**Extreme Programming (XP) — Lecture Notes & Practical Guide**

**1) What XP Is and Why It Works**

* **Definition**: A set of values, principles, and practices to deliver **high-value, high-quality software fast** by turning proven “best practices” up to the max—and combining them.
* **Not Windows XP**: The “XP” shorthand predates Microsoft’s OS naming.
* **Methodology?** Yes—**lightweight, low-ceremony, high-discipline**.
* **Origins**: Kent Beck (with Ward Cunningham), rooted in early OO/Smalltalk practice.
* **Core advantage**: **Visibility & accountability**—concrete commitments, concrete progress (deployable software), clear explanations of plan changes.

**2) The 12 Core XP Practices (what to do & how to do it)**

1. **Planning Game**
   * Business writes **User Stories** (on small cards).
   * Dev estimates effort & team **velocity**.
   * Business **prioritizes** and schedules releases accordingly.
2. **Small Releases**
   * Start with the **smallest useful** feature set; release **early & often**.
3. **System Metaphor**
   * A shared **organizing metaphor/naming scheme** to align understanding.
4. **Simple Design**
   * Build the **simplest design that works today**; expect change tomorrow.
5. **Continuous Testing**
   * **Test-first** always.
   * **Unit tests**: written by devs, automated, class/subsystem scope.
   * **Acceptance/functional tests**: defined by the customer; ideally automated; a story is **done** when these pass.
6. **Refactoring**
   * Continuously remove duplication and improve design, **safely** under the test net.
7. **Pair Programming**
   * All production code by **two devs at one machine** (driver & navigator); code is reviewed as it’s written.
8. **Collective Code Ownership**
   * Anyone can change any code at any time; standards + tests guard quality.
9. **Continuous Integration**
   * Integrate **at least daily**; tests must be **100% green** before and after.
10. **40-Hour Work Week**

* Sustainable pace; repeated overtime signals a **process problem**.

1. **On-site Customer**

* Continuous access to a real user (or a proxy like a PM).

1. **Coding Standards**

* Uniform style so code reads the same regardless of author.

**3) Simplicity in XP (how to judge “the simplest thing”)**

* **XP’s four criteria** (code + tests together):
  1. Communicates everything needed **clearly** (all tests run; intent is obvious).
  2. **No duplication** (unless removing it violates #1).
  3. **Minimum number of classes** (consistent with #1–#2).
  4. **Minimum number of methods** (consistent with #1–#3).
* Key slogans:
  1. **YAGNI** — *You Aren’t Gonna Need It.* Don’t build generality you don’t need now.
  2. **DTSTTCPW** — *Do The Simplest Thing That Could Possibly Work.* Get it running first.
  3. **OAOO** — *Once And Only Once.* Say what’s needed once—no more, no less.

**4) What an XP Project Looks Like (cadence & teamwork)**

* **Workspace**: Team sits together (or remote equivalent); shared table/boards/tools; daily stand-ups; calm, **sustainable** environment.
* **Iteration length**: Fixed **1–3 weeks**.
* **Iteration flow**
  + **Start (e.g., Monday)**: Planning with the customer → choose stories → break into **engineering tasks** → individual sign-ups and estimates (no one signs up for more than they finished last iteration).
  + **During**: Pair program; **TDD** loop (red → green → refactor); customer supplies/updates acceptance tests; **CI** on every change.
  + **End (e.g., Friday)**: Deliver a **working system**; implemented features are complete and **bug-free**; short demo + acceptance.
  + **Release**: Nearly a **non-event**—pick a good iteration build and ship.
* **Team size**: Smooth up to **~12 devs**; up to **~24** gets hard; larger efforts split into multiple XP teams with inter-team “customer” relationships.

**5) Applicability & Practical Notes**

* **Best fit**: Evolving business apps where feedback and change are frequent; value is discovered iteratively.
* **Platform/low-level software**: XP still helps, but strong backward-compat constraints may require **heavier upfront design** for stable kernels, while using XP on higher layers.
* **Remote XP**: Fixed pairing slots, good screen-share tools, strict CI gates (tests + format + static checks), daily short syncs, visible boards/metrics.

**6) XP Quick Cheat Sheet**

* Start from **User Stories** → write **Acceptance Tests** → plan the iteration.
* For each task: **RED (failing test) → GREEN (minimum code) → REFACTOR**.
* **Pair program**; rotate pairs regularly.
* **Anyone** may change **any** code—but **all tests must pass**.
* **Integrate at least daily**; no green = no merge.
* Keep asking: **YAGNI? DTSTTCPW? OAOO?**
* **Definition of Done** for a story:
  + Story + executable acceptance tests
  + Unit tests on critical paths
  + CI green; zero blocking defects
  + Code meets standards; important design cleaned up via refactor
  + **Deployable/demo-able** artifact