Building and Handling Exploratory Interfaces

Precis

Want a tester's interface to let you explore some code? Want data and mocks set up for you, visibility into underlying data, tools to iterate input, alarms for trouble? You do. Find out how to persuade LLMs to build exploratory interfaces for you.

Abstract

LLMs generate great piles of weird code (which can still pass your tests). We now need to work through far more code, looking for unfamiliar machine-made problems: We will need a corresponding step change in how we look for trouble. In this session, I'll share my recent approaches and experiments in generating ephemeral tools to help me swiftly and usefully explore a delightful open source product, FlockXR.

I'll share how I've asked LLMs to build exploratory interfaces – short-lived generated tools that let me explore code behaviour at a unit test level. You'll play with some of these interfaces on your own devices, and I will build new ones in the session so that you can see how it is done.

While playing, you will use parameterised sliders, synchronised generators, dynamic property assertions and visual analysis. You will see my approaches to default data and to object creation, to working with generated dynamic inputs and alerts for recognisable oddness, how my interfaces manage dependencies and how I can capture information about problems.

I'll share my processes, prompts and libraries. I'll describe how I use exploratory interfaces on the project, the pitfalls and solutions around finding code that fits, and how I wrangle the LLMs towards reliable and helpful interfaces.

Key Takeaways

- See how to persuade an LLM to build an exploratory interface for code components
- Recognise pitfalls and mitigations around dependencies / mocks, consumable and configuration data, and oracles
- Access an open-source library with process and prompts to build your own