テストのタイミング

昨日、Shadowsocks Outline の自動設定ツールを作成し、それを Python プロジェクトとして他の人にも使えるようにしようと試みました。`ssconfig` ファイルから Shadowsocks の URL をデコードし、`config.yaml` ファイルに Shadowsocks プロキシ設定を更新するスクリプトを開発しました。さらに、`gsutil` を使用してクライアント用のサブスクリプションファイルを Google Cloud Storage にアップロードする別のスクリプトも作成しました。

私は AI コードエディタの Windsurf を利用して支援を受けました。しかし、Python のユニットテストにおけるモック依存関係の扱いには苦戦しました。

Yin Wang さんのテストに関する教訓を振り返りながら、彼が Google で働いていた時のことを思い出しました。彼はそこで Python インタプリタを開発し、社内のコードを検索機能のためにインデックス化していました。同僚たちはテストを書くことを強く主張し、彼はそれを煩わしく感じていました。彼は、エレガントなコードを書くことがテストよりも重要だと考え、同僚たちは表面的な部分しか理解せず、本質を捉えていないと思っていました。

私は自分の間違いに気づきましたが、AI はそれを指摘しませんでした。ライブラリのメインコードがしっかりしていることを確認してから、テストに集中すべきです。この原則は、プロトタイププロジェクトにも適用されます。以前の仕事、例えばマイクロサービスの立ち上げでは、マイクロサービスがいくつかの API や機能を持った後にテストを書くべきです。

もし Windsurf がテストの部分をうまく扱っていたら、私にこの不満はなかったでしょう。しかし、ここには 2 つの異なる問題があります。テストを実装するタイミングと、テストを正しく書く方法です。現在、私たちは前者に焦点を当てています。これらの問題はある程度相互に関連しています。もし AI コードエディタや人間がテストコードを簡単に書けるなら、テストのタイミングは些細なことに思えるかもしれません。しかし、テストを書くのに必要な労力はメインのコードを書くのと同等であり、タイミングは重要な考慮事項となります。

コラボレーションの観点から見ると、テストへのアプローチは様々です。個人プロジェクトの場合、私はテストを作成する前に多くのコードを書くかもしれません。しかし、チームで作業する場合、一般的にはすべてのスニペットや機能に対してテストを書く方が良いでしょう。ただし、これが常に当てはまるわけではありません。チームがどのように協力するかによります。より正確に言うと、チームメンバーが互いに共有するコードに対してテストを書くべきです。目標はコードの品質を確保することなので、コードを提供する前に、各チームメンバーは自由にテストのタイミングを選ぶことができます。

以前の職場経験では、3 人のバックエンドエンジニアと協力して、半年かけて一つの機能を開発しました。テストの観点から見ると、この記事で議論されているポイントが、当時の開発が遅かった理由を説明しているかもしれません。

コラボレーションの観点から、メインコードを担当する者は関連するテストも担当すべきです。タスクは可能な限り絡み合わず、各チームメンバーの責任が明確かつ分離されていることが重要です。

テストの話題に戻ると、AI コードエディタもこの種の最適化を欠いており、改善の余地があることを示しています。この原則はソフトウェア工学に限定されず、ハードウェアやその他の分野にも関連しています。テストは最適化の一形態であり、よく言われるように「時期尚早な最適化は諸悪の根源」です。

仕事の主目的を忘れないことが極めて重要です。プロセスや手順は避けられないものですが、本当に重要なことは何かを常に心に留めておかなければなりません。

参考文献:

- テスト駆動開発, Yin Wang
- テストの論理, Yin Wang