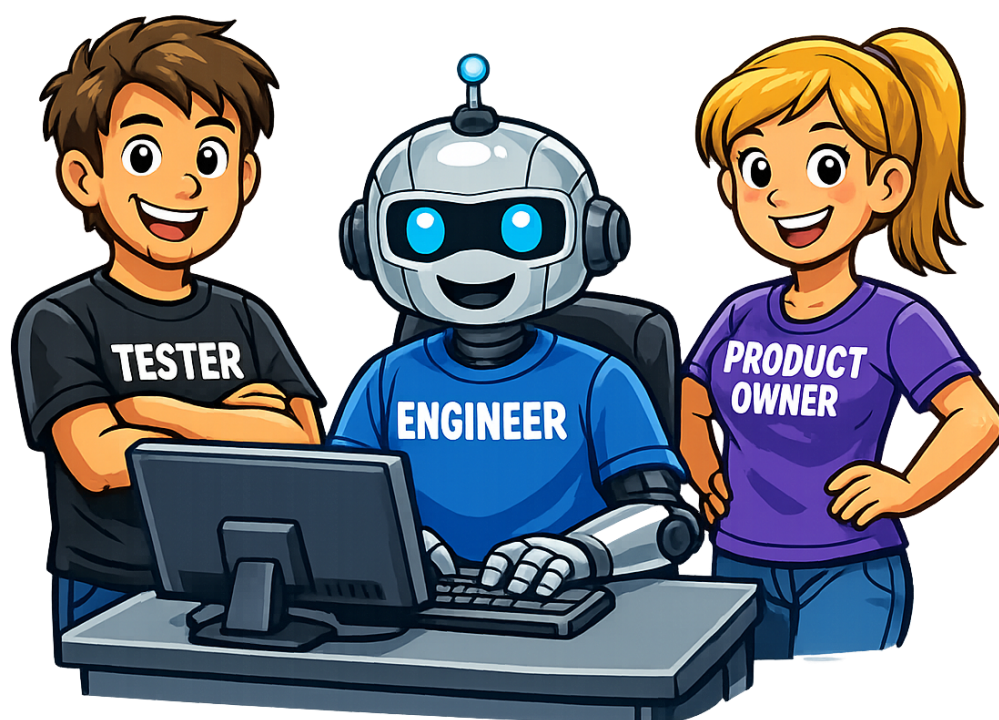


Who would have thunk it, coding was automated before testing!

13 May 2026 · testing, automation, ai, evals

It's not that analysis, solution design etc are automated but the writing of the code itself is now fully automatable. "Testing is also automated!" you say! well in a sense, yes, but much like how the broader/deeper work of analysis and system & solution design is still a human task so is full investigative testing. Yes, the tests could be written in code before Gen AI, and now that same code could be written by Claude etc

But what approach should the testing process take? what sort of analyses will find the biggest bugs? What does it feel like to use the app? what are the risks? these questions still need people to help to dream them up and investigate them. AI can greatly speed up this work, but the testing mentality and skill set is not only needed - it's now a major advantage.



Is your team starting to look a bit like this?

Why? firstly AI agents can bullshit with best of them – and as they are now doing the coding - a critical eye over its claims is invaluable. Secondly, reviewing a largish piece of software delivered by AI (or otherwise) was never easy, and always required a different skillset than used to build it.

Some engineers criticize coding agents, saying that they still must read every line of code to ensure the system works. Since when did reading all the code equate to testing a system? (never) Let alone ensuring it works? (an impossible task by the way)

I'll stop ranting and use some examples. Like the classic pocket watch, let's assume you're given the job of testing a pocket watch (yes, the mini clock on a chain from the 19th Century). while it may well be helpful to pop open the back cover & marvel at the fantastic array of cogs springs and mechanisms internal to the watch, that will only give me a hint to whether it is 'working' or not.

When did reading all the code equate to testing a system? (never)

If I spent a whole day carefully examining every cog and didn't have anything other than some thoughts like "this moves slowly that might not be right" I might soon find by self "replaced by AI" (a modern euphemism for being laid off.)

A much more effective method might be to use the watch for a day. Did it survive my key infested pocket and still "keep time"? Is it visible readable without my reading glasses? I might also compare it to a local time source (back then - probably the local church clock and later in the century, the station clock)

I might try out its various features like "setting the time" as I might have to when visiting the next town, that used a differing time (a common thing before time started to be standardised by the railway companies) I'd check its style and engravings were of interest to its target audience, the fellows of the 19th century gentleman's club I frequent (it's important to get into character).

That's not to say I wouldn't use tools and automation, in-fact when checking if the watch keeps time I'd hope I could fabricate a machine to help me - it would be tricky to do it by hand!

The questions go on (what's the spring /battery life?) and on, and that's the point. The hard part is adjusting your view from "are the cogs OK?" to "does it tell the time?" "Will the wearer be happy with it?" etc.

Again, it's not that the cogs and code don't matter, but the bigger picture is more important when assessing many types of quality. But what about security? code-based vulnerabilities are rapidly becoming the domain of LLMs and coding agents. But the adversarial test mentality will give you a way to defraud the app/watch service desk that the LLM can't, involving duping the doorman (AKA. social engineering) and some discrete pickpocketing (AKA credential theft).

And what if your app is itself AI based, who better to create and curate the complex suite of evals required to assess your app's drift and ability to give users the answers they need?

So, the advantage in the age of AI is not the engineer who spends all their time learning syntax while avoiding architecture, speaking to product owners and users. The advantage is to those experienced at examining software, assessing the various qualities of what the agent has delivered and whether that's what the product owner, CEO, devops team, support team (etc) & customer are happy with.

That's not a skill many people have, and it's not a skill that everyone realised they are going to need. If you need help you can reach out to me on [email](#) or [LinkedIn](https://www.linkedin.com/in/peter-houghton-374a36/) (<https://www.linkedin.com/in/peter-houghton-374a36/>)