THE CONCURRENT DEVELOPMENT MODEL

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

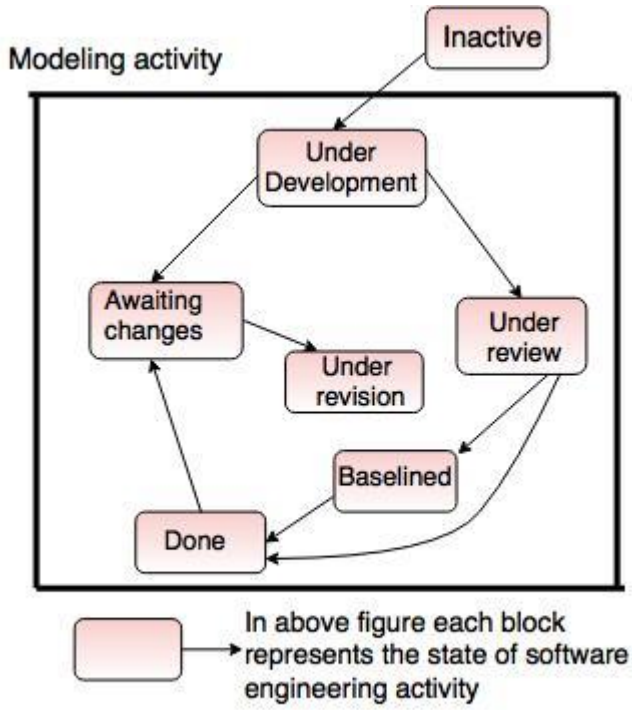


Fig. - One element of the concurrent process model

Advantages of the concurrent development model

- This model is applicable to all types of software development processes.
- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities.

AGILE DEVELOPMENT

- Agile software engineering combines a philosophy and a set of development guidelines.
- The development stress delivery over analysis and design, and active and continuous communication between developers and customer.

AGILE PROCESS:

- Agility Principal
- Human Factors

a. Agile principles

- The highest priority of this process is to satisfy the customer.
- Acceptance of changing requirement even late in development.
- Frequently deliver a working software in small time span.
- Throughout the project business people and developers work together on daily basis.
- Projects are created around motivated people if they are given the proper environment and support.
- Face to face interaction is the most efficient method of moving information in the development team.
- Primary measure of progress is a working software.
- Agile process helps in sustainable development.
- Continuous attention to technical excellence and good design increases agility.
- From self organizing teams the best architecture, design and requirements are emerged.
- Simplicity is necessary in development.

b. HUMAN FACTORS

- Competence:- In an agile development "competence" encompasses innate talent, specific software-related skills, and overall knowledge of the process

that the team has chosen to apply. Skill and knowledge of process can and should be taught to all people who serve as agile team members.

- Common focus. Although members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal—to deliver a working software increment to the customer within the time promised. To achieve this goal, the team will also focus on continual adaptations (small and large) that will make the process fit the needs of the team.
- Collaboration. Software engineering (regardless of process) is about assessing, analyzing, and using information that is communicated to the software team; creating information that will help all stakeholders understand the work of the team; and building information (computer software and relevant databases) that provides business value for the customer. To accomplish these tasks, team members must collaborate—with one another and all other stakeholders.
- Decision-making ability. Any good software team (including agile teams) must be allowed the freedom to control its own destiny. This implies that the team is given autonomy—decision-making authority for both technical and project issues.
- Fuzzy problem-solving ability. Software managers must recognize that the agile team will continually have to deal with ambiguity and will continually be buffeted by change. In some cases, the team must accept the fact that the problem they are solving today may not be the problem that needs to be solved tomorrow.
- Mutual trust and respect. The agile team must become what is called "jelled" team . A jelled team exhibits the trust and respect that are necessary to make them "so strongly knit that the whole is greater than the sum of the parts.
- Self-organization. In the context of agile development, self-organization implies three things:

1. The agile team organizes itself for the work to be done
2. The team organizes the process to best accommodate its local environment
3. The team organizes the work schedule to best achieve delivery of the software increment

EXTREME PROGRAMMING (XP)

- The Extreme Programming is commonly used agile process model.
- It uses the concept of object-oriented programming.
- A developer focuses on the framework activities like planning, design, coding and testing. XP has a set of rules and practices.

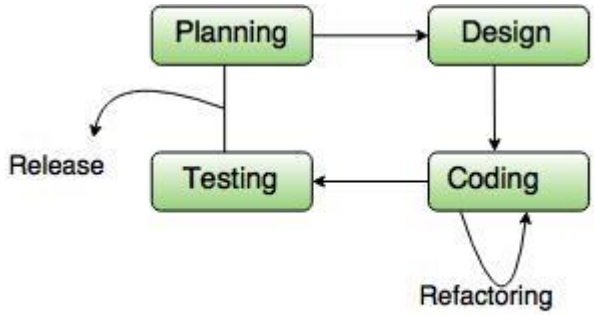


Fig. - The Extreme Programming Process

a. XP values

Following are the values for extreme programming:

1. Communication

- Building software development process needs communication between the developer and the customer.
- Communication is important for requirement gathering and discussing the concept.

2) Simplicity

The simple design is easy to implement in code.

3. Feedback

Feedback guides the development process in the right direction.

4. Courage

In every development process there will always be a pressure situation. The courage or the discipline to deal with it surely makes the task easy.

5. Respect

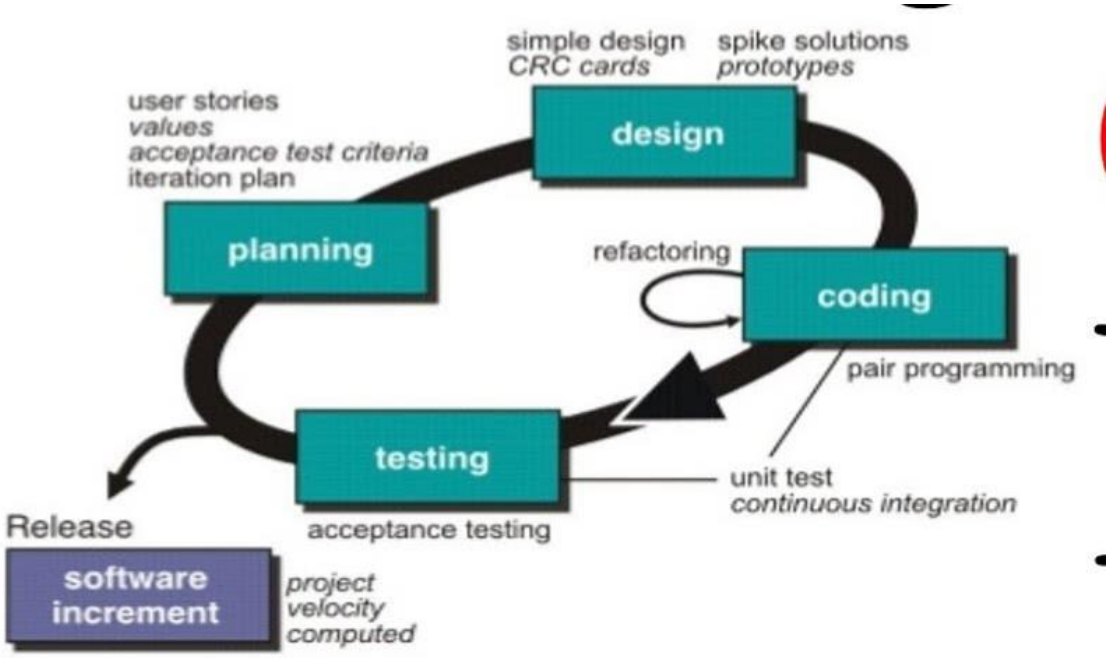
Agile process should inculcate the habit to respect all team members, other stake holders and customer.

B. The XP Process

- **The XP process comprises four framework activities:**

The most widely used agile process, originally proposed by Kent Beck in 2004. It uses an object-oriented approach.

- XP Planning
  - Begins with the listening, leads to creation of "user stories" that describes required output, features, and functionality. Customer assigns a value(i.e., a priority) to each story.
  - Agile team assesses each story and assigns a cost (development weeks. If more than 3 weeks, customer asked to split into smaller stories)
  - Working together, stories are grouped for a deliverable increment next release.
  - A commitment (stories to be included, delivery date and other project matters) is made. Three ways: 1. Either all stories will be implemented in a few weeks, 2. high priority stories first, or 3. the riskiest stories will be implemented first.
  - After the first increment "project velocity", namely number of stories implemented during the first release is used to help define subsequent delivery dates for other increments. Customers can add stories, delete existing stories, change values of an existing story, split stories as development work proceeds.
- XP Design (occurs both before and after coding as refactoring is encouraged)
  - Follows the KIS principle (keep it simple) Nothing more nothing less than the story.
  - Encourage the use of CRC (class-responsibility-collaborator) cards in an object-oriented context. The only design work product of XP. They identify and organize the classes that are relevant to the current software increment. For difficult design problems, suggests the creation of "spike solutions"—a design prototype for that portion is implemented and evaluated.
  - Encourages "refactoring"—an iterative refinement of the internal program design. Does not alter the external behavior yet improve the internal structure. Minimize chances of bugs. More efficient, easy to read.
- XP Coding
  - Recommends the construction of a unit test for a story *before* coding commences. So implementer can focus on what must be implemented to pass the test.
  - Encourages "pair programming". Two people work together at one workstation. Real time problem solving, real time review for quality assurance. Take slightly different roles.
- XP Testing
  - All unit tests are executed daily and ideally should be automated. Regression tests are conducted to test current and previous components.
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality



c. Industrial XP:-

Industrial Extreme Programming(IXP) is an organic evolution of XP. IXP incorporates new practices , they are::

· Reading assessment

· Project community

· Project chartering

· Test-driven management

· Retrospectives

· Continuous learning

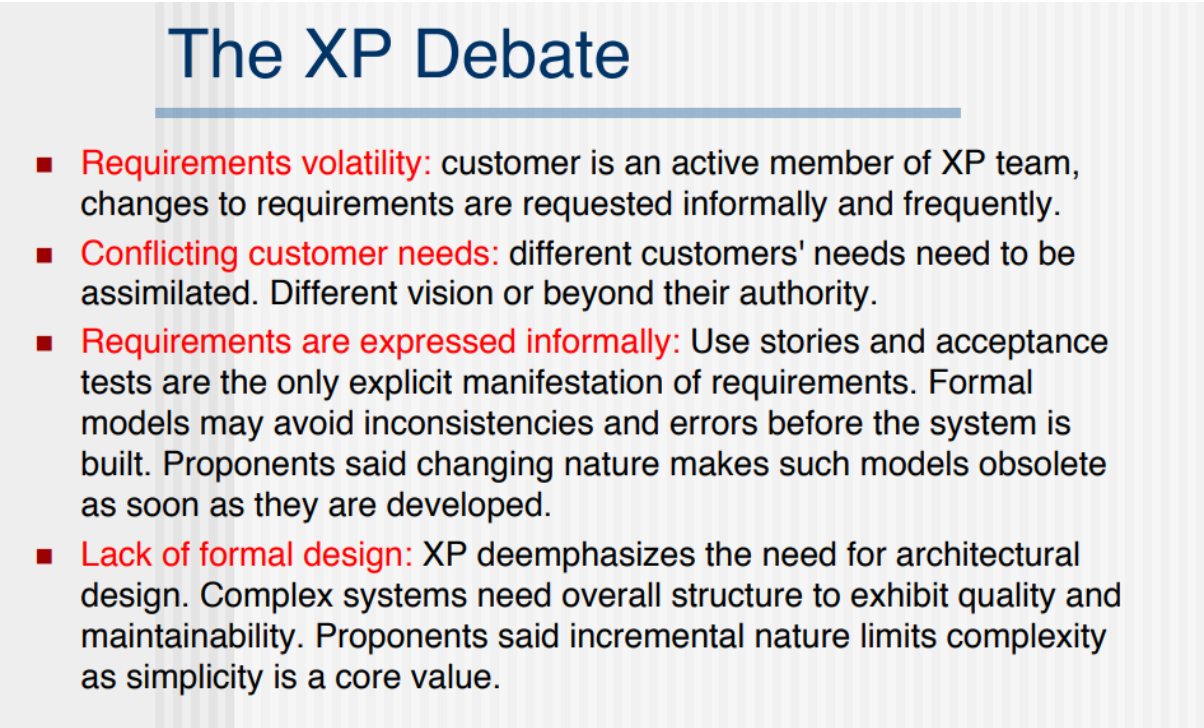
· SSD:story-driven development

· DDD:domain-driven design

· Pairing

· Iterative usability

d.



Reference:

Software Engineering: A Practitioner's Approach, 7/e (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman

www.tutorialpoint.com