Changing code you do not fully understand

Key takeaways

1. Know a boundary’s invariants, owners, and consumers before you change it.
2. The smallest coherent fix repairs the whole dependency class, not one case.
3. Verify causality narrowly, compatibility broadly, one artifact deeply.

Common misunderstandings of safe code changes

Which claim would you defend right now? Choose before the evidence.

“You must understand the whole repository before changing shared code.”

“The safest fix is always the patch with the fewest changed lines.”

“A green full build proves the dependency graph is correct.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

You do not need the whole repository in your head

Historical worked example: one Make rule, 12 dependent wrappers

The first question after diagnosis: who else depends on this?

```
$ rg -l '^from review_deck_renderer import' slides/gen_???_*.py | wc -l
12
```

You do need the invariants, owners, and consumers of the boundary you will change.

Measure blast radius before choosing a fix

Direct dependents

```
$ rg -l                '^from
review_deck_renderer import'
slides/gen_???_*.py

# wrappers that call shared code
```

Downstream consumers

```
$ rg -n '097_.*-slides.pdf'                web/
lectures.txt slides/Makefile

# schedule, build, publication copy
```

surface	must remain true
individual deck	the wrapper still emits the same PDF name
shared renderer	one outline id still selects one deck
incremental build	changed sources rebuild; unrelated sources do not
publication	make still copies every public PDF to web/lectures

Find the owners, then write the invariants

owner	responsibility	invariant for this change
gen_NNN_*.py	deck-specific entry point	target stem maps to wrapper
style_NNN_*.py	deck palette/helpers	style edits invalidate its PDF
course_style.py	shared slide primitives	shared edits invalidate decks
review renderer	first-pass deck mechanics	renderer edits invalidate wrappers
review document	first-pass deck content	outline edits invalidate wrappers
Makefile	incremental build graph	edges match real reads/imports

Do not infer ownership from proximity. The wrapper lives beside the PDF, but the renderer and outline own most of its content.

A local patch can preserve the bug

Too local: fixes one target

```
097_react_architectures-
slides.pdf:                ../docs/2026-07-05-
course-structure-review.md

# 11 other scheduled wrappers stay stale
# style and renderer edits still stay stale
```

Coherent: encode dependency classes

```
# every deck: entry + shared chrome
%-slides.pdf: gen_%.py course_style.py

# every matching style is a real edge
$(STYLE_PDF_FILES):                %-slides.pdf:
style_%.py

# wrapper-backed decks: renderer + outline
$(REVIEW_PDF_FILES):
review_deck_renderer.py $(REVIEW_OUTLINE)
```

The smallest coherent change is not the fewest edited characters. It is the smallest change that restores the invariant for the whole class.

Make the hidden dependencies explicit

Fix sketch: known shared edges, not a Python import-graph parser

```
REVIEW_OUTLINE := ../docs/2026-07-05-course-structure-review.md
STYLE_FILES := $(wildcard style_???_*.py)
STYLE_PDF_FILES := $(patsubst style_%.py,%-slides.pdf,           $(STYLE_FILES))
REVIEW_GEN_SCRIPTS := $(shell rg -l           '^from review_deck_renderer import' $(GEN_SCRIPTS))
REVIEW_PDF_FILES := $(patsubst gen_%.py,%-slides.pdf,           $(REVIEW_GEN_SCRIPTS))

%-slides.pdf: gen_%.py course_style.py
  ./$<

$(STYLE_PDF_FILES): %-slides.pdf: style_%.py
$(REVIEW_PDF_FILES): review_deck_renderer.py $(REVIEW_OUTLINE)
```

- Each matching style file is a real prerequisite for its generated PDF.
- Only review-backed PDFs depend on the shared renderer and outline.
- The existing recipe stays in one place; explicit targets add prerequisites.

Do not hide a migration inside a bug fix

decision	question	answer now
correctness	does Make rebuild when a real source changes?	yes — fix now
performance	does the fix rebuild unrelated PDFs?	avoid broad overbuild
architecture	should first-pass wrappers continue to exist?	migrate deck by deck
content	should this outline become the final lecture?	no — rewrite separately

Repair invalidation without promising that the provisional renderer is permanent. A correct build graph makes the later migration safer.

Targeted verification proves the intended edge

Positive tests: two newly declared edges trigger the recipe

```
cd slides
target=097_react_architectures-slides.pdf

make "$target"
touch ../docs/2026-07-05-course-structure-review.md
make --dry-run "$target" | tee /tmp/outline-rebuild.txt
test -s /tmp/outline-rebuild.txt

make "$target"
touch style_097_react_architectures.py
make --dry-run "$target" | tee /tmp/style-rebuild.txt
test -s /tmp/style-rebuild.txt
```

A passing full build cannot prove incremental invalidation: it may simply rebuild everything. Test the edge directly.

Verify what must not rebuild, then go broad

Negative check from slides/: preserve incrementality

```
make 097_react_architectures-slides.pdf
touch style_130_web_service_coordination.py
test -z "$(make --dry-run
097_react_architectures-slides.pdf)"

# unrelated style -> no rebuild
```

Broad check: every deck still builds

```
make -j4
uv run --project                ~/git-papers/
slide-builder                slide-audit --
metrics --render                "$PWD/
gen_097_react_architectures.py"

# repeat audit for changed deck scripts
```

Targeted tests establish causality. The broad build catches compatibility failures elsewhere. You need both.

Inspect one random downstream artifact deeply

Random challenge: artifact semantics, not just exit status

```
affected=( $(rg -l '^from review_deck_renderer import' gen_???_*.py) )
pick=${affected[$((RANDOM % ${#affected[@]} + 1))]}
pdf=${pick#gen_}
pdf=${pdf%.py}-slides.pdf

./$pick
pdftotext "$pdf" /tmp/deck.txt
rg -n 'Small.group|Key Takeaways' /tmp/deck.txt

# View every page: title, content, ordering, final activity,
# and no blank or overflow slide.
```

A green command is not a correct artifact. Random selection resists inspecting only the example that already worked.

Leave a receipt another engineer can challenge

claim	evidence to record
cause	old rule + touch experiment output
blast radius	consumer query and scheduled count
invariants	positive and negative rebuild cases
compatibility	parallel full build + layout audits
artifact quality	name and visual notes for random deck
remaining debt	first-pass wrappers still require migration

This receipt is more useful than ‘tests pass’: it states what was tested, what was not, and what architectural debt remains.

source: historical repository snapshot: slides/Makefile, review_deck_renderer.py, 12 wrappers

You vs. the Agent: safe change

Your Responsibility	Agent's Responsibility
Name the invariant and acceptable blast radius	Enumerate consumers and ownership boundaries
Separate the immediate fix from an architectural migration	Draft the smallest coherent patch and alternatives
Choose which risks require targeted and broad checks	Run positive, negative, and broad verification
Judge one downstream artifact for semantic quality	Produce an auditable change receipt with exact commands

Change the slice, verify the system

1. You can change unfamiliar code safely once you know its invariants, owners, and consumers.
2. The smallest coherent fix restores a rule for the whole dependency class, not one example.
3. Verify causality narrowly, compatibility broadly, and one random artifact deeply.

Choose the coherent fix

10 min

- Twelve deck scripts each import a matching style file; some also call a shared renderer. Editing `style_097.py` does not rebuild `097-slides.pdf`.
- Patch A adds one prerequisite for deck 097. Patch B declares the matching style dependency for every deck and the renderer dependency only where used.
- Choose A or B. Write two invariants, one positive test, one negative test, and one fact that could reverse your choice.
- Swap plans and challenge one missing consumer. Deliverable: a five-line change plan plus the challenge that survived.