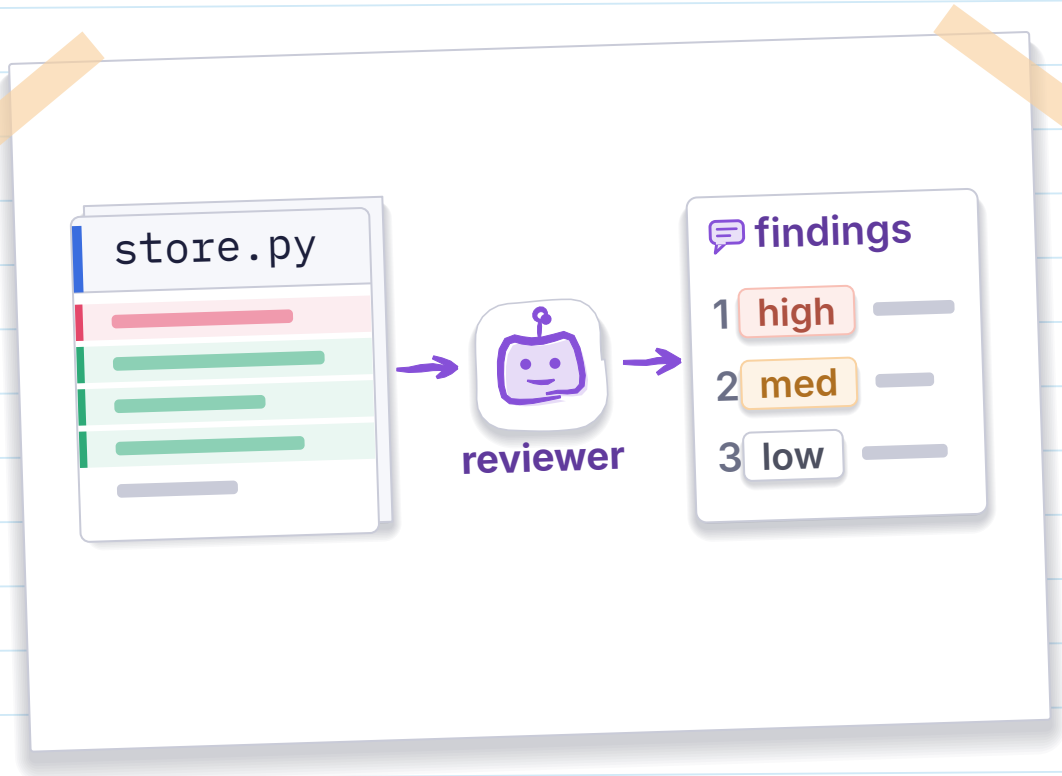
Prof. Aviv Zohar

Semester A, 2026–7

Agentic Code Review

A second reader for the agent's code



Agent review adds a fuzzy tool beside CI

notes/store.py the change −1 +4

```
12 def save_note(note: Note) -> bool:
-      NOTES_FILE.write_text(note.text)
+      try:
+          NOTES_FILE.write_text(note.text)
+      except OSError:
+          pass
17     return True
```

Formal tools: CI

conditions we have encoded

- Tests
- Type checks
- Lint

Agent review adds a fuzzy tool beside CI

notes/store.py the change −1 +4

```
12 def save_note(note: Note) -> bool:
-      NOTES_FILE.write_text(note.text)
+      try:
+          NOTES_FILE.write_text(note.text)
+      except OSError:
+          pass  (b)
17     return True  (a)
```

Formal tools: CI

conditions we have encoded

- Tests
- Type checks
- Lint

Fuzzy tool: agent review

what our checks do not encode

- Spec mismatch (a)
- Missing error path (b)
- Misleading name

Review in a fresh context

agent session `~/notes`

you typed by you

Make `save_note` handle disk errors.

author agent the session that wrote the code

`write_text` can't really fail here, so I'll catch `OSError` and return `True`.

tool call asked by the model

```
Edit notes/store.py
```

you typed by you

Now review your change.

author agent the session that wrote the code

Looks correct to me.

Its context already holds the reasons the code looked right.

So it tends to approve its own choices.

Review in a fresh context

agent session `~/notes`

you typed by you

Make `save_note` handle disk errors.

author agent the session that wrote the code

`write_text` can't really fail here, so I'll catch `OSError` and return `True`.

tool call asked by the model

Edit `notes/store.py`

you typed by you

Now review your change.

author agent the session that wrote the code

Looks correct to me.

fresh session or a subagent

Spec

"`save_note` returns whether the note was written."

the author's conversation is not here

Diff `notes/store.py`



reviewer

Claude Code writes → Codex reviews

a different model, different mistakes

Brief the reviewer

Diff `notes/store.py` −1 +4

```
def save_note(note: Note) -> bool: …
```

Spec or issue

“`save_note` returns whether the note was written.”

Review focus

Does it meet the requirements?
Does it handle failures?



Brief the reviewer

Diff `notes/store.py` −1 +4

```
def save_note(note: Note) -> bool: …
```

Spec or issue

"`save_note` returns whether the note was written."

Review focus

Does it meet the requirements?
Does it handle failures?

`REVIEW.md` — in the repository

```
1 # Review checklist
2 - Never mock the database.
3 - Every endpoint checks authorization.
```



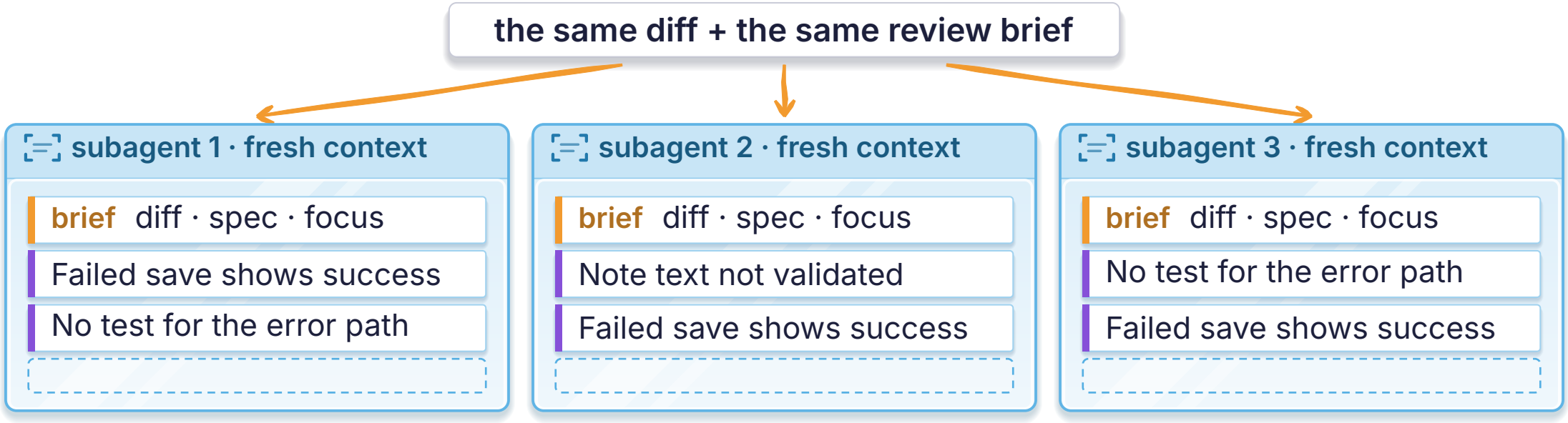
Project decisions that a generic reviewer cannot know.

or package them as a review skill:

```
.claude/skills/review/
└── SKILL.md
```

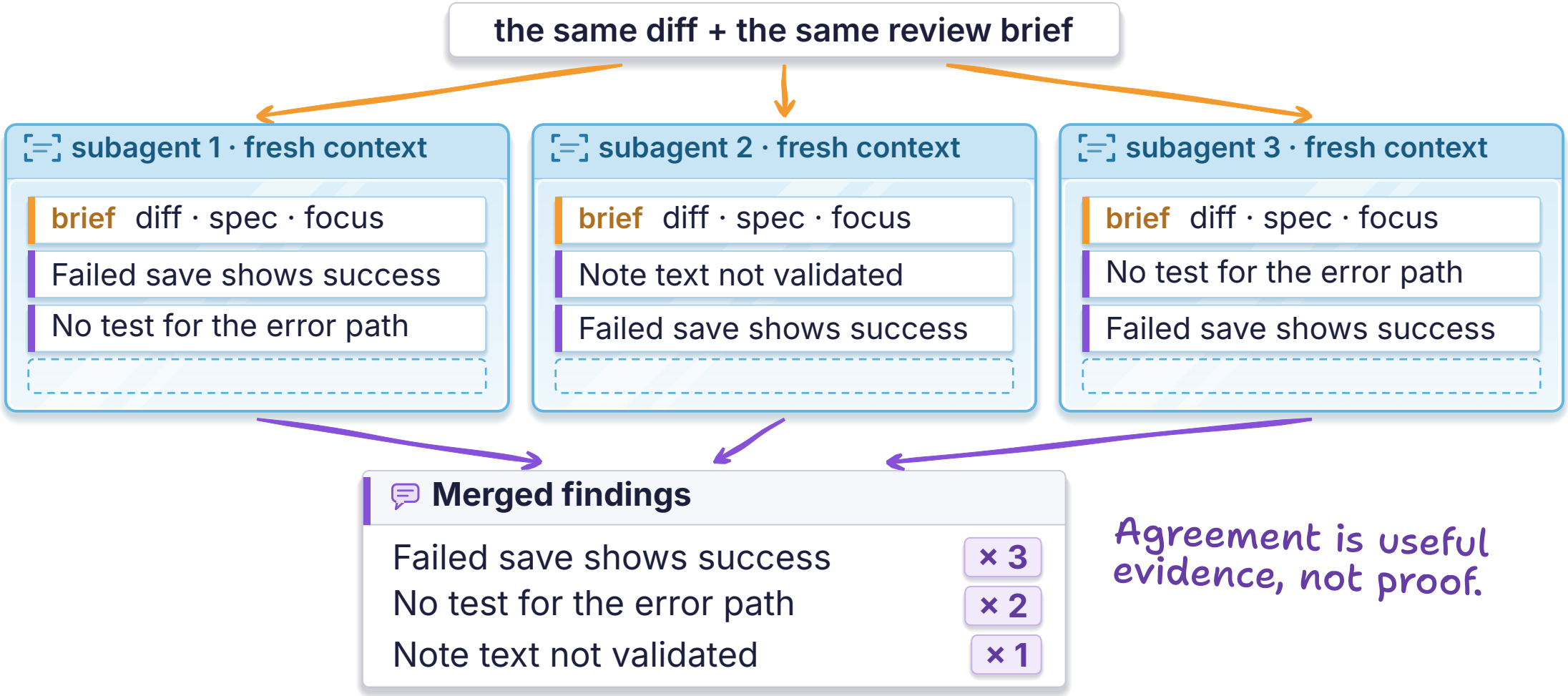
Tell the reviewer to read these rules.

Run several fresh reviews



Same code, same brief: partly different findings.

Run several fresh reviews



Ask for findings you can judge

Finding from the reviewer

Failure scenario required for every finding

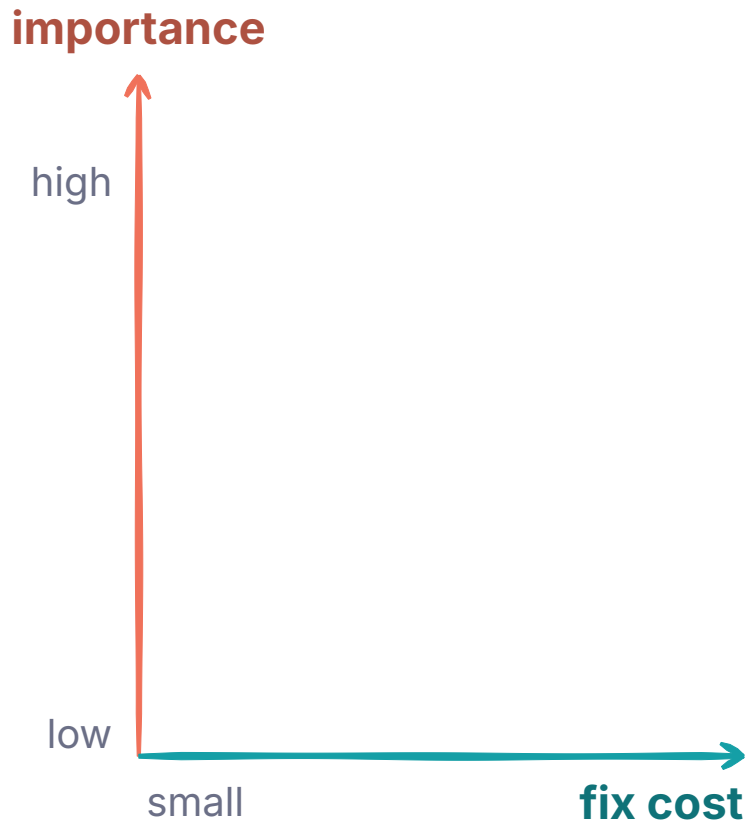
a concrete failure scenario

Importance

how serious is the consequence?

Fix cost

how much work is the fix?



Ask for findings you can judge

Finding from the reviewer

Failure scenario *required for every finding*

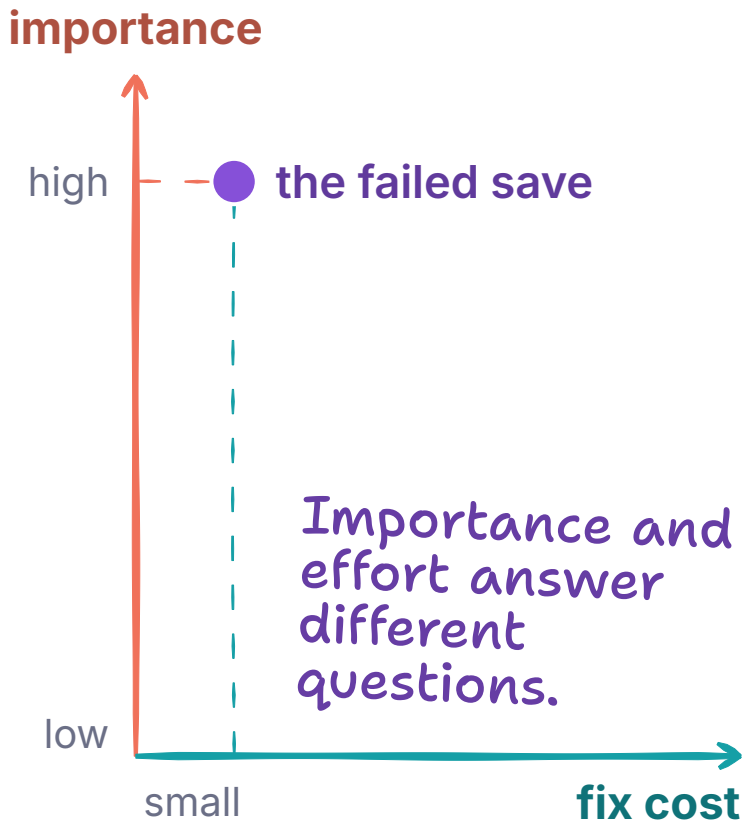
Save fails → success shown → edits lost on reload

Importance

High: data loss

Fix cost

Small: handle the error, add a test



Review commands built into the tools

Claude Code

```
❯ /code-review [effort] [PR number or branch]
```



ⓐ effort	`low` `medium` `high` `xhigh` `max`
ⓑ target	a PR number or branch; none: the current diff
`--comment`	posts findings as inline PR comments
`--fix`	applies the findings to the working tree

Leave it off while you assess the report.

Review commands built into the tools

Claude Code

```
› /code-review [effort] [PR number or branch]
```

ⓐ effort — `low` `medium` `high` `xhigh` `max`

ⓑ target — a PR number or branch; none: the current diff

`--comment` posts findings as inline PR comments

`--fix` applies the findings to the working tree

Check the installed version's help: these commands change.

Claude Code `/security-review` checks branch changes for security vulnerabilities

Codex CLI `/review` starts a code review

Verify a finding before you act on it



Finding reported by the reviewer

"If saving fails, the app still reports success, and reloading loses the edits."

Asked for findings, a reviewer finds something, real or not.

Verify it first: turn its failure scenario into a failing test.

Verify a finding before you act on it

reviewer
Reports a finding

→

you
Reproduce with a failing test

→

you
You decide

→

author
Fix

→

you
Rerun the test

Finding reported by the reviewer

"If saving fails, the app still reports success, and reloading loses the edits."

tests/test_store.py — 1 failed

```
1 def test_failed_save_is_reported(monkeypatch):
2     monkeypatch.setattr(
3         Path, "write_text", raise_oserror)
4     assert save_note(Note("draft")) is False
```

It fails on the current code: evidence for the fix, and a regression test to keep.

make saving fail …

… and expect a reported failure

Verify a finding before you act on it



Finding reported by the reviewer

"If saving fails, the app still reports success, and reloading loses the edits."

tests/test_store.py — 1 failed

```
def test_failed_save_is_reported(monkeypatch):
    monkeypatch.setattr(
        Path, "write_text", raise_oserror)
    assert save_note(Note("draft")) is False
```

A reviewer that applies its own fixes becomes a second author, whose changes still need checking.

The reviewer reports; you decide what to accept.

What agent review catches, and what it misses

Often catches

local, in the code it reads

- Local bugs
- Missed edge cases
- Spec mismatches
- Missing error handling
- Inconsistencies

Often misses

judgment, and what reading cannot show



Whether it is the right feature



Fit with the architecture

What agent review catches, and what it misses

Often catches
local, in the code it reads

- Local bugs
- Missed edge cases
- Spec mismatches
- Missing error handling
- Inconsistencies

Often misses
judgment, and what reading cannot show

- Whether it is the right feature
- Fit with the architecture
- Invariants in files it never read
- Runtime behavior

Needs a human reader: security changes · data migrations · public APIs

A second reader for the agent's code



See you in the next video.