# Excel Template Guides for Agile Project Management

## Table of Contents

# Release & Sprint Plan

## 1) Release Plan — Increment Plan

**Purpose.** Use the Increment Plan to **group sprints into releases** (e.g., "Increment 1" contains Sprints 1–3). For very small projects with only one sprint, this section can be omitted.

**When to fill it.**

- At the start of the course (initial plan): draft increments, dates, and goals.
- Update only when a release boundary changes (e.g., scope moves from Increment 1 → 2).

### Columns — what to write

- **Incr.** Sequential release number (`1`, `2`, `3`, … ). Each increment aggregates one or more sprints.
- **Start Calendar start date** of the increment (`YYYY-MM-DD`). Usually the start date of the first sprint included.
- **Days Total calendar days** covered by the increment (often a multiple of the sprint length, e.g., 28 or 42).
- **End Calendar end date** (inclusive). Should equal `Start + Days - 1`. Keep aligned with the sprints inside the increment.
- **Size** Planned size of the increment. Prefer **sum of Story Points** for all sprints in the increment. (You may use "number of stories," but keep the meaning consistent throughout the file.)
- **Status** Recommended values:
  - `Planned` – dates/goals defined, not started
  - `In Progress` – any sprint within the increment has started
  - `Released` – increment delivered
  - `Canceled` – no longer pursued

- **Release Date** Intended **external release** date (if applicable). Leave blank for internal milestones.

- **Goal** One–two lines summarizing the value delivered by the increment (outcome-focused, not task-focused). *Example:* "Minimal data→model pipeline + basic UI stub."

> **Tip:** The **Increment** field in the Sprint Plan must reference one of these numbers (`1`, `2`, …).

---

## 2) Sprint Plan

**Purpose.** Define the **timeline of sprints** (start/end, size), their **status**, the **goal** for each sprint, and which **Increment** they belong to.

**When to fill it.**

- During **Sprint Planning** for all planned sprints.
- Update **Status** and (optionally) **Size** at the end of each sprint.

### Columns — what to write

- **Sprint** Sequential sprint number (`1`, `2`, `3`, …).

- **Start Calendar start date** of the sprint (`YYYY-MM-DD`). Use your course calendar (avoid exam weeks/holidays).

- **Days Length of the sprint** in calendar days (commonly `7` or `14`). Keep consistent unless you intentionally change it.

- **End Calendar end date** (inclusive). Should equal `Start + Days - 1`.

- **Size Sum of Story Points** for the **selected User Stories** in this sprint. *(Optional at sprint end: record delivered SP, e.g., `8 (delivered 5)`).*

- **Status** Recommended values:

  - `Planned` – selected but not started
  - `In Progress` – active sprint
  - `Done` – completed and reviewed
  - `Canceled`

- **Release Date** If the sprint is tied to a **demo/release**, set that date; otherwise leave blank.

- **Goal** The **Sprint Goal** in one line—the primary outcome you commit to deliver. It should be **testable** ("achieved / not achieved" at Review). *Examples from your template:* "Planning" · "Specification and prototype development" · "Usability test, documentation, prototype corrections"

- **Increment** The **Increment number** (from the Release Plan) this sprint contributes to (e.g., `1` or `2`).

### Special rows in the template

- **Unplanned** rows Use these if an **unplanned sprint** or **emergency mini-iteration** is created. Fill them like normal sprints and set **Status** accordingly. Leave blank if unused.

- **Unallocated stories** Number of **User Stories not yet placed** in any sprint. Update after each Sprint Planning.

---

## Conventions & Best Practices

1. **Date formats** → use `YYYY-MM-DD`. Format cells as **Date** (not Text).
2. **Size consistency** → prefer **Story Points** for both Increments and Sprints; don't mix with "number of stories."
3. **Status discipline** → update at least: Sprint start (`In Progress`), Sprint Review (`Done`), Increment release (`Released`).
4. **Link to GitHub**
   - Sprint **Size** = sum of SP of the **Selected** stories in your **Product Backlog Project**.
   - Sprint **Goal** = what you can **demo** from the Kanban at Review.
   - Keep Excel synchronized with boards after **Planning** and **Review**.
5. **Holidays / gaps** → either keep **Days** fixed and reduce **Size**, or adjust **Days** and note the reason.

---

## Examples

### Increment Plan (example)

| Incr. | Start | Days | End | Size | Status | Release Date | Goal |
|---|---|---|---|---|---|---|---|
| 1 | 2006-06-13 | 42 | 2006-07-24 | 19 | Planned | | Planning; specification & prototype development |
| 2 | 2006-07-25 | 42 | 2006-09-04 | 0 | Planned | | Usability tests; documentation; corrections |

### Sprint Plan (example)

| Sprint | Start | Days | End | Size | Status | Release Date | Goal | Increment |
|---|---|---|---|---|---|---|---|---|
| 1 | 2006-06-13 | 14 | 2006-06-26 | 8 | Planned | | Planning | 1 |
| 2 | 2006-06-27 | 14 | 2006-07-10 | 3 | Planned | | Specification and prototype development | 1 |
| 3 | 2006-07-11 | 14 | 2006-07-24 | 8 | Planned | | Specification and prototype development | 1 |

| Sprint | Start | Days | End | Size | Status | Release Date | Goal | Increment |
|---|---|---|---|---|---|---|---|---|
| 4 | 2006-07-25 | 14 | 2006-08-07 | 0 | Planned | | Specification and prototype development | 2 |
| 5 | 2006-08-08 | 14 | 2006-08-21 | 0 | Planned | | Specification and prototype development, docs | 2 |
| 6 | 2006-08-22 | 14 | 2006-09-04 | 0 | Planned | | Usability test, documentation, prototype fixes | 2 |

> Replace dates and sizes with your current calendar and Story Point planning.

---

## Minimal workflow (step-by-step)

1. **Before Sprint 1** Define Increments (`Incr.`, `Start`, `Days`, `End`, draft `Goal`). Add planned sprints with `Start`, `Days`, compute `End`.
2. **During Sprint Planning** Select stories in GitHub → sum SP → set **Size**; write **Sprint Goal**; set **Status** = `Planned`; map to **Increment**.
3. **During the Sprint** Keep Kanban updated daily. Excel only changes if dates/scope shift.
4. **Sprint Review** Optionally record **Delivered SP**; set **Status** = `Done`.
5. **Release (if any)** When all sprints in an increment are `Done`, mark the increment `Released` and set **Release Date**.

---

# Product Backlog

## Purpose

The Product Backlog lists **User Stories** (and optionally Epics as placeholders) with their **status, size (Story Points), target sprint, priority**, and **comments**. It is the single source of truth for **what** the team plans to deliver and **when**.

> Keep this sheet in sync with your GitHub **Product Backlog Project**. Each row typically maps to one **User Story** (issue).

---

## Columns — what to write

- **Story ID**
  A unique, sequential identifier in the spreadsheet (e.g., `1`, `2`, `3`).

*Tip:* Also include the GitHub issue number in **Comments** (e.g., `GH#42`) or encode it in the **Story name** as `US7:` to ease traceability.

- **Story name**
  The **title** of the User Story in the "As a , I want so that " style.
  *Example:* `US3 — As a Data Scientist, I want reproducible experiments so that we can build better versions`.

- **Status**
  Current state of the story. Use the same vocabulary as your boards:

  - `Planned` – selected for a future sprint but not started
  - `Ongoing` – currently being implemented (matches Kanban "In Progress")
  - `Done` – development & review completed (meets Definition of Done)
  - `Removed` – de-scoped or canceled (keep here to preserve decision history)
    *(Optional additional statuses: `Blocked`, `Needs Info` — if you also track them in GitHub labels/fields.)*

- **Size**
  **Story Points** using Fibonacci (`1, 2, 3, 5, 8, 13`). Estimate during refinement/planning.

  - Represents **relative effort/complexity**, not hours.
  - Keep consistent across the backlog so velocity trends make sense.

- **Sprint**
  The **target sprint number** where this story is planned to be delivered (e.g., `1`, `2`, `3`).

  - Leave **blank** if **unallocated** (not yet assigned to a sprint).
  - Must match a sprint listed in the **Sprint Plan** sheet.

- **Priority**
  Business/teaching priority. Use one consistent scale across the course:

  - Recommended: `P0` (highest), `P1`, `P2`.
  - Alternatively: `1` (highest) to `5` (lowest).
    Explain your scale in a note if you don't use the recommended one.

- **Comments**
  Free-text notes for **acceptance criteria pointers, links, or risks**. Keep the story definition concise here; for larger definitions use GitHub issue body and link to it.
  *Examples:*

  - `GH#42 — AC in issue body; depends on E2 (DVC setup)`
  - `Link: https://github.com/org/repo/issues/42`
  - `Risk: dataset changes mid-sprint`

---

## How to use the sheet effectively

1. **Add/Refine Stories**

- Add a row per User Story with **Story name**, **Size** (SP), and **Priority**.
- Keep **Comments** short with links to GitHub, acceptance criteria, and dependencies.

2. **Allocate to Sprints**
   - During Sprint Planning, set **Sprint** to the chosen sprint number.
   - Ensure the **sum of SP** for that sprint matches your **team capacity/velocity**.
3. **Track Progress**
   - Update **Status**: `Planned` → `Ongoing` → `Done`.
   - If a story is dropped, set `Removed` (don’t delete rows—keep the history).
4. **Sync with GitHub**
   - Create/match a **GitHub issue** for each story.
   - Ensure GitHub Project fields (SP, Priority, Status) mirror the spreadsheet values.
   - Use **Issue Links** in **Comments** for traceability.
5. **Unallocated stories**
   - Rows with empty **Sprint** are the **pipeline** for future sprints.
   - Sort the backlog by **Priority** (and optionally **Size**) to decide the next sprint content.

---

## Example

| Story ID | Story name | Status | Size | Sprint | Priority | Comments |
|---:|---|---|---:|---:|---|---|
| 1 | This is a sample story | Done | 3 | 1 | P1 | GH#101 |
| 2 | This is another sample story | Ongoing | 5 | 1 | P0 | Add details/links to external docs |
| 3 | This is a third sample story | Planned | 3 | 2 | P2 | — |
| 4 | This is a fourth sample story | Planned | 8 | 3 | P1 | — |
| 5 | This is an unallocated sample story | Planned | 13 | | P2 | Consider splitting before scheduling |
| 6 | This is a removed story | Removed | 5 | | | Decision logged on 2025-11-04 |

---

## Quality checks (self-audit)

- Every row maps to a **real GitHub issue** (link present in **Comments**).
- **Status** matches the Kanban column at end-of-day.
- **Sprint** is set only when actually scheduled.
- **Size (SP)** uses the Fibonacci scale and is consistent across stories.
- **Priority** follows the chosen scale (`P0/P1/P2` or `1..5`).

- **Removed** stories are kept (not deleted) with a short reason/date in **Comments**.

---

## 1) What the charts mean

- **Velocity and Remaining Work (top chart)**
  - **Tops of the bars** = amount of functionality implemented **by the beginning** of each sprint (cumulative delivered SP).
  - **Bottoms of the bars** = **current total size** of the project at the beginning of each sprint (planned scope after any change). If scope grows, the bar **bottom moves down**.
  - **Red line** = current **planned scope** (projected size).
- **Development Velocity (bottom chart)**
  Shows **planned vs realized** velocity per sprint and reference lines for averages (realized average, average of last 8, worst 3 in last 8).

---

## 2) Header inputs (left panel)

Fill these manually or via formulas linked to your backlog/velocity data.

- **Original planned size**
  Total initial scope of the project (in **Story Points**). Usually the sum of all SP in the initial Product Backlog.
- **Count trend from last**
  Number of last sprints to compute the **trend** lines (e.g., 3 means use the last 3 sprints for trend).

### Velocity (points per sprint)

All are expressed in **SP/sprint**:

- **Original estimate** — the expected velocity at the start (capacity-based guess).
- **Last 3 sprints** — average realized velocity of the most recent 3 sprints.
- **Realized total average** — average realized velocity over **all completed** sprints.
- **Average last 8** — realized average over the last **8** sprints (or fewer if not available).
- **Avg. worst 3 in last 8** — average of the **three lowest** realized velocities within the last 8 sprints (useful for conservative planning).
- **Trend** — velocity based on the “Count trend from last” window (e.g., regression or simple average over the last *N* sprints).

### Predictions — *Completion at the end of sprint…*

These fields contain **predicted sprint number of completion** (i.e., “we’ll likely finish around sprint X”) under different assumptions:

- **Original estimate – Min/Avg/Max** — completion sprint assuming optimistic/average/pessimistic velocity scenarios you used initially.
- **Last 3 sprints** — using the average of last 3 realized velocities.
- **Realized average** — using realized total average.

- **Average last 8** — using realized average of the last 8.
- **Avg. worst 3 in last 8** — conservative estimate.
- **Trend** — using the trend window.
- **Realized ± St. Dev** — completion sprint if you add or subtract one standard deviation from the realized average velocity.

> **How to compute**: `Predicted sprints to finish = ceil( Remaining Work / Velocity )`. Then **add the current sprint index** if you want an absolute sprint number. "Remaining Work" at the time of the prediction equals **Current Total Size – Cumulative Realized Work**.

---

## 3) Sprint table (bottom-left)

Fill or compute one row **per sprint**.

| Column | What to write |
|---|---|
| **Sprint** | Sprint number (`1, 2, 3, …`). |
| **Remain. Work** | Remaining Story Points at **start** of the sprint. Recommended formula: `Current Total Size (start of sprint) − Cumulative Realized Work (up to previous sprint)`. |
| **Planned Work** | Story Points planned for this sprint (sum of SP of stories in Sprint Plan / GitHub "Selected"). |
| **Realized Work** | Story Points **delivered** in the sprint (meets DoD; counted at **Review**). |
| **Current Total Size** | Planned scope at the **start** of the sprint, **after** applying any scope changes (added/removed SP). This is the value used by the chart's bar bottoms. |

### Update cadence

- **Before each sprint**: Set `Planned Work` and `Current Total Size`; compute `Remain. Work`.
- **After each sprint review**: Enter `Realized Work`. Recompute realized averages, velocity metrics, and predictions.

### Scope changes

If backlog SP changes (new stories, re-estimates):

- Adjust **Current Total Size** for the **next** sprint start.
- This will automatically reflect as a change in bar bottoms (scope up/down).

---

## 4) Recommended formulas (pseudo)

Let `i` be the sprint index (1-based).

- `CumulativeRealized[i] = sum(RealizedWork[1..i])`

- `RemainingWork[i] = CurrentTotalSize[i] − CumulativeRealized[i−1]` (use `0` when `i=1`)
- `VelocityLast3[i] = average(RealizedWork[max(1,i−2)..i])`
- `VelocityAvg[i] = average(RealizedWork[1..i])`
- `VelocityLast8[i] = average(RealizedWork[max(1,i−7)..i])`
- `VelocityWorst3in8[i] = average(three smallest values of RealizedWork in window max(1,i−7)..i)`
- `StdDevRealized[i] = stdev(RealizedWork[1..i])`
- **Prediction** (generic): `SprintsToFinish = ceil( RemainingWork[i] / VelocityChoice[i] )`

---

## 5) Quality checks

- Planned vs realized velocity is **close** over time; large, repeated gaps imply planning issues.
- Remaining Work **decreases** monotonically unless scope changes are recorded.
- Scope changes are documented in your backlog and mirrored in **Current Total Size**.
- Predictions converge as the project progresses (volatility reduces).
- Zero or blank `Realized Work` appears only for **future** or **not yet reviewed** sprints.

---

# Sprint Backlog

## 1) Header inputs

- **Sprint implementation days**
  The **number of working days** in this sprint (e.g., `5` for a one-week sprint or `10` for two weeks). Do **not** include weekends/holidays if you won't work those days.

- **Trend calculated based on last**
  The **window (in days)** to compute the *Current Trend* line (e.g., `5` uses the last five daily points). Use the most recent data to smooth noise.

---

## 2) Task table — what to write

Each row represents **one Task** (a GitHub issue). Use one named owner per task.

| Column | What to write |
|---|---|
| **Task name** | A concise, action-oriented title prefixed with the story: `US<id> — <task>` (e.g., `US1 — Define data schema & validators`). |
| **Story ID** | The **Story ID** from the Product Backlog sheet the task belongs to. |
| **Responsible** | **Exactly one** owner (GitHub handle or full name). |
| **Status** | `Planned`, `Ongoing`, or `Done` (align with Kanban columns). |
| **Est.** | **Estimated hours** for this task (≤ 8h recommended for Sprint 1). |

| Column | What to write |
|---|---|
| **Remaining on implementation day … (1..N)** | For **each working day**, write the **remaining hours** for this task **at end of day**. Values must be **non-increasing** and end at `0` when the task is truly done. |

> **Tip:** If a task is blocked, keep the remaining hours unchanged and add a note in your daily Excel (or in the GitHub issue). Split tasks that exceed 8h.

---

## 3) Totals & mini-burndown

- **Totals → Effort** = sum of **Est.** across all rows (planned capacity).
- **Daily Remaining (1..N)** columns also show total remaining hours across all tasks. These totals drive the **chart**:
  - **Blue bars (Daily Progress)** = total **remaining hours** at end of each day.
  - **Grey diagonal (Ideal Progress)** = linear line from **Totals Effort** on day 1 to **0** on day *N*.
  - **Blue line (Current Trend)** = regression/average trend over the last *k* days (set by "Trend calculated based on last").

**Burndown rule of thumb.** Bars should **decrease daily** and cross the **ideal** near the end. If bars flatten or rise, re-plan or unblock tasks quickly.

---

## 4) Daily routine (students)

1. **Before the sprint starts**
   - Enter all planned tasks with **Task name**, **Story ID**, **Responsible**, **Status = Planned**, and **Est.** hours.
   - Set **Sprint implementation days** (e.g., 5).
2. **End of each day**
   - Update **Status** and enter **Remaining hours** for that day per task.
   - A finished task must have **Remaining = 0** and **Status = Done**.
   - Check that the **total** remaining decreases; if not, discuss in stand-up.
3. **During the sprint**
   - If a task grows, **split** it and adjust estimates; document the reason in GitHub.
   - Keep **one owner per task** to avoid accountability gaps.
4. **Sprint Review**
   - All delivered tasks are `Done` with **Remaining = 0** on the final day.
   - Snapshot the chart for your demo/report.

---

## 5) Suggested validations (optional formulas/pseudocode)

For each task i and day d:

- `Remaining[i, d] >= 0`
- `Remaining[i, d] <= Remaining[i, d-1]` (non-increasing)
- If `Status[i] == "Done"` on day d → `Remaining[i, d] == 0` and all subsequent days `= 0`
- On final day N → `sum_i Remaining[i, N] == 0` ideally

Totals:

- `TotalsEffort = sum_i Est[i]`
- `TotalRemaining[d] = sum_i Remaining[i, d]`

Trend (one simple approach):

- `CurrentTrend[d] = linear_regression(day=1..d, y=TotalRemaining[1..d])` or average slope over last *k* days.

---

## 6) Worked example

| Task name | Story ID | Responsible | Status | Est. | Day1 | Day2 | Day3 | Day4 | Day5 |
|---|---|---|---|---|---|---|---|---|---|
| Example task | 1 | Danny Dev | Done | 5 | 5 | 2 | 0 | | |
| Example task 2 | 1 | Tina Tester | Ongoing | 7 | 7 | 7 | 2 | 2 | |
| Example task 3 | 2 | Danny Dev | Ongoing | 12 | 12 | 12 | 12 | 10 | |
| <Delete these example… > | 2 | — | Planned | 9 | 9 | 9 | 9 | 9 | 9 |

Totals Effort = **33** hours; Daily Remaining totals: **33 → 30 → 23 → 21 → …**

---

## 7) Best practices

- Keep tasks small (≤ 8h) and **atomic**.
- Avoid multi-owner tasks; create separate tasks if two people collaborate.
- Update **at end of day**, not next morning.
- Use the chart in stand-ups to decide where help is needed.
- Mirror status in GitHub Kanban and link the issue in the task name or comments.

---

# Task Slips

## What each slip contains (fill these fields)

- **Story ID**
  The **Story ID** from the Product Backlog that this task belongs to (e.g., `1`, `2`).

- **Story**
  The **User Story title** (short version). Keep this consistent with your **Product Backlog** and the **GitHub issue** title of the story.

- **Task**
  The **task title** (action-oriented), ideally prefixed by the story: `US<id> — <task>`. Example: `US1 — Define data schema & validators`.

- **Responsible Person**
  Exactly **one owner** (GitHub handle or full name). Change ownership explicitly if reassigned.

- **Initial Estimate**
  **Estimated hours** for this task at the time of Sprint Planning. Keep tasks **≤ 8h** where possible. If you discover it's larger, split the task and create a new slip.

- **Work Done**
  Cumulative **hours already spent** on this task. Update **at end of day**. This should **increase** monotonically and never exceed `Initial Estimate` unless you re-estimate.

- **Work Left**
  **Remaining hours** to finish the task **as of today**. Update **at end of day**. This should **decrease** monotonically down to `0` when done.

> **Consistency rule:** `Initial Estimate ≈ Work Done + Work Left` (allowing for re-estimation during the sprint with a short note on the GitHub issue).

---

## Daily usage pattern

1. **Create slips at Sprint Planning**
   - For every task in the Sprint Backlog, create one slip with **Story ID**, **Story**, **Task**, **Responsible**, and **Initial Estimate**.
2. **End-of-day updates**
   - Update **Work Done** and **Work Left**. If a task finishes, set **Work Left = 0**.
   - If the estimate changes, record the new **Initial Estimate** and add a comment in GitHub (reason + date).
3. **Stand-ups / Reviews**
   - Use slips to quickly surface **blocked tasks** (Work Left unchanged, but Work Done increased slowly) and to coordinate help.

---

## Example

### Slip 1

- Story ID: `1` — Story: *This is a sample story*
- Task: *Example task*

- Responsible: *Danny Dev*
- Initial Estimate: **5h**
- Work Done: **3h**
- Work Left: **2h**

**Slip 2**

- Story ID: 1 — Story: *This is a sample story*
- Task: *Example task 2*
- Responsible: *Tina Tester*
- Initial Estimate: **7h**
- Work Done: **5h**
- Work Left: **2h**

**Slip 3**

- Story ID: 2 — Story: *This is another sample story*
- Task: *Example task 3*
- Responsible: *Danny Dev*
- Initial Estimate: **12h**
- Work Done: **2h**
- Work Left: **10h**

---

## Quality checks (self-audit)

- Every slip refers to a **real GitHub task issue** (link in the issue or attached as a note).
- **One owner** per task; if paired work happens, split the task.
- **Work Done** + **Work Left** tracks reality; if it diverges from **Initial Estimate**, re-estimate and note why.
- Slips are kept **in sync** with the **Sprint Backlog** row for the same task (remaining hours per day should be compatible).

---

# Sprint Retrospective

## How to fill each quadrant

### 1) WHAT WENT WELL?

Write **concrete successes** and practices to keep. Examples:

- "Daily updates were on time; burndown stayed close to ideal."
- "MLflow tracking URI set up early; reproducibility verified."
- "Clear ownership per task avoided context switching." Use bullet points, one idea per line. Include **evidence** where possible (link to issue/PR).

### 2) WHAT WENT POORLY?

List **problems, delays, or blockers** without blame. Examples:

- “DVC remote misconfigured → data push failed on day 2.”
- “Too many large tasks (>8h); carry-over increased.”
- “Reviews piled up at the end; PR waiting time 2+ days.”

### 3) WHAT NEW IDEAS DO WE HAVE?

Capture **improvements or experiments** to try next sprint. Examples:

- “Create a lint/test GitHub Action that runs on every PR.”
- “Introduce a ‘reviewer-of-the-day’ rotation.”
- “Split data-cleaning task into stages with DVC to get earlier feedback.”

### 4) WHAT ACTIONS WILL WE TAKE?

Turn ideas/problems into **SMART actions** (Specific, Measurable, Achievable, Relevant, Time-bound). For each action, record:

- **Action** – short imperative sentence (e.g., “Add CI job: dvc repro + unit tests”).
- **Owner** – exactly one person.
- **Due** – a date or “by Sprint N end.”
- **Success metric** – how we’ll know it worked (e.g., “PR mean wait time < 24h”).

> *Rule of thumb:* 2–4 actions per sprint; avoid overloading the next sprint.

---

## Facilitation flow (10–20 minutes)

1. **Set the stage (2′)** – Remind goal: *inspect & adapt*, not blame.
2. **Silent writing (3′)** – Team adds notes to the four quadrants.
3. **Cluster & discuss (5′)** – Merge duplicates; clarify with facts/links.
4. **Select actions (5′)** – Vote/decide 2–4 items → write **Actions** with **Owner/Due/Metric**.
5. **Close (1′)** – Confirm that actions are added to **Sprint N+1 Backlog** as tasks or checklist items.

---

## Ground rules

- Focus on **process and artifacts**, not people.
- Be **specific**: include links (issue/PR/board).
- Prefer **data**: velocity, review times, defect counts, rework hours.
- Keep it **short** and **actionable**; long post-mortems go in docs, not here.

---

## Example entries

**What went well?**

- Early schema validation caught 3 data issues (link GH#42).

**What went poorly?**

- Batch job failed nightly due to missing secrets; fixed day 4 (GH#55).

**What new ideas do we have?**

- Add `.env.template` and a setup script for local MLflow.

**What actions will we take?**

- *Create `setup_local_env.sh` and `.env.template`* — **Owner:** @alex — **Due:** by end of Sprint 2 — **Metric:** New dev setup < 15 minutes.

---

### After the retro

- Convert each **Action** into a **GitHub Task** in the next sprint's Kanban.
- Review the previous sprint's actions at the **start** of the next retro (done/not done, impact).

---

# Team Self-Assessment

## Structure of the matrix

- **Rows = Evaluators** (each team member fills their row).
- **Columns = To evaluate** (each team member receives ratings).
- **Diagonal cells** (row *i*, col *i*) = **self-assessment**.
- **Off-diagonal cells** (row *i*, col *j*, i≠j) = **peer assessment** from *i* to *j*.

> Keep names consistent with the team roster. If a member is absent this sprint, leave their **row and column blank** and note it below the table.

---

## Rating scale (1–5)

Choose **one integer** per cell:

1 — *Almost no contribution* this sprint
2 — *Less than expected*
3 — *According to capabilities and agreed role*
4 — *Above expectations*
5 — *Outstanding, clearly beyond agreed scope*

> Use the same scale for **self** and **peers**. When in doubt, default to **3** and justify deviations in a comment.

---

## How to fill it (two steps)

**STEP 1 — Self-assessment**
Each student rates **themselves** in their **diagonal** cell only.

**STEP 2 — Team assessment**
Each student rates **every other member** in the **off-diagonal** cells of their row.

Tips:

- Base ratings on **evidence** (GitHub issues/PRs, hours, board movement).
- Avoid reciprocity bias ("they rated me high, I rate them high").
- If someone joined mid-sprint, consider rating range **1–3** unless exceptional evidence exists.

---

## Optional comments (recommended)

Add a short note below the table:

- **Highlights** (what they did well)
- **Improvements** (one actionable suggestion)
- **Context** (availability, blockers)

Keep comments respectful and specific; link to issues/PRs when possible.

## Quality & fairness checks

- Every active member has **N ratings received** (N−1 peers + 1 self).
- Comments are **professional** and evidence-based.
- Check for **unusual patterns** (all 5s or all 1s from a single rater).
- Cross-validate with workload (hours, issues closed, PR reviews).
- Discuss major discrepancies in the **retrospective** and agree on actions.

---

## Example (7-person team)

```
          To evaluate →   Name1  Name2  Name3  Name4  Name5  Name6  Name7
Evaluator
Name1 (self)                4      3      3      3      4      3      3
Name2                         3      3      4      3      3      3      3
Name3                         3      3      4      3      3      4      3
Name4                         3      3      3      3      4      3      3
Name5                         4      3      3      3      4      3      3
Name6                         3      3      4      3      3      4      3
Name7                         3      3      3      3      3      3      4
```