परीक्षण का समय

कल, मैंने □□□□□□□□□□□□ □□□□□□□□ के लिए एक ऑटो-कॉन्फ़िगरेशन टूल बनाने का प्रयास किया, जिसका उद्देश्य इसे एक □□□□□□□ प्रोजेक्ट में बदलना था ताकि इसे दूसरे लोग भी इस्तेमाल कर सकें। मैंने एक स्क्रिप्ट विकसित की जो `ssconfig` फ़ाइल से □□□□□□□□□□□□ □□□□ को डिकोड करके `config.yaml` फ़ाइल को □□□□□□□□□□□□ प्रॉक्सी कॉन्फ़िगरेशन के साथ अपडेट करती है। इसके अलावा, मैंने एक और स्क्रिप्ट बनाई जो `gsutil` का उपयोग करके क्लाइंट्स के लिए सब्सक्रिप्शन फ़ाइल को □□□□□□□ □□□□□ □□□□□□□□ पर अपलोड करती है।

मैंने मदद के लिए □□□□□□□□□, एक □□ कोड एडिटर का उपयोग किया। हालांकि, इसे □□□□□□□ यूनिट टेस्टिंग में मॉक डिपेंडेंसी को हैंडल करने में कठिनाई हुई।

□□□ □□□□ द्वारा साझा किए गए परीक्षण (□□□□□□□□) के सबकों पर विचार करते हुए, मुझे उनके □□□□□□□ में काम करने के अनुभव याद आए, जहाँ उन्होंने एक □□□□□□□ इंटरप्रेटर पर काम किया और कंपनी के कोड को खोज (□□□□□□□) कार्यक्षमता के लिए इंडेक्स किया। उनके सहकर्मी परीक्षण लिखने पर जोर देते थे, जिसे वह परेशान करने वाला मानते थे। उनका मानना था कि सुंदर कोड लिखना परीक्षण से अधिक महत्वपूर्ण है, और उनके सहकर्मी केवल सतही पहलुओं को समझते थे, जबकि उनका सार नहीं समझ पाते थे।

मुझे अपनी गलती का एहसास हुआ; □□ ने इसे इंगित नहीं किया। मुझे यह सुनिश्चित करना चाहिए कि किसी लाइब्रेरी का मुख्य कोड मजबूत हो, इससे पहले कि मैं टेस्ट पर ध्यान केंद्रित करूं। यही सिद्धांत एक प्रूफ ऑफ कॉन्सेप्ट प्रोजेक्ट पर भी लागू होता है। पिछली नौकरियों में, जैसे कि एक माइक्रोसर्विस शुरू करते समय, टेस्ट को तब लिखा जाना चाहिए जब माइक्रोसर्विस में कुछ □□□□ या फंक्शन्स हो चुके हों।

अगर □□□□□□□□□ ने टेस्टिंग का हिस्सा अच्छी तरह से संभाला होता, तो मुझे यह शिकायत नहीं होती। हालांकि, यहां दो अलग-अलग मुद्दे हैं: टेस्ट लागू करने का समय और उन्हें सही तरीके से लिखने का तरीका। फिलहाल, हम पहले मुद्दे पर ध्यान केंद्रित कर रहे हैं। ये मुद्दे कुछ हद तक आपस में जुड़े हुए हैं। अगर एक □□ कोड एडिटर या इंसान को टेस्ट कोड लिखना आसान लगता है, तो टेस्ट का समय तुच्छ लग सकता है। हालांकि, टेस्ट लिखने में लगने वाला प्रयास मुख्य कोड लिखने के बराबर है, जिससे समय एक महत्वपूर्ण विचार बन जाता है।

सहयोग के दृष्टिकोण से, परीक्षण (□□□□□□□□) का तरीका अलग-अलग हो सकता है। एक व्यक्तिगत प्रोजेक्ट के लिए, मैं टेस्ट लिखने से पहले काफी सारा कोड लिख सकता हूँ। हालांकि, टीम के साथ काम करते समय, आमतौर पर हर स्निपेट या फीचर के लिए टेस्ट लिखना बेहतर होता है। लेकिन यह हमेशा ऐसा नहीं होता; यह इस बात पर निर्भर करता है कि टीम कैसे सहयोग करती है। इसे और सटीक तरीके से कहें तो, टेस्ट उस कोड के लिए लिखे जाने चाहिए जो टीम के सदस्य एक-दूसरे के साथ साझा करते हैं। लक्ष्य कोड की गुणवत्ता सुनिश्चित करना है, इसलिए कोड डिलीवर करने से पहले, प्रत्येक टीम सदस्य अपने टेस्टिंग का समय चुनने के लिए स्वतंत्र है।

पिछले कार्य अनुभव में, मैंने तीन अन्य बैकएंड इंजीनियरों के साथ एक फीचर पर सहयोग किया जिसे पूरा करने में छह महीने लगे। परीक्षण के दृष्टिकोण से, इस लेख में चर्चा किए गए बिंदु उस समय के दौरान विकास धीमा होने के कारणों को समझा सकते हैं।

सहयोग के दृष्टिकोण से, मुख्य कोड के लिए जिम्मेदार लोगों को संबंधित टेस्ट के लिए भी जिम्मेदार होना चाहिए। कार्यों को यथासंभव कम से कम जोड़ा जाना चाहिए, जहां प्रत्येक टीम सदस्य की जिम्मेदारियां स्पष्ट और अलग हों।

टेस्टिंग के विषय पर वापस आते हुए, □□ कोड एडिटर्स में भी इस प्रकार के ऑप्टिमाइज़ेशन की कमी है, जो सुधार के एक क्षेत्र को उजागर करता है। यह सिद्धांत सिर्फ सॉफ्टवेयर इंजीनियरिंग तक ही सीमित नहीं है; यह हार्डवेयर और अन्य क्षेत्रों में भी प्रासंगिक है। टेस्टिंग एक प्रकार का ऑप्टिमाइज़ेशन है, और जैसा कि कहा जाता है, “समय से पहले ऑप्टिमाइज़ेशन सभी बुराइयों की जड़ है।”

यह याद रखना महत्वपूर्ण है कि नौकरी का मुख्य उद्देश्य क्या है। जबकि प्रक्रियाएं और प्रक्रियाएं अपरिहार्य हैं, हमें यह ध्यान रखना चाहिए

कि वास्तव में क्या महत्वपूर्ण है।

संदर्भ:

- टेस्ट-ड्रिवन डेवलपमेंट, □□□ □□□□
- टेस्टिंग का तर्क, □□□ □□□□