**Sit Together**

- Develop in an open space big enough for everyone.
- Have small, private spaces nearby.
**Whole Team**

• 1st class cross-functional team

• Tipping Points [Malcolm Gladwell]
  - 12: # of people who can comfortably interact in a day
  - 150: above this you no long recognize the faces of everyone on the team

---

**Informative Workspace**
**Energized Work**

- Work only as many hours as you can be productive and only as many hours you can sustain.
  - Tired developers make more mistakes, which slows you down more in the long run (remove value from product).
  - If you mess with people’s personal lives (by taking it over), in the long run the project will pay the consequences.

**Pair Programming**

- Two software engineers work on one task at one computer
- One engineer, the driver, has control of the keyboard and mouse and creates the implementation
- The other engineer, the navigator, watches the driver’s implementation to identify defects and participates in on-demand brainstorming
- The roles of driver and observer are periodically rotated between the two software engineers
**Stories**

- Customer-visible functionality

---

**Weekly cycle**

- Highest priority stories in “time boxed” weekly increments
- Caveat: see Slack practice
Quarterly Cycle

- Timeboxed

- As small as possible, but still delivering **business value**
  - No releases to ’implement the database’

- Get customer feedback early and often

Slack

- In every iteration, plan some lower-priority tasks that can be dropped if you get behind – builds trust if you don’t miss the “important stuff.”

Ten-Minute Build

- Automatically build the **entire** system and run all tests in 10 minutes
- Feedback, feedback!
**Continuous Integration**

- Pair writes up unit test cases and code for a task (part of a user story)
- Pair unit tests code to 100%
- Pair integrates new code
- Pair runs ALL unit test cases to 100%
- Should happen once or twice a day.

**Test-first Programming**

- **Test-Driven Development (TDD)**
  - Write tests before code
  - Tests are automated
  - Often use xUnit framework
  - Must run at 100% before proceeding
- **Acceptance Testing**
  - Written with the customer
  - Acts as “contract”
  - Measure of progress
Incremental Design

- No Big Design Up Front (BDUF)

- Knowledge-based design – the most effective design is in light of experience

- “Do The Simplest Thing That Could Possibly Work”
- “You Aren’t Gonna Need It” (YAGNI)

- Refactoring: Improve the design of existing code without changing functionality
  - Relies on testing practice to ensure nothing breaks in the process of refactoring

XP Practices: Corollary

Corollary Practices

- **Real Customer Involvement.** Customer available on site to clarify stories and to make critical business decisions.

- **Incremental Deployment.** Gradually deploy functionality. Big deployment is high risk and can have high human and economic costs.

- **Team Continuity.** Keep effective teams together.

- **Shrinking Team.** As a team grows in capability, keep the workload constant but gradually reduce the size (e.g. with attrition).

Corollary Practices (cont’d)

- **Root-Cause Analysis.** (1) write failing automatic system test; (2) write failing automatic unit test(s); (3) get each to pass; (4) examine how defect was created and not caught

- **Shared Code.** Anyone on the team can improve any part of the system at any time. [prereq: pair programming, continuous integration; test-first programming]

- **Code & Tests.** Maintain only the code and tests as permanent artifacts. Rely on social mechanisms to keep alive the important history of the project.

- **Single Code Base.** Have only one code stream.
Corollary Practices (cont’d)

- **Daily Deployment.** Put new code into production every night.

- **Negotiated Scope Contract.** Fix time, cost, and quality but call for on-going negotiation of precise scope.

- **Pay-per-use.** Charge for every time the system is used.