Table of Contents

Preface to the Third Edition: TDD in the Age of AI

Is there any point in learning TDD now that AI can write code for you? A single prompt could probably generate the entire example application in this book, including all its tests, and the infrastructure config for automated deployment too.

The truth is that it's too early to tell. AI is still in its infancy, and who knows what it'll be able to do in a few years or even months' time.

AI Is Both Insanely Impressive and Incredibly Unreliable

What we do know is that right now, AI is both insanely impressive and incredibly unreliable.

Beyond being able to understand and respond to prompts in normal human language—it's easy to forget how absolutely extraordinary that is; literally science-fiction a few years ago—AI tools can generate working code, they can generate tests, they can help us to break down requirements, brainstorm solutions, quickly prototype new technologies. It's genuinely astonishing.

As we're all finding out though, this all comes with a massive "but". AI outputs are frequently plagued by hallucinations, and in the world of code, that means things that just won't work, even if they look plausible. Worse than that, they can produce code that appears to work, but is full of subtle bugs, security issues, or performance nightmares. From a code quality point of view, we know that AI tools will often produce code that's filled with copy-paste and duplication, weird hacks, and undecipherable spaghetti code that spells a maintenance nightmare.

Mitigations for AI's Shortcomings Sure Look a Lot Like TDD

If you read the advice, even from AI companies themselves, about the best way to work with AI, you'll find that it performs best when working in small, well-defined contexts, with frequent checks for correctness. When taking on larger tasks, the advice is to break them down into smaller, well-defined pieces, with clearly defined success criteria.

When we're thinking about the problem of hallucinations, it sure seems like having a comprehensive test suite and running it frequently, is going to be a must-have.

When we're thinking about code quality, the idea of having a human in the loop, with frequent pauses for review and refactoring, again seems like a key mitigation.

In short, all of the techniques of test-driven development that are outlined in this book:

- Defining a test that describes each small change of functionality, before we write the code for it
- Breaking our problem down into small pieces and working incrementally, with frequent test runs to catch bugs, regressions, and hallucinations
- The "refactor" step in TDD's red/green/refactor cycle, which gives us a regular reminder for the human in the loop to review and improve the code.

TDD is all about finding a structured, safer way of developing software, reducing the risk of bugs and regressions and improving code quality, and these are very much the exact same things that we need to achieve when working with AI.

Leaky Abstractions and the Importance of Experience

"Leaky abstractions" (https://oreil.ly/PgWjL) are a diagnosis of a common problem in software development, whereby higher-level abstractions fail in subtle ways, and the complexities of the underlying system leak through.

In the presence of leaky abstractions, you need to understand the lower-level system to be able to work effectively. It's for this reason that, when the switch to third-generation languages (3GLs) happened, programmers who understood the underlying machine code were often the most effective at using the new languages like C and Fortran.

In a similar way, AI offers us a new, higher-level abstraction around writing code, but we can already see the "leaks" in the form of hallucinations and poor code quality. And by analogy to the 3GLs, the programmers who are going to be most effective with AI are going to be the ones who "know what good looks like", both in terms of code quality, test structure, and so on, but also in terms of what a safe and reliable workflow for software development looks like.

My Own Experiences with AI

In my own experiences of working with AI, I've been very impressed at its ability to write tests, for example…as long as there was already a good first example test to copy from. Its ability to write that *first* test, the one where, as we'll see, a lot of the design (and thinking) happens in TDD, was much more mixed.

Similarly when working in a less "autocomplete" and more "agentic" mode, I saw AI tools do very well on simple problems with clear instructions, but when trying to deal with more complex logic and requirements with ambiguity, I've seen it get dreadfully stuck in loops and dead ends.

When that happened, I found that trying to guide the AI agent back towards taking small steps, working on a single piece at a time, and clarifying requirements in tests, was the best way to get things back on track.

I've also been able to experiment with using the "refactor" step to try and improve the often-terrible code that the AI produced. Here again I had mixed results, where the AI would need a lot of nudging before settling on a solution that felt sufficiently readable and maintainable, to me.

So I'd echo what many others are saying, which is that AI works best when you, the user, are a discerning partner rather than passive recipient.

> Ultimately, as software developers, we need to be able to stand by the code we produce, and be accountable for it, no matter what tools were used to write it.

The AI-Enabled Workflow of the Future

The AI-enabled workflow of the future will look very different to what we have now, but all the indications are that the most effective approach is going to be incremental, have checks and balances to avoid hallucinations, and systematically involve humans in the loop to ensure quality.

And the closest workflow we have to that today, is TDD. I'm excited to share it with you!

Comments

ALSO ON **OBEY THE TESTING GOAT!**

Splitting Our Tests into Multiple Files, and a …

9 years ago • 22 comments

As our first few users start using the site, we've noticed they sometimes make …

Prettification: Layout and Styling, and What to

2 years ago • 4 comments

We're starting to think about releasing the first version of our site, but we're a bit …

Making our App Production-Ready

2 years ago • 8 comments

Static files aren't the same kind of things as the dynamic content that comes from …

Getting to Ready Dep

9 years ago • 9

Our deployn fine but it's r ready. Let's

Emoji?

4 Responses

👍 Upvote 😝 Funny 😍 Love 😮 Surprised 😤 Angry 😢 Sad

1 Comment Kgotso Koete ▾

Join the discussion…

♡ **Share** **Best** **Newest** **Oldest**

João Pedro —

a month ago

We are seeing more and more efforts to remove the developer from the code writer position to have a manager-of-models type of responsibility.

We'll see how our market will behave on the next few years of this transition.