Your docs are green. That is not the same as your docs being true.

DocGuard reads what your documentation claims about your code, reads the code, and reports every claim the two no longer agree on — saying which it is sure about, and which it is not.

THE PROBLEM

The faster AI writes code, the faster everyone loses the map of what was built. Every agent session starts from zero and re-derives how the system works. The docs are the map. Nothing checks whether the map still matches the territory.

A REAL FINDING, FROM DOCGUARD'S OWN DEMO FIXTURE — A 4-SERVICE PAYMENTS API WITH DELIBERATE DRIFT

```
4. [MED] Traceability
   DATA-MODEL.md — exists but no matching source code found (unlinked doc)
   → Doc lives in canonical/ but isn't in the manifest — guard skips it, drift accumulates silently.
```

`npx docguard-cli demo` returns seventeen of these in half a second, against a project whose documentation *looks* complete. Each names the file, the claim, and what trusting it would cost.

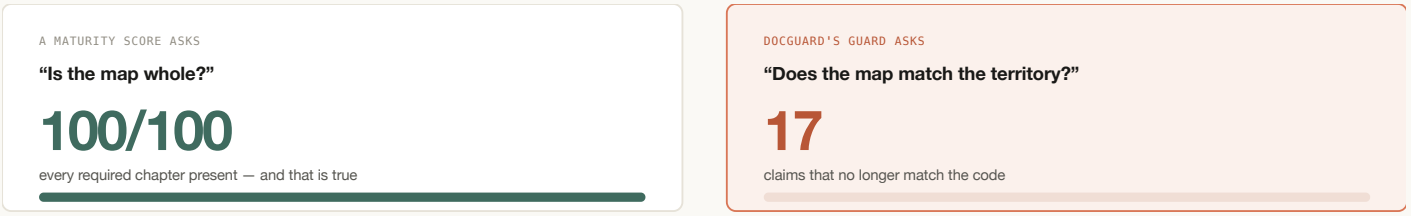


FIG. 1 — THE SAME DOCUMENTATION, UNDER TWO DIFFERENT QUESTIONS. DOCGUARD REPORTS BOTH, AND NEVER LETS ONE STAND IN FOR THE OTHER.

THE METHOD, IN ONE SENTENCE EACH — THE THIRD COLUMN IS THE ONE MOST TOOLS DO NOT HAVE

`verified`

The doc says it; the code agrees. **Believe it.**

`drift`

The doc says it; the code says otherwise. DocGuard names the fix.

`escalate`

Something changed near this claim. **The judgement is yours** — the tool says so, rather than guessing.

A document can be beautifully formatted and completely wrong. A green check is worth exactly as much as the question it answered.

THE DESIGN CONSTRAINT EVERYTHING ELSE FOLLOWS FROM · PHILOSOPHY.MD

TWO TIERS, OR THE TOOL WOULD OVERWRITE YOUR THINKING

A canonical document holds two kinds of sentence. **Some are derived from code** — endpoints, entities, environment variables. **Some are human reasoning** — why the queue is idempotent, which trade-off was rejected, the gotcha nobody would guess. A tool that regenerates the whole file destroys the second kind every time it refreshes the first.

DocGuard keeps both in one readable markdown file, separated by HTML-comment markers. It owns the bytes inside a `source=code` block and may rewrite them when the code changes. It never touches anything else.

THE MARKER, AS IT APPEARS IN A REAL ARCHITECTURE.MD

```
<!-- docguard:section id=api-endpoints source=code -->
| GET  | /api/orders/:id   | orders.controller.ts |
| POST | /api/orders       | orders.controller.ts |
<!-- /docguard:section -->
The orders service is idempotent on client-supplied keys because the mobile
client retries on any 5xx; see ADR-007 for the rejected alternative.   ← never regenerated
```

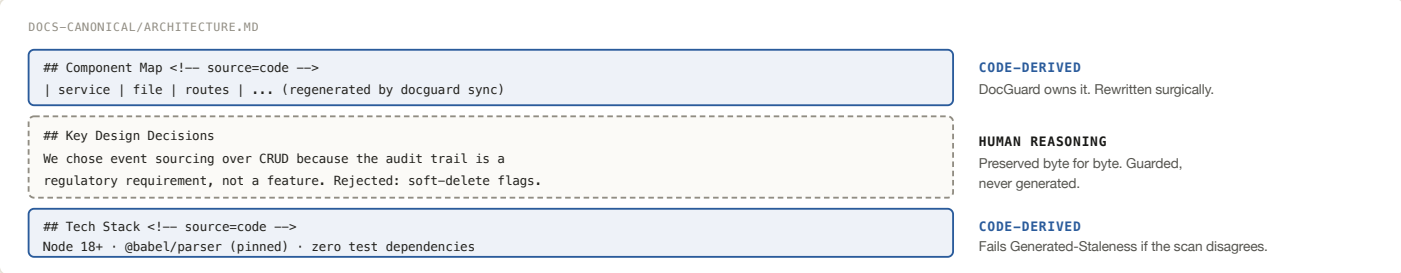


FIG. 2 — ONE FILE, TWO OWNERS. THE TOOL REFRESHES THE BLUE BANDS AND VALIDATES THE GREY ONE; IT NEVER HAS PERMISSION TO DO THE REVERSE.

THREE VERBS, ONE MEMORY

DIRECTION	VERB	WHAT IT DOES
code → docs	**generate**	Reverse-engineer a canonical memory from any codebase. Forty-six scanners build the code-truth skeleton (routes, schemas, screens, env vars, IaC); an agent writes the prose around it.
code ↔ docs	**sync**	When code changes, refresh only the affected `source=code` sections. Mechanical where deterministic; a structured agent task where not.
docs ↔ code	**guard**	Validate that the memory still matches reality: deleted endpoints still documented, routes never documented, env vars nobody reads, requirement IDs no test traces to.

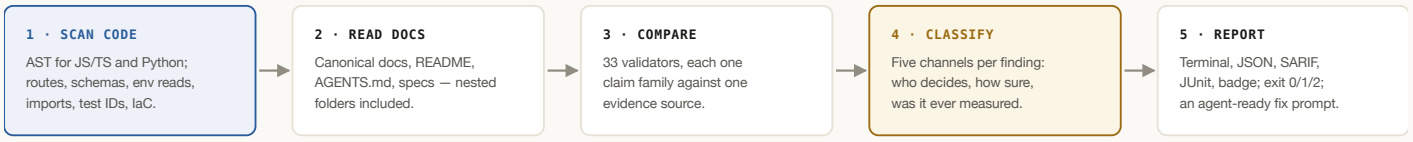
Any language: JavaScript/TypeScript and Python get a syntax tree; Rust, Go, Java/Kotlin, Ruby, PHP and C# get a shape-aware fallback, and every finding says which produced it — silence on a fallback file is weak evidence, not a pass.

The tool is allowed to be wrong about the table. It is never allowed to be wrong about your paragraph — so it is not allowed to touch it.

PILLAR 2 OF CANONICAL-DRIVEN DEVELOPMENT

ONE CLAIM, FROM DOCUMENT TO VERDICT

Every run is the same five steps, and none of them calls a language model. The whole pipeline is deterministic, offline, and reproducible — the same tree produces the same findings on every machine, which is what lets a finding be a CI gate rather than an opinion.



NO NETWORK · NO LLM AT VALIDATION TIME · ONE PINNED DEPENDENCY (@BABEL/PARSER, WITH A REGEX FALLBACK) · NODE 18, 20, 22, 24

FIG. 3 — THE PIPELINE. STEP 4 IS WHERE DOCGUARD DIFFERS FROM A LINTER: A FINDING IS NOT ONE NUMBER, IT IS FIVE INDEPENDENT ANSWERS.

WHAT A FINDING CARRIES

A finding used to answer one question — *how worried should you be?* — with one field. That field was doing three jobs: how certain the detector was, whether a human had to look, and whether the number had ever been checked against reality. DocGuard split it into five channels that are **independent**: a blocking error can still be an escalation, and a high-confidence observation can still be one only you can judge.

CHANNEL	THE QUESTION IT ANSWERS	VALUES
severity	Does CI block?	`error` · `warn` · `info`
disposition	Who decides — the tool or you?	`act` DocGuard asserts a defect and names the correction · `escalate` the judgement is yours
confidence	How sure is the detector of its *observation*?	`high` · `low`
evidence	Has the reviewed corpus ever measured this code?	`measured` with *n* and a Wilson bound · `not-measured` (a maintainer's prior, nothing more)
parserTier	Which analyzer could see this file?	`js-ast` · `py-ast` · `regex-fallback` · `fallback-language` · `mixed`

`escalate · confidence: high`

"13 code commits since ARCHITECTURE.md was reviewed." The count comes from `git log`, so it is exact — and it proves only that a review is **due**, never that the document is wrong. Editing until it stops printing fixes nothing.

`act · confidence: high`

"README says 15 validators; the package ships 33." Both sides are counted from disk, so the tool may rewrite the number — and it stamps the provenance of the count it used, so a later reader can see the fix was about the same subject.

Across fourteen real repositories, 671 findings split 234 the tool would fix and 437 it handed to a human. It says which is which on every line.

CHANGELOG · CALIBRATED FINDING CHANNELS, SEPTEMBER 2026

THIRTY-SIX DETECTORS, AND EVERY ONE HAD TO EARN ITS DEFAULT

Each validator compares one family of documented claims against one source of evidence, grouped here by what they read — which decides how much a finding is worth.

36	73	1	0
validator modules; new rules start advisory	code scanners feeding them, two full AST parsers	pinned runtime dependency, with a regex fallback	model, network or telemetry calls at validation time
CLI/VALIDATORS/	CLI/SCANNERS/	@BABEL/PARSER	PRIVACY.MD

FAMILY	READS	VALIDATORS	THE CLAIM IT TESTS
Code truth · 9	AST: routes, schemas, env reads, imports	Docs-Sync · Docs-Diff · API-Surface · Schema-Sync · Environment · Architecture · Surface-Sync · Docs-Coverage · Generated-Staleness	“This endpoint / entity / variable / layer rule exists and looks like this.” The findings the tool fixes itself.
Structure · 8	File tree, headings, manifests	Structure (with Doc Sections) · Changelog · Document-Lifecycle · Spec-Registry · Spec-Kit · Metadata-Sync · Doc-Ownership · Decision-Records	“Required chapters exist, requirements reach tasks, versions agree, decisions name their successor.” Completeness, not correctness.
References · 6	Links, symbols, IDs across two revisions	Cross-Reference · Reference-Existence · Traceability · Test-Spec · Path-Scoped-Rules · AI-DLC	“This link resolves; this symbol exists; this requirement has a test; a path-scoped rule matches files; an AI-DLC record names real files.”
Change & time · 5	`git log`, the diff since a ref, code fingerprints	Freshness · Diff-Suspicion · Drift-Comments · Doc-Dependency · Requirement-Anchors	“Something changed near this claim.” Exact, but escalated: a review is *due*.
Numbers & self · 3	Counts on disk, typed JSON, its own package	Metrics-Consistency · Canonical-Sync · Evidence	“The number in this sentence equals the number on disk.” Includes the tool's own README, so its count claims are checked, not remembered.
Quality & hygiene · 5	Prose metrics, secret patterns, TODOs, suppressions	Doc-Quality · API-Doc-Smells · TODO-Tracking · Security · Suppressions	“Readable, not bloated, not documented in name only; no secret committed; no TODO untracked; no suppression past its expiry.” Soft by design.

A validator with nothing to validate returns **N/A with a reason**, never a pass. A green line says what it looked at.

THE BAR FOR SHIPPING ONE

Built from a method

Where a published method exists, the detector implements it (page 5). Where none does, the finding's own help text calls the heuristic a heuristic.

Run read-only on real repos

First-run conditions on production codebases; every finding hand-classified as true, false, or inert (page 6).

Kept, tuned, or cut

A detector that floods gets precision levers and is re-measured. One that adds nothing is removed — not shipped default-off.

BUILT FROM PAPERS, THEN MEASURED ON REAL REPOSITORIES

The change-aware detectors are not inventions. Each implements a published result, and each needed a precision lever the paper never had to worry about before it could run unattended in someone else's CI.

DETECTOR	PUBLISHED BASIS	WHAT WAS TAKEN	WHAT HAD TO BE ADDED BEFORE DEFAULT-ON
`Diff-Suspicion`	Outdated-comment detection, arXiv 2010.01625	A deterministic rule — prose whose tokens overlap a deleted span of the diff is suspect — reached F1 74.7, beating every post-hoc neural model. Applied to docs instead of comments.	Two signals must *both* hold (the doc references the changed file *and* shares its removed symbols); a generic-token filter; a per-doc cap. See Fig. 4.
`Reference-Existence`	Two-revision symbol check, arXiv 2212.01479	A backticked symbol present when the doc was last updated but gone at HEAD is outdated; field-tested at roughly 50% maintainer acceptance.	CLI flags excluded and the two-revision gate made mandatory — the two documented false-positive modes of the source method.
`API-Doc-Smells`	API-documentation smell taxonomy, 1,000-unit benchmark	The two smells with strong deterministic detectors: Bloated (F1 0.90) and Lazy (F1 0.95), keyed on documentation length per signature-headed section.	The three semantic smells that need a BERT model were left out on purpose and routed to staged agent judgement instead.
`Finding labels`	TRACE — calibrated explainability, IEEE TMLCN 2026	The HIGH / MEDIUM / LOW vocabulary, used as deterministic strata (a validator's check pass-ratio).	An explicit statement that the strata were never calibrated against outcomes — precision is the quantity DocGuard actually measures (page 6).
`Doc generation`	AITPG — multi-agent debate + RAG, IEEE TSE 2026	The staged prose pattern behind `generate` and `diagnose`: the CLI structures, the agent writes, the validators check.	Nothing generated is trusted: every `source=code` section is re-validated on the next run.

AND THE PAPER THAT ARGUES AGAINST THE WHOLE IDEA

Evaluating AGENTS.md (arXiv 2602.11988, revised June 2026) found that repository context files did not generally improve agent task success in its evaluated settings, and increased inference cost. It also found agents generally followed the instructions they were given.

DocGuard's README cites this study in its own “Why” section and says plainly that the results **do not establish DocGuard's effectiveness**. They motivate two things the tool now does: keep context concise (retired plans leave the tree; `memory` packs task-specific context, not everything), and measure outcomes rather than assume them.

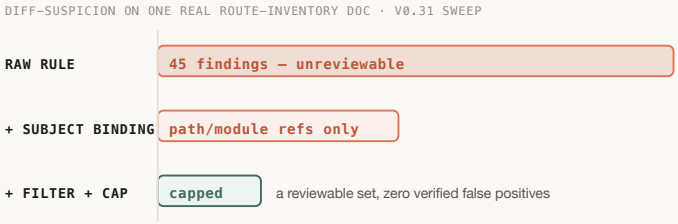


FIG. 4 — WHY A PAPER'S F1 IS THE START, NOT THE END. THE SAME RULE, THREE LEVERS LATER.

The recall-maximizing variants of these detectors were built, tested, and rejected: a false-positive flood destroys trust faster than a miss does.

VALIDATION.MD · “WHAT WE DO NOT CLAIM”

THE NUMBER COMES WITH ITS N, OR IT DOES NOT SHIP

Most tools publish an accuracy figure. DocGuard publishes a frozen benchmark, the cases behind it, the confidence interval, the caveat, and a loader that refuses to quote the number if any of those have been edited by hand.

1.000

finding precision · 12 expected findings, 0 unexpected

WILSON 95% · 0.757 — 1.000

0.000

false-positive rate on 12 clean controls

WILSON 95% · 0.000 — 0.243

0.7%

of 671 real-world findings carry any benchmark evidence at all (5 of 671)

14 REPOSITORIES · DISCLOSED, NOT HIDDEN

12 DEFECT CASES · 12 CLEAN CONTROLS · 12 REPOSITORY GROUPS · 12 CAUSAL FAMILIES



PER-DETECTOR CELLS ARE SMALL: SECURITY N=20 · STRUCTURE N=2 · ARCHITECTURE N=0 (NULL)

FIG. 5 — THE FROZEN CORPUS, BENCHMARKS/BASELINE.JSON. EVERY DOT IS A RETAINED CASE THE LOADER RECOMPUTES FROM.

THE CAVEAT, VERBATIM FROM THE ENVELOPE — THE LOADER REJECTS THE FILE IF THIS TEXT IS CHANGED

> Benchmark precision on a deliberately balanced corpus of 12 defect and 12 clean-control cases … This is DocGuard's precision on labelled cases, **not the probability that a finding in your repository is real**; quote every ratio with its n and Wilson 95% bound.

DocGuard does not publish a calibration document. The real base rate of a stale claim is nowhere near the corpus's 50/50 split, so a probability read off this corpus would mislead — and the README says so in the same paragraph that cites the number.

THREE MECHANISMS THAT KEEP THE NUMBER HONEST

Fail-closed loader

The baseline is an envelope of retained cases plus derived ratios. The loader recomputes every ratio from the cases and refuses an envelope whose numbers, caveat, or measure have been hand-edited. A test asserts each rejection path; TestGuard probes that the guard cannot be silently removed.

Adjudication record

The corpus could only absorb a false positive that had already been repaired — so a disagreement the maintainers reviewed and declined to act on left no trace, and "precision 1.0" described the contribution pipeline rather than the detectors. An adjudication row now records the disagreement without corrupting the measurement.

Feedback samples the unmeasured

`docguard feedback` used to sample only findings the confidence label already doubted — the channel that would validate the label sampled nothing it could learn from. Selection is now "unmeasured code, or low confidence", and the command states what it left out.

"Precision 1.000" is true, and it is a claim about twelve labelled cases. The tool is built to stop that sentence from ever being shortened.

SPECS/007 · INDEPENDENT PRECISION EVIDENCE LOOP

FIELD REPORTS BECOME TESTS, NOT TICKETS

More than a hundred releases in under eight months. The cadence is not the point; what fed it is. Each report from a real working session — most of them written by an AI agent that hit friction on a real tree — was reproduced as a regression test *with a control*: the pre-fix behaviour, asserted, so a future change cannot silently re-break it.

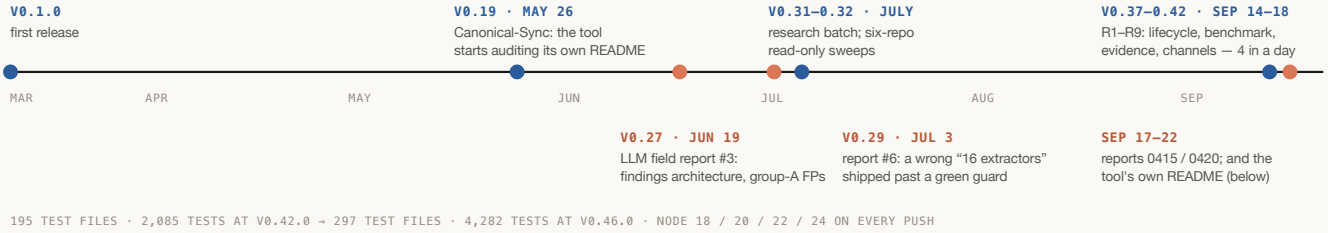


FIG. 6 — BLUE ABOVE THE LINE: CAPABILITY. CLAY BELOW: SOMETHING REAL BROKE, AND BECAME A TEST.

DOGFOODING, INCLUDING THE PART THAT IS EMBARRASSING

DocGuard guards its own repository on every push: `guard`, `score`, `diff`, `badge`, and the packed npm tarball run without installed dependencies. Its README count claims are machine-governed. Its own claims are fault-probed by its sibling, TestGuard. Three things that discipline found:

found · v0.32

The semantic-claim extractor ignored `.docguardignore`; an excluded audit contributed 28 of 39 "unverified claims". Fixed; the count fell to 12 — and one of those 12 was **real drift**, a stale suite-runtime claim in TEST-SPEC.md.

found · v0.32

REF002 flagged DocGuard's own source: doc comments used realistic `ADR-012` examples and fixtures cited ADRs. Non-product scoping and digit-free placeholders; shipped with zero self-findings.

missed · Sep 22

The README said "33 tests" from March 15 through all 92 releases while the suite grew past 2,000 — and every self-guard was green. Metrics-Consistency had computed the test count since v0.8.2 and never compared it to anything. A human found it, not the tool.

WHAT THE FIX COULD AND COULD NOT DO — PR #448

```
declared test cases (static, AST)   1,721     reported by node:test   2,085
// loops generate cases a static count cannot see — so the static number is a FLOOR.
MET003 flags a documented count below the floor (stale for certain) and passes anything above it.
It carries no mechanical fix: writing the floor would replace a stale number with a wrong one.
```

A green self-guard proves the checks that exist passed, not that the right checks exist. The honest version of this story — a half-built check, a human reader, a lower-bound design that refuses to guess — is the loop this page is about, and it is why the tool's limitations page comes next.

FROM CHECKING THE MAP TO GOVERNING HOW IT CHANGES

Through 0.42, DocGuard asked one question: *does the documentation still match the code?* Versions 0.43 and 0.44 add the questions around it — which section a change should have touched, whether a feature went through its spec first, what an agent needs to read, and whether the tool got slower or noisier. Fifty-one specs (014–064) were written, planned and tasked before their code; thirty-four carry TestGuard claims probed with planted faults.

51

specs delivered through the Spec Kit pipeline since 0.42

SPECS/014 — SPECS/064

55%

smaller MCP guard response: each fact once, nothing lost

SPEC 037 · BUDGET FIXTURE

10%

growth in agent-facing bytes or package size before a PR must state a reason (guard time: 15%)

SPEC 031 · EVERY PR

3,281

tests on Node 18, 20, 22 and 24 at v0.44.0

NPM TEST

AREA	WHAT CHANGED	WHY IT MATTERS
Docs know their code	A section declares the code it describes (`covers=`); a reviewed fingerprint lock names the section to re-read when that code changes. Module and entity diagrams are drawn from code.	Freshness said a review was *due* somewhere. This says *which paragraph*, and why.
Spec first, enforced	A spec-first gate classifies each change to governed paths as covered, exempt or uncovered. Completions record the reviewed revision and survive squash merges. As-built specs describe older code.	"We follow Spec Kit" becomes a checked property of the history, not a habit.
Agents read less, better	MCP doc tools answer *which docs describe this file* and return one bounded section. Path-scoped rules report which instruction files each harness loads for a path.	The AGENTS.md study on page 5 measured cost; these cut what an agent must read to find the rule that applies.
The tool measured	Budgets run base and head side by side on every PR. An install older than 14 days tells the agent so — from its own CHANGELOG, no network call — and names `docguard upgrade` .	A guard that slows down or grows noisier erodes trust the same way drift does.
Reads real projects	Served routes for every framework it detects, NestJS to ASP.NET Core; unreadable paths counted, never guessed.	Gaps found on ten real repositories, closed by specs 040–064.
Exceptions explained	Ignore entries carry a reason and an expiry; `docguard ignore` lists every suppression.	A suppression now says why, and until when.

kept · a promise with a test

The compact guard response was meant to lose nothing. A reconstruction test reads every field back and caught the first draft dropping validator messages.

held back · measured

The symbol map ships **opt-in**. A frozen 54-trial benchmark found no regression but no benefit on navigation tasks (steps unchanged, latency 6% lower against a 15% bar), so it stays off by default.

Every feature in these releases started as a written spec.

SPECS/014 — SPECS/064 · TESTGUARD.CLAIMS.JSON

FROM GOVERNING CHANGE TO PROVING EVERY VERDICT

Through 0.44, a green line meant the check found nothing. Versions 0.45 and 0.46 make it say what it *looked at*: which function a requirement lives in, and whether a check could observe what it needed. Ideas from AWS AI-DLC and DORA were rebuilt here, not copied.

22

specs delivered through the Spec Kit pipeline since 0.44

SPECS/070 — SPECS/092

6

closed verdicts; “could not look” is never a pass

SPEC 081 · EVERY VALIDATOR

0

findings lost across 45 real projects, release against release

HERMETIC A/B · V0.45.2 → 0.46

4,282

tests on Node 18, 20, 22 and 24 at v0.46.0

NPM TEST

AREA	WHAT CHANGED	WHY IT MATTERS
Requirements in the code	An `@implements` tag anchors to the function under it, and turns suspect when that code or the requirement changes.	Traceability that knows *which function*, not just which file.
Every check decides	Each validator ends in one of six verdicts. A git call that fails is a recorded gap, never a finding.	“Could not look” stops reading as “looked and found nothing”.
Rules you grow into	New rules ship advisory; `docguard policy` shows which ones CI history says are ready to promote. Suppressions name an owner and an end date.	Stricter over time, on evidence from your own runs.
Chains and decisions	Requirements must reach tasks; ADR supersession must run both ways; authoring guidance comes from the registry the validators read.	The tool’s own skill had told agents to write a section its checker rejected.
AWS AI-DLC	Reads AI-DLC’s records and memory; a plugin sensor fails a gate on drift the stage introduced, not drift it inherited.	A gate that fails on old warnings gets switched off.
DORA evidence	`score --dora` maps DORA’s documentation items and AI capabilities to repository evidence.	DORA found documentation multiplies what every practice pays back.
Releases compared	Each release is compared with the last, finding by finding; nineteen parsers are fuzzed on every run.	A batch of fixes cannot quietly change what users see.

found · by its own fuzzer

A 265 KB line of HTML comments took the comment reader 9.7 seconds. Now under a millisecond, with identical output.

kept honest · advisory first

The release A/B added 159 findings across twelve projects, all advisory: listed, never counted against an exit code.

A check that could not see is not a check that passed. Since 0.46, DocGuard says which one it was.

SPECS/070 — SPECS/092 · CHANGELOG.MD

WHAT IT DOES NOT CLAIM

- **No recall guarantee.** Precision-first means some real drift is missed by design. The variants that caught more were tested and rejected because they also flooded.
- **No judgement of prose.** Whether a paragraph is *true* is never auto-decided. Exact declarations in `.docguard-evidence.json` can verify selected statements; every other statement stays unverified, and the badge prints how many.
- **No calibrated probabilities.** HIGH / MEDIUM / LOW are deterministic strata. The one number DocGuard measures is precision on labelled cases, with n and a bound, and a caveat that cannot be deleted.
- **Not proof that context helps.** The AGENTS.md study on page 5 is cited in the README precisely because it cuts the other way. What DocGuard offers is a way to find out on your repository, with outcomes measured.
- **Soft by default.** The research-backed detectors ship as low-confidence warnings. Exit code 2 still fails an ordinary shell step; the severity policy is yours.
- **A maturity score is not a verdict.** 100/100 means the map is whole; only `guard` says whether it is true. Nor is `score --dora` a DORA survey.

WHERE IT FITS

ALONGSIDE	WHAT THAT DOES	WHAT DOCGUARD ADDS
GitHub Spec Kit	Generates feature specs, plans, tasks — spec → code, once.	Governance after generation: drift tracking, lifecycle (retire what shipped), spec-quality validation. DocGuard is an official Spec Kit community extension.
AGENTS.md	One instruction file: build, test, style.	AGENTS.md is one of the required files. DocGuard validates it, packs task-specific context from the canonical docs, and keeps every agent on the same map.
Kiro · Cursor rules	IDE-bound spec or rule files.	Portable across any IDE and any agent; enforced in CI rather than read by one editor.
AWS AI-DLC	A gated lifecycle where agents write each stage's artifacts.	A plugin sensor that fails a gate on new drift; reads AI-DLC's records and memory.
TestGuard	Breaks a promise on purpose and asks whether a test noticed.	The other half of the same question: whether the *documentation* of that promise is still true. They share an evidence format, and TestGuard fault-probes DocGuard's own claims in CI.

THREE MORE THINGS THAT FOLLOW FROM IT

Ships everywhere, phones nowhere

npm and PyPI packages with provenance attestation, Homebrew, a GitHub Action, a Docker image, an MCP server and Claude Desktop bundle, a Spec Kit extension. No telemetry, no network at validation time.

Evidence, not output

JSON, SARIF and JUnit with all five channels on every finding; a badge that prints the unverified-claim count next to the check count; a benchmark envelope with its cases attached.

Built for AI-written code

When an agent writes both the code and the docs, the failure mode is confident, plausible documentation of a system that no longer exists. A deterministic comparison against the tree can see it.

A score tells you the map is complete. DocGuard tells you which parts of the map still match the territory — and which parts it cannot tell.

GITHUB.COM/RACCIOLY/DOCGUARD · NPM DOCGUARD-CLI · MIT · RICARDO ACCIOLY