MISSION HANDBOOK

Mission handbook reference manual

Use this handbook as the main reference for how a long-running mission is planned, executed, validated, and handed off without having to inspect the code first.

PDF EXPORT

Download the handbook as PDF

The same HTML source powers the website and the printable handbook. Run `npm run export:pdf` to refresh `exported/mission-handbook.pdf` with diagrams and handbook content intact.

Open current PDF

The export command reuses an existing preview or starts one automatically for the export run.

1. MISSION OVERVIEW

What this framework is for

A mission turns a broad objective into smaller, verifiable assignments. Each worker owns a bounded feature, proves what changed, and hands evidence back to the orchestrator.

Treat this handbook as the reader-facing reference manual for those roles, transitions, validators, and delivery rules.

- The handbook explains the framework in plain language.
- The companion skill repository encodes the executable rules.
- Validation is mandatory before work is considered done.

2. REFERENCE MAP

Use the handbook like a reference manual

Roles and surfaces

Read the overview, companion skill, and terminology sections when you need to know who does what.

State transitions

Use the lifecycle and rules sections when you need to decide whether to continue, retry, or escalate.

Proof of completion

Use delivery expectations and export guidance when you need to verify what a finished outcome must contain.

3. COMPANION SKILL

Mission marathon skill: the execution partner for the handbook

The companion skill is for long-running missions where work must be decomposed, validated, and handed off safely when a worker can no longer make bounded progress.

What it does

Breaks a large mission into smaller features, assigns workers, runs validators, and records outcomes.

How it is shaped

Its structure mirrors the handbook: orchestrator, workers, validators, handoffs, and evidence artifacts.

How to use it

Use the handbook to understand the model. Use the skill to apply that model to a real mission.

4. TERMINOLOGY

Core terms you will see throughout the handbook

Mission

A coordinated unit of work that may involve multiple features, workers, validators, and handoffs.

Feature

A bounded assignment with expected behavior and verification steps.

Orchestrator

The controller that assigns work, reads handoffs, and decides whether to continue, retry, split, or stop.

Worker

The agent that researches, implements, validates, and reports on one assigned feature.

Validator

The check that proves the public result matches the intended behavior.

Handoff

The evidence package that tells the next agent exactly what was done and what still needs attention.

5. LIFECYCLE

How a task moves from request to verified delivery

The lifecycle is a reference workflow. Each transition must be observable, justified, and bounded.

1. Clarify and scope

Read the mission plan, feature brief, rules, and repo state before changing anything.

2. Implement

Make a focused change in the correct repository without drifting into unrelated work.

3. Validate

Run the required checks and inspect the result that users or reviewers will actually see.

4. Handoff or escalate

If checks pass, report evidence. If the work is blocked or stale, return control instead of looping.

What triggers lifecycle changes

- **Progression:** a requirement is understood and a focused change can be made.
- **Blocking:** a missing dependency, failed baseline, or repo-boundary conflict prevents safe progress.
- **Retry:** a validator finds a fixable issue that can be addressed with a bounded follow-up change.
- **Takeover:** repeated failed retries, stale outputs, or ambiguity require orchestrator review.

6. RULES

Rules that keep long-running work useful

Anti-spin rules

Repeating the same failed action, rerunning unchanged validators, or searching without a new hypothesis all count as non-progress.

- Do not keep retrying after the same blocker appears.
- Do not claim progress without a user-visible change or new evidence.
- Escalate when the next step is unclear or no longer bounded.

Takeover rules

The orchestrator or a human should take over when a worker can no longer make safe forward progress inside the current feature.

- Take over when a required service or private dependency fails.
- Take over when validation cannot prove the claimed behavior.
- Take over when the feature needs to be split or reassigned.

These rules are part of the lifecycle. The correct move after failed evidence is escalation, not hidden looping.

7. DELIVERY EXPECTATIONS

What a finished feature must leave behind

A delivery is complete only when another person or agent can reproduce the checks and understand the outcome.

- A short summary of what was implemented.
- The exact commands that were run and what they showed.
- Any tests added or updated, with the behaviors they cover.
- Clear notes on blockers, leftovers, or discovered issues.
- A clean commit that captures the completed change.

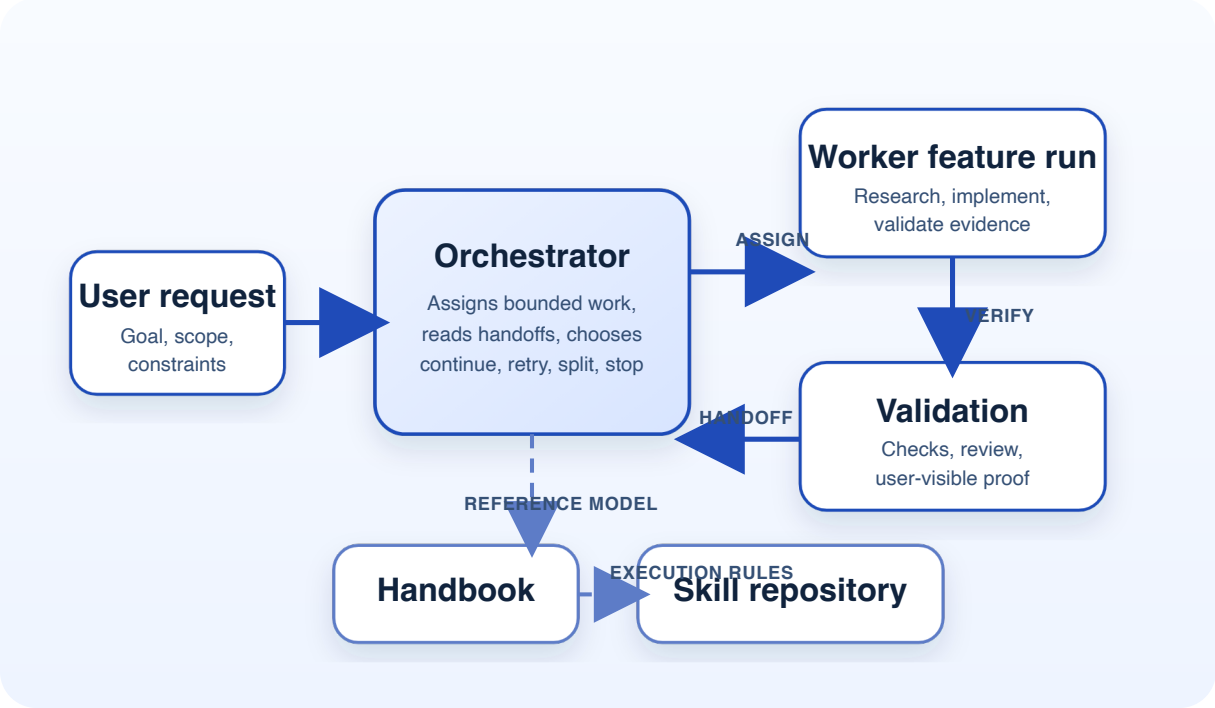
8. DIAGRAMS

Visible architecture and flow diagrams

Use these diagrams as quick-reference maps. They are embedded inline so they render on the site and stay visible in PDF export.

Architecture diagram

The orchestrator is the control center. It routes work to workers, receives validation evidence, and keeps the handbook aligned with the execution skill.



Lifecycle flow diagram

Work moves forward only when validation passes. Fixable issues allow a bounded retry; blocked or stale work returns to the orchestrator.