Agile Software Development

Practicing XP

(Unit 3)

Dr. Sreedhar Bhukya

Professor, Department of CSE, SNIST

Extreme Programming

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques
- Extreme Programming (XP) takes an „extreme" approach to iterative development
  - New versions may be built several times per day
  - Increments are delivered to customers every 2 weeks
  - All tests must be run for every build and the build is only accepted if tests run successfully.

Key Practices You Need to Explore for Your Team

Key Practice #1 – Pair Programming. ...
Key Practice #2 – Planning Game. ...
Key Practice #3 – Continuous Process. ...
Key Practice #4 – Coding Standards. ...
Key Practice #5 – Sustainable Pace. ...
Key Practice #6 – Test Driven Development (TDD)

Practicing XP: Thinking

- ✧ XP doesn‟t require experts. But, it does require a habit of mindfulness.
- ✧ We discuss five practices to help mindful developers excel
- ✧ Pair programming
  - Doubles the brainpower available during coding, and gives one person in each pair the opportunity to think about strategic, long-term issues
- ✧ Energized work
  - Acknowledges that developers do their best, most productive work when they‟re energized and motivated

Practicing XP: Thinking

- ✧ An informative workspace
  - Gives the whole team more opportunities to notice what‟s working well and what isn‟t.
- ✧ Root-cause analysis
  - A useful tool for identifying the underlying causes of your problems
- ✧ Retrospectives
- ✧ (an opportunity for agile development teams to reflect on past work together and identify ways to improve)
  - Provide a way to analyze and improve the entire development process

Pair Programming

Pair programming is a software development technique in which two programmers work together at one workstation.

Pair Programming

- Pair programming involves programmers working in pairs, developing code together.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from improving the system code

Pair Programming – contd.

- In pair programming, programmers sit together at the same computer to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.

FORMATIVE WORK

1. Establishment of a classroom culture that encourages interaction and the use of assessment tools.
2. Establishment of learning goals, and tracking of individual student progress toward those goals.
3. Use of varied instruction methods to meet diverse student needs.
4. Use of varied approaches to assessing student understanding.
5. Feedback on student performance and adaptation of instruction to meet identified needs.
6. Active involvement of students in the learning process.

Collaboration: Trust, Sit together.

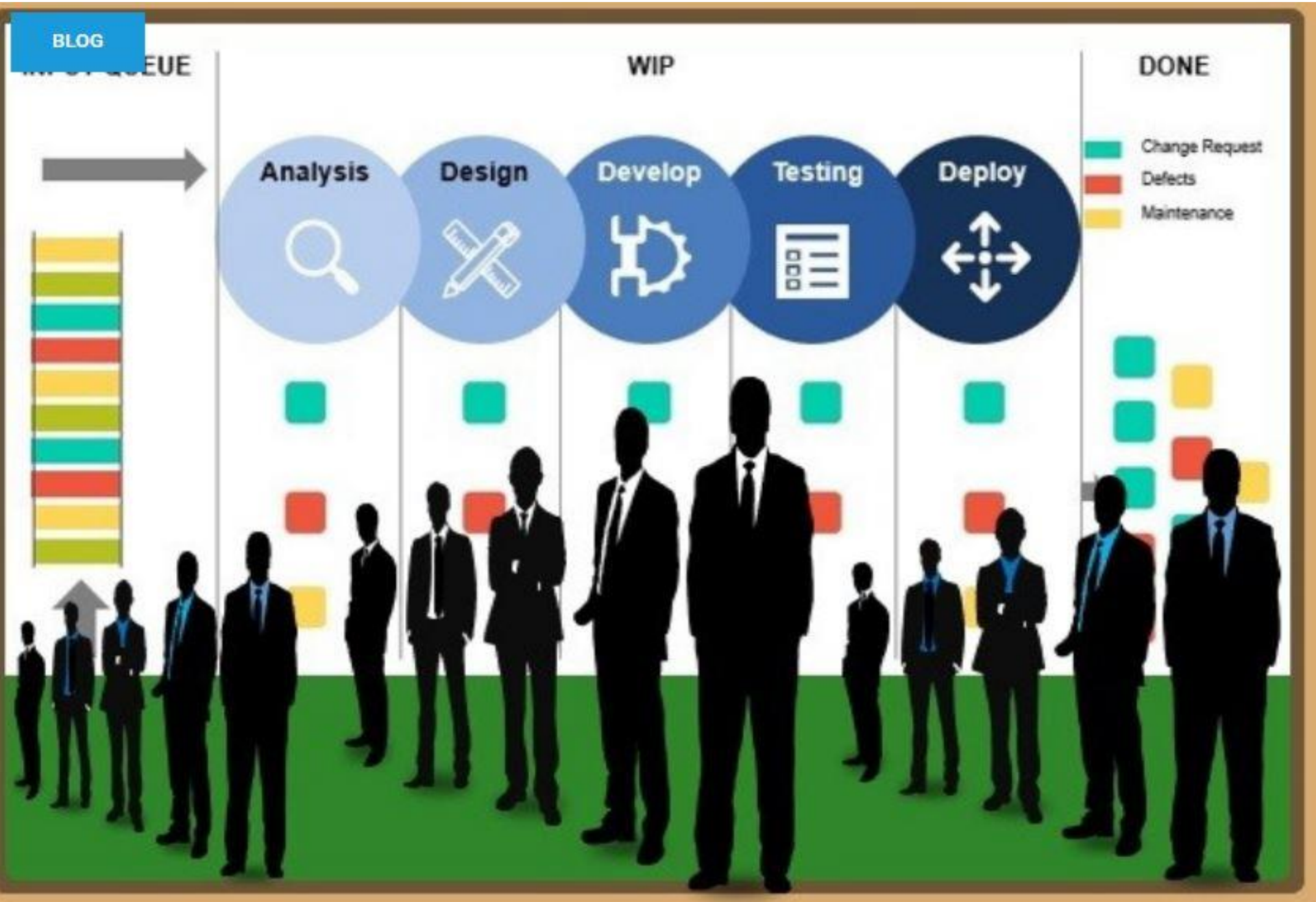
1. Agile collaboration helps connect the right people from different groups to work together on tasks,
2. boosting cross-team collaboration and productivity.
3. Breaking complex projects down into sprints creates a better understanding of the timeline for the project and the tasks involved in meeting it.


Try new things
& improve continuously
Build trust
(self) reflection
Collaborative learning
Shared understanding

stand up meeting

1. The daily stand-up is a short, daily meeting to discuss progress and identify blockers.
2. The reason it's called a “stand-up” is because if attendees participate while standing, the meeting should be kept short.
3. For software teams, a stand-up is like a sports team's huddle

stand up meeting



7 Coding Practices for Agile Software Development

1. Automate Testing.
2. Focus on readability.
3. Use third-party tools.
4. Back up your code daily.
5. Use low-code development.
6. Standardize headers for different modules.
7. Implement peer reviews.

Thinking about a new program

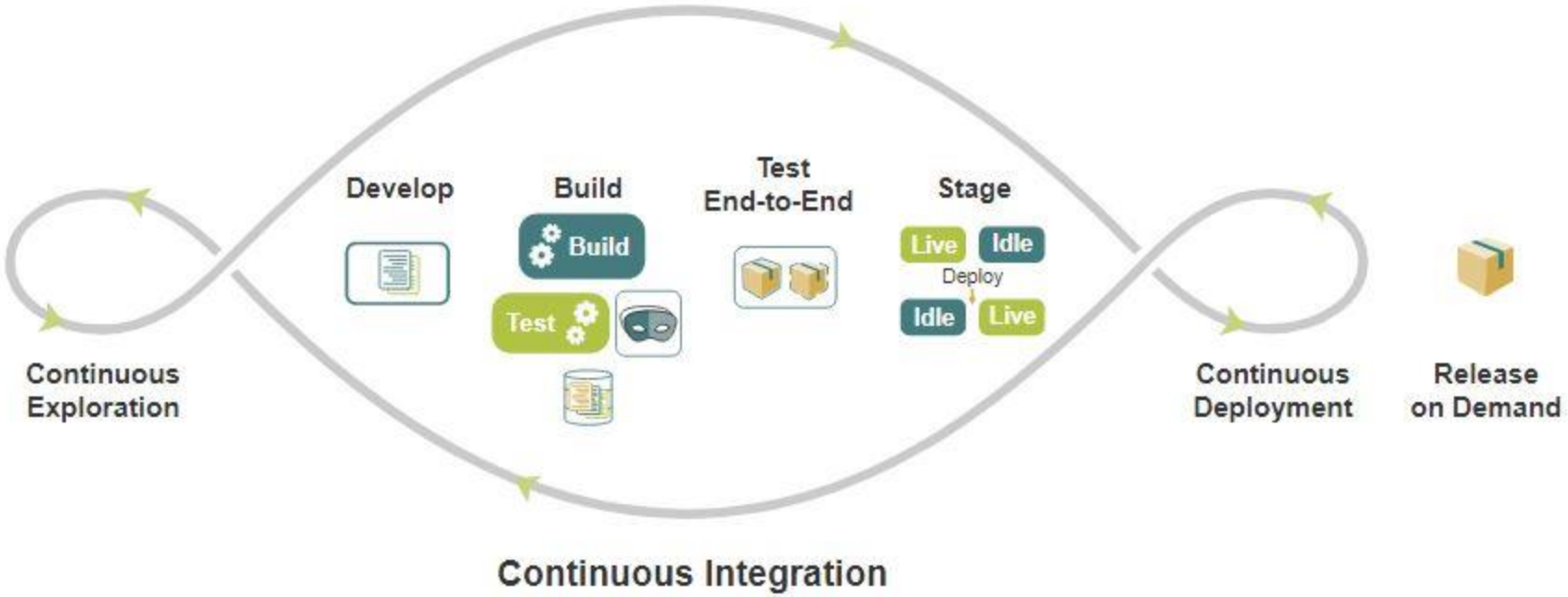
Writing a new program

Ten minute build in agile

1. The 10-Minute Build is the gold standard for getting what, in Agile/Scrum, we call Fast Feedback.
2. With the click of a button, you should be able to build your software, run comprehensive automated tests, and deploy to a testing environment.
3. This matters. It allows the team to “fail fast”

Continuous integration

1. Develop describes the practices necessary to implement stories code and components to version control
2. Build describes the techniques needed to create deployable
3. Test end-to-end describes the practices necessary to validate the solution
4. Stage describes the steps required to host and validate solutions in a



Collective code ownership

1. Collective code ownership, as the name suggests, is the explicit convention that “every” team member is not only allowed,
2. but in fact has a positive duty, to make changes to “any” code file as necessary: either to complete a development task, to repair a defect, or even to improve the code's overall structure.

What is documentation in agile?

1. Agile documentation is a way of creating and maintaining documentation for a project that is based on the principles of agile software development.
2. In agile development, the focus is on delivering working software quickly and iteratively, with an emphasis on flexibility and collaboration.

Root-Cause Analysis

- We prevent mistakes by fixing our process.
- When I hear about a serious mistake on my project, my natural reaction is to get angry or frustrated. I want to blame someone for screwing up.
- Unfortunately, this response ignores the reality of Murphy‟s Law.
  - If something can go wrong, it will.
  - People are, well, people. Everybody makes mistakes. I certainly do.

Root-Cause Analysis – Contd.

- Aggressively laying blame might cause people to hide their mistakes, or to try to pin them on others, but this dysfunctional behavior won‟t actually prevent mistakes.
- Try this:
  - Everybody is doing the best job they can given their abilities and knowledge.
  - Rather than blaming people, I blame the process.
  - What is it about the way we work that allowed this mistake to happen?
  - How can we change the way we work so that it‟s harder for something to go wrong?

Root-Cause Analysis – Contd.

- How to Find the Root Cause Problem:
- When we start working on a new task, we spend a lot of time getting the code into a working state.
  - Why? Because the build is often broken in source control.
  - Why? Because people check in code without running their tests.
  - It‟s easy to stop here and say, “Aha! We found the problem.
  - People need to run their tests before checking in.” That is a correct answer, as running tests before check-in is part of continuous integration.
  - But it‟s also already part of the process. People know they should run the tests, they just aren‟t doing it.
- Dig deeper.

Root-Cause Analysis – Contd.

- ✧ Why don‟t they run tests before checking in?
  - Because sometimes the tests take longer to run than people have available.
- ✧ Why do the tests take so long?
  - Because tests spend a lot of time in database setup and teardown.
- ✧ Why?
  - Because our design makes it difficult to test business logic without touching the database.

Customer Involvement

- The **role of the customer in the testing process** is to help develop acceptance tests for the stories that **are to be implemented in the next release of the system.**
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

Coding Standards

- ✧ Development practices
- ✧ Tools, key bindings, and IDE
- ✧ File and directory layout
- ✧ The most important thing you will learn is how to disagree.
- ✧ Build conventions
- ✧ Error handling and assertions
- ✧ Approach to events and logging
- ✧ Design conventions
  - Such as how to deal with null references

Iteration Demo

- We keep it real.
- An **XP team produces working software every week**, starting with the very first week.
- It takes a lot of discipline to keep that pace.
- **Programmers need discipline to keep the code clean so they can continue to make progress.**
- **Customers need** discipline to fully understand and communicate one set of features before starting another.
- **Testers need discipline to work on software that changes daily**

Iteration Demo – Contd.

- **Calmly describe problems** and how you handled them.
- At the end of the demo, ask your executive sponsor two key questions:
  - **Is our work to date satisfactory**?
  - **May we continue?**
- These questions help keep the project on track and remind your sponsor to speak up if he"s unhappy.
- **You should be communicating well enough with your sponsor** that his answers are never a surprise.

GcuRetro

Iteration