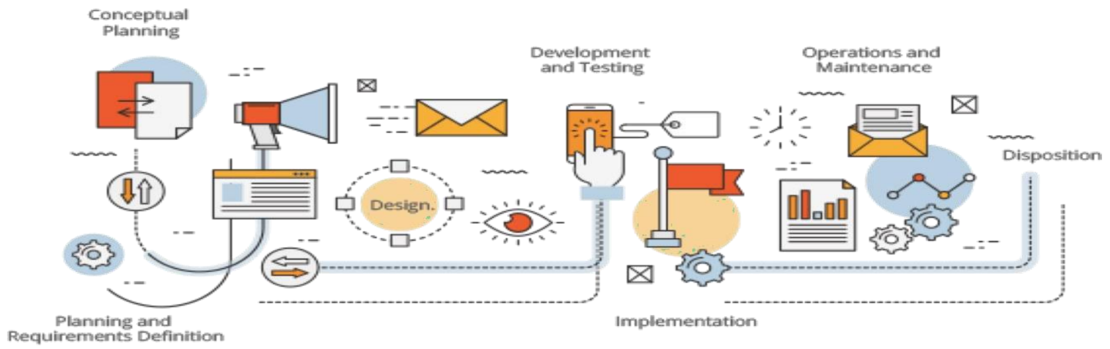
Software Engineering

Software Process Models – Agile Models



“Being Agile is to take that first of many steps and the best way to be Agile is; go Agile, step-by-step.”

- CA Vikram Verma

NATIONAL UNIVERSITY OF COMPUTING AND EMERGING SCIENCES

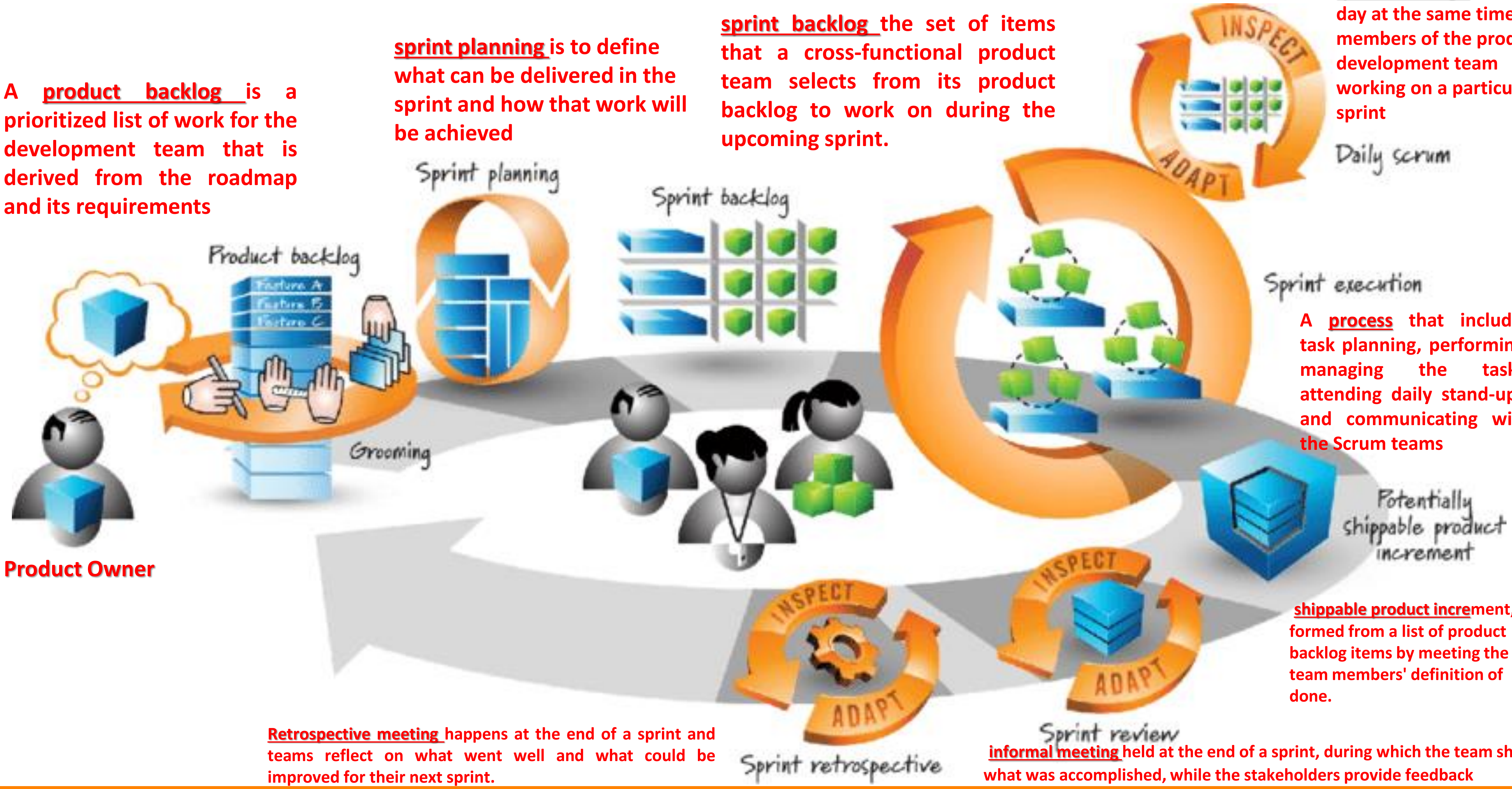
Scrum sprint cycle diagram (Revision)

A **product backlog** is a prioritized list of work for the development team that is derived from the roadmap and its requirements

sprint planning is to define what can be delivered in the sprint and how that work will be achieved

sprint backlog the set of items that a cross-functional product team selects from its product backlog to work on during the upcoming sprint.

quick meetings held each day at the same time for members of the product development team working on a particular sprint



A **process** that includes task planning, performing, managing the tasks, attending daily stand-ups, and communicating with the Scrum teams

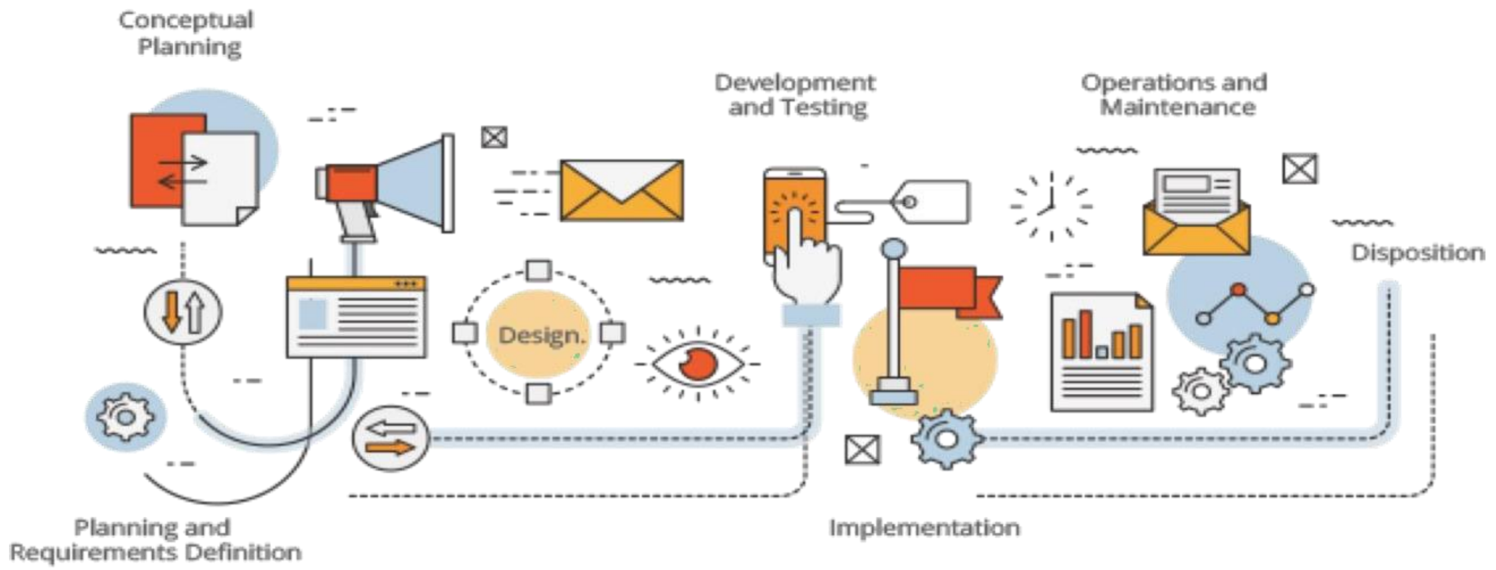
Product Owner

shippable product increment, formed from a list of product backlog items by meeting the team members' definition of done.

Retrospective meeting happens at the end of a sprint and teams reflect on what went well and what could be improved for their next sprint.

informal meeting held at the end of a sprint, during which the team shows what was accomplished, while the stakeholders provide feedback

Extreme Programming (XP)



What Is Extreme Programming (XP)?

- Extreme programming is a software development methodology that’s part of what’s collectively known as agile methodologies.
- Developed in 1990 and still relevant to modern day software development.
- XP is built upon **values, principles, and practices**, and its goal is to allow small to mid-sized teams to produce high-quality software and adapt to evolving and changing requirements.
- XP emphasizes the technical aspects of software development.
- Extreme programming is, in a nutshell, about good practices taken to an extreme. Since pair-programming is good, let’s do it all of the time. Since testing early is good, let’s test before the production code is even written.

How XP Works?

1. **Make small updates often** – Instead of big changes, release little improvements quickly.
2. **Work in pairs** – Two people write code together to catch mistakes early.
3. **Test first, code later** – Write tests first to make sure everything works right.
4. **Keep it simple** – No extra complexity, just what’s needed.
5. **Improve constantly** – Clean up and refine the code regularly.

Extreme Programming Explained (Beck [2000])

- XP takes common-sense principles and practices to extreme levels
  - If code reviews are good, we'll review code all the time (pair programming).
  - If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).
  - If design is good, we'll make it part of everybody's daily business (refactoring).
  - If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality (the simplest thing that could possibly work).
  - If architecture is important, everybody will work defining and refining the architecture all the time (metaphor).
  - If integration testing is important, then we'll integrate and test several times a day (continuous integration).
  - If short iterations are good, we'll make the iterations really, really short—seconds and minutes and hours, not weeks and months and years (the Planning Game).

The 5 Core Values of Extreme Programming (XP)

XP is built on **five key values** that guide how teams work together to create great software:

1. **Communication**
   - Team members talk openly and share ideas to avoid misunderstandings.
   - Regular discussions with customers and teammates keep everyone on the same page.
2. **Simplicity**
   - Write only the code needed—nothing extra or complex.
   - Focus on solving today’s problem instead of guessing future needs.
3. **Feedback**
   - Get continuous feedback from customers and teammates.
   - Use testing and reviews to catch mistakes early and improve quickly.
4. **Courage**
   - Be brave enough to change or throw away bad code.
   - Speak up about problems and make decisions based on facts, not fear.
5. **Respect**
   - Everyone’s ideas matter, and team members trust each other.
   - Work together in a supportive and positive way.

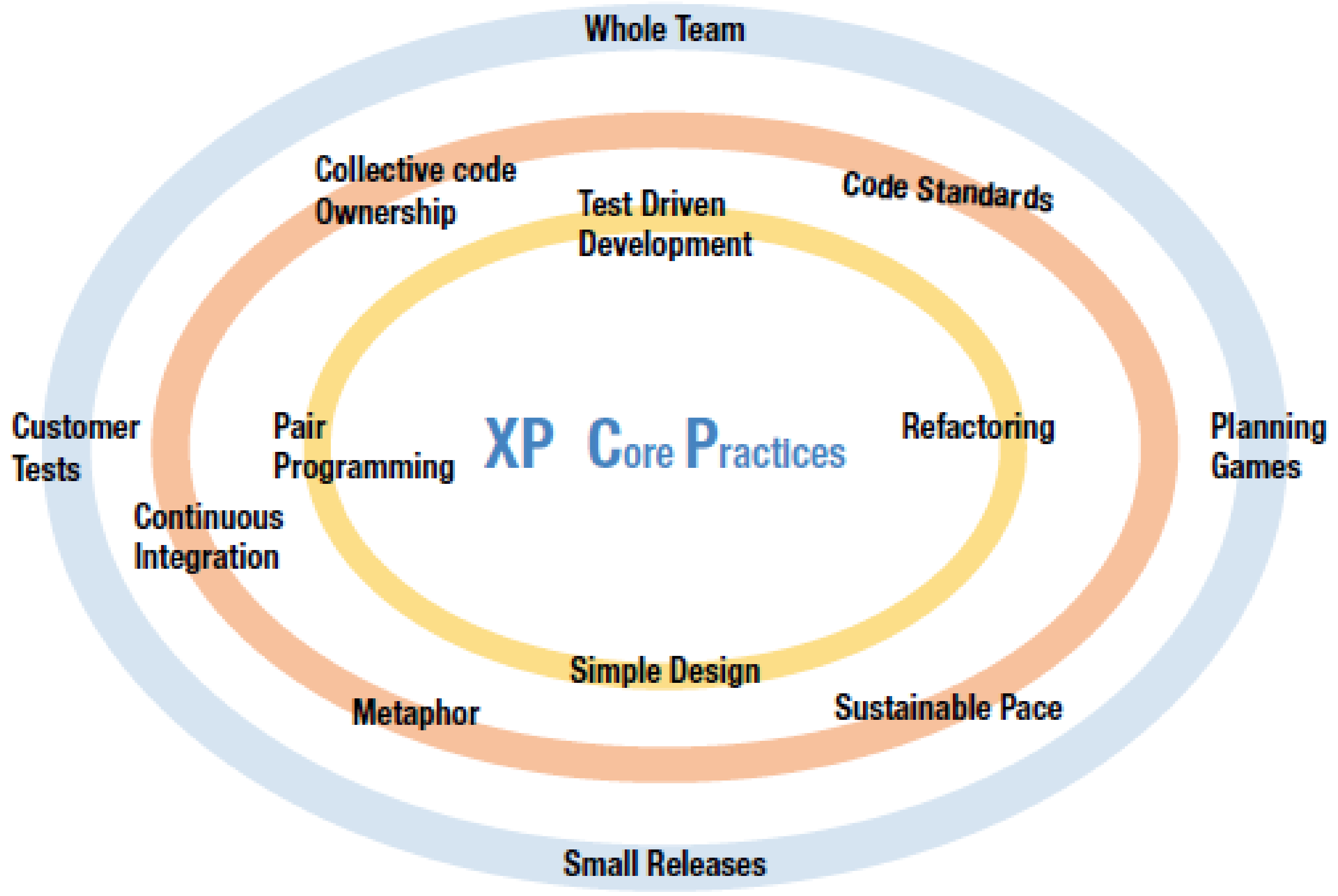
XP Process Model Steps

1. **Planning**
   - The customer and team define user stories (small, clear requirements).
   - The team estimates the time needed for each story.
   - A release plan is created for frequent small updates.
2. **Design**
   - Focuses on simple and clear architecture to avoid complexity.
   - Uses CRC cards (Class-Responsibility-Collaborator) to model the system.
   - Pair programming ensures better design quality.
3. **Coding**
   - Developers work in pairs to write clean, efficient code.
   - Test-Driven Development (TDD): Tests are written before actual code to ensure reliability.
   - Frequent refactoring keeps the code simple and maintainable.
4. **Testing**
   - Automated tests are run continuously to catch issues early.
   - Acceptance tests ensure the software meets customer requirements.
   - Any failed test leads to immediate fixes.
5. **Integration & Deployment**
   - Code is integrated multiple times a day (Continuous Integration).
   - Frequent small releases allow quick customer feedback.
   - The system evolves gradually without large, risky changes.

Agile Practices from Extreme Programming

- User Stories
  - In their words, what the system needs to do for customers
- Acceptance Tests
  - Customer specifies scenarios to test a user story
- Release Planning
  - Together, developers and customers decide on user stories to implement in the next release/iteration
- Test-Driven Development
  - Write tests before implementing each feature
- Refactoring
  - Restructure to clean up the design as new code is added

XP Practices



XP Practices

- **Faster feedback**
  1. Test driven development
  2. Onsite customer
  3. Pair programming
- **Continuous process**
  4. Continuous integration
  5. Refactoring
  6. Short releases
- **Shared understanding**
  7. The planning game
  8. Simple Design
  9. System Metaphor
  10. Collective ownership
  11. Coding Standards
- **Developers welfare**
  12. 40-Hour Week

#1 – The Planning Game

The **Planning Game** helps the team and customer collaborate on **prioritizing and planning** what work to do and when. It breaks work into smaller, manageable pieces, ensures that the team focuses on **delivering high-value features**, and provides a **feedback loop** to continuously adjust based on customer needs and team capacity.

There are 2 levels of plans in XP; level one is release planning and level 2 is iteration planning.

- The first phase is **<u>release planning</u>**. The team and stakeholders/customers collaboratively decide what are the requirements and features that can be delivered into production and when. This is done based on priorities, capacity, estimations and risks factor of the team to deliver.

- In **<u>Iteration planning</u>** the team will pick up the most valuable items from the list and break them down into tasks then estimates and a commitment to delivering at the end of the iteration.

#2 – Simple Design

- In the XP, the team will not do complex or big architecture and designs upfront; instead the team will start with a simple design and let it emerge and evolve over a period of iterations.

- Flexible design early to set direction: Start with a design that is flexible enough to handle later changes without major rework

- Design incrementally: Start with a minimal plausible design, add and refactor at the end of each iteration

#3 – Using Metaphors

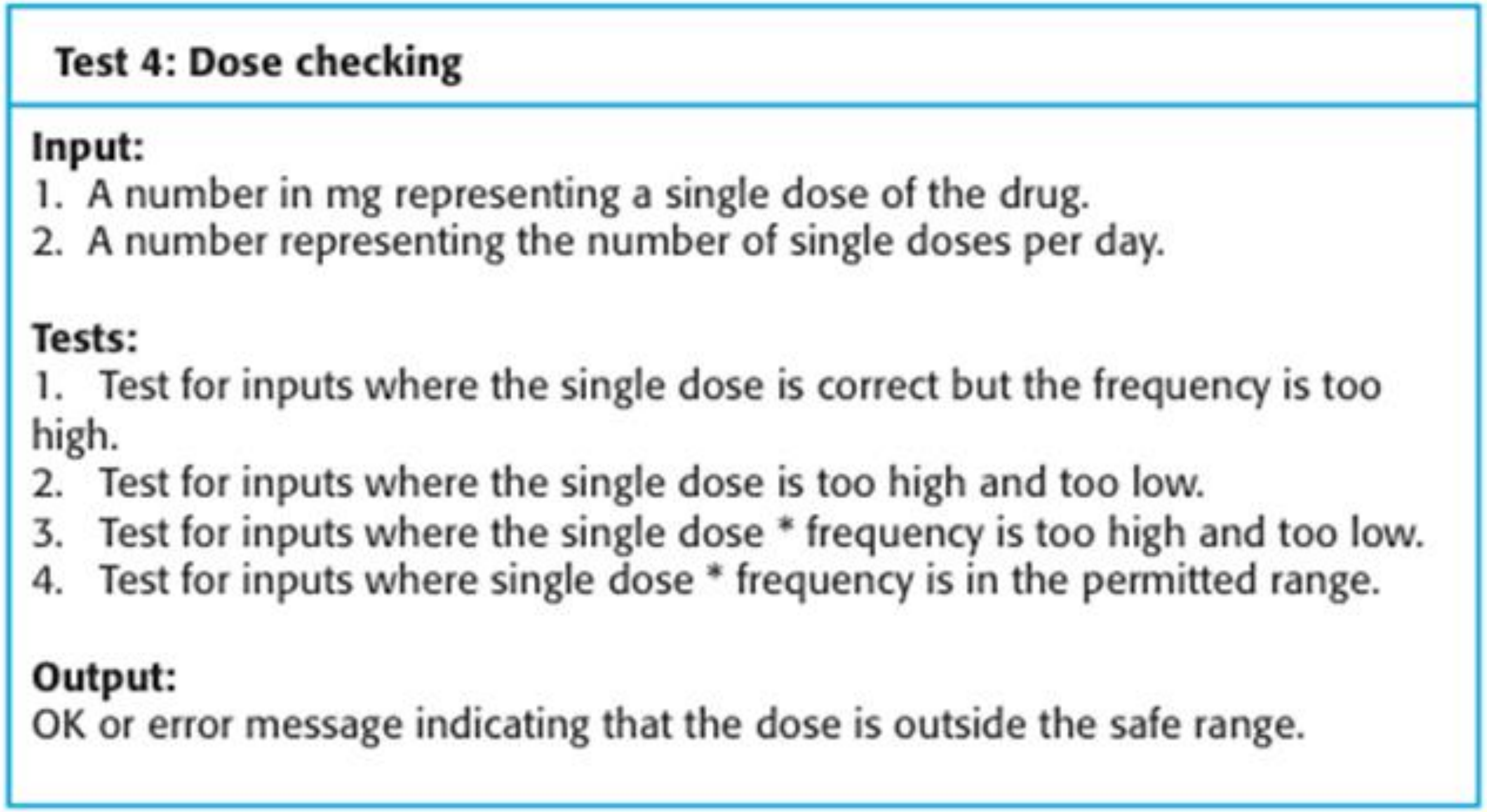
- Find a suitable metaphor to convey information.
- Use everyday language and metaphors to explain the system's design and functionality.
- Metaphors help make complex concepts clearer and improve communication between team members and the customer.
- The metaphors should be simple and relatable, making it easy to discuss how the system works.
- It could be a naming conversion practice used in design and code to have a shared understanding between teams. For example Order_Food() is easily explained -- this will be used to order food.

#4 – Test-Driven Development (TDD)

- In XP Developers write the unit test cases before starting the coding. The team automates the unit tests and it helps during the build and integration stage. The main benefit of TDD is that programmers have to only write the code that passes the tests.

The "practice of test-first development, planning and writing tests before each micro-increment" was used as early as NASA's Project Mercury, in the early 1960s
–Larmanand Basili[2003]

Test Case Description – For Dose Checking



Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

#5 – Refactoring

- Refactoring as the word suggests, is restructuring or reconstructing existing things. In XP over a period of time the team produces lots of working code and it increases the complexity and that contributes technical debt. To avoid this, we should consider the below points,

  - Ensure code or functions are not duplicated.
  - Ensure all variables in scope, defined and used.
  - No long functions or methods
  - Removing unnecessary stuff and variables
  - Proper use of access modifiers etc.

- By refactoring, the programmers look to improve the overall code quality and make it more readable without altering its behavior.

#6 – Pair Programming

- Two programmers at one desk operating on the same code and unit test cases, on the same system (one display and one keyboard).

- One programmer plays the pilot role and focuses on clean code, and compiles and runs. The second one plays the role of navigator and focuses on the big picture and reviews code for improvement or refactoring.

- Every hour or given a period of time this pair is allowed to switch roles so that everyone gets to know about the code and functionality of the whole system.

Benefits of Pair Programming

- **Fewer Bugs** – Mistakes are caught early with two sets of eyes.
- **Better Code Quality** – Code is cleaner and follows best practices.
- **Faster Problem Solving** – Two minds working together find solutions quicker.
- **Skill Sharing** – Junior developers learn from experienced developers.
- **Better Team Communication** – Developers collaborate more effectively.

Challenges of Pair Programming

- **Takes More Time** – Two people work on the same task, which can slow things down initially.
- **Personality Clashes** – Not all developers work well together.
- **Requires Focus** – Constant discussion can be exhausting.
- **Increased Cost** – Two developers working on one task can be costly for businesses.

#7- Collective Ownership

- No single person owns the code
- Any team member can work any time on the code.
- Success or failure is a collective effort and there is no blame game.
- There is no one key player here, so if there is a bug or issue then any developer can be called to fix it.

#8 – Continuous Integration

- CI is Continuous Integration. In XP, Developers do pairs programming on local versions of the code. There is a need to integrate changes made every few hours or on a daily basis so after every code compilation and build we have to integrate it where all the tests are executed automatically for the entire project.

- If the tests fail, they are fixed then and there, so that any chance of defect propagation and further problems are avoided with minimum downtime.

- The team can use CI tools like Jenkins, shippable, Integrity, Azure DevOps Pipelines, etc.

#9 – 40 hour a week

- No late sittings
- Multiple consecutive weeks of overtime are treated as “Something wrong”
- Project team members are required to work no more than 40 hours a week, and overtime should not exceed two consecutive weeks, otherwise, productivity will be affected.

#10 – Open Workspace

- **Open Workspace** is an Extreme Programming (XP) practice where the development team works in a **shared, open environment** instead of private offices or cubicles. This setup encourages **better communication, faster problem-solving, and stronger teamwork**.

#11 – On Site Customer

- Add a real, functional user to the team who will answer questions full-time.
- at least one actual customer representative is required to be on-site throughout the project development cycle to determine requirements, answer team questions, and write functional acceptance tests.
- Continuous access to customer
- For large number of customers
  - Customer representative

Challenges & Solutions

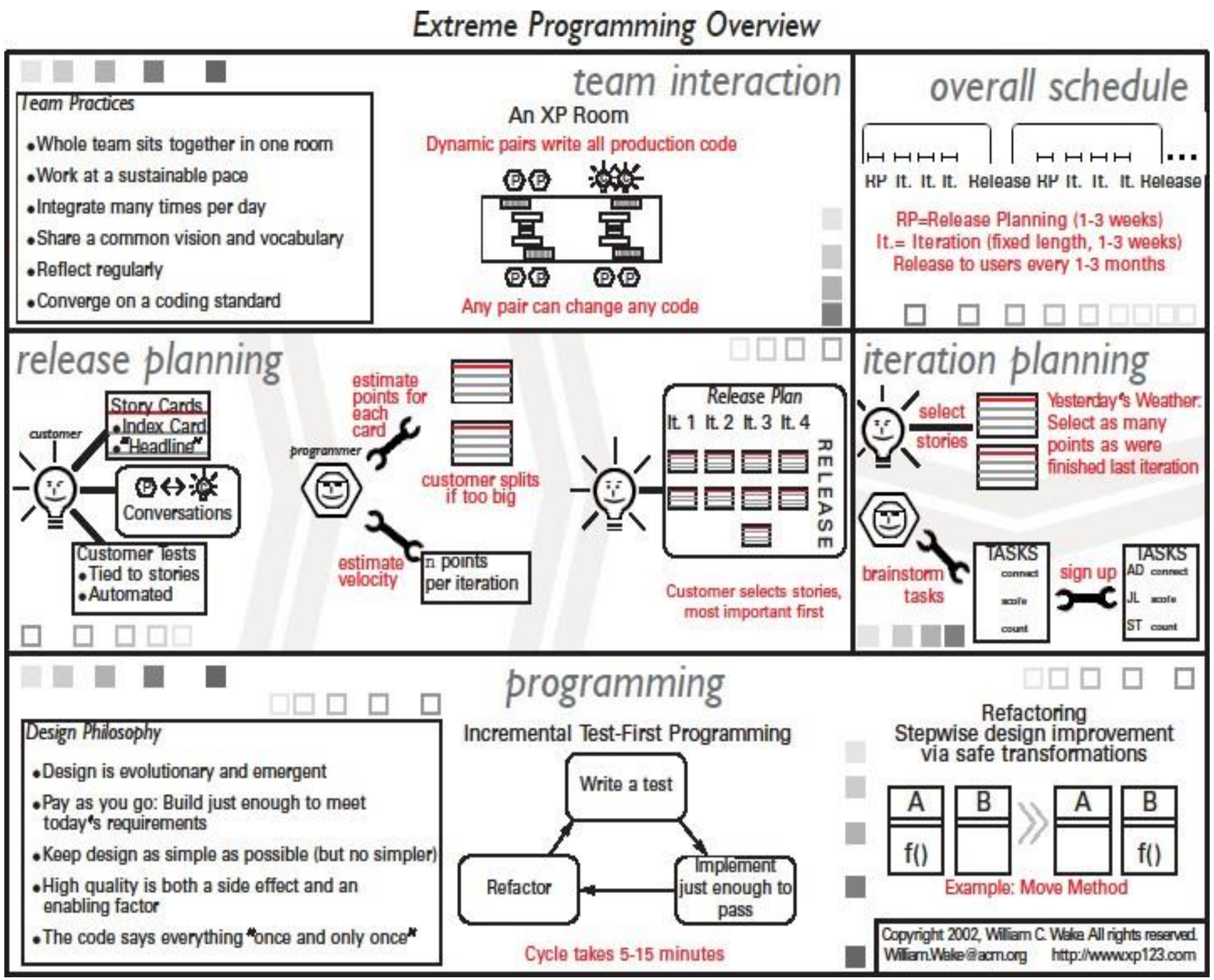
Challenge	Solution
Customer may not always be available	Appoint a **dedicated product owner** to act as their representative
Developers might depend too much on the customer	Use **user stories and documentation** to maintain clarity
Customer might struggle with technical discussions	Developers should explain things in **simple, non-technical terms**

#12– Coding Standards

- Organizations want their programmers to hold to some well-described and standard style of coding called coding standards. It is a guideline for the development team as in XP. Since there are multiple programming pairs at play coding standards are very useful to make consistency in code, style, naming conversion, exception handling and use of parameters.
- These standards must define and agreed before the team starts the coding.
- It will make the code simple to understand and helps detect the problem or issues quickly and also increases the efficiency of the software.
- Same standard by all members
- Code look familiar

Common XP Coding Standards

- **Use Meaningful Variable & Function Names** – Avoid x, y, or temp. Use customer_name instead of cn.
- **Consistent Formatting** – Follow indentation, spacing, and line breaks.
- **Write Simple & Clean Code** – Avoid unnecessary complexity.
- **Follow Language-Specific Best Practices** – E.g., PEP8 for Python, Java conventions for Java.
- **Automate Standards** – Use tools like ESLint (JavaScript), Prettier, Black (Python).


Extreme Programming Overview

team interaction

Team Practices

- Whole team sits together in one room
- Work at a sustainable pace
- Integrate many times per day
- Share a common vision and vocabulary
- Reflect regularly
- Converge on a coding standard

An XP Room
Dynamic pairs write all production code
Any pair can change any code

overall schedule

RP It. It. It. Release RP It. It. It. Release

RP=Release Planning (1-3 weeks)
It.= Iteration (fixed length, 1-3 weeks)
Release to users every 1-3 months

release planning

iteration planning

programming

Design Philosophy

- Design is evolutionary and emergent
- Pay as you go: Build just enough to meet today's requirements
- Keep design as simple as possible (but no simpler)
- High quality is both a side effect and an enabling factor
- The code says everything "once and only once"

Incremental Test-First Programming
Write a test
Implement just enough to pass
Refactor
Cycle takes 5-15 minutes
Refactoring
Stepwise design improvement via safe transformations
A
f()
B
A
B
f()
Example: Move Method