Extreme Programming

Lecture 7

Recap

- Activities in XP
- XP Values
- XP Practices

Extreme Programming - Rules

- The rules provide a common reference for everyone on the team so that they remind everyone of what and how they need to do when things go smoothly and also when they are not going well.
- The basic rules of planning game can be categorized into the following areas –
- Planning
- Managing
- Designing
- Coding
- Testing

Planning

Planning is done during release planning and iteration planning. The rules are same for both.

In Release planning,

- Business and the team are the players.
- Story cards are used.
- User stories are written by the customer on story cards.
- User stories are written by the customer on story cards.
- Business is decided on the priority of the functionality for implementation.
- Estimates are given by the team based on the story cards.
- Short (frequent small) releases are to be planned
- Release schedule is to be created with mutual agreement.
- The next release is to be divided into iterations.

In Iteration planning

- The team members are the players.
- Task cards are used.
- For each story selected for the iteration, the task cards are produced.
- The team members have to estimate the tasks based on the task cards.
- Each team member is assigned with task cards.
- Each team member has to then re-estimate based on his or her assignment, for
  - Accepting the work.
  - Taking responsibility of the completion of the work.
  - Getting feedback about the actual time taken.
  - Improving the estimates.
  - Respecting those estimates.

Managing

- The team is given a dedicated open workspace.
- Each workstation is to be arranged such that two developers can sit side by side and easily slide the keyboard and mouse.
- Sustainable pace is to be set (40-hour week and no overtime for more than one week) and managed.
- Enforce the rules of the planning game.
- Fixing any extreme programming practice when it breaks.
- Ensure Communication among the Team.
- Discourage Communication that is −
  - not helpful
  - not at the right time
  - done in great detail
- Make people move around.
- Measure the actual times and convey to team periodically so that each team member will know the performance as against prediction. This ensures that the team member improves in estimating.
- Make food available to the team as and when required.

Designing

The rules of designing are –

- Choose a metaphor for the system and evolve it as development progresses.
- Keep the design simple.
- No functionality is added early.
- Put as much architecture in place now as you need to meet your current needs, and trust that you can put more in later
- Refactor whenever and wherever possible.

Coding

The rules of coding are −

- The business (who represents the customer) should always be available.
- Developers should write all the code in accordance with rules emphasizing communication through the code.
- Pair programming should be ensured.
- Coding standards are to be followed.
- All code must have unit tests.
- Write a unit test first, before the code is written for each piece of the system so that it is much easier and faster to create your code. The combined time it takes to create a unit test and create some code to make it pass is about the same as just coding it up straight away. It eases regression testing.

Coding cont….

- When you are coding you should be wearing only one of the following four hats –
  - Adding new functionality, but only changing the implementation.
  - Adding new functionality, but only changing the interface.
  - Refactoring code, but only changing the implementation.
  - Refactoring code, but only changing the interface .
- A dedicated integration workstation is provided for the entire team.
- Only one pair integrates code at a time and ensures all the tests are passed.
- Integration should be done often.
- Collective ownership should be used.

Testing

The rules of testing are –

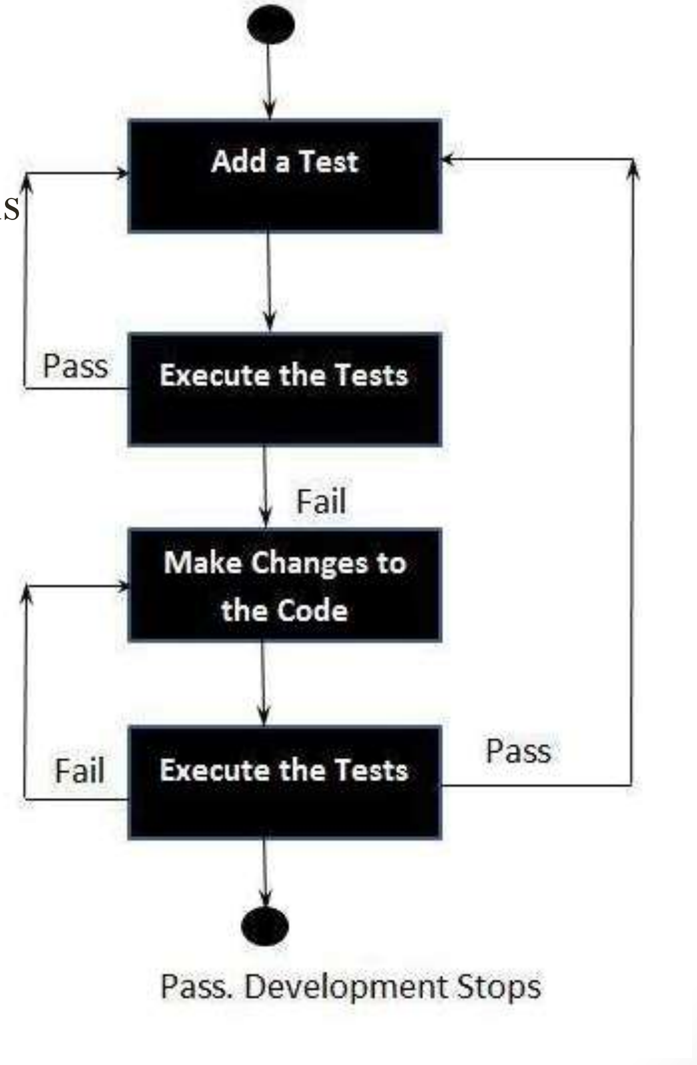
- All codes must pass all unit tests before it is released.
- Tests are to be written when a defect is found.
- Acceptance tests are to be run often.

Test-Driven Development (TDD)

- Test-driven development starts with developing test for each one of the features. The test might fail as the tests are developed even before the development. Development team then develops and refactors the code to pass the test.
- Test-driven development is related to the test-first programming evolved as part of extreme programming concepts.

Test-Driven Development Process

- Add a Test
- Run all tests and see if the new one fails
- Write some code
- Run tests and Refactor code
- Repeat



Context of Testing:

- Valid inputs
- Invalid inputs
- Errors, exceptions, and events
- Boundary conditions
- Everything that might break

Benefits ofTDD:

- Much less debug time
- Code proven to meet requirements
- Tests become Safety Net
- Near zero defects
- Shorter development cycles

Misconceptions about TDD

Misconception	Clarification
TDD is all about testing and test automation.	TDD is a development methodology using Test-First approach.
TDD does not involve any design.	TDD includes critical analysis and design based on the requirements. The design emerges during development.
TDD is only at Unit level.	TDD can be used at the integration and system levels.
TDD cannot be used by traditional testing projects.	TDD became popular with Extreme Programming and is being used in other Agile methodologies. However, it can be used in traditional testing projects as well.
TDD is a tool.	TDD is a development methodology, and after every new Unit Test passes, it is added to the Automation Test Suite as all the tests need to be run whenever a new code is added or existing code is modified and also after every refactoring. Thus, Test Automation Tools supporting TDD facilitate this process.
TDD means handing Acceptance tests to the developers.	TDD does not mean handing Acceptance Tests to the developers.

Acceptance TDD

Acceptance Test Driven Development (ATDD) defines Acceptance Criteria and Acceptance Tests during the creation of User Stories, early in development. ATDD focuses on the communication and common understanding among the customers, developers and the testers.

The Key practices in ATDD are as follows –

- Discuss real-world scenarios to build a shared understanding of the domain.
- Use those scenarios to arrive at acceptance criteria.
- Automate Acceptance tests.
- Focus the development on those tests.
- Use the tests as a live specification to facilitate change.

The benefits of using ATDD are as follows –

- Requirements are unambiguous and without functional gaps.
- Others understand the special cases that the developers foresee.
- The Acceptance tests guide the development.