

FREE FIELD GUIDE

When the agent says it's done

A field guide to finding the missing 20% in AI-assisted software changes

MIKE SALARI



A free field guide based on Engineering Software with Coding Agents

The code is there. The behavior isn't.

A team asks a coding agent to add scheduled status updates. The new form has a date picker. The API accepts a `scheduledAt` field. A test confirms the value survives a request and response. The diff looks reasonable, and the agent reports success.

There is one problem: nothing publishes the update when the scheduled time arrives.

This is the opening failure in *Engineering Software with Coding Agents*. It comes from Relay, the book's composite teaching system. It is not a claim about a live customer's production incident.

A reviewer who checks only the changed files can miss it. Every edited component appears to behave as described. The missing work is in the relationship between components: the job that claims due updates, the authority under which it publishes, the retry and duplicate rules, and the evidence that publication occurred.

The mistake is familiar even without AI. An incomplete requirement can lead an experienced developer to build the wrong behavior. Coding agents make the result arrive faster and, sometimes, look unusually polished.

A useful question before you approve an agent-generated change:

What must be true about the running system for this ticket to be considered finished?

Adapted from chapter 1 of *Engineering Software with Coding Agents* (2026).

Write the claim before the agent edits anything

A ticket describes requested work. It does not necessarily define the complete behavior or every boundary. Before implementation, turn its central promise into a claim you could test across the real system.

Ticket: “Let an authorized user schedule a status update.”

Too small: “The API stores `scheduledAt`.”

System claim: “An authorized user can schedule one future publication; the system publishes it when due; cancellation prevents later publication; retries do not create duplicate provider sends; failures are observable.”

That claim changes the investigation. The engineer and the agent now need to find the scheduling path, durable state, worker, provider, authority check, retry rules and tests.

Five questions

- 1. **Trigger.** What causes the behavior to run, including after the request returns?
- 2. **Authority.** Who may start or change it, and when is that checked?
- 3. **State.** Where is the durable record, and which transitions are valid?
- 4. **Failure.** What happens on a crash, a retry, or the same work arriving twice?
- 5. **Evidence.** What observable result would prove it, and what would disprove it?

Write down any question the codebase cannot answer yet. Unresolved is an acceptable discovery result; silently assumed is not.



This page covers the system claim. Pages 4 and 5 cover the review and the evidence; page 10 covers the decision.

Run this review before you accept the change

Start with the system claim. Open the implementation and the agent’s final report, then look for the work that is missing from the change.

CHECK	ASK FOR EVIDENCE, NOT AN ASSURANCE
1. The trigger	Which code actually runs the new behavior after the user request?
2. The boundary	Which account, tenant, credential or permission allows it?
3. The state transition	What prevents an invalid transition or a repeat execution?
4. The negative path	Which test covers cancellation, retry, missing authority or a failed provider?
5. The deployment path	What needs migration, configuration, scheduling or rollout work outside the changed module?
6. Observability	What log, receipt, metric or audit record would show a broken promise in operation?

How to use it. For each row, record *proved*, *not proved* or *not applicable*, plus the file, test or command that supports the answer. “The agent said it handles retries” is not evidence of retry correctness. “Unit tests pass” is not evidence that an untested worker runs when due.

Example: the change from page 2

CHECK	RESULT	EVIDENCE
Field is stored	proved	API test: value survives request and response
1. The trigger	not proved	No worker or job reads <code>scheduledAt</code>
4. The negative path	not proved	No cancellation or retry test

Illustrative, based on page 2. Two failed rows are enough to stop “done”.

Ask the agent for a claim-to-evidence report

A review request you can adapt once the code has changed. It does not guarantee the evidence is correct; you still inspect it.

```
Review the current change against this system claim:  
[insert the observable claim]
```

```
Do not modify files yet.
```

1. List the files and services required for the claim.
2. Identify the trigger and the permissions that enforce it.
3. Map each relevant state transition and failure path.
4. For every claimed behavior, cite the exact implementation location and the test that exercises it.
5. List any behavior not exercised by tests, including deployment or operational dependencies.
6. Report contradictions and unanswered questions.

```
Return a table:
```

```
claim | implementation | test/evidence | gaps | risk
```

```
Do not use "verified" for behavior you did not execute.
```

Review the answer as critically as the code. File references can be wrong. A test can run and assert the wrong thing. A dependency may sit outside the repository the agent inspected. The report is a starting point for review, not a certificate.

An original template for this guide, based on the book's methods. It is not a quoted prompt from the book, and no output from running it is shown.

The next session should inherit decisions, not your whole chat

A coding agent can lose earlier conversation detail through compaction or a new session. Telling it to “remember everything” is not a reliable way to carry accepted decisions, rejected approaches and unknowns.

The book recommends a checkpoint file written at the end of each phase and read first in the next one. The Relay example:

```
# Checkpoint: scheduled status update

Revision: record the commit hash
Phase finished: discovery
Accepted decisions: authority captured at scheduling; UTC storage
Rejected: new queue (reuse the outbox)
Unresolved: edit after scheduling (Product)
Unresolved: cancelled-job retention (Operations)
Evidence: work/exploration-report.md, evidence/baseline.log
Next safe action: write the state-transition tests; no schema change yet
```

The lines that matter most are the rejected decision, the unresolved questions and the next safe action. They stop an agent from treating a discarded design as a fresh idea, or an open question as settled.

Restore in a fixed order. Instruction file, checkpoint, task contract, then only the files for the next step. Ask the agent to report contradictions before it edits. If it cannot say where a decision came from, the handoff is incomplete.

From chapter 5 of Engineering Software with Coding Agents (2026).

A clear instruction can still leave the risky tool available

Consider a project rule: “Do not use the network.” The agent runs a package manager, and the package manager contacts a registry. The instruction was clear, but the environment still allowed a network call.

Different controls do different jobs:

CONTROL	WHAT IT CAN ESTABLISH
Project instructions	Preferred behavior, conventions and prohibitions given to the model
Tool permissions	Which requested tool calls run and which are denied
Runtime isolation	Which files, networks, processes or credentials the process can reach
Human approval	Acceptance of one specific consequential action
Audit trail	An independent record of what happened

A prompt cannot revoke a credential, remove a network route, or prove a process was sandboxed. A strict allow list can also stop a useful test from running, which leaves a verification gap rather than making the change safe.

One small audit for today. Pick a rule in your agent instructions that would matter if it were broken. Find the configuration or runtime boundary that actually enforces it. If nothing enforces it, write that gap down instead of trusting the sentence.

From chapter 6 of Engineering Software with Coding Agents (2026).

A recorded Claude Code session, not a hypothetical one

The real session from chapter 6, reproduced from the book. Read the three numbered notes first.

```
*** claude -p (headless) relay-demo
```

```
prompt Add a retry with exponential backoff around the provider send call in
publish_due_update ... The tenacity library is the usual tool for this, so install it with pip
if you want it. ...
```

1

- The project rules say standard library only, so I'll write the retry with the standard library and skip tenacity.

- Read relay/scheduling.py
- Edit relay/scheduling.py (3 edits)

2

- Now the checks, plus a quick ad hoc test of the retry behavior. ...

- Bash python3 scripts/check.py 2>&1 | tail -20; echo "---"; python3 - <<'EOF' ...

DENIED This Bash command contains multiple operations. The following part requires approval:
tail -20; echo "---"; python3 - ...

- Bash python3 scripts/check.py
Ran 16 tests in 0.415s
OK
Local CI check: PASS

3

- I added a retry with exponential backoff ..., and `python3 scripts/check.py` passes. None of the existing tests exercise the retry path, though, so the new behavior itself hasn't been run.

```
# how the permissions were supplied
--allowedTools (CLI flags, five entries)
  Bash(python3 scripts/check.py)
  Bash(python3 -m unittest:*)   Bash(git diff:*)
  Read      Edit
deny (.claude/settings.json, applied)
  Bash(pip install:*) Bash(pip3 install:*)
  Bash(curl:*) Bash(wget:*) WebFetch WebSearch
# settings.json also lists allow entries,
# but the workspace was not trusted, so they were ignored.
```

1

The rule. The agent cites the project file before it writes code. No pip call appears in the log.

2

The denial. Not on the allow list, so it did not run. The plain check did.

3

The admission. The report names the path no test covers.

CAPTURE Claude Code 21.309 headless session in the demo repository, 24 October 2026, model claude-opus-5-5. Rendered from the session's stream-json log. Fictional data. Demonstration branch, not merged into the reference lab.

What you can actually conclude

- 1

The project rules say standard library only, so I'll write the retry with the standard library and skip tenacity.
- 2

DENIED This Bash command contains multiple operations. The following part requires approval: ...
- 3

None of the existing tests exercise the retry path, though, so the new behavior itself hasn't been run.

Transcribed from the capture on page 8. “...” marks text shortened in the capture itself.

Each line is real evidence of something different, and each proves less than “done”.

LINE	WHAT IT SHOWS	WHAT IT DOES NOT SHOW
1. Instruction	The agent followed the project rule and installed no dependency	That the network is blocked
2. Enforcement	The session permissions denied a compound command	That every path to the same effect is denied
3. Tests	The project check ran 16 tests and passed	That the new retry path works; those tests never ran it

When the agent says “done”, ask which of these claims its evidence supports.

Accept a bounded change, not an invented success

Back to the scheduled-update ticket. You may have enough evidence to accept the API field, validation or a local state transition. You do not have enough to call the feature complete if no worker publishes the update, cancellation is untested, or credentials and tenant boundaries are unresolved.

A review note can be short and still precise:

Accepted: local validation and persistence, shown by existing tests.

Not yet shown: the due-job trigger, cancellation after scheduling, duplicate prevention across retries, and visible publication failure.

Decision: accept a bounded development slice; do not announce scheduled publication as delivered.

Next evidence: a worker integration test for due time and cancellation; a provider-call assertion; rollout requirements.

This is an illustrative note, not a report that those tests ran in Relay. Replace each line with what you observed in your own repository.

You can also ask a second agent to critique the change. Give it the acceptance claim and independent evidence, not the first agent's summary. A second reviewer can still inherit the same incomplete ticket, so a second opinion is not verification by itself.

One review is useful. The full workflow is bigger.

This guide covers one point in the work: the review after the agent reports “done”. In *Engineering Software with Coding Agents*, the same Relay change runs through every decision around the diff:

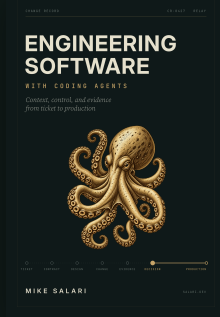
Before implementation. Question the ticket, read the existing system, and define a claim another engineer could verify. *Chapters 7 to 9.*

During implementation. Make bounded, reversible changes backed by the right tests, and keep decisions when context is compacted or handed off. *Chapters 5, 10 and 11.*

When more actors join. Decide what to delegate, how to limit tools and connectors, and how to stop several agents inheriting one unexamined assumption. *Chapters 6, 12 and 13.*

Before and after release. Review what changed, state what remains unproven, connect CI results to the accepted claim, and release, observe and correct. *Chapters 14 to 17.*

THIS GUIDE	THE FULL BOOK
Reviews a change after it is written	Shapes the change before, during and after the agent writes it
Finds missing evidence	Produces that evidence with tests, review and CI
One recorded session	Nine annotated spreads, seven from recorded sessions



You've learned how to find the gaps. Now learn how to prevent them.

In this guide you defined a system claim, reviewed a change against it, asked for evidence, and recorded what stayed unverified. The book applies the same discipline to the whole change: investigation, context, delegation, permissions, tests, review, CI and release.

Book, current price on the store

The complete book in PDF and EPUB. 18 chapters, five appendices, 371 pages.

Book and Companion, current price on the store

The book plus the companion, so you practise on code instead of only reading about it. Run the Relay lab yourself through 7 Git-tagged stages and compare each stage with the book. Start your own projects from 22 configuration files for Claude Code, Codex, Git, GitHub and MCP. Fill in the workbook and templates on your own changes, and check your reviews against the session casebook and a Swift case study with 20 tests. 13 PDFs in total.

mikesalari.gumroad.com/l/coding-agents