

Prompt-First Delivery

Ship faster by orchestrating AI agents — without losing quality.



Define outcome + acceptance



Dispatch parallel workstreams



Integrate, test, and ship safely



Executive summary

- Default workflow shifts from **code-first** to **specify** → **dispatch** → **review** → **ship**.
- Even if a model is slower per task, **parallel capacity** dominates throughput.
- Architecture, interfaces, and quality gates matter more; micro-style matters less.
- Prompting, decomposition, and orchestration become a **core production capability**.

Key idea

Humans are sequential; agents are parallel. Your job becomes orchestration + review bandwidth.

We don't code first anymore.

“Prompting becomes the primary interface; review + integration becomes the control system.”

- Coding is still necessary — but it's no longer the default first step.
- Quality is a function of **goal clarity, constraints, examples, and guardrails.**

Reframe: “How do we implement?” → “What outcome must hold, and what constraints apply?”

The shift: code-first → prompt-first

Train the reflex: specify intent before typing code.

OLD DEFAULT

Code-first

- Start by writing code
- Serial execution (one thread)
- Debug while building
- Quality = style + correctness

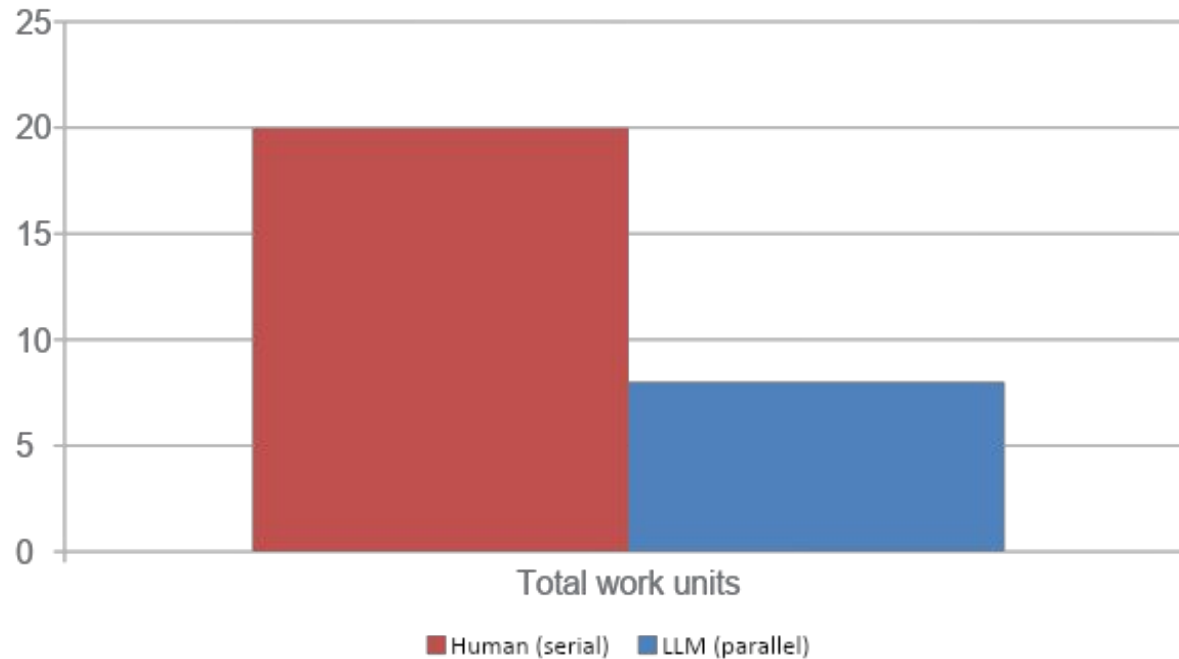
NEW DEFAULT

Prompt-first

- Start with intent + constraints
- Decompose into parallel tasks
- Model drafts → you review
- Quality = architecture + tests

Throughput beats speed

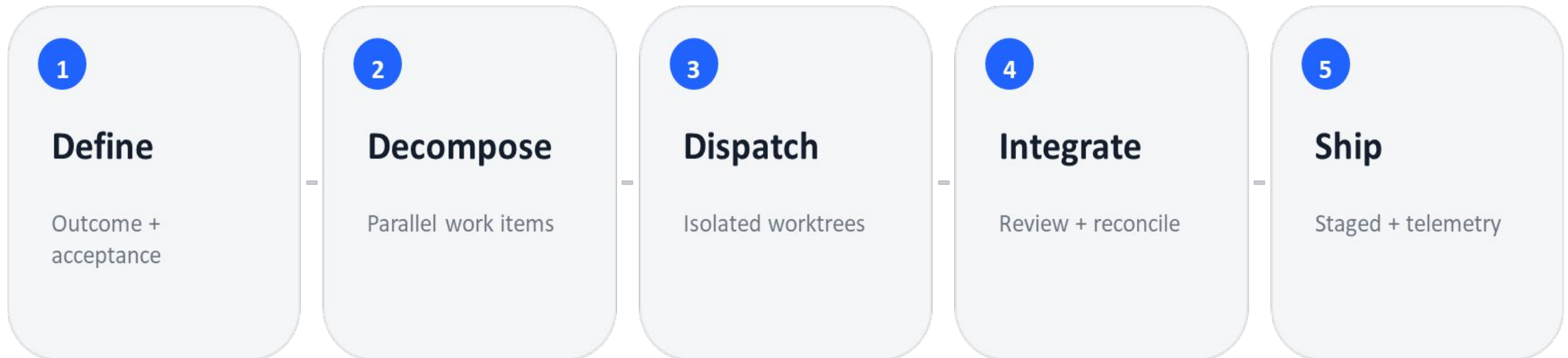
Illustrative example: five similar tasks.



What changes the outcome

- Your time is serialized; agent work can run concurrently.
- Constraint becomes **review + integration capacity**.
- Solve with decomposition, checklists, and automated tests.

Prompt-first operating loop



Control system: humans decide scope, constraints, and integration; agents draft execution.

- Review diffs, not raw generation
- Integrate early and often
- Ship safely: staged rollout + rollback path

What still matters (more than ever)

Non-negotiables

- **Architecture & module boundaries**
- Interfaces / contracts (invariants)
- Test strategy (unit + integration)
- Security & secrets handling
- Observability (logs, metrics, alerts)

De-emphasize

- **Hand-crafted code as the default**
- Premature polish / over-engineering
- Endless bikeshedding (lint, micro-style)
- Consensus meetings instead of tests
- Big-bang releases with zero learning

Tooling: the minimum viable stack

Tooling is secondary; habits are primary.

Orchestration UI

Task juggling, previews, terminal routing

Parallel branches

Git worktrees + isolation (secrets, DB)

Coding agent

Claude Code / Codex (drafts code from prompts)

Optimization target

- Orchestration (clear tasks + boundaries)
- Fast review loops (diffs + tests)
- Guardrails in CI (quality & security)
- Repeatable prompt templates

If you improve only one thing:

Standardize prompts + Definition of Done.

Parallel agents without collisions

A 3-part isolation blueprint.



Worktrees

File + branch isolation
No overwrites, no tangled history



Symlink secrets

One source of truth
Avoid config drift



Isolated runtime

DB + ports per worktree
Parallel tests + migrations

Automate it:

one command creates the worktree, links required secrets, and provisions an isolated DB.

Operating model: roles + Definition of Done

Architect / Reviewer

- Owns system design
- Approves integration & standards

Prompt Author

- Writes specs + constraints
- Provides examples & stop conditions

Integrator

- Merges worktrees
- Keeps main branch green

QA / Validator

- Runs checks & probes
- Verifies acceptance criteria

Definition of Done

- Tests pass
- Acceptance criteria met
- Risks + tradeoffs documented
- Rollout plan & rollback path clear

Operating principle

Strong opinions, loosely held — disagree fast, commit, ship, learn.

Prompt patterns that scale

Bad outputs are usually a spec problem. Standardize prompts.

Spec-first template

- 1 Goal:
- 2 Context:
- 3 Constraints (deps, perf, security):
- 4 Acceptance criteria (tests/behaviors):
- 5 Non-goals:
- 6 Output: diff + brief rationale

Tasking (parallel worktree)

- 1 Create worktree: feature/<task>
- 2 Do only: <scope>
- 3 Avoid: unrelated files
- 4 Add/update tests: <where>
- 5 Stop when: tests pass + diff ready

6 Return: summary + commands run + logs

Rules of thumb

- Ask for diffs and tests (not “write code”)
- Pin constraints: deps, perf, security, style
- Provide edge cases + expected outputs
- Force a stop condition (handoff point)

Prompt quality compounds.

Standard templates reduce retries and make outcomes predictable.

Guardrails: quality, security, and process

Make safety the default so parallelism doesn't create chaos.



Quality

- Tests required for changes
- Prefer smaller diffs
- Automated checks in CI
- Architecture review for cross-cutting changes



Security & privacy

- No secrets in prompts
- Redact customer data
- Dependency allow-list
- Threat-model sensitive changes



Process

- One owner per integration
- Main branch always green
- Time-box prompt iteration
- Document assumptions + TODOs

When to slow down

Payments/auth • data migrations • security-sensitive code • high-blast-radius refactors

Rapid experimentation

If it can be tested, stop debating and run the experiment.

Operating cadence

- Plan in 2-month cycles (sync to model/tool upgrades)
- Bi-weekly commitment lists (PM + Eng)
- Daily shipping: improvements/experiments go live
- Maintain 6–12 month strategic bets (architecture that compounds)

Rule: Most experiments fail, learning is the win. Don't run so long you can't conclude.

5-day experiment loop

- **Day 1** Hypothesis + success metric
- **Day 2** MVP (truly minimal)
- **Days 3–5** Ship to subset + collect signal
- **Week 2** Analyze → proceed / pivot / stop

Decision velocity: two-way doors win

Decide fast when reversible; deliberate when irreversible.



Rustam Akhmadullin

ahmadullin.com

[linkedin.com/in/rustam-akhmadullin](https://www.linkedin.com/in/rustam-akhmadullin)

Two-way door

- Reversible / feature flags (most changes)
- Decide fast, test immediately
- Single-person accountability
- Prefer bold hypotheses over small tweaks

One-way door

- Hard to reverse (rare)
- Add safeguards: security, migrations, comms
- Prototype before committing
- Still bias for learning

Communication protocol:

async first — announce decision + owner + timing; share context, not just the plan.

30-day adoption plan



Rustam Akhmadullin

ahmadullin.com

[linkedin.com/in/rustam-akhmadullin](https://www.linkedin.com/in/rustam-akhmadullin)

Shift default behavior without breaking delivery or morale.

Week	Focus	Deliverables
Week 1	Baseline + training	Prompt templates, review checklist, “done” criteria, first AGENTS.md
Week 2	Parallel isolation	Worktree workflow, secrets strategy, isolated test DB template
Week 3	Experiment engine	5-day loop, rollout guardrails, dashboard + alerts
Week 4	Institutionalize	Skills library, modular AGENTS.md, metrics review + iteration

Change management:

train the “default reflex” (prompt first). Tooling follows habits.

Metrics to prove it works

Increase delegation while keeping quality stable.

Velocity

- Lead time (idea → prod)
- Deployment frequency
- Throughput (items/week)
- PR cycle time (open → merge)

Quality

- Change failure / rollback rate
- Bug escape rate (incidents)
- Test pass rate + flake trend
- SLO adherence (uptime/latency)

Security

- Secrets incidents
- Vuln findings trend
- Dependency policy violations
- High-risk changes reviewed

Operational metric

% of work delegated to agents vs hand-coded → target increases over time while quality remains stable.

Clarity: how to win



Rustam Akhmadullin

ahmadullin.com

[linkedin.com/in/rustam-akhmadullin](https://www.linkedin.com/in/rustam-akhmadullin)

Principles

- Outcome > implementation: specify intent, constraints, acceptance tests.
- Parallelize everything that can be parallelized.
- Review is the control system: diffs + tests + integration discipline.
- Automate guardrails (CI, checklists, templates) so speed doesn't create risk.
- Ship to learn: short cycles, clear metrics, decisive follow-through.

Next step:

pick one repo + one feature and run the 30-day plan.

Technical reference

Engineer-facing setup notes for safe parallel agents (worktrees, secrets, DBs, runbooks).

Appendix

Appendix A: CLI quick install

Treat usage tiers as a compute budget for throughput.

Claude Code (example)

```
1 # macOS / Linux / WSL
2 curl -fsSL https://claude.ai/install.sh |
  bash
3
4 # Windows PowerShell
5 irm https://claude.ai/install.ps1 | iex
6
7 # Start
8 cd /path/to/repo && claude
```

- Authenticate once (per CLI prompts)
- Run only inside the intended folder
- Review diffs + run tests before merge

Safety habit

- Treat outputs as drafts
- Keep main branch green
- Prefer small, reviewable diffs
- Never paste secrets or customer data

Note

Install commands vary by tool/vendor; keep your internal runbook updated. Check here <https://code.claude.com/docs/en/quickstart>

Appendix B: Git worktrees — quick start

One worktree per task/agent = zero file collisions.

Commands

```
1 # Create an isolated workspace (new branch +  
folder)  
2 git worktree add -b feature/foo  
../repo-feature-foo  
3  
4 # List active worktrees  
5 git worktree list  
6  
7 # Remove when done  
8 git worktree remove ../repo-feature-foo
```

- Worktrees share the same .git metadata (no extra clone)
- Never use the same branch in two worktrees
- Clean up aggressively when merged

Why it matters

- Separate folders + separate branches
- Parallel tasks without overwrites
- Run tests/migrations concurrently with DB isolation

High-leverage

Wrap worktree + secrets + DB setup into one script.

Appendix C: Isolation details

Secrets + DB per worktree (example patterns).

Symlink secrets/config

```
1 # macOS / Linux
2 ln -s ../repo/.env .env
3 ln -s ../repo/.npmrc .npmrc
4
5 # Windows
6 mklink .env ..\repo\.env
```

DB isolation (example)

```
1 # in .env.local (not committed)
2 DB_NAME=app_test_<worktree-suffix>
```

Rule

- One agent = one worktree = one DB
- Parallel migrations/tests without lock fights
- Symlinked secrets avoid duplicated config drift

Automation pattern

A script reads a “worktree include” list, creates the worktree, links files, provisions DB, and bootstraps deps.

Appendix D: AGENTS.md + long-running runbooks

Keep guidance short, deterministic, and local.

AGENTS.md best practices

- Keep it short + high-signal; remove duplicates.
- Put commands + runbook early (install, dev, tests, deploy).
- Use examples > essays (bad vs good).
- Modularize by folder; nearest file wins.
- Add guardrails (what not to do) + PR acceptance criteria.

Long-running commands

- Encode the procedure in a dedicated Skill/runbook (not a 10k-token wall).
- Standard polling interval (e.g., 10–30s) + max attempts.
- Define success/failure log markers.
- Capture stderr/stdout + exit codes; attach outputs.
- Stop and ask for a human decision when thresholds are hit.