// WHITE PAPER · PUBLISHED 18 SEPTEMBER 2026

Hosted Code Review: The YAML Parsed and Nothing Read It

How a registry of past defects outgrew every reviewer meant to read it, and the four moves that got it back inside the window

Jason Newell · MAX Research Collective · Report WPR-MAX-018 · Published 18 September 2026
Licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0), https://creativecommons.org/licenses/by/4.0/
Original: https://www.maxresearchcollective.com/articles/hosted-code-review.html
Cite as: Newell, J. (2026). Hosted Code Review: The YAML Parsed and Nothing Read It. MAX Research Collective, WPR-MAX-018. https://www.maxresearchcollective.com/articles/hosted-code-review.html

The change that parsed and did nothing

One entry went into the configuration of a hosted code reviewer in August. (A hosted reviewer is a commercial service that reads a pull request, meaning a proposed change to a codebase, and comments on it the way a colleague would.) That entry pointed the service at a set of instruction files. Three separate graders would now work from one shared body of criteria instead of three private ones. The configuration parsed. A test went green. Everything shipped.

It did nothing at all.

Entries of that form are directory-scoped, a detail buried in the vendor's own documentation and easy to read past. Such a guideline governs one folder and everything below it, and nothing outside. Every rule inside those files aimed at the application trees. The files themselves sat in a configuration folder somewhere else entirely. So no rule reached a path it had been written for. Three graders carried on reading what they had read the week before.

A comment sits above that entry now, and it carries the whole lesson: the test we validated it with only proved the YAML parsed, which is not the same claim.

That shape of failure is the one worth being afraid of. Something opens the first file, reports success, and never touches the rest. The board stays green. In the defect registry it files under surrogate metrics — a measurement that stands beside the thing you care about and then moves independently of it. Two of the three graders flagged the entry once it reached review. That is at least small evidence that two of them were reading something.

What follows is the fix, and it starts with the size of what the graders were supposed to read.

Two bounds, and they fail differently

Every automated reviewer is bounded twice. The two bounds get discussed together; they behave nothing alike.

BOUND	WHAT IT LIMITS	HOW IT FAILS
Size	The tokens of instruction and diff it will accept	Loudly. The request is rejected and somebody sees it.
Shape	Which files it will look at at all	Silently. A skipped file looks exactly like a clean one.

Table 1. The two ceilings on a hosted reviewer, and why only one of them gets noticed.

A token is the unit a language model counts in: roughly three quarters of an English word. Size is a hard number, published by the vendor, and it announces itself the first time a request crosses it. Shape is a set of filters, mostly written by the customer, and it announces nothing ever. Take two reviewers: one never opened the file, and the other opened it and found nothing wrong. They hand back the same artifact, which is silence.

Size cost about a week. Shape cost the August change above, and it would have cost considerably more if two graders had not caught it in the act.

The size, measured

One grader loads a single instruction file into every review it runs. The published ceiling for the whole prompt is 110,000 tokens. That file had reached 365 KB — roughly 90,000 tokens — so about eighty percent of the budget was gone before a line of the proposed change had been read. Reviews from that lane failed with a complaint that the prompt was too big.

Diagnosis arrived by accident. The same failure fired on a documentation-only change whose diff ran to a handful of lines. That ruled out the size of the change and put the blame where it belonged. A large diff would have been the comfortable explanation; the uncomfortable one is that the instructions had eaten the window.

Here is what the governing files weighed once somebody measured them instead of estimating.

FILE	SIZE	TOKENS, APPROXIMATE	WHO LOADS IT
The defect registry	645 KB	161,000	Nobody directly; it is the source the rest derive from
The document index	248 KB	62,000	People, and working sessions
The criteria file	240 KB	60,000	One hosted grader, as its code guidelines
The instruction file	175 KB	44,000	A second hosted grader, as a custom instruction file
The standing rules	150 KB	38,000	Working sessions

Table 2. The five governing files, measured after the carve described below. The instruction file stood at 365 KB before it.

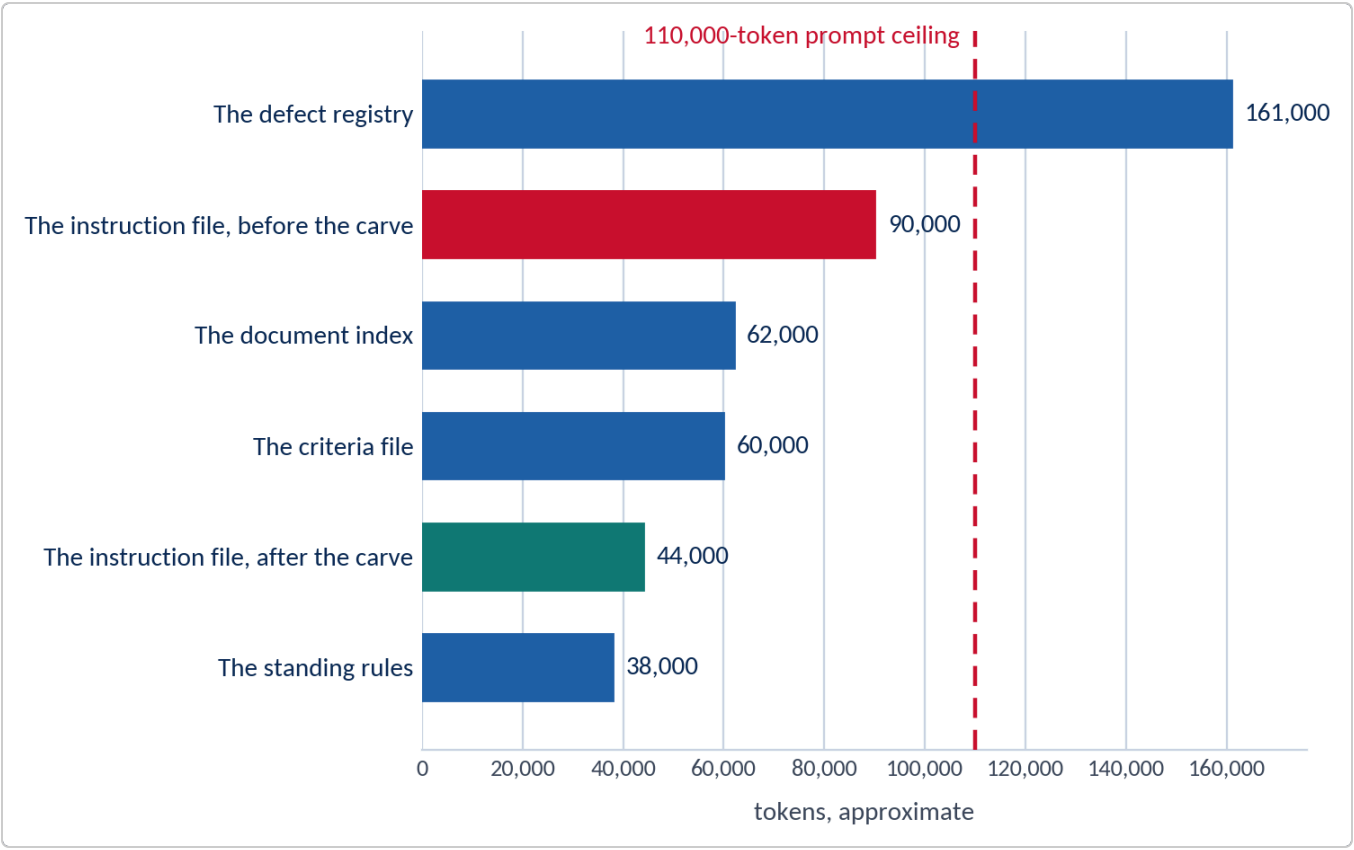


Figure 1. The governing files against the 110,000-token prompt ceiling, with the instruction file shown before and after the carve.

Set the registry against that ceiling and the arithmetic stops looking like a tuning problem: the registry alone runs to one and a half times the entire window. It is also the number most likely to keep climbing, since a window is fixed and a record of defect classes only ever grows. Each entry holds a kind of bug that already shipped here and already failed. It carries three things: the symptom, the root cause, and the check that would have caught it. Nothing leaves. Where a class can be mechanised it becomes a lint rule and the entry retires itself. That trims the growth rate; it never reverses it.

An append-only record and a fixed window diverge. That is arithmetic rather than a vendor defect, and past some point the two simply part company.

One repository makes the pressure look gentler than it is. The same standard governs a fleet of them. Each tree accumulates its own classes and carries its own copy of the criteria, so the total climbs on two axes at once: the number of repositories, and the age of each. A single repository crossing a ceiling is a

scheduling problem. A fleet crossing it at different times, quietly, is what keeps a maintainer awake.

The shape, which is worse

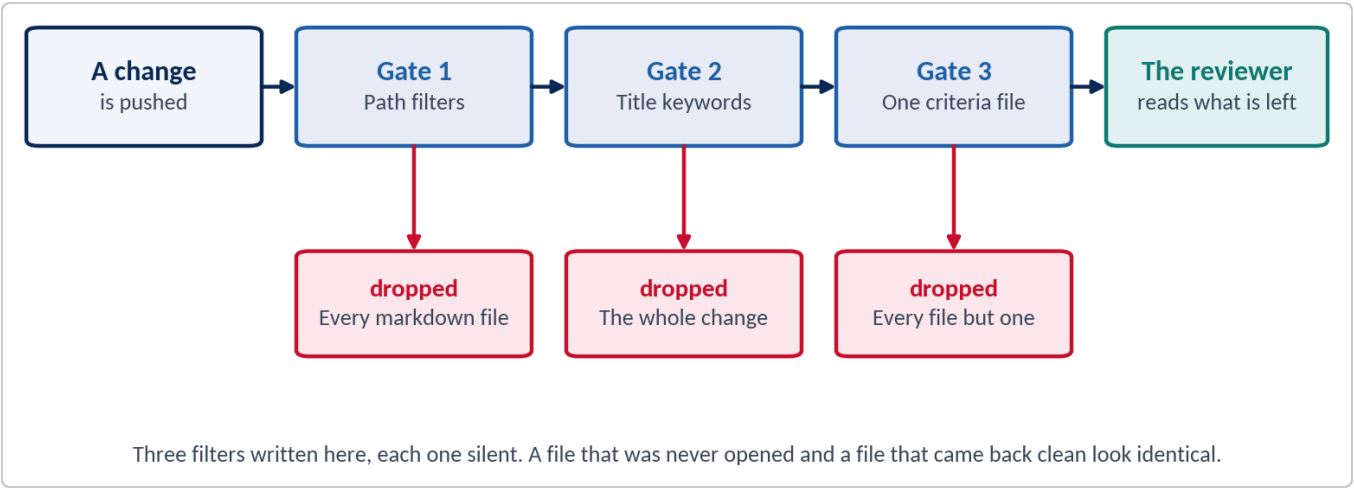


Figure 2. The three gates a proposed change passes before a hosted reviewer ever sees it, and what each one drops.

Eight constraints stand between a change and a hosted review here. Five came out of a configuration written in this building. That is the part that stings.

CONSTRAINT	SOURCE	CONSEQUENCE
Markdown excluded from diff review	Our configuration	Every documentation file is invisible to the reviewer
Title-keyword skip	Our configuration	A change labelled as documentation is skipped entirely, and silently
One criteria file	Our configuration	Exactly one file of ours is read as review criteria
Negative-only filter lists	Vendor behaviour	A single positive pattern flips the list into an allowlist and drops every unmatched file
Directory-scoped guidelines	Vendor behaviour	An entry governs its own folder and nothing outside it
Ten reviews in a rolling hour	Vendor plan	A push inside a fix loop can meet a throttle where a review should be
Twenty requests a day	Vendor free tier	The lane returns a quota error and stops
Container-based action, Linux only	Vendor packaging	The build machine here runs Windows, so that lane cannot run at all

Table 3. Every gate between a change and a hosted review. The first three were written here.

Read the third row on its own and the design problem stands out. A hosted grader reads exactly one file of ours. The registry stays shut, and so do the standing rules and the document index. Everything a reviewer ought to know about this codebase has to fit through that hole, and the hole keeps its width while the knowledge grows.

Four moves

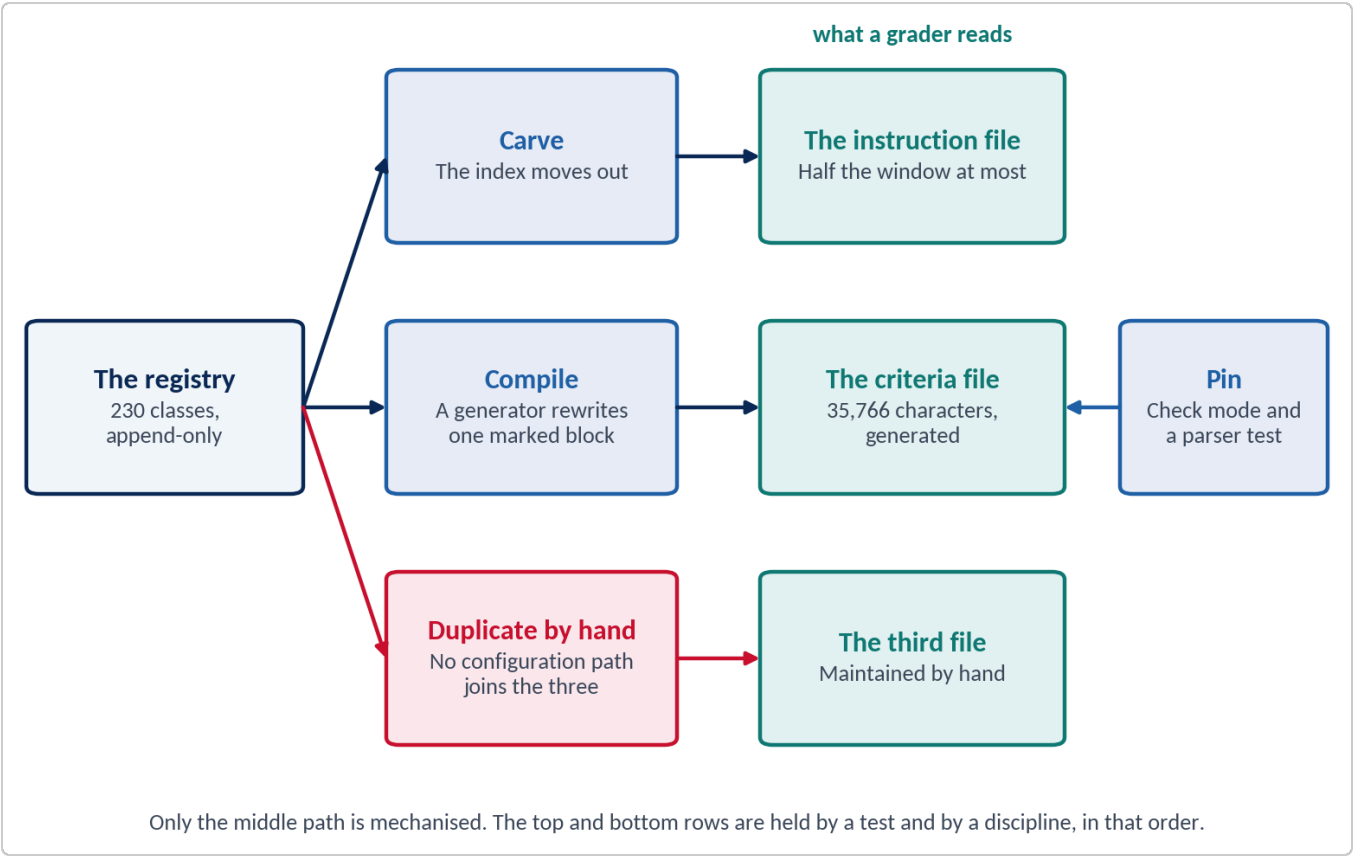


Figure 3. One append-only source, compiled into the file the reviewer reads, pinned by the build, and written by hand into the two places a generator cannot reach.

Carve

Take out of an instruction file everything that does not need to be there. A 219-row index of documents accounted for 55.4 percent of that file: more than half a review budget spent on a table of contents. It moved to a file of its own at the top of the repository, so every relative link inside it kept resolving. What remained came down to 175 KB.

A system test holds the line now, and it asserts the budget rather than a byte count (a byte count invites somebody to raise it by one). Half the grader’s window is the ceiling. Its failure message ends with a sentence aimed squarely at whoever reads it at eleven at night: do not raise this limit to make the test pass.

Compile

No reviewer can be made to read more files. Any reviewer can be handed more inside the one file it already reads.

So a generator walks every class heading in the registry and rewrites a marked block inside the criteria file. Two hundred and thirty classes land there as 35,766 characters, deterministic, with no model anywhere in the path. Copying entries across by hand was the alternative; an append-only registry copied by hand drifts inside a week. Compilation turns a file the reviewer will not read into a section of a file it will.

Pin

A generated file nobody checks is a stale file with extra steps.

One test runs the generator in check mode and fails with the exact command needed to regenerate. A second pins that the parser sees every heading in the registry, so a new heading style cannot quietly drop classes out of the reviewer's working set while every check stays green. That second test exists because the first alone would have caught nothing.

Pinning also has a shape inside the build, learned at cost. Both the registry and the criteria file are classified as code rather than documentation, which keeps them off the documentation-only fast path. One change added a single bullet, took that path, skipped the test shards, and merged green. It left the local reviewer's model definition stale on the main branch — briefing it on criteria that no longer matched their source. The next change along ate the red build that should have fired on the first.

Duplicate, deliberately

Three graders read three different files, and no configuration path joins them. August settled that question.

So a new defect class goes into the registry, into the criteria file, and into the third grader's instruction file, by hand, every time. Only the registry block inside the criteria file is generated. That duplication is a stated discipline rather than a mechanism, and it is written down as one; a discipline somebody believes is a mechanism is a discipline that quietly stops being performed.

Three claims that get conflated

Underneath the sizes and the filters sits a harder question. How do you confirm a reviewer read what you gave it?

CLAIM	STRENGTH	WHAT SHOWS IT
The configuration parses	Weakest, and it proves nothing	A schema test
The file was loaded	Real, and available per run	A receipt: a byte count, an entry count, a tool trace
The content was applied	Strongest, and largely unprovable from outside	A finding that could only have come from that content

Table 4. Three different claims that a single green check is regularly asked to stand in for.

August was a first-row test being read as a third-row guarantee. Schema satisfied, everybody moved on.

What a receipt looks like

On the hosted side the only honest receipt is a reviewer's own tool trace, and exactly one of the three services emits one. Its review body on a single pull request carried the traces inline: a knowledge-base load returning 13,343 bytes, then two repository-wide searches returning 8,272 and 38,100. On that run the service demonstrably went past the diff and read the tree. Stated precisely, that is

evidence about one run rather than a guarantee about every run. No equivalent exists from the other two. One offered only its complaint about prompt size; the other offered a quota error.

A softer signal came out of the same change. That service raised a finding. Shown contrary evidence in a reply, it re-ran its own inspection, withdrew the finding, and wrote down what it had learned. Any lane that can be moved by evidence is reading something.

Locally the receipt is structural instead of lucky. A report header states five things every run: which model answered, over what window, how many findings came back, how many registry entries the brief carried, and whether each supporting piece loaded (the teaching corpus, the structural context, the review slices, the build notes).

Two details there carry the weight. Registry entries render as a count rather than a flag. A failed read shows as unavailable, never as a quiet zero, because "the brief held all 230 classes" and "the brief loaded" are different claims. Every line renders whenever it is supplied, including the cheerful ones. An ungoverned context window went unnoticed here for months for a single reason: nothing ever printed it. Anything able to stop loading quietly has to be made unable to do it quietly.

Sitting under that is an honest-empty rule, which matters more than it looks. An unreadable ledger degrades to a brief that states it is empty; it never degrades to an empty string. Splice in an empty string and a section heading renders with nothing beneath it, which a model reads as other reviewers having found nothing rather than as a collection that has not happened yet.

Which leaves the strongest claim available, and it is narrower than anyone would like. Nobody here can prove a vendor read all 230 classes. Anybody here can prove, on every commit or the build fails, that all 230 sit in the one file the vendor says it reads.

What none of this fixes

LIMIT	THE HONEST STATEMENT
The ceiling moved and it did not go away	The criteria file now stands at 240 KB, roughly 60,000 tokens. Nothing available here proves a grader ingests all of it rather than truncating.
Loading is not applying	Every receipt above proves ingestion. A model can be handed 230 classes and reason from none of them.
Our own filters hide work from review	The markdown exclusion and the title skip together mean this very article receives no hosted diff review. That is a deliberate trade against the quota, and it is still a gap.
One run of evidence is not a guarantee	The tool trace covered one pull request. No per-run assertion exists on the vendor side, and none can be bought.
Duplication is a discipline and not a mechanism	Three files updated by hand per class. Only one block in one of them is generated.
The local lane is not the better reviewer	A 27-billion-parameter model on a workstation will not out-reason a frontier model on a novel logic bug.

Table 5. The limits that survived the fix, each stated as the strongest true sentence available.

The mismatch underneath

One shape shows up at every layer here once you have seen it once. Something of ours grows; something of theirs is fixed.

Defect classes accumulate; a context window does not. Output rate climbs, because AI-assisted development moves the number of pushes and leaves the quota exactly where it was. A per-push meter stays put. Repositories multiply, and the one file a hosted grader will read stays one file. Each time the answer is the same move

wearing a different name: measure what is actually being sent, take out whatever fails to earn its place, generate whatever can be generated, then pin the result so the next person cannot quietly undo it.

None of which settles the question that August entry asked. The receipt problem has no vendor-side answer, so it gets solved on the side of the machine that can be opened — which is why the reviewer worth running first is the one that can be made to show its work, rather than the one most likely to be right.

Glossary

TERM	DEFINITION
Append-only registry	A record that accepts new entries and never removes old ones. Here it holds defect classes, so a class that shipped once cannot quietly leave the institutional memory.
Context window	The bounded amount of text a language model can attend to at once, counted in tokens. It is fixed for a given model, and instructions, code and the model's own answer all share it.
Custom instruction file	A file a hosted reviewer loads into every review as standing criteria, separate from the change under review.
Defect class	A kind of bug rather than a single occurrence, recorded with its symptom, its root cause, and the check that would have caught it.
Diff	The set of lines a proposed change adds, removes or alters. A reviewer handed only the diff sees the change and nothing around it.
Directory-scoped	Applying to one folder and everything beneath it, and to nothing outside. A rule written that way reaches only the tree it sits in.
Gate	A check returning pass or fail over a whole change, as distinct from a reviewer, which returns findings. Counting a gate's failures as findings makes a lane look most productive on its worst days.
Hosted reviewer	A commercial service that reads a proposed change and comments on it, billed per push or per request.
Lane	One reviewer or gate that can look at a change and answer. The roster of them is declared once, and everything else derives from it.
Path filter	A configuration rule naming which files a reviewer may look at. A negative-only list silently becomes an allowlist the moment one positive pattern joins it.

TERM	DEFINITION
Pull request	A proposed change offered for review before it joins the main line of a codebase.
Quota	The number of reviews or requests a vendor plan permits in a window. It is the denominator that AI-assisted development leaves untouched while moving the numerator.
Receipt	A per-run statement of what a reviewer actually loaded, carrying counts rather than flags, so ingestion can be checked instead of assumed.
Throttle	A vendor-side pause arriving where a review should be. Counted as a review, it makes a lane look busiest when it was least available.
Token	The unit a language model counts in, roughly three quarters of an English word.
YAML	A plain-text configuration format. It parses happily into a structure that means nothing, which is the property this article is named after.

MAX3 WPR-MAX-018

// ABOUT THE AUTHOR

Jason Newell leads organizations and people in technology, and is a researcher, deep technologist and builder — from a CRAY to agentic AI. He founded MAX Research Collective to build AI for regulated industries that runs where its owner has control, with healthcare as its strongest focus. About Jason → Connect on LinkedIn →

Newell, J. (2026). MAX Research Collective, WPR-MAX-018. CC BY 4.0