مدرس: دکتر احسان خامس‌پناه طراحان: سامان اسلامی نظری، آرش شاهین، علی هدایی

بخش تئوری

سوال اول

تفاوت میان Behavior Verification و State Verification را بیان کنید. به منظور انجام هر یک از این روش‌ها از چه شیوه‌ای (Mock یا Stub) می‌توانیم استفاده کنیم؟

پاسخ:

در حالت Behavior verification، همانطور که از نامش پیداست، اعتبارسنجی رفتار مدنظر ماست. در این حالت درست صدا زدن توابع و ترتیب صدا زدنشان برای ما اهمیت دارد. با این حال در حالت State Verification ما با استفاده از آزمون‌ها سعی می‌کنیم تا مطمئن شویم پس از انجام یک عملیات، سیستم در حالت و وضعیت درستی قرار می‌گیرد. برای مثال فرض کنیم یک سیستم سفارش‌دهی داریم که تعدادی سفارش را با موجودی انبار منطبق داده و در صورتی که این تعداد موجود نباشد، یک ایمیل به کاربر ارسال می‌کند؛ آزمودن اینکه تابع ایمیل فراخوانی شده و فقط یک بار این اتفاق می‌افتد، نوعی Behavior Verification است. این کار با استفاده از Mock-ها قابل انجام است. برای انجام State Verification از Spy-ها که نوع پیشرفته‌تری از Test stub-ها هستند استفاده می‌کنیم. Spy به ما این قابلیت را می‌دهد که وضعیت داخلی Stub را نیز بررسی کرده و از صحت آن اطمینان حاصل کنیم.

سوال دوم

test spy-ها چه چیزی هستند و چرا استفاده می شوند؟ انواع آن‌ها را نام ببرید و توضیح دهید.

پاسخ:

Test Spy یک نوع از Test Double است که برای مشاهده و ثبت اطلاعاتی که در طول اجرای یک تست به آن دسترسی پیدا می‌کند، استفاده می‌شود. تفاوت اصلی آن با دیگر انواع Test Double مانند mock یا stub این است که علاوه بر اینکه به تست کمک می‌کند، اطلاعاتی را که به آن پاس داده می‌شود یا رفتارهای مورد انتظار را در زمان اجرای تست ثبت می‌کند، تا بعداً بتوان این اطلاعات را برای بررسی استفاده کرد.

Test Spy زمانی استفاده می‌شود که شما بخواهید عملکرد یک قطعه از کد را بررسی کنید بدون اینکه عملکرد واقعی آن را مختل کنید. به عبارتی، Test Spy به شما امکان می‌دهد تا ورودی‌ها و خروجی‌های یک تابع یا

متد را ذخیره و بررسی کنید و اطمینان حاصل کنید که تابع موردنظر در زمان مناسب فراخوانی شده است و پارامترهای مورد انتظار را دریافت کرده است.

انواع Test Spy:

1. **رابط بازیابی (Retrieval Interface):**

در این روش، Test Spy به‌عنوان یک کلاس جداگانه تعریف می‌شود که شامل یک رابط بازیابی است. این رابط اطلاعات ضبط شده توسط Test Spy را برای تست در دسترس قرار می‌دهد تا بعداً بتوان آن را بررسی کرد. این روش برای بازیابی اطلاعات دقیق تعاملات مفید است.

2. **خود شانت (Self Shunt):**

این نوع شامل ادغام Test Spy و کلاس Test Case به یک شیء واحد به نام Self Shunt است؛ به این معنی که خود شیء تست به عنوان Test Spy عمل می‌کند. هر زمان که سامانه تحت تست (SUT) به یک وابسته به مؤلفه (DOC) واگذار می‌شود، در واقع متدهای آن شیء در کلاس Test Case فراخوانی می‌شوند.

3. **Test Double داخلی:**

Test Spy می‌تواند به‌عنوان یک Test Double کدنویسی‌شده در کلاس تست پیاده‌سازی شود. این معمولاً شامل کدنویسی Test Spy به‌عنوان یک کلاس داخلی ناشناس یا یک بلوک closure است که برای بررسی تعاملات در داخل تست به کار می‌رود.

4. **ثبت خروجی غیر مستقیم (Indirect Output Registry):**

در این نوع، Test Spy داده‌ها را به‌طور غیرمستقیم در یک فایل، رجیستری یا ساختار داده دیگری ذخیره می‌کند. سپس تست می‌تواند این داده‌ها را از محل ذخیره‌سازی برای اعتبارسنجی تعاملات بازیابی کند

سوال سوم

فرض کنید می‌خواهید آزمونی برای یک سیستم بنویسید که نیاز به مدیریت یک Fixture مشترک بین چندین آزمون را دارد. در این شرایط:

الف) چه زمانی استفاده از Shared Fixture نسبت به Fresh Fixture مناسب‌تر است؟

ب) مزایا و معایب استفاده از Lazy Setup در مقایسه با Suite Fixture Setup را بررسی کنید.

ج) اگر بخواهید مطمئن شوید که آزمون‌ها در مقابل تغییرات ناخواسته به یک Fixture مشترک مقاوم هستند (مانند تغییرات در داده های دیتابیس)، چه رویکردی را برای مدیریت Fixture پیشنهاد می‌کنید؟

پاسخ:

الف)

استفاده از Shared Fixture زمانی مناسب است که ساخت یک Fixture جدید برای هر تست هزینه‌بر یا پیچیده باشد؛ مانند زمان‌هایی که نیاز به پر کردن یک پایگاه‌داده بزرگ داریم. Shared Fixture همچنین برای

آزمون‌هایی که به داده‌های یکسان و پایدار نیاز دارند مفید است. اما اگر هر آزمون نیاز به محیطی مستقل داشته باشد تا از تداخلات جلوگیری شود، Fresh Fixture بهتر است.

ب)

Lazy Setup فقط زمانی Fixture را ایجاد میکند که آزمون به آن نیاز داشته باشد که باعث بهبود کارایی و استقلال تست‌ها می شود. اما مدیریت آن پیچیده‌تر است زیرا هر تست باید بررسی کند که Fixture موجود است یا نه. در مقابل، Suite Fixture Setup یکبار Fixture را قبل از همه تست‌ها ایجاد می‌کند، که ساده‌تر است ولی اگر یک تست Fixture را تغییر دهد ممکن است بر سایر تست‌ها تأثیر بگذارد.

ج)

استفاده از Delta Assertion یا Database Partitioning می‌تواند مؤثر باشد. با Delta Assertion می‌توان وضعیت قبل و بعد از هر تست را بررسی کرد تا مطمئن شویم که تغییری در داده‌های اصلی ایجاد نشده است. همچنین استفاده از روش‌های Isolation (مانند استفاده از یک دیتابیس مجزا برای هر آزمون) به جلوگیری از تداخل کمک می‌کند.