

Quick Reference

A companion pack for RAD and AI Coding Agents: Why Fast Prototypes Fail at the Cutover. Drop-in templates, tests, and CI gates that turn the article's checklist into something your pipeline actually enforces.

The core idea

A working AI-generated prototype proves the happy path exists. It says almost nothing about the rules nobody thought to state out loud — the missing “no self-approval” branch, the hallucinated package name, the secret baked into the client bundle. This kit closes that gap with three things:

- A fillable cutover checklist for pre-launch review.
- A spec template + worked example with REQ-IDs and GIVEN/WHEN/THEN.
- A drop-in CI workflow that turns the checklist into a merge gate.

The 10 cutover gates

Run as a real pre-launch review with a qualified reviewer — not a solo self-check. Any fail or blocked blocks launch. “Yes” is not evidence.

#	Gate	What it really asks
1	Negative cases written down	Who shouldn't be able to do this? What happens if they try?
2	Every business rule is an explicit REQ	If it lives only in Slack, it gets lost before cutover.
3	Authorization reviewed separately	A feature working for the right user says nothing about the wrong one.
4	Secrets confirmed server-side	Not in client code, not committed, not logged.
5	Mocks audited	A passing suite that never hits real deps proves less than it appears.
6	Human review of generated code	Look at the implementation of every rule — not just the demo.
7	Rollback & monitoring before launch	If you can't describe the rollback in one sentence, don't ship.
8	Data migration at prod scale	Not just the sample data from prototyping.
9	Dependencies checked against registry	Slopsquatting: 4.6–19.7% of AI imports are hallucinated names.
10	App registered in internal inventory	Who owns it, what data, where deployed.

The spec format

A spec is only load-bearing if it turns into tests. Use REQ-IDs as the contract between spec and test suite.

Business rules:

```
## Business rules & security constraints
| REQ-ID | Rule | Source |
|-----|-----|-----|
| REQ-01 | Self-approval is forbidden. | Security §4.2 |
| REQ-02 | Amounts > $500 require dual sign-off. | Finance §3.1 |
```

Acceptance criteria (GIVEN / WHEN / THEN):

```
### REQ-01: Self-approval forbidden

GIVEN an employee has submitted their own expense report
WHEN they attempt to trigger the approval action
THEN the system must reject with HTTP 403
```

AND log the unauthorized attempt in the audit log.

Test wired to the REQ-ID:

```
it('REQ-01: should forbid self-approval and return 403', async () => {
  const res = await request(app)
    .post(`/api/expenses/123/approve`)
    .set("Authorization", `Bearer ${employeeToken}`)
    .send();
  expect(res.status).toBe(403);
});
```

The CI gate

A checklist in a wiki gets skipped the first time a release is late. The durable version is a set of checks wired into the pull-request pipeline that block a merge rather than politely suggest one.

Check	Tool	What it catches
REQ & rarr; test traceability	grep + CI step	A REQ in spec.md with no matching test name.
SAST scan	Semgrep	OWASP Top 10 vulns in generated code.
Dependency verification	verify-dependencies.js	Hallucinated or recently-registered package names.
Secret scan	gitleaks	Credentials committed to the repo.

Top anti-patterns

Anti-pattern	The fix
Treating the prompt as a spec	A prompt captures what was said. A spec captures what's enforced.
User testing = security testing	You can't click on a rule that was never written.
Green checkmark = correct	Mock-heavy tests prove you talk to mocks, not real deps.
Spec-driven on every change	Match process to risk. CSS tweaks don't need 16 acceptance criteria.
Checklist in a wiki	Wire it into CI. Wiki pages get skipped under deadline pressure.

When to use what (60-second version)

Answer in order. Stop at the first yes.

1	Safety-critical, regulated, or failure has severe consequences?	Web or SaaS with formal verification, or agentic with extra cutover controls.
2	Large, mature, tightly coupled codebase?	Agile with human-led refactoring. Agent = research assistant, not author.
3	End users can give feedback on a prototype with RAD or A?	RAD or Agentic engineering. CI enforcement? Agentic. No CI? Classic RAD.
4	Problem is genuinely novel — no one knows the right answer yet?	Vigilant cutover. Throw away prototype to learn. Throw it away. Then pick from above.

The one question that matters: Can you describe, in one sentence, what would make this feature unsafe to ship? And does your process actually test for that specific thing?

If the answer is “I'm not sure what would make it unsafe,” you don't have enough specification to ship — regardless of which methodology you picked. Write the spec first. Then pick the methodology. Not the other way around.

Kit contents: README • 01 Cutover Checklist • 02 Spec Template • 03 Example Spec (Expense Approval) • 04 Jest Test Template • 05 GitHub Actions CI • 06 Decision Guide • 07 Anti-Patterns • 08 Dependency Verification Script • this Quick Reference PDF