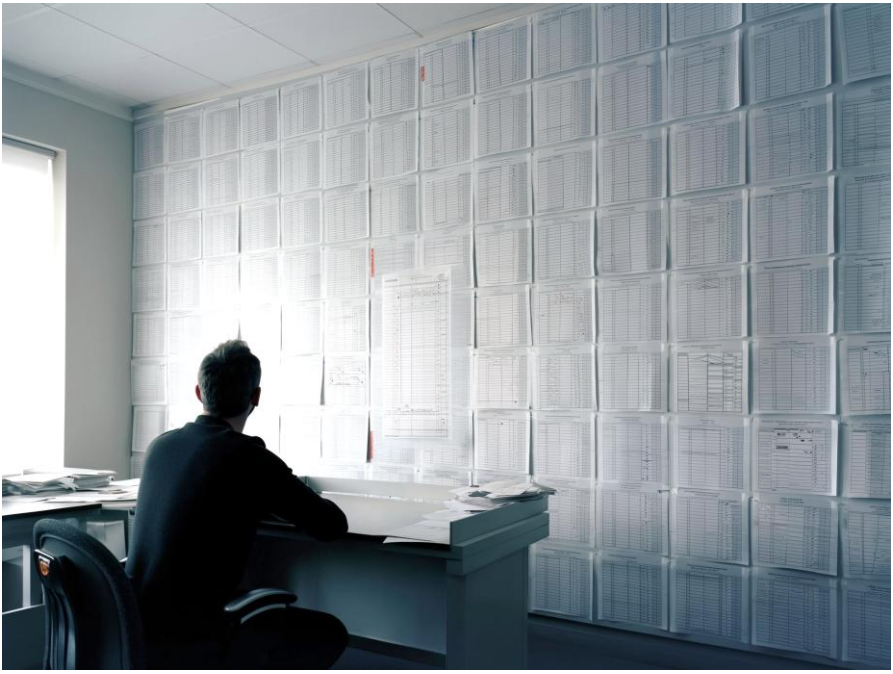
I don't want to spend the rest of my career reviewing AI outputs



Arguably, generative Artificial Intelligence (AI) can be added to safety flows as long as we keep humans in the loop – humans can check the AI outputs as one might the output of a junior engineer. However, AI can vastly accelerate output generation and simply overwhelm reviewers with data. How, therefore, can we assure the AI outputs, especially when tool quality increases and errors become rarer? Ben Fulford discusses these issues and presents an approach to filtering the data to focus on what's important from a safety engineering perspective.

I used to think the answer to putting AI into safety workflows was simple enough. Keep humans firmly in control of every output. Treat what the model produces the way you would treat work from a junior engineer: read it, question it, check the reasoning, sign it off. Do that thoroughly and the risk is managed.

Over the last few months I have stopped believing that is feasible.

I have been building a GraphRAG (Graph-based Retrieval-Augmented Generation) tool for hazard identification. It reasons over a structured model of the system and produces hazard analysis that is accurate and well-evidenced, and it produces a great deal of it. Thousands of lines of technical content in a few minutes.

The first time I sat down in front of a full run, I could not work out where to start. The quality was there. The volume defeated me.

The problem the tool was built to solve is a real one. Hazard analysis is slow and expensive. It occupies experienced engineers for days at a time, it usually gets done once, and the amount a programme can afford is set by how many engineer-days can be found rather than by how much analysis the system warrants. The tool removes that constraint almost completely, and I am convinced that what I have built is a real step forward in how this work gets done.

Removing that constraint created a second problem. Somebody now has to sit down and confirm that all that output is correct. The point of building it was to give engineers their time back. What I built alongside it was a new job, and it is not a job I want to do.

This is worth working through in public, because it is not specific to my tool. Any team putting generative AI into an assurance workflow will meet this within weeks of the tool starting to work properly.

A world of abundance

For most of history, information was scarce. Creating it took effort, recording it took materials, and copying it took somebody's time. That cost acted as a filter. If a thing had been written down and kept, somebody had decided it was worth the trouble, and the reader could take that decision as a signal.

Generative AI almost completely removes the cost of production. Set alongside the infinite scroll feed and the thousand-word email that used to be three lines, we now live with an abundance of information and no matching increase in our capacity to take it in. The filter that scarcity provided has gone, and nothing has replaced it. Herbert Simon saw this coming in 1971 [1]:

"We now live with an abundance of information and no matching increase in our capacity to take it in"

> *"In an information-rich world, the wealth of information means a dearth of something else: a scarcity of whatever it is that information consumes. What information consumes is rather obvious: it consumes the attention of its recipients. Hence a wealth of information creates a poverty of attention."*

Attention is now the scarce resource. The people who can hold theirs on the thing that matters, in an environment engineered to fragment it, have an advantage over everyone else, and that advantage is growing.

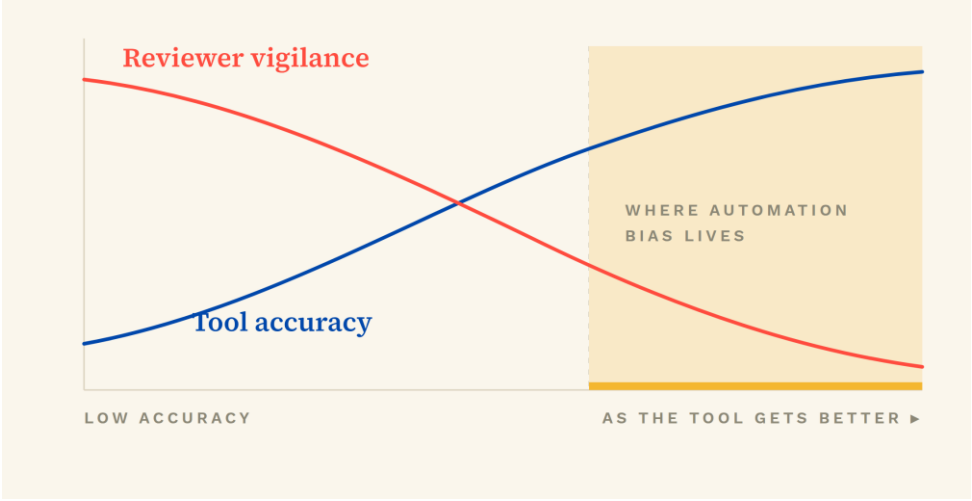
The same logic applies to engineering work, with one difference: we get to design the system rather than rely on individual discipline. Safety engineering has always been an exercise in directing limited attention. A hazard log entry cost somebody an afternoon of thought, so entries were deliberate. A safety case was long because somebody spent months on it. Scarcity did a lot of that filtering for us, without anybody designing it that way. With generation now close to free, the filtering must be designed deliberately, and I think the safety engineering community is better placed than most to design it.

Why reading everything stops working

Reading everything fails for two reasons, and the second is worse than the first.

The first is arithmetic. A hazard and operability study (HAZOP) team working through a two-day workshop might produce a few hundred worksheet rows. The same coverage now comes back in an afternoon, at greater depth, with the option of running Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA) and System-Theoretic Process Analysis (STPA) over the same system model on the same day. The volume of analysis went up by an order of magnitude. Review capacity did not move. Adding reviewers is the obvious answer, and it is a poor one, because the cost curve is linear in exactly the dimension the tool made cheap.

The second reason is human factors. At low accuracy, errors are uncomfortable and visible. The failure rate is high enough to feel, so engineers stay alert. As the tool improves, trust rises and checking drops. Automation bias sets in; the tendency to accept an output because outputs are usually right. The errors that survive this are the ones that most resemble correct answers, which makes them hardest to catch at precisely the point people are looking least carefully.



Lisanne Bainbridge described the mechanism in 1983 [2], forty years before a Large Language Model (LLM) walked into it. Ironies of Automation is four pages long and every paragraph of it applies here. Three lines in particular:

> *"It is impossible for even a highly motivated human being to maintain effective visual attention towards a source of information on which very little happens, for more than about half an hour."*

> *"Physical skills deteriorate when they are not used, particularly the refinements of gain and timing."*

> *"By taking away the easy parts of his task, automation can make the difficult parts of the human operator's task more difficult."*

Every control-room lesson in that paper maps onto an engineer working through several hundred rows of AI-generated analysis. Human in the loop, implemented as a person reading everything, is a control that degrades as the tool gets better. That is worth stating plainly, because it is the control most organisations are currently relying on, and the one most likely to turn up unexamined in a safety argument.

"Human in the loop, implemented as a person reading everything, is a control that degrades as the tool gets better"

There is a third cost, less dramatic and more corrosive. Reading everything spends your most experienced engineers on the part of the work that needs them least.

What software engineering already did

Software engineering worked through a version of this problem, and the answer is instructive.

A compiler converts source code into machine instructions. Very few review the machine instructions. That was not always obviously safe, and early compilers were distrusted for good reason. Trust was earned by building apparatus around the tool rather than by checking its output line by line, in three steps.

- One: constrain what the tool can produce. Type systems, language rules and a defined specification of valid output mean whole classes of error cannot be expressed in the first place.
- Two: verify the output against that specification automatically. Test suites check behaviour against what the code was supposed to do. The check is machine-run, repeatable and cheap enough to run on every change.
- Three: monitor in service and make reversal cheap. Production monitoring catches the errors that got through and the drift that builds up over time, and rollback is a decision rather than a project.

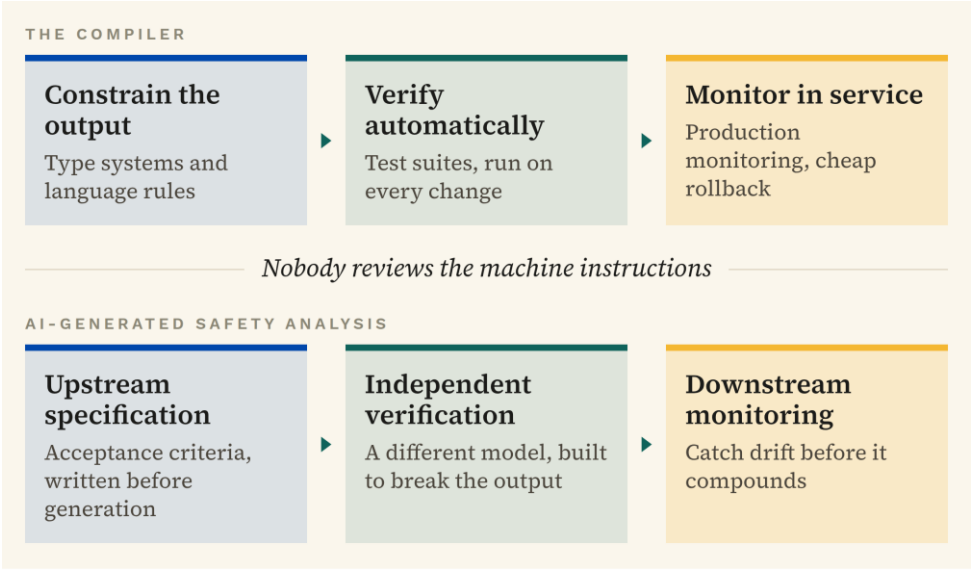
Tool qualification under IEC 61508 [3] and RTCA/DO-178C [4] formalises the argument. The manual middle was removed, and the profession is more confident in compiled output today than it was when people were checking it by hand.

That structure maps onto safety assessment directly.

Upstream specification. Define what a good output has to satisfy before generation starts, precisely enough to test against. The standard in play, the risk appetite, the acceptance criteria, the scope boundary. Most teams do this after the fact, inside a reviewer's head, one output at a time. Writing it down first is what makes everything downstream possible.

Independent verification. A second model, ideally a different architecture, built specifically to try to break the output against that specification. This is the only layer that scales at the volume these tools produce. It is also the layer people are most sceptical about, and the scepticism is well founded: the checker has failure modes of its own, and they correlate with the generator's if they are careless about diversity. Common-cause failure is a live concern here, and it is one this field already knows how to reason about.

Downstream monitoring. Track how the review process performs over time, with enough granularity to catch drift before it compounds. If a class of finding is being accepted and later found wrong, that is a process capability problem and it should surface as one.



One result from my own build is worth reporting here, because it bears directly on where the effort should go. On the GraphRAG tool, the guardrails, validators and self-checks I put around the model improved output quality more than swapping in a more capable model did. Engineering the system around the AI mattered more than the AI. I did not expect the gap to be as large as it was, and it is the single finding that most changed how I am spending development time.

Design the output to be cheap to check

There is a fourth move, and it changes the economics more than any of the above.

Verifying a claim costs far less than generating it, provided the output was built for verification. A finding that arrives as a paragraph of prose has to be re-derived before anyone can check it. The same finding, arriving as a typed claim marked as fact, inference or assumption, carrying a citation into the system model it came from, can be checked by following the citation. Identical content, an order of magnitude difference in review cost.

The tool I am building carries a claims ledger for this reason. Every claim has a status and a provenance link. Review burden falls with no change to the underlying model.

This argument is usually made from trust and regulation. The version engineering managers act on is the labour economics one.

Consequence and confidence

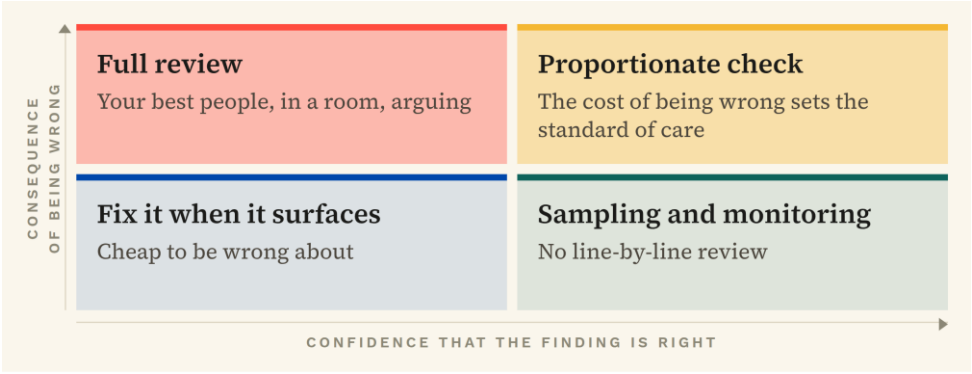
Apparatus reduces the volume that needs a human. It does not decide where the humans go. That decision needs two variables: the consequence of a finding being wrong, and the confidence that it is right.

High consequence, low confidence is where human attention earns its cost. These get full review, and they get your best people. Pull together a mixed group: the domain specialist, the operator, the person who worked the last incident, the resident sceptic. This is the small population of items where a room full of expensive expertise is the correct answer.

Low consequence, high confidence needs no line-by-line review at all. Sampling and monitoring are adequate, and the argument for why they are adequate can be written down and defended.

High consequence, high confidence still earns a proportionate check, because the cost of being wrong is what sets the standard of care.

Low consequence, low confidence is cheap to be wrong about. Fix it when it surfaces.



None of this should feel unfamiliar. It is what safety engineering has always done. Every risk is ranked by severity and likelihood before anybody decides how much attention it deserves. As Low As Reasonably Practicable (ALARP) is, in the end, an argument about where to stop spending effort. Manufacturing solved "we cannot inspect everything" in the 1920s with acceptance sampling, and the resulting framework, later MIL-STD-105 [5] and ISO 2859 [6], gives a written, auditable argument for partial inspection with quantified producer's and consumer's risk. Regulators and quality departments have had a century to object to it.

We have never claimed to examine everything with equal care. We have claimed to be able to explain where we looked hardest and why. That is the discipline this problem needs, and it is why the safety engineering community is better placed to lead this work than most AI teams. Ranking by consequence before deciding what deserves scrutiny is our default setting.

Angus Keddie put the point to me more sharply than I had managed. He is comfortable with the tool hallucinating, provided it reliably catches the high-consequence issues. He calls those HAZOP gems: the specific, difficult edge cases that lead to a serious outcome. The team's job is to argue about those. Sometimes the argument ends with the team dismissing the flagged issue as not credible, and that is still time well spent, because it was spent where being wrong costs the most.

I would add one thing to that. The gems are the interesting part of the job.

The part I have not solved

All this rests on a confidence score you can trust, and that is the part I have not solved.

A model's self-reported confidence is a generated token like any other. If the number directing engineer attention is itself uncalibrated model output, the human-in-the-loop control inherits the exact weakness it exists to manage.

What the apparatus needs is an independent judge, on the same principle as the compiler: the check sits outside the thing being checked, it runs against a stated specification, and its own performance can be measured over time. A judge that shares the generator's architecture, its context and its blind spots is not an independent check at all.

There is a second requirement that gets discussed less. The judge must be asking the right question. A score that measures whether a finding is well formed, internally consistent or plausible will point engineers towards tidy output. The question that matters is whether the finding is correct, and whether it is correct about something that counts. Ask the wrong question and the score confidently directs attention to the wrong place, which is the failure the whole system exists to prevent.

Building that judge is the hardest problem in the stack. It is the subject of my next article.

What the job becomes

Here is the version of the work I want, and the one I think is achievable.

The tool generates the analysis. The apparatus around it, specification, independent verification and monitoring, absorbs the volume that no human should be reading. What comes out arrives with clear markers on it, showing engineers where their effort is needed and why it is needed there.

The engineers go where the markers point. The difficult cases, where being wrong carries a real consequence and the system cannot vouch for its own answer, get a room and an argument. Some of those end with the team dismissing the finding as not credible, which is a good use of the time. Some of them change the design.

That is a better day than the one I had staring at a full run with no idea where to begin. It is also, by some distance, the most interesting part of this profession. The judgement calls at the edges, argued out by people who have seen things go wrong, are why most of us are still doing this work.

The risk is that we get the other version by default. Adopt these tools without building the apparatus and the review job arrives anyway: unfunded, undesigned, and quietly loaded onto the engineers who were promised their time back. Human in the loop becomes a line in a safety argument that nobody has tested. That is a worse position than not adopting the tools at all, because it carries the appearance of a control.

We already have the methods. Ranking by consequence, arguing about where to stop, writing down why partial coverage is adequate and defending it to a regulator: that is the job description. The new part is applying it to the output of a machine rather than the output of a team and doing it before the volume becomes unmanageable rather than after.

I do not want to spend the rest of my career reviewing AI outputs. I do not think I will have to. That depends on this community designing the system that means none of us do, and the time to design it is now, while the volume is still merely uncomfortable.

References

[1] H. A. Simon, "Designing Organizations for an Information-Rich World", in M. Greenberger (ed.), Computers, Communications and the Public Interest, Johns Hopkins Press, 1971, pp. 37-72

[2] L. Bainbridge, "Ironies of Automation", Automatica, Vol. 19, No. 6, 1983, pp. 775-779

[3] IEC 61508:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems, International Electrotechnical Commission, 2010

[4] RTCA DO-178C / EUROCAE ED-12C, Software Considerations in Airborne Systems and Equipment Certification, RTCA Inc., 2011

[5] MIL-STD-105E, Sampling Procedures and Tables for Inspection by Attributes, US Department of Defense, 1989 (cancelled 1995)

[6] BS EN ISO 2859-1:1999, Sampling procedures for inspection by attributes, Part 1: Sampling schemes indexed by acceptance quality limit (AQL) for lot-by-lot inspection, ISO, 1999



Ben Fulford, Safety and Assurance Engineering, BMT

Ben builds tools for safety and assurance engineering: hazard identification, hazard log modernisation, and training that develops judgement rather than attendance records. He has 15 years across defence, maritime and nuclear, including safety manager on the Astute submarine platform management system and lead independent safety auditor on an uncrewed underwater vehicle programme. He is currently building a hazard identification tool on a knowledge graph, and a browser game that teaches AI assurance through consequence.

The author retains copyright of this article.