**Engineering Tasks + CRC Cards — Teaching Notes & How-To**

**Part I — Engineering Task**

**1) What is an Engineering Task?**

* **Definition**: Once a **User Story** is selected for the current iteration, the team **brainstorms the concrete work** needed to deliver it. Each unit of work is an **Engineering Task**.
* **Purpose**: Turn a business-facing story into **developer-executable pieces** that can be **estimated (Ideal Programming Time)**, **owned**, **tracked**, and **finished**.
* **Helpful aid**: Use **CRC Cards** first to reveal candidate classes, responsibilities, and collaborations; then map those into tasks.

**2) When and who creates them?**

* **When**: During **Iteration Planning** (the story is in scope).
* **Who**: The **whole team** (dev, test, UX, ops, etc.). The **assignee uses their own estimate** in the iteration plan (different engineers may estimate differently).
* **Recording**: Give each task a **short name** on the board/plan. If you write a paper task card, **the card itself isn’t a tracked artifact**—the **task** is.

**3) Size & Estimation**

* **Preferred size**: **≤ 1 ideal day** (uninterrupted doing time).
* **Upper guardrail**: **> 3 days → split or spike** (you may allow up to 5, but expect trouble).
* **Common split patterns**
  + **By TDD/implementation layers**: acceptance test scaffold → domain logic → persistence/integration → UI.
  + **By collaboration boundary**: external service, auth, audit, error handling as separate tasks.
  + **By risk/unknowns**: time-boxed **Spike** to explore, producing a conclusion + minimal example.

**4) Definition of Done (per task)**

* Code implemented **with unit tests**; acceptance test data/steps updated if relevant.
* **Review/pairing** completed (if practiced).
* **CI green** (incl. linters/formatters).
* Supporting **scripts/docs/monitoring** updated.
* **No blocking defects**, mergeable to main.

**5) Example — From User Story to Tasks (RJ30)**

**Story (summary)**: RJ30 records overtime hours; Mon–Sat paid **time-and-a-half**, Sunday **double time**. Record both **hours and dollars** into **regular/premium/double-premium** “bins”; notify if >30 overtime hours.

| **Engineering Task** | **What it does** | **Estimate (ideal days)** |
| --- | --- | --- |
| Define RJ30 input record | Create RJ30 schema; route to **Hours Raw Input** bin | 0.5 |
| Create target bins | Add **premium/double-premium** hours & money bins; roll money bins into gross pay | 0.25 |
| **OvertimeEvaluationStation** | Classify each RJ30 into regular/premium/double and place **hours** into bins | 0.5 |
| **OvertimePayStation** | Apply pay rate and place **dollars** into premium/double money bins | (included in 0.5 above) |
| Over-threshold notice | Notify when an employee reports **> 30 overtime hours** | 0.25 |

If the “>30 hours notification” is **outside** the story’s acceptance scope, promote it to a **separate story**; otherwise keep it as a task.

**6) Task Card Template (suggested fields)**

* **ID / Name** (verb–object, ≤ 50 chars)
* **Linked Story** (Story ID)
* **Brief description** (deliverable/result)
* **DoD** (how *this task* is proven done)
* **Estimate (IPT)**
* **Dependencies/Risks**
* **Owner (Pair)**
* **Status** (To Do / Doing / Review / CI Green / Done)

**Example**

* Name: Implement OvertimeEvaluationStation classification & binning
* DoD: unit tests for rules green; errors audited; CI green
* Estimate: 0.5 day; Deps: bins defined; calendar/pay-rate services available

**7) Planning & Tracking**

* **Board**: Task tickets flow with WIP limits; optionally show remaining ideal hours.
* **Iteration plan**: Use the **assignee’s estimate**; task sums ≈ story effort.
* **Metrics**: Burndown / Cumulative Flow to spot bottlenecks (reviews/CI/integration).
* **Changes**: Small scope → add a task; big/out-of-scope → create a **new story** or defer.

**8) Pitfalls & Fixes**

* Tasks too big → **split or spike**.
* Implementation checklist with **no verifiable output** → add DoD-bound deliverables.
* Treating tasks as “mini stories” → commitments/priority stay at the **story** level.
* “Standard” team estimate used for all → use **assignee’s** estimate for the plan.
* Invisible work (test data/CI/migration) → **make it a task**.

**9) Minimal workflow**

Select story → brainstorm tasks → name + DoD + estimate → size check (≤1 day; >3 split) → assign to plan → board flow + CI green → done & measure

**Part II — CRC Cards (Class–Responsibility–Collaboration)**

**1) What is a CRC Card?**

* **Definition**: An index card that captures a class’s **Responsibilities** and **Collaborators**—hence **C**lass–**R**esponsibility–**C**ollaboration.
* **Purpose**: Early-phase, **moveable** design aid to discover candidate classes and clarify **who does what** and **who works with whom**. You **act out** the scenario with cards, then map outcomes into **Engineering Tasks**.
* **Why cards**: Minimal writing, **maximal movement**—layout and proximity convey relationships; great for brainstorming and seeing the big picture quickly. Paper + table works best; snap a **photo** to archive.

**2) How to write CRC Cards**

**A. Prep**

* 3×5 index cards (or stickies), bold markers, a **large table**.
* Up to **5 colors** to highlight key object types; start with ~**15 cards** to avoid clutter.
* Guideline: **Write as little as possible**; add a keyword only to resolve confusion.

**B. Card template (three essentials per card)**

* **Class** — domain-centric noun (e.g., *Project*, *OvertimePolicy*).
* **Responsibilities (3–5)** — concise verb–object bullets of **what** the class should do (not *how*).
* **Collaborators** — other classes this one must interact with to fulfill responsibilities.

**C. Workshop steps (20–40 minutes per scenario)**

1. **Read the scenario**: choose one concrete user-story flow.
2. **Noun/verb scan**: highlight **nouns** (candidate classes) and **verbs** (candidate responsibilities/messages).
3. **Draft cards**: one per candidate class; start with 2–3 core responsibilities.
4. **Play the cards** (walk the timeline):
   * “Who owns this step?” → add a **responsibility** to that class.
   * “Who helps?” → add a **collaborator**, place cards close together.
   * Too many/too broad responsibilities → **split the card**; overlaps → **merge or move**.
5. **Add non-happy paths**: map **errors/timeouts/permissions/audit** into responsibilities/collaborations.
6. **Refine naming**: unify domain language; keep **3–5 responsibilities** per card.
7. **Archive**: photo or tool entry; **the card isn’t a long-term spec**—the captured outcome is.

**3) Writing tips & a tiny example**

**Do & Don’t**

* ✅ Domain-centric names; each responsibility should be **verifiable/testable**.
* ✅ Collaborations should imply a **message/data exchange** (what I ask/what I get).
* ❌ Don’t write **HOW** (DB tables/framework minutiae).
* ❌ Don’t flood the table; expand gradually.

**Mini example (2 cards)**

* **Class**: OvertimeEvaluationStation
  + **Responsibilities**: parse RJ30; classify overtime; write hours to TimeBin; audit errors
  + **Collaborators**: OvertimePolicy, TimeBin, AuditLog
* **Class**: OvertimePolicy
  + **Responsibilities**: return multiplier (1.5×/2×) based on calendar/rules
  + **Collaborators**: CalendarService, CompanyRules

From these, you naturally derive tasks like **classification logic + tests**, **bin write API**, **audit logging**, etc.

**4) Logistics tips**

* **Table first** (fastest for moving pieces); **wall with stickies** also works—photograph to persist.
* 3×5 cards encourage **brevity**; if it doesn’t fit, it’s probably **two cards**.
* Keep color use **limited** (≤5) and card count modest at start (~15).

**5) One-line takeaway**

**CRC = “what a class does + who it collaborates with.”** Use minimal words and physical movement to **act out the scenario**, then convert responsibilities & collaborations into **small, clear Engineering Tasks**.