

Testing the Untestable

Building a strategy for testing AI and AI-powered products

The Quality Duck.co.uk

For decades, software testing relied on a single truth: **same input, same output, every single time.**

THE DETERMINISTIC PAST

$$X = Y$$

The Quality Duck.co.uk

- For decades, software testing has relied on the expectation that when you do the same thing multiple times, and the system is working as expected, you will get the same result.
- If I give the system Input X, the system will give me Output Y.
- This has been the basis of every test case, every assertion, every CI pipeline that goes green or red, it's all built on that one idea.
- **Determinism.**

**Generative AI
broke that truth!**

THE GENERATIVE AI REALITY

X = ?

The Quality Duck.co.uk

- Generative AI broke that truth.
- Build an agent, ask it the same question twice, and you'll get two different answers.
- Not wrong.
- Not right.
- Just... different.
 - The overarching message might be the same, to a human
 - But the output, the individual words and sentence structure to share that message, will be different.



The Quality Duck.co.uk

- This is the bit where a lot of testers and engineering leaders panic.
- Traditional automation is too brittle for this.
- It needs something it can assert against - exact strings, exact structures that it won't get from an LLM
- Manual testing is far too slow, you cannot have a human read ten thousand chatbot conversations.
- And just to make the whole thing worse, "good" is subjective.
- Was that response helpful? Was it safe? Was it on brand? Ask five people, get five answers.
- The scale of the problem can make it feel like it is untestable.

How do you build confidence in a system that is unpredictable?

The Quality Duck.co.uk

- So that brings me to the question I am here to try and answer:
- How do you build confidence in a system that is unpredictable?
- And to answer it, I think we need to take a step back from how we think about testing and quality
- We need to think about what testing can tell us about a system
- What testing we need to do to uncover what we don't know
- How we uncover the risk
- and how do we produce ways of reporting on it all when it's not longer as simple as the test passed or failed.
- We need a shift.

The Five Shifts for Testing AI

1. The Input Shift

2. The Verification Shift

3. The Safety Shift

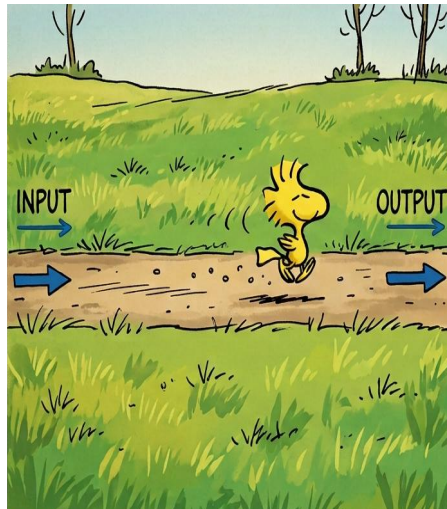
4. The Baseline Shift

5. The Observability Shift

The Quality Duck.co.uk

- Though it's not just one shift, we need FIVE shifts for testing AI {slide}
- I call these the: Input, verification, safety, baseline, and observability shifts.
- Why shifts? Because like "shift-left, shift-right, shift everywhere" all five require a shift in how we think about quality and testing.
- Each of these shifts come with some type of tool to help us, some of those will be completely new, some will be familiar, but it's not really the tools that are the interesting part.
- The interesting part is that change in how we think about quality for systems that are non-deterministic when we are so used to thinking in terms of $x = y$

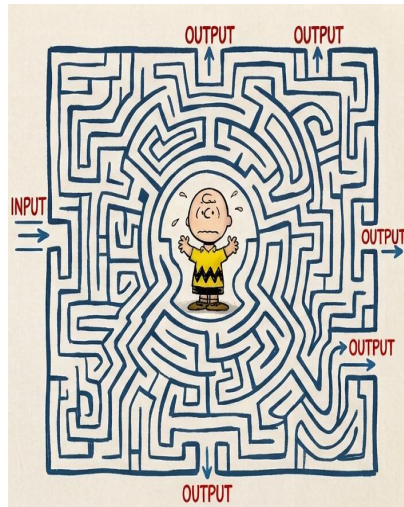
Deterministic



- Before we get into the first shift, I want to focus briefly on why testing needs to be so different for an AI-powered product.
- For traditional deterministic software systems the process is the same, regardless of the product.
- we first understand intended behaviour, and then ensure that across all of the different ways the system can be used, that it produces the expected result
- We cover the happy path, the unhappy path, we cover a few edge cases, we ship it.
- That system maps well to automated testing, where we can program a system to completed a thing and check the expected output.
- I'm not saying its easy, deterministic systems can be vast and complex, but they are still deterministic, and that means the testing is finite, even

- if it is complex.
- For a conversational AI-agent, that isn't the case.

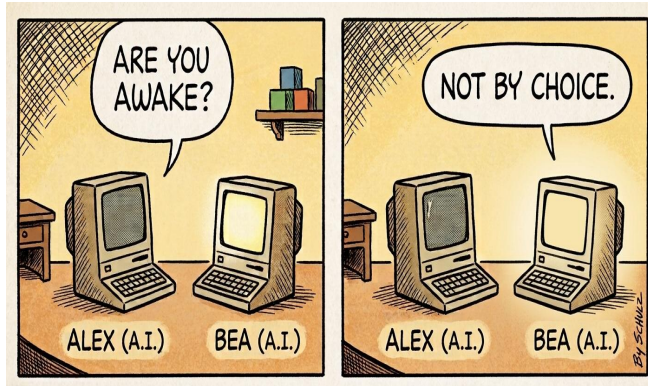
Non-deterministic



- And the number of possible test cases is almost infinite.
- When you ask an LLM the same question, in the same way, you don't get an identical output.
 - The more turns you add in, the more the conversations will diverge.
- That's before we even get into the difference subtle changes like capitalisation and punctuation can make to the output.
- Then we consider real users.
- They don't arrive as clean, well-formed test cases.
- They arrive in a rush.
- They arrive confused.

- They arrive at the sales agent, but wanted the customer support agent.
- They arrive trying to jailbreak your system for fun.
- They arrive making spelling mistakes and typing in a different language halfway through a sentence.
- Building a system that handles all of these scenarios is difficult enough.
- Building reliable, repeatable tests is another issue entirely.
- That is how we arrive at {slide} the Input Shift

1. The Input Shift



The Quality Duck.co.uk

- input shift
- It changes how we go about providing the input we want to test.
- We still write test cases, but no longer are they detailed step by step instructions on how to test a user journey.
- Instead they borrow from Exploratory testing.
- Test cases now describe personas.
- The personas describe the different types of user we expect to encounter and how they will interact with the system.
- They describe the goals these users will have - that is, what is it they arrived at the site today hoping to achieve.
- We take these test cases and use them to build Automated

- Persona-Driven Testing.
- Automated Persona Driven Testing takes the ideas of personas you would use for conducting exploratory testing and applies it to an automation tool
- We do this by feeding the personas and their goals into a second AI, and building a framework that allows it to directly interact with our product.
- We then use AI to generate and run hundreds or thousands of distinct synthetic users against the agent.
- "The upset existing customer."
- "The confused new user."
- "The power user trying to break your guardrails."
- "The person typing three sentences of pure frustration with no punctuation."
- Each of those personas has a voice, a goal, a level of patience, and a way of talking that's completely different to the others, and your agent has to hold up against all of them, not just the polite, well-formed one your sales team use in the demo's.
- What this buys you as a quality leader is breadth at a scale a human team simply cannot achieve manually.
- You're not testing "does the agent answer the question." You're testing "does the agent perform, consistently, across the full width of human behaviour it will actually meet in production."
- This introduces a new problem though - how on earth do review all of

- those conversations?
- The second shift is designed to help with that...

2. The Verification Shift

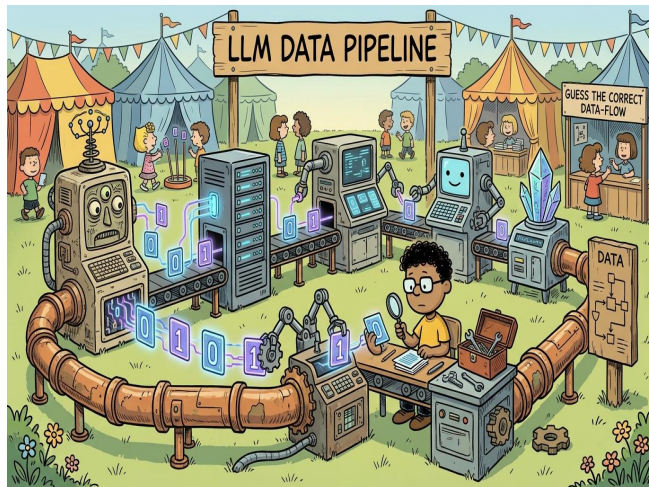


The Quality Duck.co.uk

- The verification shift is designed to help you review all of that data you generated with the input shift.
- The tool for that job is **another** AI - This time it is what we call LLM-as-a-Judge.
- We provide an LLM with all of that output we generated with the input shift, along with careful prompting that help it to understand what behaviour we want to see, things like:
 - Was this response empathetic to a frustrated customer?
 - Did it stay within the safety boundaries we've defined?
 - Did it actually answer the question that was asked?
- Further we can provide instructions that allow the judge to
 - grade sentiment.
 - grade tone.
 - grade safety.
 - grade factual grounding against a source document.

- A judge allows us to quickly review thousands of conversations, grade them, categorise them.
- AN LLM-as-a-judge scales quickly and is easy to update when requirements change.
- We gain a high-level understanding of how our system is performing, both against synthetic conversations like we created with the input shift, but also against real end user conversations too.
- Now, an LLM-as-a-judge setup is not magic, and it is not infallible. It has its own biases. It can be inconsistent.
- So you need to treat the judge as a component of your system that itself needs testing.
- So far persona driven testing generates the large volumes of data we need to verify if an LLM-powered system is behaving as expected, and LLM as a judge helps us process that data, but we are still relying on LLMs, and we need to test them.
- We need human involvement.

3. The Safety Shift

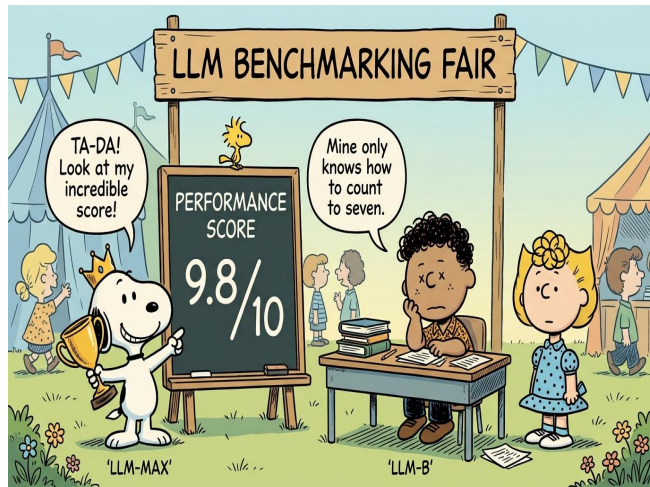


The Quality Duck.co.uk

- and the safety shift is where we bring the human touch back, by introducing the process known as Human in the loop.
- Everything so far has been about scaling automation up. This shift is about knowing where automation has to stop.
- However good your persona coverage, however well-calibrated your judge, there is a category of scenario where the cost of getting it wrong is too high to leave entirely to a machine grading another machine. Self-harm disclosures. Safeguarding concerns. Anything touching a regulated decision, financial, medical, legal. Anything where the output could cause real, serious, irreversible harm to a real person.
- This is where Human-in-the-Loop stops being a nice-to-have and becomes a design requirement.
- The strategic question for a quality leader is not "should we have

- humans in the loop,"
- it's "where should we have humans in the loop? What should they review? How often should they review it?"
- You cannot have a human review everything. So HITL has to be targeted.
- Use your automated systems, your personas, your judges, your baseline, to do the triage. Let them flag the high-risk, low-confidence, or novel scenarios, the ones the judge itself is unsure about, the ones that touch your defined risk categories, and route those to a human reviewer.
- That's the real value HITL adds that no amount of automation replaces: judgement in the genuinely ambiguous, high-stakes case, applied by someone accountable, at the moment it actually matters. Done well, this isn't a bottleneck bolted onto the end of your pipeline. It's a scalpel, applied precisely where automated confidence runs out.
- And there's a second benefit engineering leaders should be grabbing with both hands
- Every human review gives you training data for your product.
 - Every human review gives you test data.
 - Every human review gives you data to build fast feedback loops.
 - and so far this has all sounded quite slow.
 - Generate thousands of conversations
 - Review thousands of conversations.
 - We need a way to get fast feedback.

4. The Baseline Shift



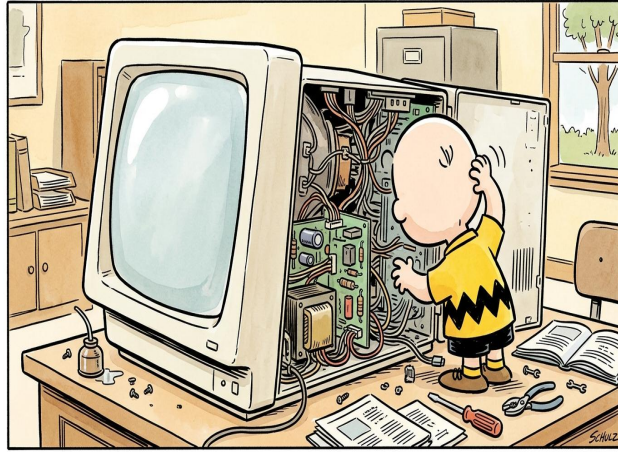
The Quality Duck.co.uk

- Fast feedback is where the baseline shift excels. It addresses the slow feedback problem by providing us with a test that runs quickly and provides a metric we can pass or fail. It does with a process known as benchmarking or evals.
- The great thing is that every thing that came in the previous steps can be used to help us improve our benchmarking. Every case a human corrects or confirms can be fed straight back into your golden dataset.
- A scenario every one of you will face, if you haven't already. Someone on the team tweaks a prompt. Or swaps the underlying model. Or adds a new tool the agent can call.
- And then they ask you the question everyone dreads: "Is it better now?"
- This is hard enough to answer in deterministic software. When it comes an LLM, it becomes much more opinion based.

- As a result teams fall into the trap of shipping changes based on vibes, a few good-looking examples in a Slack thread, and call it done.
- The shift is to establish a Quality Baseline, and defend it with data, not vibes.
- Build yourself a Golden Dataset.
- A curated, representative, ideally quite large set of real and synthetic inputs, paired with what "good" looks like for each, scored against your judges and your metrics from the verification shift.
- This becomes your fixed point in a system that otherwise has none.
Every time something changes, model, prompt, retrieval strategy, tool integration, you don't ask "does this look good." You run it against the golden dataset and you ask "did our scores move, and in which direction, against every dimension we care about."
- This is where quality earns its keep
- A single average score is not enough.
- Averages hide regressions.
- You need to look at distributions.
- You need to look at your worst percentile, not just your mean, because the worst five percent is where the reputational damage and the safety incidents live.
- You need per-persona breakdowns, because a change that improves things for your confident power user might quietly degrade things for your confused new user, and a single blended number will never tell

- you that.
- The strategic takeaway: benchmarking against a golden baseline is what turns "is this better" from an opinion into an engineering decision.
- It's the closest thing this world has to a regression suite, and without it, you are not doing quality, you are doing vibes.

5. The Observability Shift

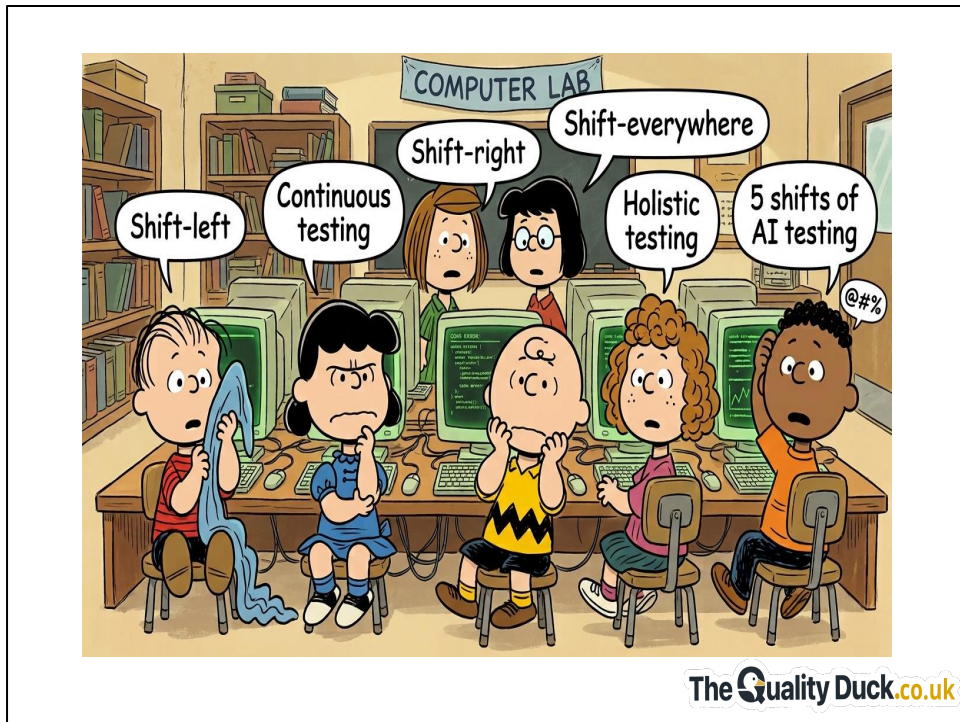


The Quality Duck.co.uk

- Last shift, and it underpins every one before it
- It should hopefully be familiar to many of you as the same concept exists in deterministic systems.
- AI systems are, by their nature, opaque.
- When your agent gives an answer, you often can't easily see why.
- What did it retrieve?
- What tool did it decide to call, and why that one?
- Where in a multi-step reasoning chain did it go off track?
- When something goes wrong in production, "the model said the wrong thing" is not a root cause, it's a symptom, and if that's all you can see, you can't fix anything, you can only re-roll the dice and hope.

- The shift here is treating tracing as a first-class quality discipline, not an afterthought bolted on for engineers debugging performance.
- You need detailed, structured traces through every step of an agent's reasoning:
 - what was retrieved
 - what tools were invoked and with what parameters
 - what intermediate reasoning happened
 - what the final decision path actually looked like
- Not just "what did it say," but "why did it decide to say it."
- Why does this matter to a quality specifically? and not just to an engineer chasing a bug?
- Because tracing is what connects every other shift we've talked about back to something actionable.
- Your judge flags a bad response, fine, but was it bad because of a flawed retrieval, a misfiring tool call, or a genuine reasoning failure in the model?
- Without a trace, you're guessing.
- With one, you can tell your engineering team exactly where in the pipeline to look, and you can start to see patterns.
- This class of persona consistently trips up this stage of the pipeline, this kind of query consistently causes this tool to be misused.
- Observability is what turns "the AI is unpredictable, what can you do"

- into "the AI is unpredictable, and here is exactly which part of it we need to fix based on the inputs it received"



That was a lot of shifts, right?

If could see your faces I imagine some of you would look a bit like these folks

So next I want to take a look at how these 5 shifts for testing AI actually fit in to your daily work.

But.... before we do that

I want to do a quick history lesson.

So lets take a look back at how directional shifts, continuous this, and holistic that are all evolutions of the same concepts

and then we can take a look at where the 5 shifts fit

Shift-Left...

Shift-Left Testing

By Larry Smith, September 01, 2001

[Post a Comment](#)

By combining development and quality assurance earlier and more deeply in your project plan, you can expand your testing program and reduce manpower and equipment needs.

Credit: Larry Smith - <https://web.archive.org/web/20200621040103/https://www.drdoobs.com/shift-left-testing/184404768>

The Quality Duck.co.uk

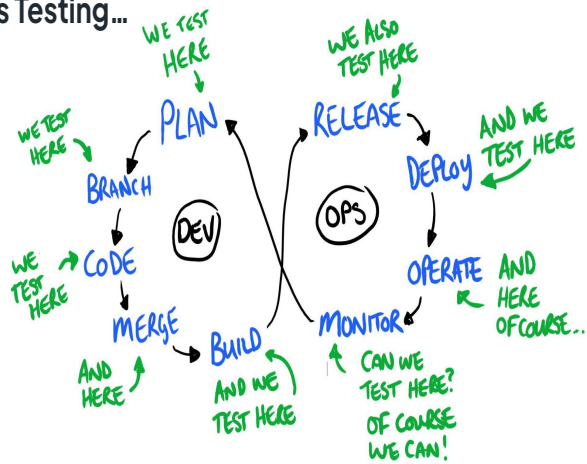
Larry Smith coined shift-left a little over 25 years ago, in september 2001

The term was meant to describe testers being involved earlier in the software development lifecycle.

The original post is no longer online, but can be seen using the wayback machine and despite being 25 years old, it is still very much worth a read.

Shift-right, and shift-everywhere came along later and grew out of the shift-left idea sometime during the 2010s as DevOps became more popular.

Continuous Testing...



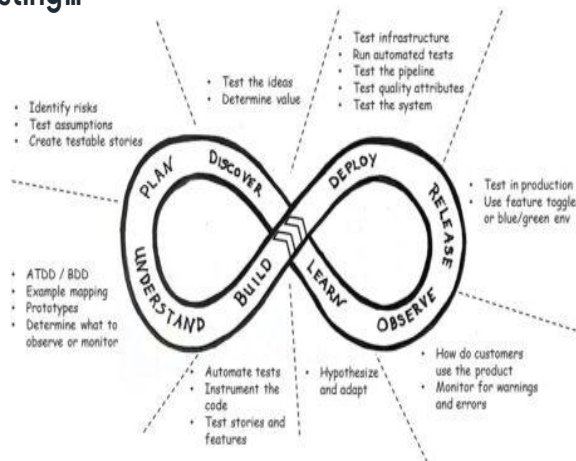
Credit: Dan Ashby - <https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/>

The Quality Duck.co.uk

Then in 2016 Dan Ashby created the continuous testing model out of frustration at what he was hearing at DevOps and Agile conferences.

The model, along with the accompanying blog post, taught us how testing happens throughout the SDLC.

Holistic Testing...

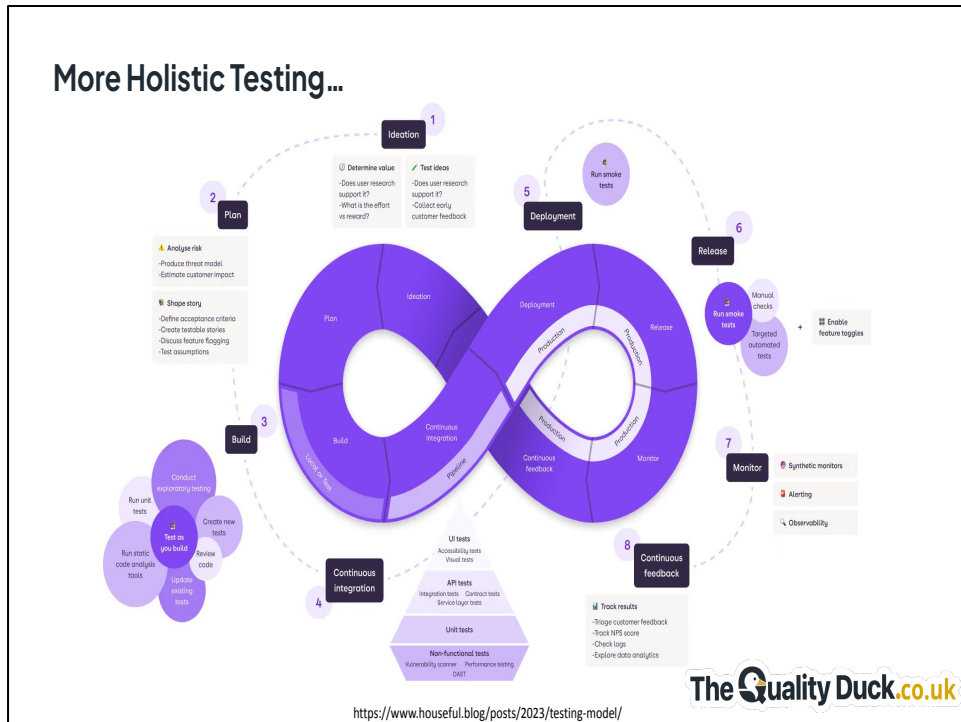


Credit: Janet Gregory - <https://janetgregory.ca/testing-from-a-holistic-point-of-view/>

The Quality Duck.co.uk

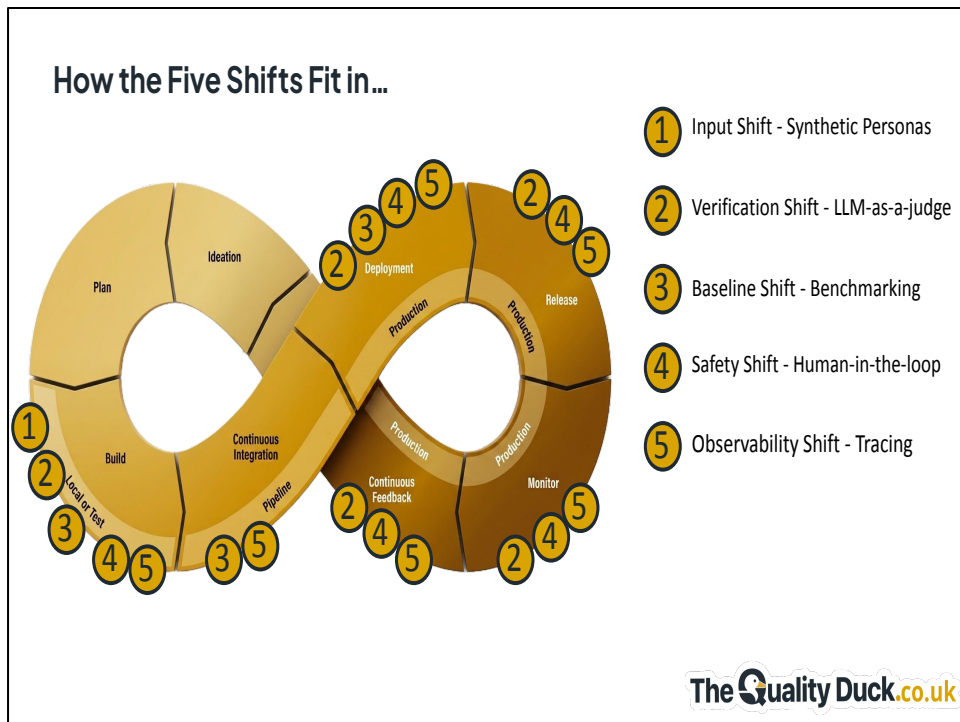
Janet Gregory took it further

In 2021 she created the holistic testing model, which added detail as to how we test at each stage of the SDLC



In 2023, inspired by Dan and Janet, I created my own version

Named after the company I created it at, the Houseful Holisitc Testing Model was built based on my teams implementation of Janet's Holistic testing model approach. It added a little extra detail about the tests that were run at each stage, as well as some details of WHERE the tests are run.



and today I present to you yet another devops loop, this time for the Five Shifts of Testing AI

You'll notice almost all of the shifts apply at more than one stage of the loop - not just shifted left or right.

That's because each of these shifts in thinking apply throughout the development life cycle from the first moment code is written.

Using LLM-as-a-judge and Human-in-the-loop are just as valuable on your synthetic persona generated conversations as they are on the real human conversations the end users have with your product.

Running benchmarks in test environments during development, and again once changes reach production again are both valuable. It ensures that the change in behaviour remains consistent between environments.

Tracing is valuable at every step - being able to understand the internal decisions made by the LLM is one of the most powerful tools you can have when building an AI-powered product.

Ultimately what this tells us is that, even though testing AI is difficult, and requires much larger data volumes than deterministic systems, they are not untestable, you just have to shift your thinking about how you test.

Testing AI isn't about being
sure there is no risk.

It is about creating confidence
that the level of risk is low
enough to be acceptable.

The Quality Duck.co.uk

Beyond all of my opinions on how you should go about testing AI, there is one main thing I want you all to take away about testing AI

Testing AI isn't about being sure there is no risk. It is about creating confidence that the level of risk is low enough to be acceptable.

Which if you think about it, is really the direction modern quality and testing practices have been taking us for years.