Software Engineering

Dr. Meisam Nazariani

Email: m_nazariani@aut.ac.ir

February 2022

Outline

AGILE DEVELOPMENT

Agile Development

Agile Methods

1. XP
2. Scrum

Objectives

- The objective of this chapter is to introduce you to agile software development methods. When you have read the chapter, you will:

  - ✓ Understand the rationale for agile software development methods, the agile manifesto, and the differences between agile and plan driven development.

  - ✓ Know about important agile development practices such as user stories, refactoring, pair programming and test-first development.

  - ✓ Understand the Scrum approach to agile project management; understand the issues of scaling agile development methods and combining agile approaches with plan-driven approaches in the development of large software systems.

Manifesto

In 2001, Kent Beck and 16 other signed the “Manifesto for Agile Software Development.” It stated:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ✓ Individuals and interactions over processes and tools
- ✓ Working software over comprehensive documentation
- ✓ Customer collaboration over contract negotiation
- ✓ Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Agile Development

Agile Development (Quick look)

1.What is it?

Agile software engineering combines a philosophy and a set of development guidelines.

- The philosophy encourages customer satisfaction and early incremental delivery of software; small, highly motivated project teams; informal methods; minimal software engineering work products; and overall development simplicity.

- The development guidelines stress delivery over analysis and design and active and continuous communication between developers and customers.

Agile Development (Quick look)

2.Who does it?

- Software engineers and other project stakeholders (managers, customers, end users) work together on an agile team.
- A team that is self-organizing and in control of its own destiny.
- An agile team fosters communication and collaboration among all who serve on it.

Agile Development (Quick look)

3.Why is it important?

- The modern business environment that spawns computer based systems and software products is fast paced and ever changing.
- Agile software engineering represents a reasonable alternative to conventional software engineering for certain classes of software and certain types of software projects.
- It has been demonstrated to deliver successful systems quickly.

Agile Development (Quick look)

4. What are the steps?

- Agile development might best be termed "software engineering lite."
- The basic framework activities communication, planning, modeling, construction, and deployment remain.
- But they morph into a minimal task set that pushes the project team toward construction and delivery.

Agile Development (Quick look)

5. What is the work product?

- Only really important work product is an operational "software increment" that is delivered to the customer on the appropriate commitment date.

WHAT IS AGILITY?

What is agility in the context of software engineering work?

- ❑ An agile team is a nimble team able to appropriately respond to changes.

  Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product.

- ❖ The pervasiveness of change is the primary driver for agility.

Agile development methods emerged in the late 1990

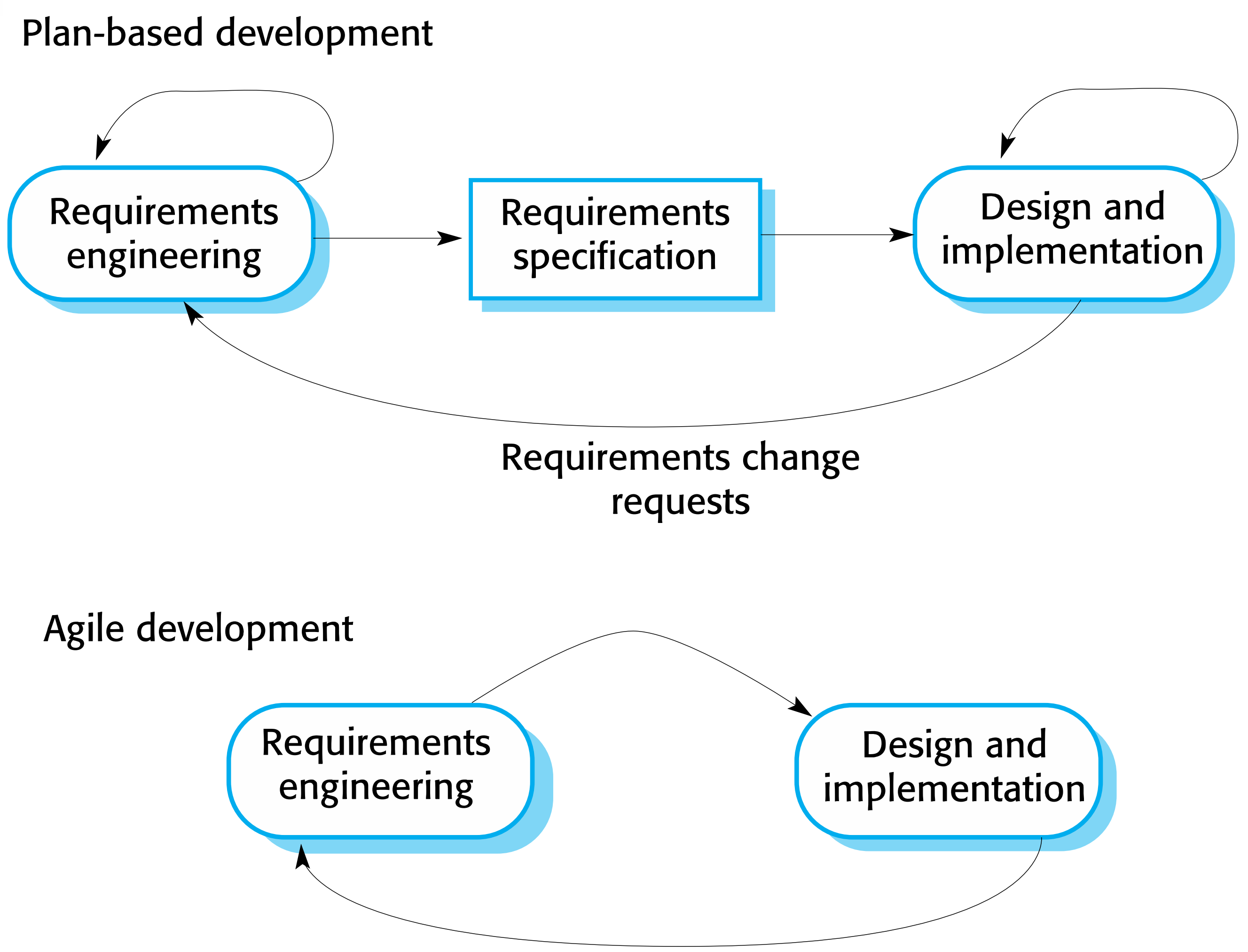
WHAT IS AGILITY?

The pervasiveness of change is the primary driver for agility.

- **But agility is more than an effective response to change**

- It encourages team structures and attitudes that make communication more facile.
- It emphasizes rapid delivery of operational software and deemphasizes the importance of intermediate work products.
- It adopts the customer as a part of the development team.
- It recognizes that planning in an uncertain world has its limits and that a project plan must be flexible.

Plan-driven and agile development


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

Plan-driven and agile development

❑ Plan-driven development

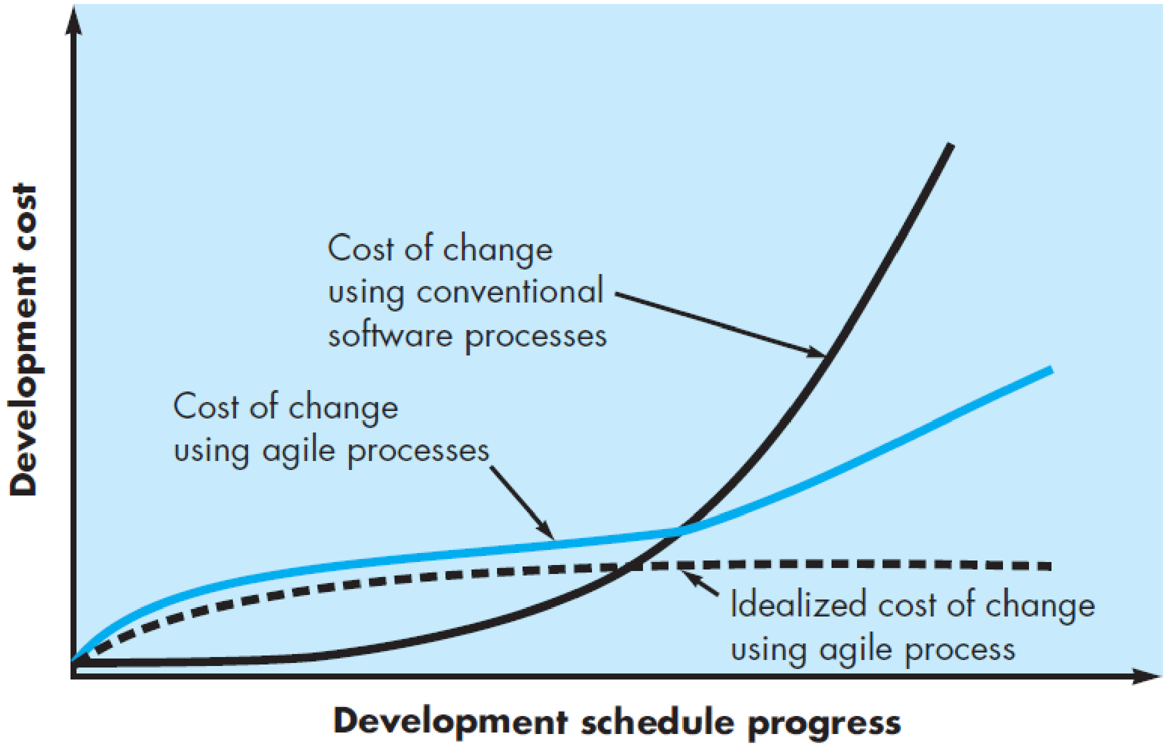
- A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- Not necessarily waterfall model plan driven, incremental development is possible
- Iteration occurs within activities.

❑ 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.

AGILITY AND THE COST OF CHANGE

- The conventional wisdom in software development is that the **cost of change increases nonlinearly as a project progresses**.



It is relatively easy to accommodate a change when a software team is gathering requirements (early in a project)

An agile process reduces the cost of change because software is released in increments and change can be better controlled within an increment.

WHAT IS AN AGILE PROCESS?

- How do we create a process that can manage unpredictability? The answer, lies in process adaptability (to rapidly changing project and technical conditions).

An agile process, must be adaptable.
But continual adaptation without forward progress accomplishes little.
Therefore, an agile software process must adapt incrementally.
To accomplish incremental adaptation, an agile team requires customer feedback.

An effective catalyst for customer feedback is an operational prototype or a portion of an operational system.
Hence, an incremental development strategy should be instituted.

Software increments (executable prototypes or portions of an operational system) must be delivered in short time periods so that adaptation keeps pace with change.

This iterative approach enables the customer to evaluate the software increment regularly, provide necessary feedback to the software team, and influence the process adaptations that are made to accommodate the feedback.

Agility Principles

❑ 12 agility principles

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

2. Welcome changing requirements, even late in development. Agile processes 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.

Agility Principles

❑ **12 agility principles**

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

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, 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.

Agile methods

XP

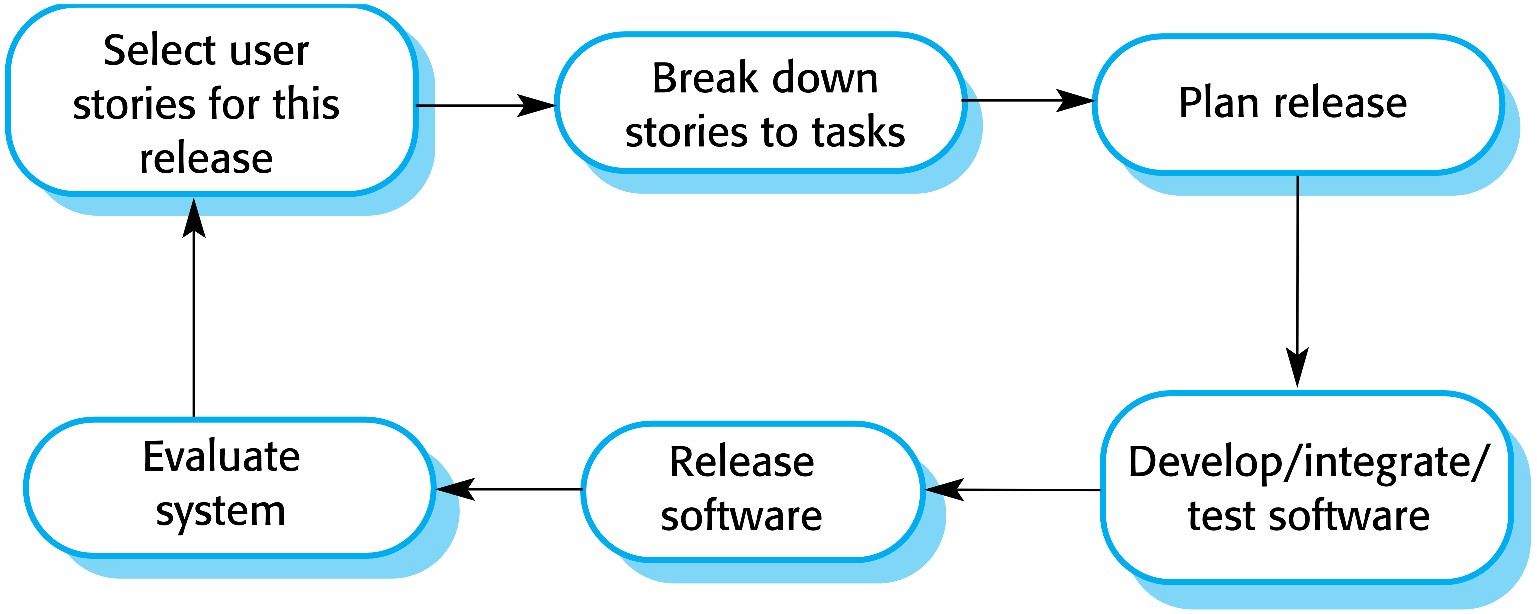
EXTREME PROGRAMMING

❑ XP

- The most widely used approach to agile software development : **Extreme Programming (XP)**

- Industrial XP (IXP)

The extreme programming release cycle


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

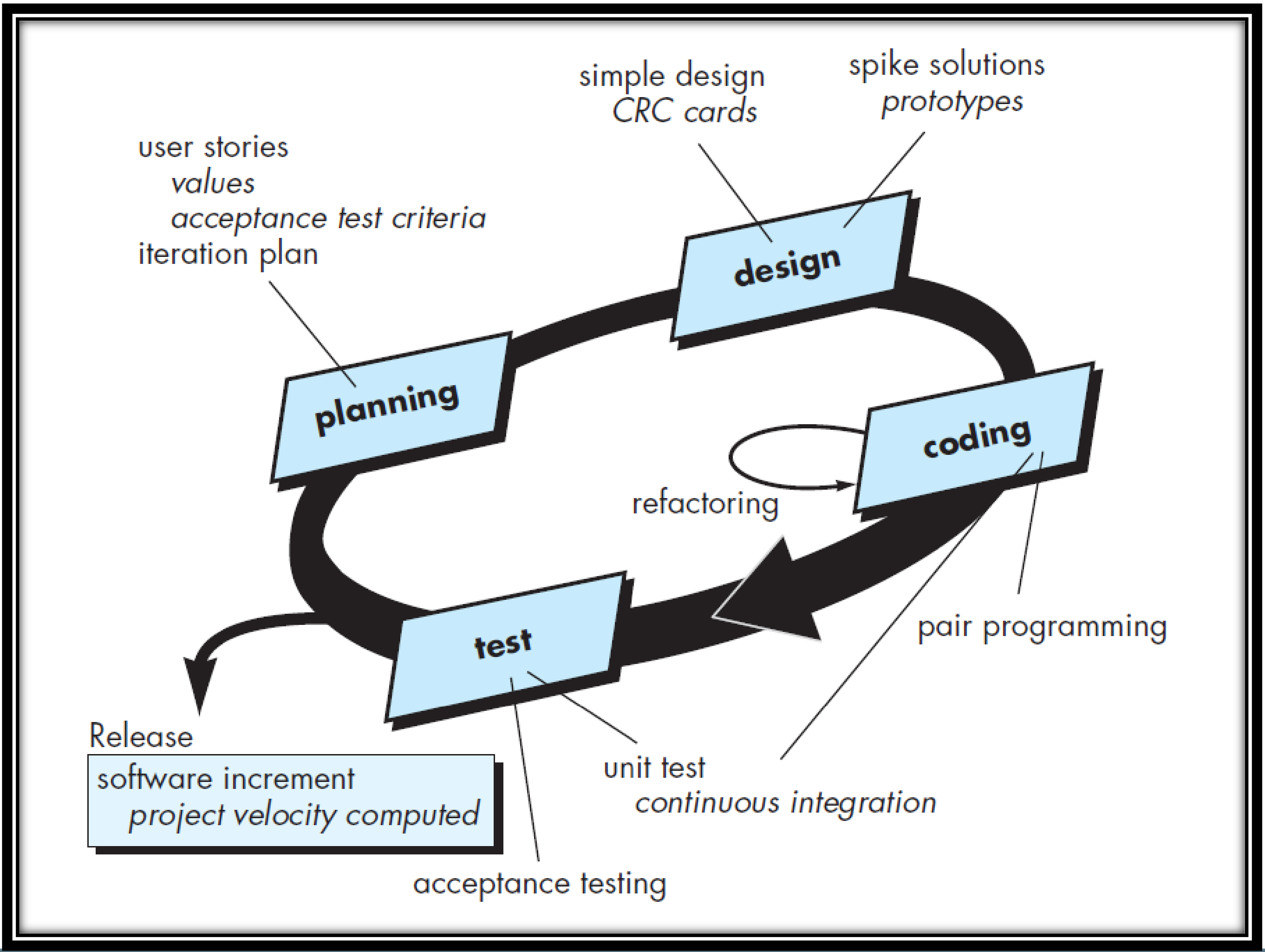
The XP Process

- Extreme Programming **uses an object oriented approach** (Appendix 2) as its preferred development paradigm.

- And encompasses a set of rules and practices that occur within the context of **four framework activities**:

  1. **Planning**
  2. **Design**
  3. **Coding**
  4. **Testing**

The Extreme Programming process



1.Planning(a)

What is an XP "story"?

A. The planning activity begins with listening: a requirements gathering activity that enables the technical members of the XP team to understand the business context for the software.

B. Listening leads to the creation of a set of " stories " (user stories) that describe required output, features, and functionality for software to be built.

C. Each story is written by the customer and is placed on an index card. the customer assigns a value (priority) to the story based on the overall business value of the feature or function

D. Members of the XP team then assess each story and assign a cost to it. If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.

❖New stories can be written at any time

Examples of story

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

1.Planning(b)

E. Once a basic commitment is made for a release, the XP team **orders the stories** that will be developed in one of three ways:

   I. All stories will be implemented immediately.

   II. The stories with highest value will be moved up in the schedule and implemented first.

   III. The riskiest stories will be moved up in the schedule and implemented first.

F. After the first project release has been delivered, the XP team computes **project velocity** (project velocity is the number of customer stories implemented during the first release)

❖ As development work proceeds, the customer can add stories, change the value of an existing story, split stories, or eliminate them.

2. Design(a)

A. XP design rigorously follows the **KIS (keep it simple)** principle. A simple design is always preferred over a more complex representation. The design of extra functionality (because the developer assumes it will be required later) is discouraged.

B. XP encourages the use of **CRC (class-responsibility-collaborator)** cards which identify and organize the object oriented classes that are relevant to the current software increment.

C. If a difficult design problem is encountered as part of the design of a story, XP recommends the immediate creation of an operational prototype of that portion of the design. Called a **spike solution** , the design prototype is implemented and evaluated. The intent is to lower risk when true implementation starts and to validate the original estimates for the story containing the design problem.

D. Members of the XP team then assess each story and assign a **cost** to it. If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.

Examples of task cards

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

2. Design(b)

XP encourages **Refactoring**

- ✓ **Refactoring** is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure.
- ✓ It is a disciplined way to clean up code [and modify/simplify the internal design] that minimizes the chances of introducing bugs.
- ✓ In essence, when you refactor you are improving the design of the code after it has been written. The intent of refactoring is to control these modifications by suggesting small design changes that "can radically improve the design".
- ✓ A central notion in XP is that design occurs both before and after coding commences. Refactoring means that design occurs continuously as the system is constructed.

Examples of refactoring

- ✓ Re-organization of a class hierarchy to remove duplicate code.
- ✓ Tidying up and renaming attributes and methods to make them easier to understand.
- ✓ The replacement of inline code with calls to methods that have been included in a program library.

3. Coding

A. After design work is done, the team does not move to code, but rather develops a series of unit tests that will exercise each of the stories that is to be included in the current release. Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test.

B. A key concept during the coding activity is **pair programming** . XP recommends that two people work together at one computer workstation to create code for a story. This provides a mechanism for real-time problem solving and real-time quality assurance (the code is reviewed as it is created).

C. As pair programmers complete their work, the code they develop is integrated with the work of others. This "**continuous integration**" strategy helps to avoid compatibility and interfacing problems and provides a "**smoke testing**" environment that helps to uncover errors early.

❖ **Smoke Testing**: is a software testing process that determines whether the deployed software build is stable or not.

4. Testing

A. The unit tests that are created should be implemented using a framework that enables them to be automated.

B. As the individual unit tests are organized into a "universal testing suite" integration and validation testing of the system can occur on a daily basis. This provides the XP team with a continual indication of progress and also can raise warning flags early if things go awry.

C. XP **acceptance tests** , also called customer tests, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer. Acceptance tests are derived from user stories that have been implemented as part of a software release

Wells states: **Fixing small problems every few hours takes less time than fixing huge problems just before the deadline.**

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.