Der Zeitpunkt des Testens

Gestern habe ich mich daran gemacht, ein Auto-Konfigurations-Tool für Shadowsocks Outline zu erstellen, mit dem Ziel, es in ein Python-Projekt zu verwandeln, das andere nutzen können. Ich habe ein Skript entwickelt, das eine `config.yaml`-Datei mit Shadowsocks-Proxy-Konfigurationen aktualisiert, indem es Shadowsocks-URLs aus einer `ssconfig`-Datei decodiert. Zusätzlich habe ich ein weiteres Skript erstellt, das `gsutil` verwendet, um die Abonnementdatei für Clients auf Google Cloud Storage hochzuladen.

Ich habe Windsurf, einen KI-gestützten Code-Editor, zur Unterstützung verwendet. Allerdings hatte er Schwierigkeiten, Mock-Abhängigkeiten in Python-Unittests zu handhaben.

Bei der Reflexion über die Test-Lektionen, die Yin Wang geteilt hat, erinnerte ich mich an seine Erfahrungen bei Google, wo er an einem Python-Interpreter arbeitete und den Code des Unternehmens für die Suchfunktion indexierte. Seine Kollegen bestanden darauf, Tests zu schreiben, was er als lästig empfand. Er glaubte, dass das Schreiben von elegantem Code wichtiger sei als das Testen, und dass seine Kollegen nur oberflächliche Aspekte verstanden, ohne das Wesentliche zu begreifen.

Mir wurde mein Fehler bewusst; die KI hat ihn nicht angesprochen. Ich sollte sicherstellen, dass der Hauptcode einer Bibliothek solide ist, bevor ich mich auf Tests konzentriere. Das gleiche Prinzip gilt für ein Proof-of-Concept-Projekt. In früheren Jobs, wie dem Start eines Microservices, sollten Tests erst geschrieben werden, nachdem der Microservice einige APIs oder Funktionen hat.

Wenn Windsurf den Testteil gut gehandhabt hätte, hätte ich diese Beschwerde nicht. Es gibt jedoch zwei unterschiedliche Probleme: den Zeitpunkt der Implementierung von Tests und die korrekte Art, sie zu schreiben. Derzeit konzentrieren wir uns auf das Erstere. Diese Probleme sind bis zu einem gewissen Grad miteinander verbunden. Wenn ein KI-Code-Editor oder ein Mensch es einfach findet, Testcode zu schreiben, könnte der Zeitpunkt der Tests trivial erscheinen. Der Aufwand für das Schreiben von Tests ist jedoch vergleichbar mit dem Schreiben des Hauptcodes, was den Zeitpunkt zu einer wichtigen Überlegung macht.

Aus der Perspektive der Zusammenarbeit kann der Ansatz zum Testen variieren. Bei einem persönlichen Projekt könnte ich eine beträchtliche Menge an Code schreiben, bevor ich Tests erstelle. Wenn ich jedoch in einem Team arbeite, ist es im Allgemeinen besser, Tests für jeden Codeausschnitt oder jedes Feature zu schreiben. Das ist jedoch nicht immer der Fall; es hängt davon ab, wie das Team zusammenarbeitet. Eine genauere Aussage wäre, dass Tests für den Code geschrieben werden sollten, den die Teammitglieder miteinander teilen. Das Ziel ist es,

die Codequalität sicherzustellen, daher kann jedes Teammitglied vor der Abgabe des Codes frei entscheiden, wann es die Tests durchführt.

In einer früheren Berufserfahrung habe ich mit drei anderen Backend-Ingenieuren an einer Funktion zusammengearbeitet, deren Entwicklung ein halbes Jahr in Anspruch nahm. Aus der Perspektive des Testens könnten die in diesem Artikel diskutierten Punkte erklären, warum die Entwicklung damals langsam voranschritt.

Aus der Perspektive der Zusammenarbeit sollten diejenigen, die für den Hauptcode verantwortlich sind, auch für die zugehörigen Tests verantwortlich sein. Aufgaben sollten so wenig wie möglich miteinander verflochten sein, mit klaren und getrennten Verantwortlichkeiten für jedes Teammitglied.

Zurück zum Thema Testing: Auch KI-Code-Editoren weisen in dieser Hinsicht Optimierungsmängel auf, was einen Verbesserungsbedarf aufzeigt. Dieses Prinzip beschränkt sich nicht nur auf die Softwareentwicklung; es ist auch für Hardware und andere Bereiche relevant. Testing ist eine Form der Optimierung, und wie das Sprichwort sagt: „Vorzeitige Optimierung ist die Wurzel allen Übels.“

Es ist entscheidend, sich an das Hauptziel der Aufgabe zu erinnern. Während Prozesse und Verfahren unvermeidlich sind, müssen wir im Hinterkopf behalten, was wirklich wichtig ist.

Referenzen:

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