Testing

Yesterday, I set out to create an auto-configuration tool for Shadowsocks Outline, aiming to turn it into a Python project for others to use. I developed a script that updates a `config.yaml` file with Shadowsocks proxy configurations by decoding Shadowsocks URLs from an `ssconfig` file. Additionally, I created another script that uses `gsutil` to upload the subscription file for clients to Google Cloud Storage.

I used Windsurf, an AI code editor, for assistance. However, it struggled with handling mock dependencies in Python unit testing.

Reflecting on the testing lessons shared by Yin Wang, I recalled his experiences at Google, where he worked on a Python interpreter and indexed the company' s code for search functionality. His colleagues insisted on writing tests, which he found annoying. He believed that writing elegant code was more important than testing, and that his colleagues only understood superficial aspects without grasping the essence.

I realized my mistake; the AI didn' t point it out. I should ensure the main code of a library is solid before focusing on tests. The same principle applies to a proof of concept project. In previous jobs, such as starting a microservice, tests should be written after the microservice has some APIs or functions.

If Windsurf handled the testing part well, I wouldn' t have this complaint. However, there are two distinct issues at play: the timing of implementing tests and the correct way to write them. Currently, we' re focusing on the former. These issues are interrelated to some extent. If an AI code editor or a human finds it easy to write test code, the timing of tests might seem trivial. However, the effort required for writing tests is comparable to that of writing the main code, making the timing an important consideration.

From a collaboration perspective, the approach to testing can vary. For a personal project, I might write a significant amount of code before creating tests. However, when working in a team, it' s generally better to write tests for every snippet or feature. But this isn' t always the case; it depends on how the team collaborates. A more accurate way to put it is that tests should be written for the code that team members share with each other. The goal is to ensure code quality, so before delivering the code, each team member is free to choose their testing timing.

In a previous work experience, I collaborated with three other backend engineers on a feature that took half a year to deliver. From a testing perspective, the points discussed in this article may explain why development was slow during that time.

From a collaboration standpoint, those responsible for the main code should also be responsible for the related tests. Tasks should intertwine as little as possible, with clear and separate responsibilities for each team member.

Returning to the topic of testing, AI code editors also lack this kind of optimization, highlighting an area for improvement. This principle isn' t limited to software engineering; it' s relevant to hardware and other

fields as well. Testing is a form of optimization, and as the saying goes, "Premature optimization is the root of all evil."

It's crucial to remember the main objective of the job. While processes and procedures are unavoidable, we must keep in mind what's truly important.

References:

- Test-Driven Development, Yin Wang
- The Logic of Testing, Yin Wang