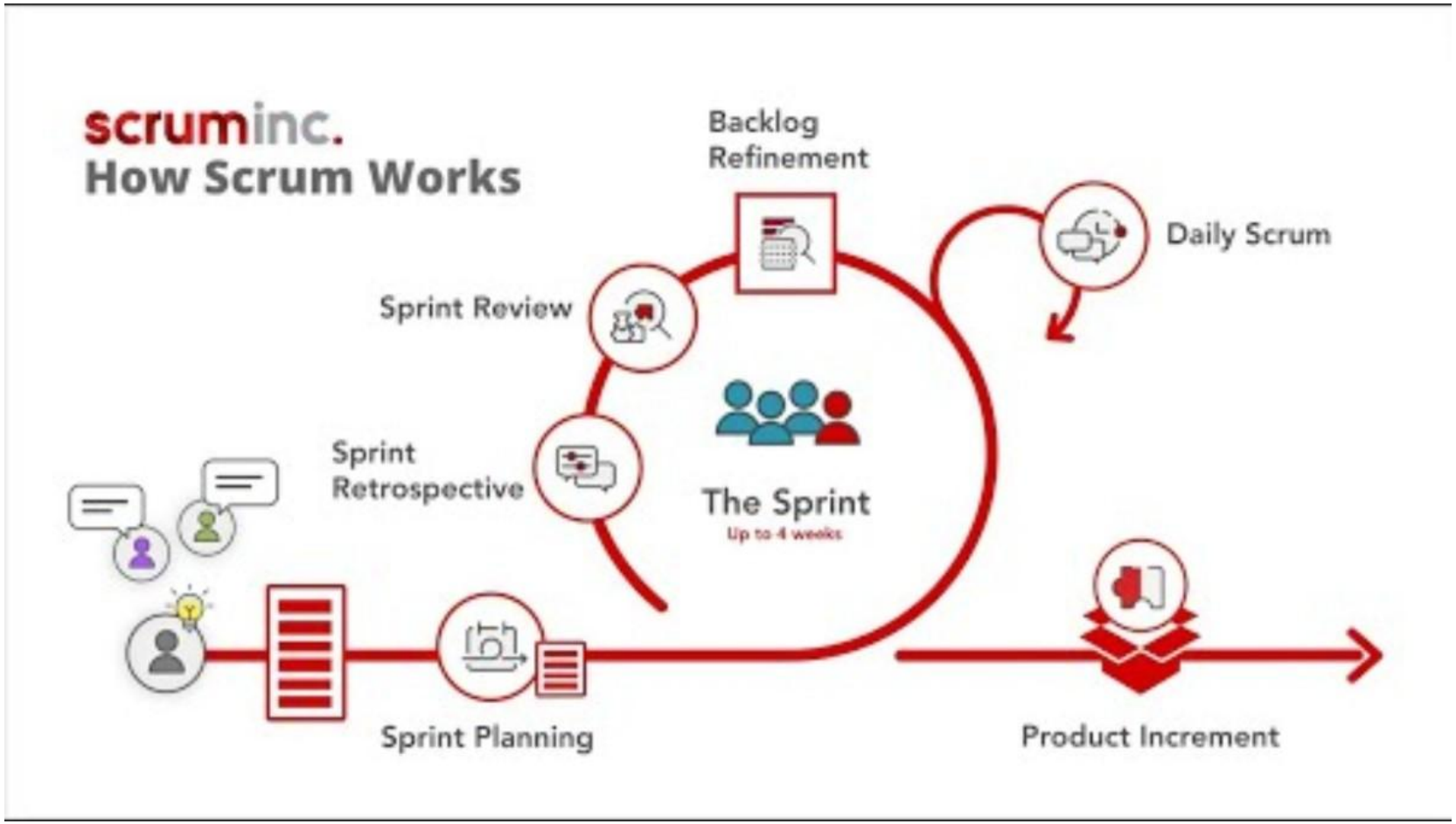
1. **Sprint Planning** – Define Sprint Goal and select backlog items.
2. **Daily Scrum (Stand-up)** – 15-minute meeting to synchronize.
3. **Sprint Execution** – Development of product increment.
4. **Sprint Review** – Demonstration and feedback session.
5. **Sprint Retrospective** – Reflect and improve process.

🌀 Sprint Flow

- Product Backlog → Sprint Planning → Sprint Backlog → Sprint →
- Daily Scrums → Increment → Sprint Review → Retrospective

- Scrum is **cyclical** and emphasizes **continuous improvement**.

How Scrum works



Scrum Practices and Techniques

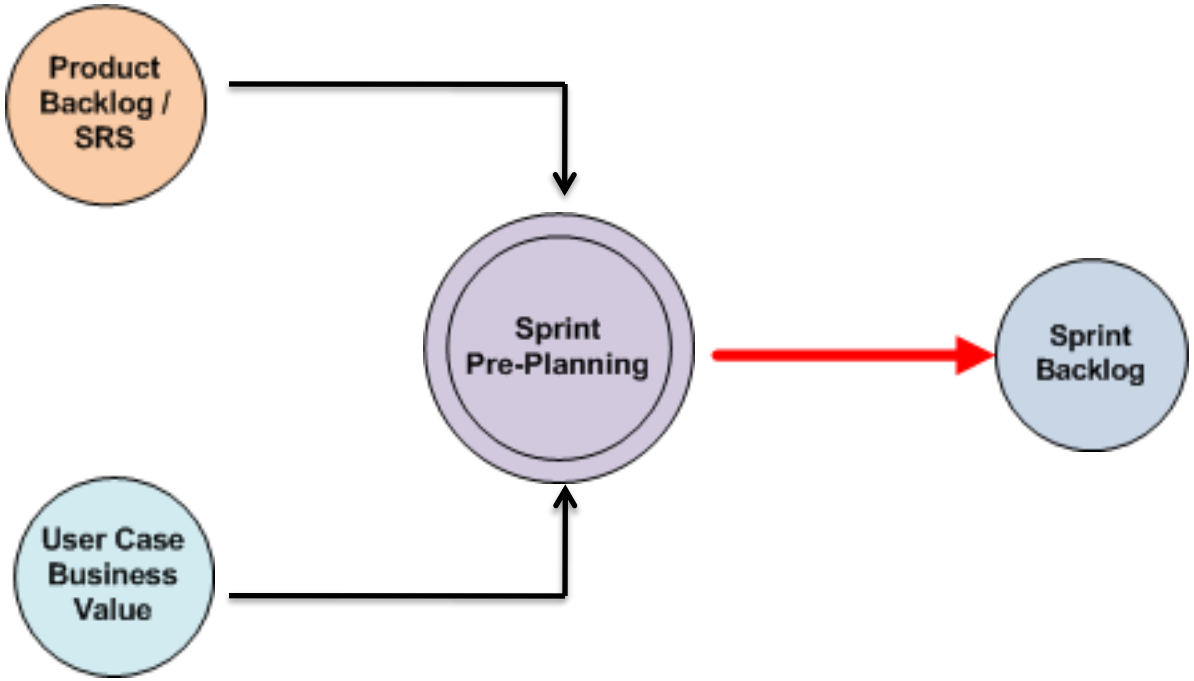
- **User Stories** to define work.
- **Estimation with Story Points or Ideal Hours**.
- **Velocity Tracking** to forecast future capacity.
- **Definition of Done (DoD)** to ensure quality.
- **Task Boards** (digital or physical) to visualize work.

The Daily Scrum Meeting

- Goal
  - Enable the Team to share progress with each other
  - Identify what's blocking or slowing down the Team (not problem solving)
  - Enable the Team to inspect and adapt daily
- Team stands in a circle and each member reports only 3 things
- Time-boxed to 15 mins

Sprint Pre-Planning

- Sprint Pre-Planning involves prioritizing requirements so that the most important Business Value is delivered early in the form of a working software



Tools and Artifacts

- Artifacts
  - Product Backlog, Sprint Backlog, Increment

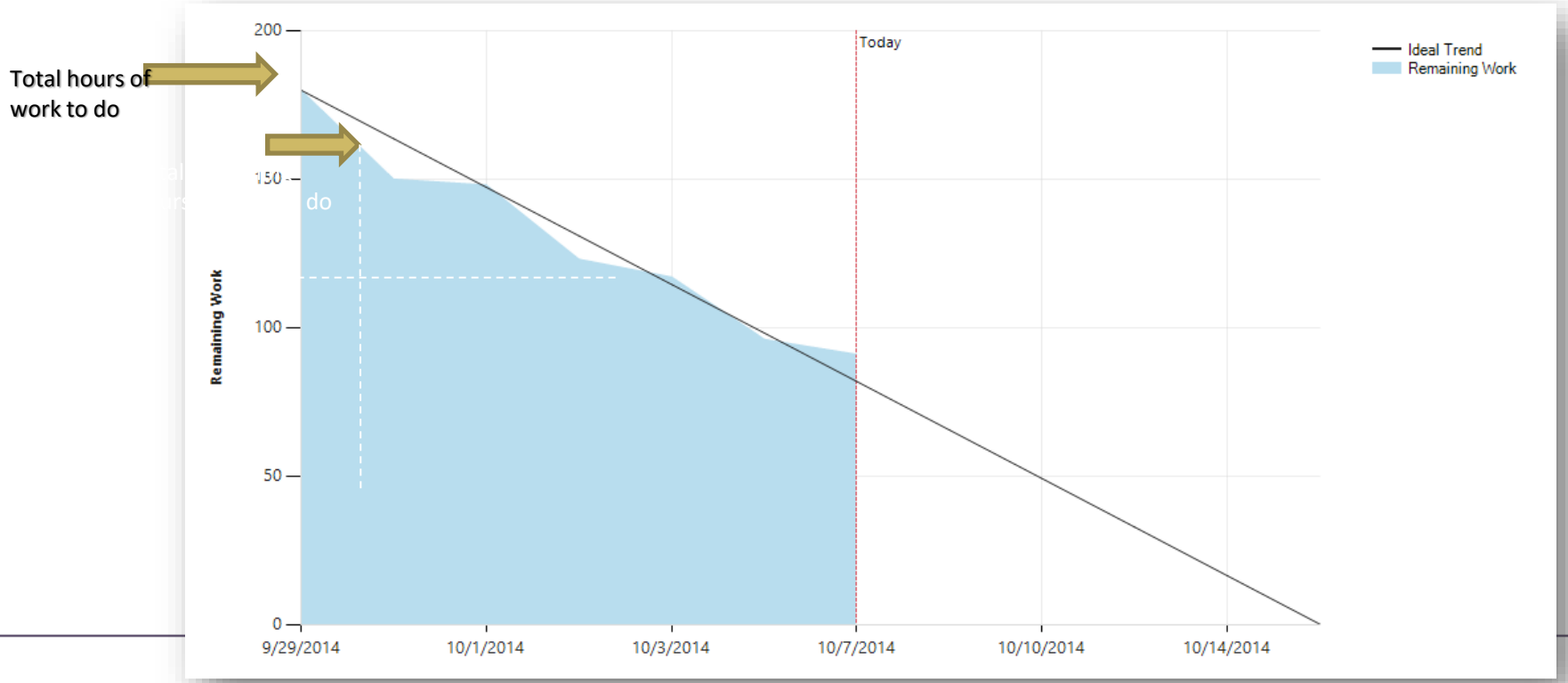
Visual Tools

- **Burndown Charts** (work left vs time)
- **Task Boards** (e.g., Trello, Jira, Azure Boards)
- **Velocity Charts**

Tools: **Jira, Azure DevOps, ClickUp, Monday.com, Miro, Trello, ……**

Burndown chart

- The Team updates and reviews Burndown chart daily
- Shows teams progress through the Sprint



Scrum Challenges

Common Pitfalls:

- Treating Scrum as a rigid process instead of a flexible framework.
- Product Owners with unclear authority or unavailability.
- Scrum Masters acting like project managers.
- Teams not truly cross-functional or self-managing.
- Lack of understanding of Agile values (mechanical Scrum).

Scrum requires cultural change, not just role re-naming and meeting scheduling.

Observations - Scrum Pitfalls

Many problems of implementation

- The wrong product owner
- Incorrectly constructed teams
- Scrum perceived as a “silver bullet”
- Scrum without understanding
- Insufficient cooperation
- No continuous improvement
- Someone else decides
- Not full-time employees
- Too little documentation

Scrum - Managing the Change

During a Sprint:

- Once team has committed, no changes to Sprint scope
- No Changes to Deliverable
- Details will emerge during Sprint, but no new work or substantially changed work
- Customer can terminate the Sprint if necessary
- No Changes to Sprint Duration
- Sprint ends on planned date whether team has completed its commitment or not

Customer can make any changes to the remaining Product Backlog before the start of the next Sprint

How AI Tools Can Support Scrum Practices

- Share your finds in Discussion Forums

XP Summary

Aspect	Scrum Highlights
Origin	Ken Schwaber & Jeff Sutherland, 1990s
Values	Commitment, Focus, Openness, Respect, Courage
Roles	Scrum Master, Product Owner, Developers
Process	Sprint Planning → Daily Scrum → Review → Retrospective
Practices	Velocity, Backlog Refinement, DoD
Tools	Jira, Trello, Burndown Chart, Task Board
Pitfalls	Role confusion, rigid execution, team immaturity
AI Help	Estimation, NLP summaries, risk detection, retros analysis

EXTREME PROGRAMMING (XP)

Extreme Programming (XP)

- **methodology** focused on **engineering excellence** and **rapid, frequent delivery** of high-quality software.
- XP pushes "best practices" to the extreme — for example, testing is done first, not after coding.
- Emphasizes customer collaboration, short release cycles, and adaptive planning.

"a deliberate and disciplined approach to software development."

A Lightweight Discipline of Software Development

Humanistic (emphasizes team work)

History

A result of project carried out (to develop better software) in 1990

Kent Beck and Ward Cunningham

"Extreme Programming Explained: …" 1999

First experimented in 1996 at DaimlerChrysler

Started as an experiment, later became one lightweight software development methodology

lightweight: few rules and practices

heavyweight: many rules, practices and documents

XP: Core Values & Philosophy

- **Communication** – Constant face-to-face interaction
- **Simplicity** – Always do the simplest thing that works
- **Feedback** – Fast feedback from customers, tests, and team
- **Courage** – Accept and adapt to change, even late
- **Respect** – Every team member's contribution is valued

XP believes **good technical practices** lead to faster feedback and better adaptability.

XP Key Roles & Responsibilities

Programmer

- Writes production code and tests
- Refactors code continuously

Customer (On-site)

- Writes **user stories** and **acceptance tests**
- Prioritizes work
- Always available for clarification

Coach

- Guides the team in XP practices

XP Key Roles & Responsibilities …

Tracker

- Monitors progress and velocity

Tester

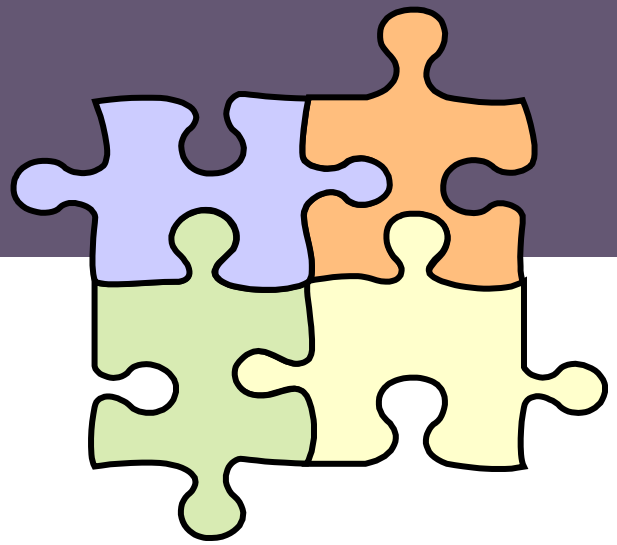
- Supports with test automation and feedback loops

Manager

- Handles coordination, scheduling, and external issues

Everyone works as a unit with shared responsibility.

XP Process and Workflow



XP operates in **short iterations** (1–3 weeks) with continuous delivery.

XP Lifecycle Phases:

1. **Exploration** – Customer writes stories, team explores tech
2. **Planning** – Estimation and release planning
3. **Iterations to Release** – Develop, integrate, test
4. **Productionizing** – Final testing and stabilization
5. **Maintenance** – Ongoing development
6. **Death** – Project ends; no further user stories

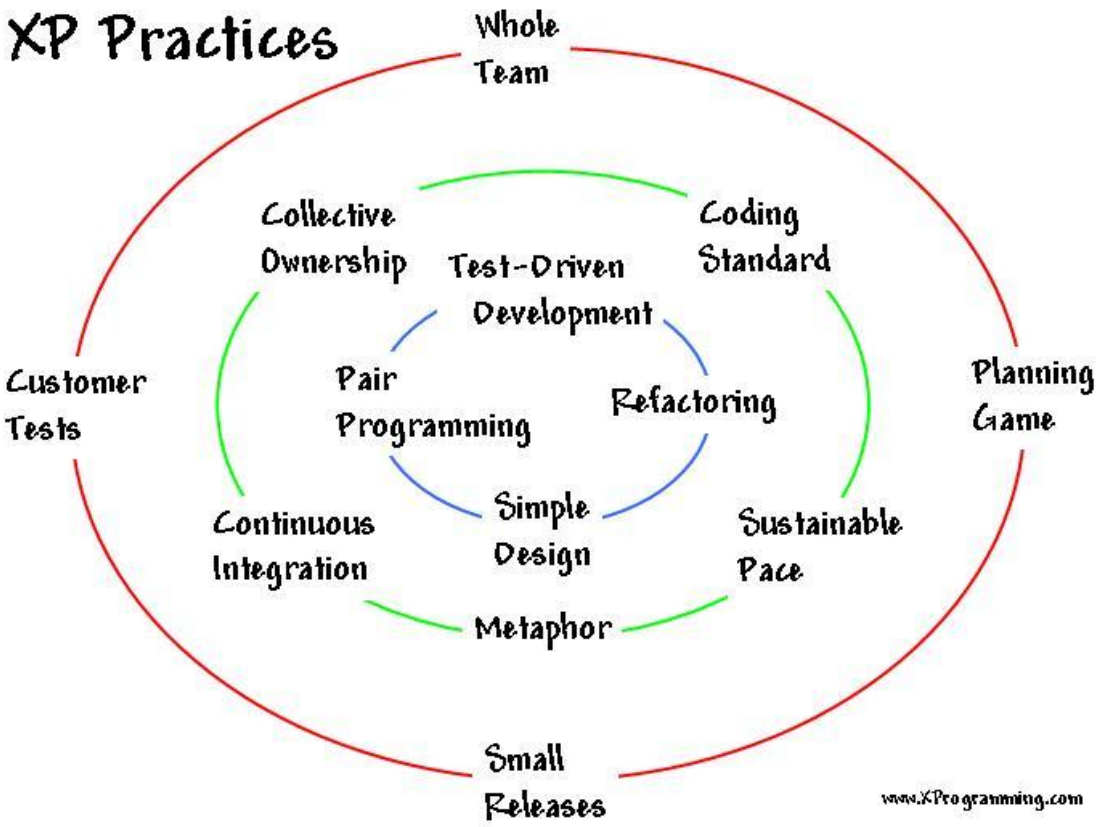
Pillars of XP flow.

- **Test-First Development**,
- **Continuous feedback**,
- **Small Releases**

XP Practices and Techniques

XP is most known for its **12 core practices**, grouped as

1. Programming Practices
2. Team Practices
3. Planning & Feedback Practices



XP Programming Practices (1)

- **Test-Driven Development (TDD)** – Write tests *before* code
- **Pair Programming** – Two developers share one machine
- **Refactoring** – Continuously improve code design
- **Simple Design** – Do only what is needed for now
- **Continuous Integration** – Frequent builds (multiple times/day)
- **Coding Standards** – Everyone follows the same style

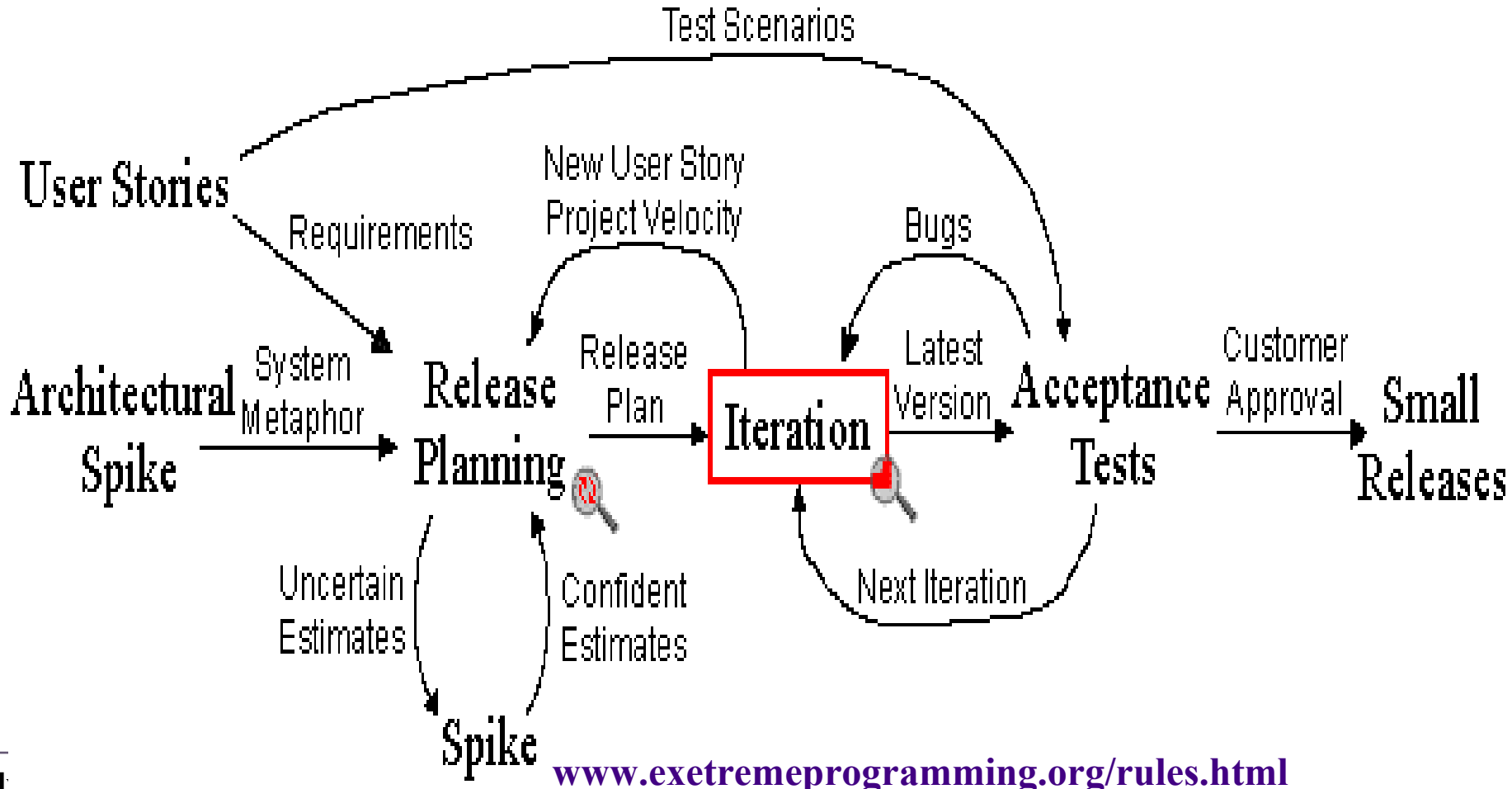
XP Team Practices (2)

- **Collective Code Ownership** – Anyone can improve any code
- **40-hour Work Week** – Sustainable pace, no burnout
- **Open Workspace** – Close collaboration, fast feedback

XP Planning & Feedback Practices (3)

- **Planning Game** – Customer defines scope; developers estimate effort
- **Small Releases** – Deliver working software frequently
- **On-site Customer** – Always available for feedback and decisions

Extreme Programming Project



XP Tools and Artifacts

XP promotes **light documentation** and **working software** over traditional artifacts, but still uses some tools

Key Artifacts

- **User Stories** – Simple cards describing customer needs
- **Acceptance Tests** – Validate user stories
- **Unit Tests** – Written and automated by developers
- **Velocity Charts** – Measure team progress per iteration

Writing a User Story in XP

- What is a User Story?
  - A user story is a simple, user-oriented description of a software feature.
  - Captures the who, what, and why of a feature.
- **Structure of a User Story:**

1. **Title:**
   - A brief, descriptive title for the story.
2. **User Role:**
   - "As a [type of user],"
3. **Goal:**
   - "I want to [perform some action],"
4. **Reason:**
   - "So that [I can achieve some outcome]."

Example – User Story

- Title: User Login
- Story:

  "As a registered user,

  I want to log in to the website,

  So that I can access my personalized dashboard.“
- Acceptance Criteria:
  - Define specific conditions that must be met for the story to be considered complete.

"*Given a registered user,*

When they enter their valid username and password,

Then they should be logged in and see their dashboard."

Characteristics of a Good User Story

- Independent:
  - Stories should be self-contained, not dependent on other stories.
- Negotiable:
  - Stories are not contracts; they are reminders for conversations.
- Valuable:
  - Stories should deliver value to the end user.
- Estimable:
  - Stories should be small enough to estimate and complete within an iteration.
- Small:
  - Stories should be concise, typically one to three sentences, and completable within a few days.
- Testable:
  - Stories should have clear criteria for acceptance tests.

Assigning Values to User Stories in XP

- Customer Assigns a Value:
  - Purpose: To identify the priority of each story relative to others.
- XP Team Estimates Cost:
  - Estimation: The team estimates the time required to implement each story in weeks.
- Handling Long Stories:
  - Threshold: If the estimated time exceeds 3 weeks:
    - Request to Split: The story is split into smaller, manageable parts.
    - Assign New Values: New values are assigned to the split stories.
- Flexibility:
  - New Stories: It is possible to create and prioritize new stories at any time.

Grouping Stories in XP

- Collaboration:
  - Client and XP Team: Work together to group stories for the next release.
- Basic Commitment:
  - Agreement: Defines delivery date, project matters, etc.
  - Development Order: Decided when the development order is set.
- Development Order Prioritization:
  1. All Implemented Immediately:
     - If feasible, implement all stories immediately.
  2. Highest Value First:
     - Prioritize and schedule stories that deliver the highest value first.
  3. Priority for Riskiest Stories:
     - Address the riskiest stories early to mitigate potential issues.

XP Project Velocity

- Definition:
  - Number of stories implemented in the first release.
- Estimation Benefits:
  - Delivery Date: Helps estimate delivery date and schedule for subsequent releases.
  - Overcommitment: Investigate and adjust over commitments.
- Managing Velocity Changes:
  - Adjustments: Add, remove, modify, or split stories.
  - Plan Variations: Plans may vary based on velocity changes.

XP Tools

- Test frameworks: **JUnit, NUnit, PyTest**
- CI tools: **Jenkins, GitHub Actions, TravisCI**
- Pair programming support: **Visual Studio Live Share**
- Planning: **Trello, Miro, Planning Poker apps**

XP Challenges

- Pair programming may feel inefficient or uncomfortable for some
- Requires **high discipline** in writing and maintaining tests
- Continuous integration needs strong automation setup
- May not fit **non-technical stakeholders** who expect documentation
- Teams may resist practices like **collective code ownership**

XP works best in **technically strong**, **collaborative teams** and is harder in siloed environments.

Practice of Small Releases in XP

Delivering a simple, functioning system into production early and updating it frequently in very short cycles (few weeks or even days).

Benefits:

1. Early Feedback:
   - Allows customers to use the system and provide feedback early in the process.
   - Helps the team to adapt and make changes quickly based on user feedback.
2. Frequent Updates:
   - Regularly updated with fully tested features.
   - Reduces risk by catching and fixing bugs early.
3. Customer Involvement:
   - Customers can use the product early and continuously see progress.
   - Ensures the product meets customer needs and expectations.

Practice of Small Releases in XP – **How does it work?**

1. Define Requirements:
   - Collaborate with the customer to define and prioritize user stories.
2. Break Down Work:
   - Split requirements into small, manageable chunks.
3. Frequent Delivery:
   - Deliver each small release to the customer.
   - Each release is fully tested before delivery.
4. Iterate and Improve:
   - Gather feedback from customers after each release.
   - Use feedback to refine and improve subsequent releases.

Practice of Metaphor in XP

- The metaphor practice in XP helps simplify complex systems, ensuring everyone on the team shares a common understanding and vision, which enhances collaboration and productivity.
  - A **common description** that guides development and communication.
- Purpose:
  - System Architecture Representation:
    - Helps represent the system architecture in an easily understandable way.
    - Describes How Programs Work: Provides a simple analogy or metaphor to describe complex system behaviors and interactions.
- Benefits:
  - Common Vision:
    - Establishes a common vision among the team members.
  - Shared Vocabulary:
    - Facilitates a shared vocabulary that everyone understands, enhancing communication and collaboration.

How Metaphors Work in XP:

- Guiding Development:
  - Metaphors serve as a mental model for developers, guiding the design and implementation process.
- Enhancing Communication:
  - By using familiar terms and concepts, metaphors make it easier for team members to discuss and understand the system.
- Examples:
  - Library System Metaphor: Comparing a system's components to parts of a library (books, shelves, librarians) to explain their roles and interactions.
  - Warehouse Metaphor: Describing data storage and retrieval systems using the analogy of a warehouse with shelves and boxes.

Planning Game in XP

- The planning game in XP is essential for setting clear, achievable goals and ensuring continuous alignment between customer expectations and development efforts.

Two Phases:

- Release Planning:
  - Goal: Define the set of features required for the next release.
- Iteration Planning:
  - Customer Role: Chooses the stories for the iteration.
  - Programmers Role: Estimate the effort required for each story.

40-Hour Week in XP

- The 40-hour week practice in XP is essential for maintaining a healthy, productive, and high-quality development process, ensuring programmers are always at their best.
- Standard Work Week:
  - 40 hours per week: 8 hours per day, 5 days a week.
  - No Overtime: Strictly no overtime to maintain productivity and quality.
- Rationale:
- Tired Programmers Make More Mistakes:
  - Ensures programmers are well-rested and maintain high performance.
  - Reduces the risk of errors and improves overall code quality.

Benefits: 40 Hrs Work

1. **Sustainable Pace:**
    1. Maintains a steady and manageable work pace.
    2. Prevents burnout and keeps the team motivated.
2. **Higher Quality Work:**
    1. Well-rested programmers produce higher quality code.
    2. Reduces the number of bugs and rework.
3. **Better Work-Life Balance:**
    1. Promotes a healthier balance between work and personal life.
    2. Increases job satisfaction and employee retention.

On-Site Customer Practice in XP

- The on-site customer practice in XP is crucial for maintaining effective communication, quick decision-making, and ensuring the delivery of a high-quality product that meets customer needs.
  - **Improve Communication:** Enhances direct communication between the customer and the XP team.

1. **Customer Integration:**
   1. **Team Member:** The customer stays with the XP team and is regarded as a team member.
   2. **Benefits:** Facilitates real-time feedback and decision-making.
2. **Responsibilities:**
   1. **Write and Run Acceptance Tests:** Ensures the developed features meet the required standards.
   2. **Answer Questions:** Provides immediate answers to any queries from the programmers.
   3. **Make Decisions:** Decides on priorities and clarifies requirements for the team.
   4. **Accept Releases:** Reviews and accepts the developed features for release.

Keep It Simple (KIS) in XP

- The Keep It Simple practice in XP ensures that designs remain straightforward, avoiding unnecessary complexity and focusing on immediate, practical implementation.
- **Definition:** Keep It Simple (KIS)Design should provide clear implementation guidelines when stories are written.
  - **Key Idea:** Nothing less, nothing more.

Guidelines for KIS in XP

1. **Avoid Extra Functionality:**
    1. Discourage adding extra features for future use.
2. **CRC Cards:**
    1. **Class-Responsibility-Collaborator (CRC) Cards:** Used to identify and organize object-oriented classes.
    2. **Purpose:** Simplify design and ensure clear responsibilities.
3. **Immediate Prototyping:**
    1. **Spike Solutions:** Create operational prototypes to validate requirements and original estimates.
    2. **Benefit:** Lower risk and ensure accurate implementation.

Benefits – KIS in XP

- Simplicity:
  - Principle: KISS (Keep It Simple, Stupid)
  - Goal: Most systems work best when kept simple, avoiding unnecessary complexity.
  - Effect: Prevents creeping featurism and reduces system failures.
- Intelligent Design:
  - Misconception: Simplicity is often mistaken for lack of sophistication.
  - Reality: Simple designs are often the most intelligent and effective.
- Risk Reduction:
  - Spike Solutions: Implemented and evaluated early to lower risk and validate estimates.

Refactoring in XP

- Refactoring is essential in XP to maintain a clean, efficient, and well-designed codebase, ensuring the system remains adaptable and easy to enhance.
  - **Definition:** A construction and design technique used in Agile to improve code quality without changing its external behavior.

Purpose of Refactoring:

1. **Clean Code:**
   1. Improve the readability and maintainability of the code.
2. **Remove Bugs:**
   1. Identify and fix hidden issues in the code.
3. **Improve Design:**
   1. Enhance the overall design and structure of the application.

When to Refactor

1. Before Adding Features:
   - Ask: "Is there a way to change the program to make the addition simple?“
2. After Adding Features:
   - Ask: "Is there a way to make the program simpler while still running all the tests?“
3. Refactor on Demand:
   - Refactor when the system and process require it, not based on specifications.

Effort for Refactoring as Application Grows:

- **Considerations:**
  - As the application size increases, refactoring efforts may also increase.
  - Regular refactoring prevents the accumulation of technical debt.

Design and Coding in XP

- **Integrated Approach:** Design and coding go together.
- **Preliminary Design:** Perform a preliminary design.
- **Unit Tests:** Develop unit tests for each story to be included in the release before starting coding.

Coding Practices:

1. **Pair Programming:**
    1. Two developers work together at one workstation.
    2. Improves code quality and knowledge sharing.
2. **Coding Standards:**
    1. Follow consistent coding standards.
    2. Ensures code is easy to read, maintain, and integrate.

Integration

- Daily Integration:
  - Part of the construction phase.I
  - ntegrate code on a daily basis.
- Developer Responsibility:
  - Developers are responsible for integrating their code.
  - Avoids critical compatibility and interfacing problems.
- Smoke Testing Environment:
  - Use smoke testing to ensure basic functionality works after each integration.

Pair Programming

- Pair programming is a core practice in XP that enhances code quality, team collaboration, and overall productivity, making it a valuable approach in Agile software development.
- **Overview of Pair Programming:**
- **Coding in Pairs:** Two programmers work together at one machine.
  - **Role of Driver:** Writes the code.
  - **Role of Observer:** Reviews the code and prepares test cases.
  - **Role Switching:** Frequently switch roles to maximize collaboration.

Benefits of Pair Programming

1. **Speed and Continuity:**
    1. **Benefit:** Enhances speed and ensures continuous progress.
2. **Quality and Design:**
    1. **Benefit:** Continuous code review leads to higher quality and better design.
    2. **Cost Efficiency:** Improved quality offsets the increased development cost.
3. **Enjoyable Experience:**
    1. **Benefit:** Increases job satisfaction and employee morale.
    2. **Team Spirit:** Promotes collaboration and communication within the team.
4. **Learning and Skill Development:**
    1. **Benefit:** Programmers learn from each other, improving both technical and soft skills.
    2. **Knowledge Sharing:** Reduces the risk of losing critical knowledge if a team member leaves.
5. **Efficient Defect Removal:**
    1. **Benefit:** Effective defect removal through continuous review.
    2. **Productivity:** Long-term improvement in productivity.

Pair Programming

- **Suitability:**
  - **Ordinary Programmers:** No need for highly specialized degrees.
  - **Team Size:** Effective for small teams (2 to 10 members).
- **Tools and Best Practices:**
  - **Collaboration Tools:** Use tools like Visual Studio Live Share, CodeTogether, or IntelliJ IDEA's Code With Me for remote pair programming.
  - **Regular Role Switching:** Ensure regular role switching to keep both programmers engaged and effective.
  - **Communication:** Maintain clear and open communication for better coordination and problem-solving.

Collective Code Ownership in XP

- **Whole Team Ownership:**The entire team owns the code.
  - Any team member can change any part of the code they need to
- Collective code ownership in XP fosters a collaborative and flexible development environment, leading to faster and higher-quality software delivery.

Key Principles:

1. Opportunistic Value Addition:

1. **Responsibility:** Anyone who sees an opportunity to add value to any portion of the code must do so.
2. **Guidelines:** Changes must align with current requirements and adhere to simple design principles.

2. Shared Responsibility:

1. **System-Wide Responsibility:** Everyone is responsible for the entire system.
2. **Benefit:** Increases development speed and agility.

Benefits of Collective Code Ownership:

- **Enhanced Collaboration:**
  - Encourages team members to collaborate and share knowledge.
- **Improved Code Quality:**
  - Continuous improvements by different team members enhance code quality.
- **Increased Flexibility:**
  - Any team member can fix issues or add features, reducing bottlenecks.
- **Faster Development:**
  - Speed increases as changes can be made by anyone at any time.

Best Practices:

- **Regular Code Reviews:**
  - Ensure code changes are reviewed regularly to maintain quality.
- **Consistent Coding Standards:**
  - Follow coding standards to make the codebase easy to understand and modify by anyone.
- **Clear Communication:**
  - Maintain clear communication within the team to coordinate changes effectively.

Continuous Integration in XP

- **Frequent Integration:**
  - Developers integrate their work multiple times a day.
  - Ensures that code changes are regularly incorporated into the main codebase.

Key Practices:

1. **Regular Code Reviews:**
   1. Ensure code changes are reviewed regularly to maintain quality.
2. **Consistent Coding Standards:**
   1. Follow coding standards to make the codebase easy to understand and modify by anyone.
3. **Clear Communication:**
   1. Maintain clear communication within the team to coordinate changes effectively.
4. **Integration Frequency:**
   1. Integrate code at the end of each work session.
   2. Typically, code is integrated multiple times a day.

9. Coding Standard

All team members follow a common standard

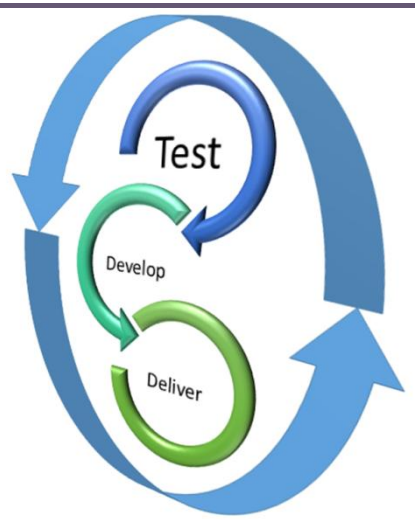
Easy to understand/readable for all members

will facilitate refactoring

Coding Standards in XP

- Adhering to coding standards in XP ensures a consistent, high-quality codebase that is easy to maintain and refactor, fostering better collaboration and efficiency among team members.
- Common Standard:
  - All team members follow a common coding standard.
  - Ensures consistency across the codebase.
- Readability:
  - Code is easy to understand and readable for all team members.
  - Facilitates collaboration and reduces misunderstandings.
- Facilitates Refactoring:
  - Standardized code makes refactoring easier and more efficient.
  - Enhances maintainability and scalability of the code.

Testing in XP



Unit Test before developing code

Use a framework for automated testing

Daily Testing

Regression Testing Strategy whenever the code is modified

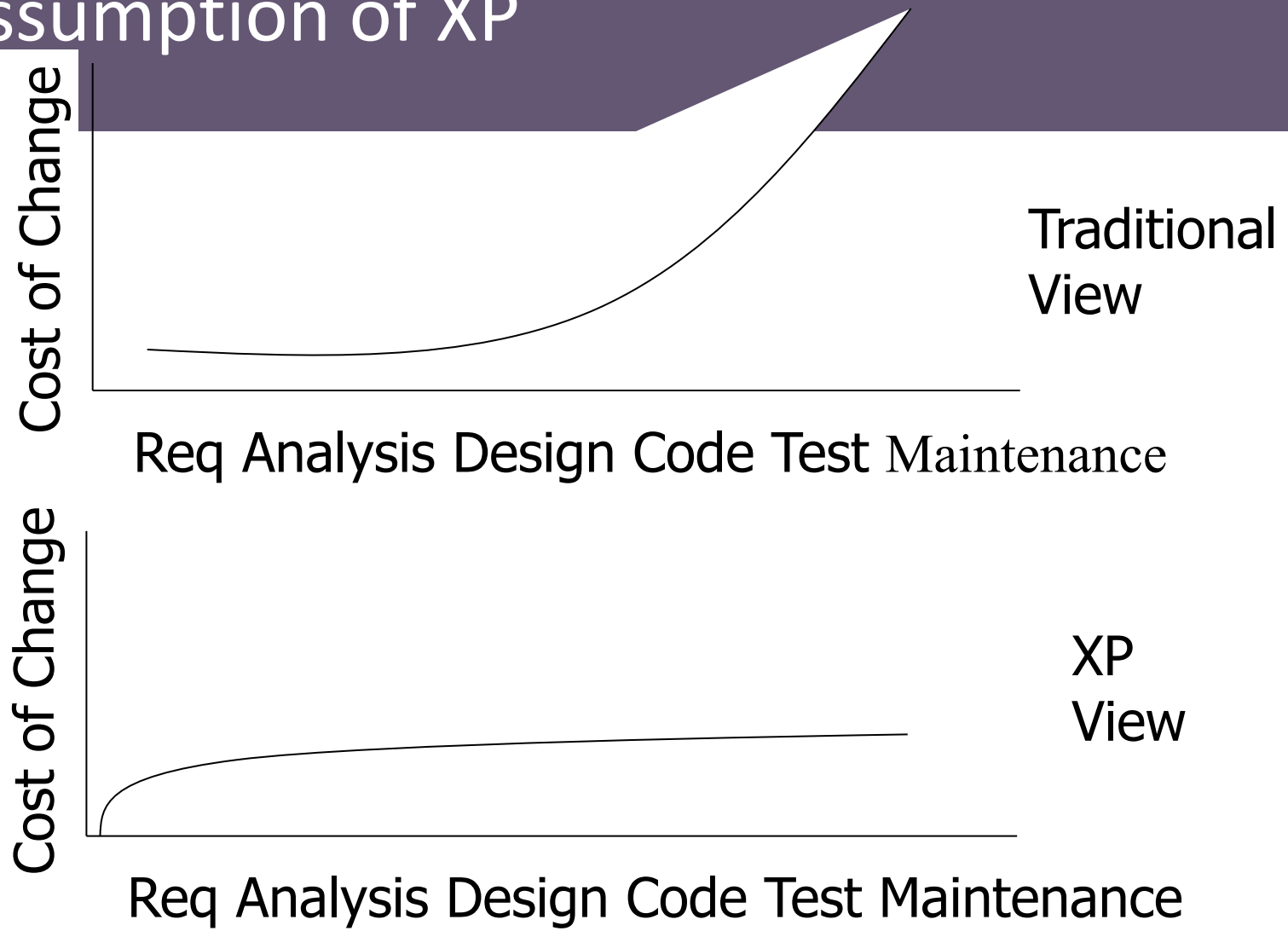
Acceptance Testing

Fixing small problems takes few minutes/hours
than fixing huge problems before the deadline

How AI Tools Can Support XP Practices

- <<Discussion Forum>>

Assumption of XP



Keeping Cost of Change Low in XP -Is it Really Possible?

- Emphasis on Simple Design:
  - Principle: Simple design to avoid complexity.
  - Benefit: Easier and cheaper to modify.
- Continuous Refactoring:
  - Practice: Regularly improve code structure.
  - Benefit: Reduces technical debt, easier changes.
- Test-Driven Development (TDD):
  - Practice: Write tests before code.
  - Benefit: Ensures changes do not break functionality.
- Pair Programming:
  - Practice: Two developers, one computer.
  - Benefit: Better code quality, easier changes.
- Continuous Integration:
  - Practice: Frequent code integration and testing.
  - Benefit: Early issue detection, cheaper fixes.
- Collective Code Ownership:
  - Practice: Any team member can change any code.
  - Benefit: Flexibility, reduced dependency on individuals.

Conclusion:

XP practices help keep change costs low.

Regular testing, refactoring, and simple design maintain flexibility and lower costs.

XP Approach and Practices

Three layers

Programming

Simple design, testing, refactoring, coding standards

Team Practices

- Collective ownership, continuous integration, metaphor, coding standards, 40 hours week, pair programming, small releases

Processes

On-site customer, testing, small releases, planning game

XP requires

Feedback

- Ask the system
  not a document
- Write test cases
- Do experiments

Courage

- Fix it!
- Toss it!
- Try it!

Activities

- Coding
- Testing
- Listening
- Designing

XP Summary

Aspect	XP Highlights
Origin	Kent Beck, late 1990s
Values	Simplicity, Communication, Feedback, Courage, Respect
Roles	Programmer, Customer, Coach, Tester, Tracker
Process	Short iterations, continuous testing, small releases
Practices	TDD, Pair Programming, Refactoring, Planning Game
Tools	JUnit, Jenkins, Live Share, Planning Poker
Pitfalls	Hard to adopt without discipline, team resistance
AI Help	Generate tests, assist refactoring, smart code suggestions

KANBAN and LEAN

- https://www.youtube.com/watch?v=R8dYLbJiTUE