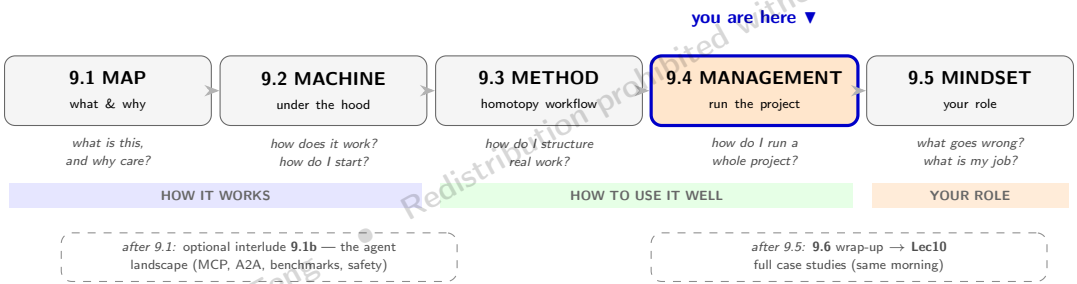
AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 9.4: Project Organization — The Researcher as Project Manager

Zhigang Feng

2026-07-06 07:05:11

The Map: Act 4 — The Management

This deck answers the fourth question: how do I run a whole project with this — not just one task?



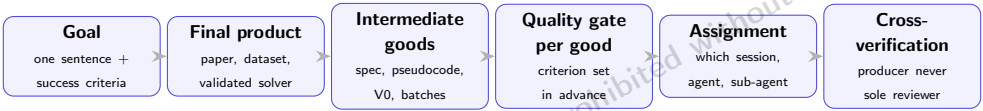
Route: the paradigm shift (researcher as project manager) → how HA-Ramsey was actually managed → the manager's instruments on disk (CLAUDE.md, skills, agents, rules, hooks) → your sub-agent RA team and cross-verification → remote/HPC execution.

The Paradigm Shift

From solo coder to project manager

The Paradigm Shift: You Are the Project Manager

When execution becomes cheap, the scarce input is management: deciding what gets built, to what standard, by whom.

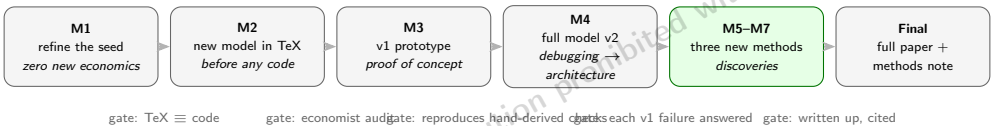


- Decompose the goal into **intermediate goods** — real artifacts (spec, validated V0, batch CSV, review report), not vague "progress"
- Decide the **quality gate** for each good *before* it is produced (T3's acceptance criteria, now for everything, not just code)
- **Assign** each good to the right producer — this session, a sub-agent, a different model tier, or yourself; then **cross-verify**: whoever produced it, someone else checks it

T3 disciplined one artifact; T4 runs the factory. Every object in this deck is one of the manager's instruments.

A Managed Project in One Slide: HA-Ramsey's Milestones

The HA-Ramsey program is the PM chain instantiated: seven milestones, each a validated, preserved intermediate good.



- Every milestone preserved as a citable artifact in the project's `Case_Study/` package — *intermediate goods, literally*; no milestone skipped, no gate waived
- The green box is the payoff of management: the discoveries (Story 1 in T1) *surfaced from gated iteration* — they were not the plan

Economics and figures: Lec10 T8. Source: HA-Ramsey case-study package.

Write the Contract First: Who Owns What

Before any code, the HA-Ramsey project wrote down the division of labor — T2's table, now with signatures.

AI owns (throughput)	**Economist owns (judgment)**
Drafting and revising the LaTeX model note	Economic correctness of every equation
Translating math into vectorized PyTorch	Which simplifications are acceptable
Penalty/boundary implementation patterns	Catching conceptual bugs in timing and probabilities
Running diagnostics and summarizing sweeps	Validation benchmarks and when a result is credible

"AI supplies throughput; the economist supplies judgment" — stated *before* any code was written. The contract is an assignment rule: which goods the AI may produce alone, and which gates only you may sign.

Quality Gates Catch What No Compiler Catches

Two conceptual bugs the economist caught in AI-drafted equations — the code ran, the losses fell, the results would have been wrong. No compiler catches these:

1. **Wrong timing:** asset transitions written with next-period consumption — but the budget constraint is a *current-period* object
2. **Wrong direction:** transition probabilities indexed by state occupied, not by *direction of movement*

When v1 then failed numerically, each failure became v2 architecture:

v1 failure	v2 structural answer
Naive sampling of a 5D state space	Adaptive sampling loops
Fragile feasible-set boundary learner	Normalized coordinates + robust geometry
Simulated paths escape the valid region	Explicit projection back + penalty term

"Debugging becomes architecture" only if someone with judgment reads the failures — the gate is where the economist earns authorship.

Project Organization

The manager's instruments: files on disk

The PM's Instruments Live on Disk

A manager who re-explains the standards every morning is not managing. Files on disk are how you delegate without re-explaining.

- Conventions live in files, not in your memory — or the agent's
- Validation steps become reproducible lab procedure
- New sessions start from the project's actual state (T2: initialization reloads it all)
- Coauthors and students can inspect what the workflow is supposed to be

Economist translation: the harness turns tacit RA practice into written lab protocol — and the protocol, not the chat, is what your name ultimately stands behind.

Six Instruments, Six Pain Points

Instrument	Management pain point it solves	Running example
`CLAUDE.md`	re-explaining the project every session	keep the MEPS weights rule or Aiyagari benchmark visible
Skill	a repeated workflow improvised differently every time	paper-review pipeline with the same quality gates
Agent	review roles need specialization and parallelism	code reviewer vs. domain reviewer
Rule	some instructions must *always* be on	every filled field carries a source URL
Verifier	output should pass named checks before it counts	rerun script, inspect diagnostics, confirm files exist
Provenance log	source discipline must survive beyond the chat	fellows dataset with field-level URLs and notes

Reason for this slide: these objects are not product trivia. Each one retires a coordination or credibility failure you would otherwise absorb personally.

Minimal Project Tree

```
my-research-project/
+-- CLAUDE.md
+-- .claude/
|   +-- skills/
|   |   +-- paper-review/
|   |       +-- SKILL.md
|   |       +-- references/
|   |           +-- referee-template.md
|   +-- agents/
|   |   +-- code-reviewer.md
|   |   +-- domain-reviewer.md
|   |   +-- verifier.md
|   +-- rules/
|   |   +-- iterative-workflow.md
|   +-- settings.json
+-- scripts/
+-- output/
+-- sources/
+-- quality_reports/
```


This is enough to explain the workflow without burying students in infrastructure. The exact research question can change; the organizational logic stays.

CLAUDE.md: The Standing Brief

`CLAUDE.md` is the main shared instruction file for the project — the brief a good manager writes once instead of repeating daily.

```
# CLAUDE.md
Project: MEPS uninsurance pilot for lecture 9
Language: Python
Goal: harmonize 2012--2014 files and produce one validated figure
Conventions: no hardcoded paths; save outputs to output/
Checks: use survey weights; save the year map; report what passed
```

Why students should care: it captures the project context you would otherwise re-explain every session.

The fellows project runs the same play with public-web material: the seed CSV, the batch file, and the schema rules are "the public-web analog of `CLAUDE.md`" (Lec10 T4) — project files holding the rules and state across sessions.

Three Memories, Three Writers

Three different persistence ideas should not be confused.

Mechanism	Who writes it	Why it exists
`CLAUDE.md`	you / team	durable project instructions and conventions
Auto memory	the tool	per-project learned patterns and corrections
Optional course log	you / course	explicit progress notes such as `progress.md`

Course recommendation: rely on `CLAUDE.md` for shared rules, treat auto memory as helpful but not authoritative, and keep explicit logs when you want students to inspect the evolution of a project.

Skills, Agents, and Rules: Three Different Jobs

Loading order matters. Rules are read before the first user message; Skills load when invoked; Agents launch with the rules and any inherited context as their system prompt.

Plain English vs. Encoded Skill

Plain English

Review this paper and tell me the top problems with exact quotes.

Use when

- Exploring
- Doing a one-off task
- Still deciding the workflow

Encoded skill

`/paper-review draft_A.pdf`

Use when

- The workflow should repeat exactly
- A student team needs one standard procedure
- Validation must be consistent across versions

Exploration starts in English. Reproducibility lives in skills.

Capstone note: Track D2 asks you to deliver exactly this — a validated skill for a recurring research task.

Skill File Structure

```
.claude/skills/paper-review/
  SKILL.md
  references/
    referee-template.md
```

`SKILL.md` stores the repeatable protocol:

- what the command is called
- what steps it should follow
- what checks must pass before it reports success

Reference files store task-specific detail that would clutter the main instruction file.

Skills Are Saved Research Playbooks

A skill is the right home for long, repeated procedures. Keep always-loaded project context short; put domain workflows in reusable skills.

Skill	Reusable workflow	Validation hook
`data-analysis`	Load, explore, regress, produce tables/figures	Output schema + script rerun
`review-paper`	Read paper, audit argument, generate referee objections	Structured review report
`weighted-survey`	Compute survey-weighted rates/means	Formula check against benchmark
`hpc-job`	Write `sbatch` script, submit, parse logs	Scheduler status + parser

Portable idea: Claude may call them `skills`; Codex may package them differently. The durable object is a saved playbook with triggers, allowed tools, steps, and acceptance tests.

Rules and Hooks: Encoding the Fellows Provenance Discipline

Guidance the agent should follow → a rule. A check that must fire mechanically → a hook. The fellows project needs both.

The rule (`.claude/rules/provenance.md`):

```
- Every filled field carries a source URL
  from an official or institutional page.
- Never infer gender, birth year, or living
  status from names, photos, or country.
- Record missing as missing; missingness
  is a result, not a failure.
```

Always-on: loaded at session start, batch after batch.

The hook (`settings.json`):

```
"PostToolUse": [{
  "matcher": "Write(data/*.csv)",
  "hooks": [{"type": "command",
    "command":
    "python checks/require_source_url.py"}]
}]
```

Mechanical: any CSV row missing `source_url` is rejected — no matter what the conversation said.

Object	Fires	Use for
Rules	during work	natural-language conventions and guardrails
Hooks	on tool events	mechanical enforcement: format, test, block, log
Settings	at session start	permissions and project-level behavior

Permissions and Hooks in `settings.json`

Two jobs: *permissions* define the blast radius; *hooks* run mechanical checks at tool boundaries.

```
{
  "permissions": {
    "allow": ["Bash(python *)", "Bash(pytest *)", "Read(*)", "Edit(*)"],
    "deny":  ["Bash(rm -rf *)", "Bash(git push --force *)",
              "Read(./.env)", "Read(./secrets/**)"]
  },
  "hooks": {
    "PostToolUse": [{"matcher": "Write|Edit|MultiEdit",
                     "hooks": [{"type": "command",
                                "command": ".claude/hooks/format-python.sh"}]}],
    "Stop":        [{"hooks": [{"type": "command",
                                "command": "pytest tests/ -q || true"}]}]
  }
}
```

Hook types worth knowing: `PreToolUse` (block risky commands), `PostToolUse` (format / lint), `Stop` (full test suite), `SubagentStop` (verify report format), `PreCompact` (save progress).

Three Lifetimes: Per-Event, Per-Session, Forever

The harness is the umbrella name for what makes a project survive across sessions.

Layer	Lifetime	Purpose
Hooks (`settings.json`)	Per-event	Mechanical enforcement: format after edits, run tests on stop, block dangerous tools
Project memory / rules	Per-session	Bootstrap context. Project instructions and path-scoped guardrails
Auto memory + progress files	Cross-session	Durable learnings and explicit handoff notes you can inspect and edit

Why this matters. Choosing the right layer is half the design: if the rule must fire every time, a hook is safer than a natural-language reminder; if it must survive a fresh session, it belongs in memory or rules, not a transient prompt.

Rule of thumb: natural-language guidance → rules and skills. Mechanical enforcement → hooks.

Your RA Team

Sub-agents, assignment, and cross-verification

Sub-Agents: Your Parallel RA Team

When the orchestrator spawns a sub-agent, the harness launches a separate agent instance with its own context window.

- Multiple sub-agents run **in parallel** — like Python's `multiprocessing`
- Each receives an agent file as its system prompt and returns a structured report
- The specialist roles are the RA team you met in the project tree:
  - `code-reviewer.md`: file hygiene, implementation quality, reproducibility
  - `domain-reviewer.md`: benchmark logic, identification, economic consistency
  - `verifier.md`: rerun the script, inspect diagnostics, confirm outputs exist

Aiyagari example. After writing `aiyagari_v1.py`, the orchestrator spawns

`code-reviewer` ‖ `domain-reviewer`

in parallel; both finish in roughly one sequential run's wall-clock time. **Review time is halved.** *Why separate roles? One prompt asking about everything does a mediocre job at each — one RA checks style, another checks economics, another reruns the table.*

Assignment and Cross-Verification

The manager's two questions for every intermediate good: who produces it, and who checks it?

Good fit for sub-agents:

- Code review (parallel: code, domain, verification)
- Lookups and web research that return short reports
- Independent unit-test generation per module

Bad fit:

- Tightly coupled work (each step depends on the previous)
- Anything you must steer interactively mid-task
- Sub-agents spawning sub-agents (depth causes drift)

Cross-verification: the producer is never the only reviewer. Blast-radius rules of thumb: plan mode before granting execution on novel pipelines; each sub-agent's checklist *spec'd*, not free-form; checkpoint validation between phases (the homotopy gates from T3).

Foreshadow for T5: parallel reviewers are parallel executionally, not epistemically — they read the same project files you wrote.

Orchestration at Research Scale

The HA-Ramsey lesson list ends with a management skill: to benefit the most, you need four things.

1. **A good seed** — T3's criteria: works, documented, modular, points somewhere
2. **A clear vision** — the final product and the milestone ladder to it
3. **Good judgment** — the quality gates only an economist can sign
4. **Orchestration** — run *different sessions and agents on different layers*, then integrate

What orchestration looked like in practice:

- Parallel sessions for formalization (TeX), implementation (PyTorch), auditing (code-vs-doc), sweeps, and write-up — all feeding one preserved package
- 32+ configuration sweeps run and summarized by agents; the economist read summaries and decided at the milestones — the job is *integration*: the PM chain on a real paper

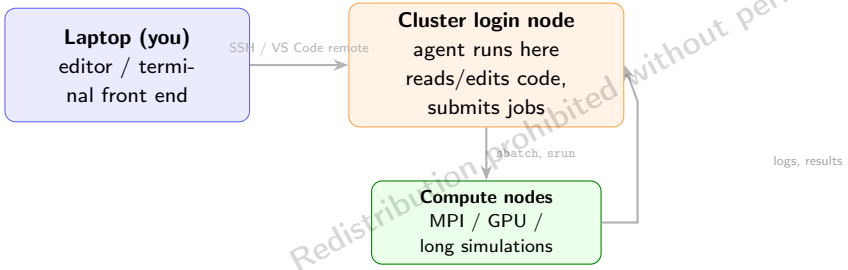
Source: HA-Ramsey case-study package (Lec10 T8).

Remote and HPC

When the laptop is no longer the right host

Remote/HPC: Move the Agent to the Compute

For heavy workloads the agent should run where the data, modules, and GPUs already are.



- With a local agent, cluster files must be copied back and forth — easy for toy code, fragile for research
- With a remote agent, project files stay in the remote worktree; your laptop is only the front end

Rule: if the data, modules, GPUs, and job logs live on the cluster, install the agent on the cluster.

Login Node vs. Compute Node: The Hard Boundary

What the agent can do on a login node:

- Read/edit code, specs, context files, and job scripts
- Run tiny smoke tests: import checks, unit tests, `-dry-run`, small-grid V0
- Submit jobs, monitor logs, parse errors, and revise scripts

What must go through the scheduler:

- MPI jobs, GPU training, large simulations, long calibrations
- Anything that consumes many cores, much memory, or more than a few minutes

Research rule: the agent may orchestrate HPC work; it should not turn the login node into a workstation.

Bridge: Management Assumes Trust — Verify It

You now delegate through files and a sub-agent team. Delegation works only if you can audit what was actually done — and know when to distrust the polish.

- Act 4 gave you the management system: goods, gates, assignment, cross-verification, and the instruments that encode them
- The final question: *"what can go wrong, and what is my job?"*
- Act 5 answers with three moves: **trace** what the agent did, catalog the **failure modes** the harness cannot prevent, and confront the **Context Dilemma** — the same context that improves the answer also pulls it toward you

Arc: T1 Framing → T2 Under the Hood → T3 Homotopy → **T4 Project Org** *→ T5 Mistakes/Context → T6 Wrap.*