AI Agents & the aptly named Butt report

19 March 2026 · agile, ai, automation, banking, questioning

It’s August 1941, during the opening years of WW2, the axis powers had conquered much of mainland Europe -and the Allies were desperately trying to limit the capacity of the axis military industrial complex. They did this with little feedback on their success - much like some modern teams using AI tools.

The Royal Air Force’s (RAF) strategic bombing campaign had applied considerable resources to the task of diminishing continental targets such as occupied French ports or the factories of the Ruhr valley. Much as the northwest and northeast of England were the industrial centres of Britain, the Ruhr was where a large part of the Axis industrial might lay, and so was an important military target.

The air crews reported back after their sorties (missions), stating how successful their missions had been. If such reports were to be believed, Germany and the axis powers should be decimated, be ‘on their knees’, and would fall or at least falter in short order. Wisely the air ministry asked for evidence and began fitting cameras to bombers and reconnaissance aircraft.

An analysis of several hundred sorties, comparing mission targets to photographic evidence became known as the Butt report (https://etherwave.wordpress.com/wp-content/uploads/2014/01/butt-report-transcription-tna-pro-air-14-12182.pdf) (after is author) and it was brutal in its conclusions.

A summary, overall: only 1 in 3 aircraft recorded as attacking their target got within five miles (approx. 8km). Note: that’s crews that recorded that they had attacked the target and excludes sorties where they were unable to reach or locate their targets.

But the high-level figure of 1 in 3 hides other details. The phase of the moon (and hence ambient light levels) had a dramatic effect. Under a full moon, attacks were successful (within 5 miles/8km) 2 in 5 times, but on a darker ‘new moon’ night this dropped to just 1 in 15 (about 7%).

A closer look showed that while occupied French ports could typically be “hit” 2 in 3 times (again I question whether being 4.9 miles away classes as ‘hit’), the industrial heartland of the Ruhr was typically hit just 10% of time. Given success was measured as “within a radius of 5 miles” or approx. 75 square miles the notion that the strategic bombing campaign was anything more than an annoyance to the Axis industrial base, was fanciful.

It’s admirable that the air ministry became sceptical and decided to test the claims with photography and statistical analysis. But how many aircrews and civilians on the ground had

been killed or injured due to these unchecked & errant raids? let alone the waste of materiel, that could have been used elsewhere in the war to greater effect.

What this highlights is in agile-speak is the theory of constraints. New aircraft and technology had enabled larger scale aerial bombardment across the continent, but that had (eventually) exposed the pitiful state of the navigation and verification parts of the “strategic” bombing process.

SUMMARY

STATISTICAL CONCLUSIONS.

An examination of night photographs taken during night bombing in June and July points to the following conclusions:

1. Of those aircraft recorded as attacking their target, only one in three got within five miles.
2. Over the French ports, the proportion was two in three; over Germany as a whole, the proportion was one in four; over the Ruhr, it was only one in ten.
3. In the Full Moon, the proportion was two in five; in the new moon it was only one in fifteen.

A snippet of the Butt report’s conclusions.

This same process is playing out in the AI agents domain right now. Enabled by new technology (Large Language Models trained for programming & agent harnesses like Claude code etc) teams can write software quicker, but the means of assessing their success still lags. As Zack Akil (https://www.zackakil.com/) (Google) states: what you want is Immediately Verifiable Output.

If your agents’ output can be rapidly assessed, then you can both cover more ground and reach a higher degree of quality (and reliably know you have). That’s not to belittle the quantity of such problems, it’s not uncommon to have problems that are hard “to do” but easy to verify (think Sudoku, jigsaw puzzles, Rubik, cubes or for the mathematical among you factoring large integers into constituent primes - and several - more business focused areas).

This is a boon for some companies. If your business is automated validation, this is your time to shine, your validator can provide just the sort of Immediately Verifiable Output that agents need, to home-in on a solution - That can be quickly verified. (see my previous post (https://www.investigatingsoftware.co.uk/2026/03/validators-as-coding-agent.html) for an example of how this can be done)

Where things become messy is pretty much everywhere else. How do I know that AI agent-built system for managing your company's finances is correct? The agent created some tests - but are they good? are they checking the right things? As Tim Harford recently put it (https://www.ft.com/content/c0e0bcd2-f8c9-4d45-ae2b-73a31195f74c?syn-25a6b1a6=1) in the Financial Times:

"The most problematic cases are the ones where it is hard to know whether the AI has done a good job, and expensive if it turns out it has not"

Another agent focused on testing could help (I imagine some of you are replying) - and your right it could. but then we are looking at a common - method scenario, where both building and testing are done by the same tool, using the same data and preconceptions. It's not to say models/agents cheating deliverables is worthless - it isn't, it's just not the be-all-and-end-all, especially given the capabilities of current models. Future models will no doubt improve.

Another approach is to provide such a high degree of fidelity and detail into the AI agents instructions that can't help but deliver the correct result. There is a long history of design specification approach to software development, ranging from formally defined software using tools like Z notation (https://en.wikipedia.org/wiki/Z_notation) through to more 'agentic' approaches like specification driven development (SDD). They suffer from an inability to handle the messy realities of some software. As one of my computer science professors once said: youtry formally specifying the sort of GUI/App the customer just drawn on the whiteboard. Assuming the customer even knows what they want. (They usually think they do). But once again, better specifications, especially those that can be iterated on quickly after the agent completes coding each version - will be an important part of the solution - essentially a new and higher level of abstraction.

But of course, this is specs, models and turtles all the way down. At some point we need a person to decide "does the software I paid for - do what's required of it." Does it make my business more money? will it upset the financial regulator? or land me on the receiving end of a lawsuit?

Some of those questions can be answered by "Shipping" the code, that is just give the app to the customers and they will either love it and subscribe etc or not. But many businesses can't do that. And they tend to be big & profitable and make up the bulk of the market outside the bubble of big tech. (think energy, finance, banks, law firms, aerospace etc)

Those sorts of enterprises need evidence and a person to testify (sometimes literally) that the app does what it "says on the tin" and let them know if the app does these other things, they may not like... And hence implicitly guide their agents to improvement. The investigators might need to use AI agents, engineers, testers or other products and automated tools to provide the feedback. The job title of the people doing that work might be not one in use today - but that testing process will need to be there.

So, what will happen in practice? Who knows for sure! probably all the above (including some lawsuits), but I think the role of engineers will move towards the role of detecting problems and specifying a better solution rather than coding in the traditional sense. So, in old money more solution architecting and test architecting, with probably still a split between the two roles - to better facilitate the better checking and investigation aspects of the test architect while freeing the software solutions architect to explore novel means of system design.

Connect on LinkedIn (https://www.linkedin.com/in/peter-houghton-374a36/)