CSCC01 – Software Engineering

Agile Development

What is Agile Development?

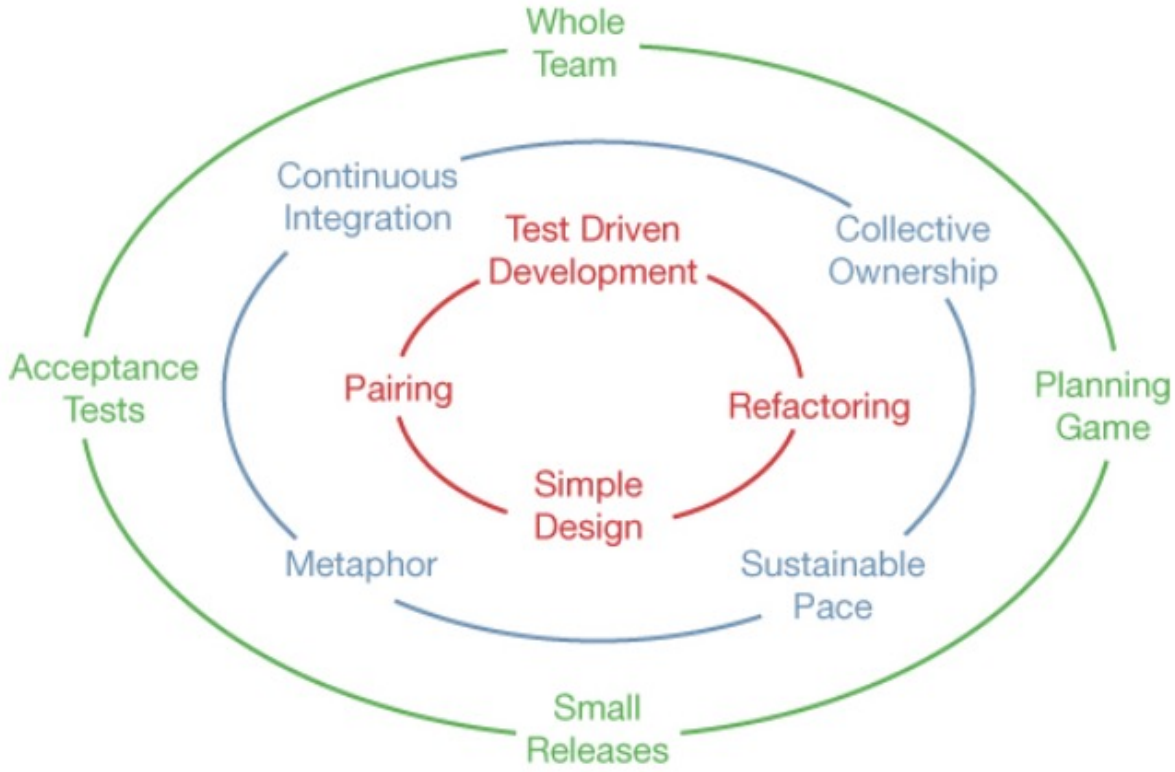
- Feedback-driven approach
  - A project is subdivided into iterations
  - The output of each iteration is measured and used to continuously evaluate the schedule
- Features are implemented in the order of business value so that the most valuable things are implemented first
- Quality is kept as high as possible

The Agile Manifesto

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

Circle of Life

- Most Agile processes are either a subset or a variation of Extreme Programming (XP)
- The *Circle of Life* is a diagram describing the practices of XP
- Three rings:
  - Outer (business practices)
  - Middle (team practices)
  - Inner (technical practices)



Business Practices

- A user story is an abbreviated description of a feature of the system, told from the point of view of a user. For example:
  - *As the driver of a car, in order to increase my velocity, I will press my foot harder on the accelerator pedal.*
- Stories must start out with limited details because a lot of them are going to be modified, split, merged, or even discarded.
  - *The details are recorded later as acceptance tests*
- The effort associated with a user story can be estimated through story points. There are several schemes to do that (e.g. *Planning Poker*)
- Story points should be roughly linear

Business Practices

- Stories should follow the *INVEST* guidelines
  - *Independent*
  - *Negotiable*
  - *Valuable*
  - *Estimable*
  - *Small*
  - *Testable*
- Tests will be written by QA, will be automated, and will be used to determine whether a story is complete.

Business Practices

Planning the iteration

- The iteration begins with the *Iteration Planning Meeting* (IPM)
- The whole team attends the IPM
- The stakeholders choose the stories that will be implemented by the programmers and testers during the iteration. To do this, they need to know how many story points the programmers think they can complete. This number is called the *velocity* (the initial value is a guess).
- Stories are prioritized based on a *Return on Investment* (ROI) calculation. For example, stories that are valuable but cheap will be done right away.

Managing the iteration

- The goal of each iteration is to produce data by getting stories done.
- The team should focus on stories rather than tasks within stories.
- As soon as the planning meeting ends, the programmers should choose the stories for which they will each individually be responsible.

Business Practices

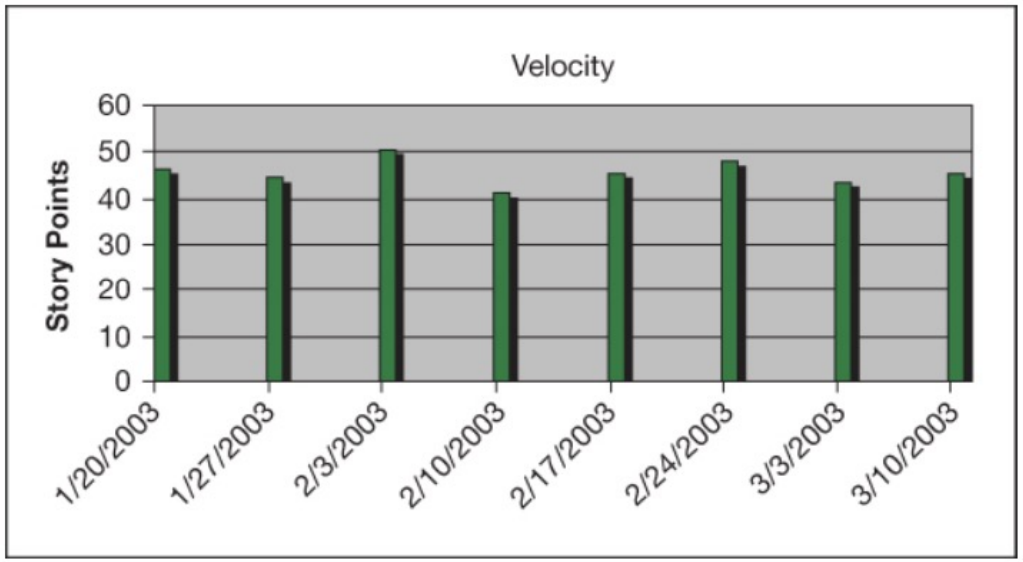
- The midpoint check
  - At the midpoint of the iteration, the point total of the completed stories should add up to *velocity*/2
  - If the actual total is less/greater than *velocity*/2, the stakeholders remove/add enough stories from the plan to adjust the remaining points
- QA and acceptance tests
  - If QA has not already begun to write the automated acceptance tests, they should start as soon as the IPM ends
  - Developers and QA should be communicating intensely about these tests
- The iteration ends with a brief demonstration of the new stories to the stakeholders showing that all the acceptance tests run – including the previous ones.

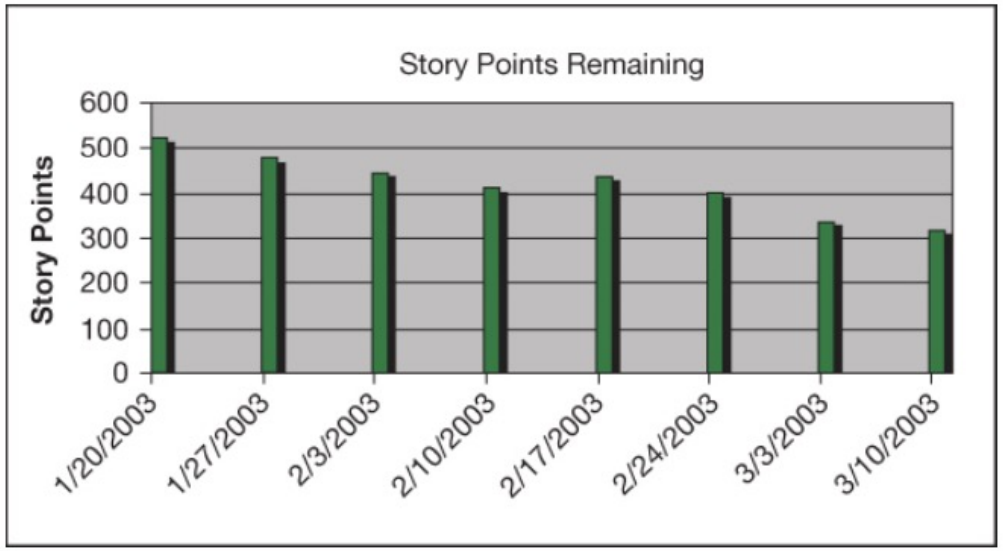
Business Practices

- The last act of the iteration is to update the velocity and burn-down charts.
- Only the points for stories that have passed their acceptance tests are recorded on these charts.
- The burn-down slope predicts the date for the next major milestone.
- The velocity slope tells us how well the team is being managed.

Business Practices

- Examples of velocity and burn-down charts:





Business Practices

- The practice of *Small Releases* suggests that a development team should release their software as often as possible
- *Whole Team* conveys the notion that a software development team is composed of many different functions, including programmers, testers, and managers, who all work together toward the same goal.

Team Practices - Metaphor

- To communicate effectively, the team requires a a model of the problem domain, which is described by a vocabulary that everyone agrees on (programmers, QA, managers, customers, users, etc)
- Such model is called a *Data Dictionary*, which is a simple representation of the data manipulated by the application and the processes that manipulates that data
- It is a thread of consistency that interconnects the entire project during every phase of its lifecycle

Team Practices – Sustainable Pace

- It is the practice that keeps a development team from burning their resources too quickly and running out of steam before the finish line.
- Working overtime should be extremely rare, and the cost of that overtime will likely be greater than the time saved on the schedule.
- Sufficient sleep is crucial in the life of programmers as it affects their productivity.

Team Practices – Collective Ownership

- No one owns the code in an Agile project. Any member of the team can check out and improve any module in the project at any time.
- *Collective Ownership* drastically improves the ability of the team to communicate and make decisions.
- *Collective Ownership* does not mean that you cannot specialize.

Team Practices – Continuous Integration

- This practice often goes hand-in-hand with a version control system
- Developers should merge their changes with the main line frequently with all unit and acceptance tests passing
- Continuous integration relies heavily on automating build and test processes

Technical Practices – Test Driven Development (TDD)

- Every required behaviour is entered twice: once as a test, and then again as production code that makes the test pass
- TDD can be described with three simple rules:
  1) Do not write any production code until you have first written a test that fails due to the lack of that code.
  2) Do not write more of a test than is sufficient to fail—and failing to compile counts as a failure.
  3) Do not write more production code than is sufficient to pass the currently failing test.

Technical Practices – Refactoring

- Refactoring is the practice of improving the structure of the code without altering the behaviour, as defined by the tests.
- The process of refactoring is woven intrinsically into the three rules of TDD in what is known as the Red/Green/Refactor cycle:
  1. First, we create a test that fails.
  2. Then we make the test pass.
  3. Then we clean up the code.
  4. Return to step 1.

Technical Practices – Simple Design

- *Simple Design* is the practice of writing only the code that is required with a structure that keeps it simplest, smallest, and most expressive.
- One of the goals of Refactoring.
- Rules of *Simple Design:*
  1) Pass all the tests
  2) Reveal the intent
  3) Remove duplication (for example, by using design patterns)
  4) Decrease elements (i.e. classes, variables, etc)

Technical Practices - Pairing

- Pairing is the act of two people working together on a single programming problem, which promotes knowledge sharing.
- It should be optional and intermittent.
- Pairs are not scheduled. They form and break up according to the programmers' preferences.
- Stories are not assigned to pairs. Individual programmers, and not pairs, are responsible for completing stories.
- Pairing is a form of code review, but with a significant advantage.
- Sometimes three or more programmers will decide to work together on a given problem. This is known as *mob programming*.

Exercise

Match each of the following practices with the relevant Agile Manifesto goal(s)

- ❑ Planning game
- ❑ Metaphor
- ❑ Acceptance tests
- ❑ Small releases
- ❑ Sustainable pace
- ❑ Whole team
- ❑ TDD
- ❑ Refactoring
- ❑ Collective ownership
- ❑ Pairing
- ❑ Simple Design
- ❑ Continuous Integration