Software Process — Practical Core

A lightweight, repeatable way to turn ideas into working software through short, testable increments.

1. Process Discipline

Plan Small

Define scope, constraints (time/cost), and a short roadmap (2–6 week horizon).

Work in Increments

Slice features into thin, end-to-end verticals; ship frequently.

Make Risk Visible

Keep a top-5 risk list; attack high risk early (spikes, prototypes).

Define Done

Code + tests pass + review + integrated + documented where needed.

Control Change

Accept changes, but trade for time/scope; keep a simple change log.

Useful artifacts: Vision (1 page), Feature list/backlog, Milestones, Acceptance criteria, Risk list.

2. Dependability & Performance

Make it work well and keep it working. Qualities to design, build, and test against:

Availability

Service is ready when users need it. Track uptime, MTTR.

Reliability

Correct, predictable behavior. Use invariants, assertions, and tests.

Security

Least privilege, input validation, safe defaults; patch regularly.

Efficiency/ Performance

Measure before tuning; watch p95 latency, throughput, memory.

Evolvability

Code that can change—clear boundaries, small modules, refactoring budget.

Tactics that help:

- Defensive coding (validate inputs, fail fast), idempotent operations, retries with backoff.
- Timeouts and circuit breakers, caching where it helps, backpressure to avoid overload.
- Monitoring and logs tied to user journeys (not just servers).

3. Requirements

Know what to build before you build it. What “good” looks like (checklist):

- **Clear and testable:** someone can write a passing/failing test for it.
- **Valuable and feasible:** user benefit is explicit; effort roughly fits the timebox.
- **Independent and small:** can be delivered in days, not months.
- **Prioritized:** Must/Should/Could (MoSCoW) with reasons.
- **Traceable:** each feature maps to a goal and has acceptance criteria.

Capture methods (use a mix):

- User stories + acceptance criteria, simple use cases, quick prototypes, example data.
- Don’t forget non-functional requirements: latency, throughput, error rates, security, accessibility, observability, portability, maintainability.

4. Reuse

Don’t reinvent the wheel.

What to Reuse

- Libraries/frameworks
- Services/APIs
- Components/templates
- Prior code and docs

Adopt with a Small Due-Diligence Pass

- **Fit:** does it solve this problem with minimal glue?
- **Health:** updates, issues, stars, docs, examples, release cadence.
- **License & security:** compatible license; vulnerability scan; pin versions.
- **Cost of ownership:** size, transitive deps, lock-in, migration path.

Benefits

- Faster delivery
- More time for testing and polish
- Usually higher quality—provided the fit is good.

Testing & Flow

Minimal loop that ties it all together:

- Plan
- Implement
- Review
- Test
  (Unit + a few high-value integration/acceptance tests)
- Integrate
- Release
- Observe
- Learn
- Next Slice

Requirements — Gathering, Analysis, Prototyping, and Use Cases

A) Gathering Methods

- Surveys
- Interviews
- Focus groups
- Observations
- Use case analysis

B) Use Case Analysis — Writing Steps

1. Describe a real usage scenario.
2. Write from the user's point of view, without jargon or technical detail.
3. Present as a clear, ordered list that emphasizes the end result and the steps taken.

C) Requirements Analysis

Functional Requirements

What the system actually does (functions of the software).

Non-functional Requirements

Description of the software's qualities (characteristics/properties of the system).

D) Prototyping — Quick Definition

A short exercise that builds a (usually non-functional) prototype representing the visual layout and how features are linked/flow.

E) Interviews or Surveys → User Stories

User stories are a simpler, less-detailed way to express requirements.

> **Example user story:** "As a sales staff member, I want to add new sales to the system so that I have a record of all transactions."

F) Example Use Case — “Record Sale in the System”

Role: Sales Staff

Main Success Scenario

- ### Initiate "Add Sale"
  The user initiates “Add Sale”.
- ### Request Item Details
  The software requests the item name and quantity.
- ### Enter & Confirm
  The user enters the name and quantity, then confirms.
- ### Store Entry
  The software stores the new entry.

Extension E1 — Quantity < 1

⊗ **Condition:** The user enters a quantity less than 1 (e.g., 0 or negative), or leaves it blank/non-numeric.

System action:

- Display: “Quantity must be a positive integer (≥ 1).”
- Highlight the Quantity field; keep the Item Name as entered.
- Do not create or modify any record.

Flow: Return to Step 2 (the software requests item name and quantity again).

Preconditions

(Must be true before starting)

- The user is signed in as Sales Staff.

Postconditions

(Conditions at end of process)

- **Success:** the entry is stored in the system.
- **Failure:** no change is saved (if you want to keep the original note, label this as “failure”).