آزمون نرم‌افزار تمرین دوم

بخش تئوری

امیر پارسا موبد | ۸۱۰۱۰۰۲۷۱

ارشیا عطایی | ۸۱۰۱۰۰۲۵۲

آدرس مخزن : https://github.com/ParsaMOBB/SWT-Course-Projects

شناسه آخرین کامیت : 5ceb14aad3844851dd2aab593824166c9370bb56

سوال ۱

behavior verification به بررسی این می‌پردازد که آیا یک متد خاص (مانند یک تابع یا عمل) در یک زمان خاص فراخوانی شده است یا خیر. به عبارتی دیگر، این نوع تأیید بر روی رفتار سیستم متمرکز است برای این verification، معمولاً از mockها استفاده می‌شود، زیرا این امکان را به ما می‌دهند که از dummyهای آموزش‌دیده‌ای که برای پاسخگویی به سناریوهای خاص تنظیم شده‌اند بهره ببریم. به این ترتیب، می‌توانیم انتظارات خود را درباره نحوه تعامل آنها با یکدیگر مشخص کنیم. از سمتی دیگر state verification به بررسی وضعیت نهایی یک شیء یا سیستم پس از اجرای یک یا چند عمل می‌پردازد. در این حالت، ما بررسی می‌کنیم که آیا وضعیت شیء به درستی تغییر کرده است یا خیر. این کار را با استفاده از stubها انجام می‌دهیم که پیچیدگی خاصی ندارند و فقط پاسخ‌های ثابتی به فراخوانی متدها ارائه می‌دهند، بدون اینکه نحوه فراخوانی آنها را بررسی کنند. به این ترتیب، stubها نتایجی را که برای ارزیابی وضعیت نهایی نیاز داریم فراهم می‌کنند.

سوال ۲

spy-Testها نوعی از double-testها هستند که اطلاعات مربوط به نحوه فراخوانی‌هایشان را در طول یک تست ذخیره می‌کنند. این قابلیت به برنامه‌نویسان امکان می‌دهد که در حین تست منطق کلی سیستم، رفتار یک بخش خاص را نیز مشاهده و بررسی کنند. spy-Testها به دلیل انعطاف‌پذیری بالایشان می‌توانند در سناریوهای تایید (verification) و مشاهده (observation) استفاده شوند، و به همین دلیل ابزاری چندمنظوره در toolkit تست به شمار می‌روند. از مزایای spy-testها می‌توان به امکان مشاهده تعاملات، انعطاف بالا و نیاز به تنظیمات کمتر اشاره کرد. این تست‌ها با کاهش نیاز به code setup پیچیده، فرآیند تست را ساده کرده و در عین حال وضوح تعاملات بین بخش‌ها را حفظ می‌کنند.

انواع spy-testها:

- **Spies Basic:** این اسپای‌ها دور متدهای فعلی پیچیده می‌شوند تا بتوانند فراخوانی‌ها و پارامترها را ثبت کنند، در حالی که رفتار متد اصلی را تغییر نمی‌دهند.
- **Spies Behavioral:** این نوع اسپای‌ها علاوه بر ثبت فراخوانی‌ها، قادر به تغییر رفتار در هنگام فراخوانی نیز هستند. برای مثال، ممکن است بر اساس آرگومان‌های ورودی، خروجی‌های مختلفی ارائه دهند.
- **Spies Thread-safe:** این اسپای‌ها در تست‌های همزمان به‌کار می‌روند تا تعاملات را به صورت دقیق و بدون تداخل ناشی از تردهای موازی ثبت کنند.

سوال ۳

الف) در شرایطی که آماده‌سازی یک **Fixture** (یا محیط آزمون) زمان‌بر است و چندین آزمون از یک مجموعه داده ثابت استفاده می‌کنند، استفاده از **Fixture Shared** مناسب‌تر است. در این حالت، به جای بازسازی کامل Fixture برای هر آزمون، یک نمونه مشترک از آن ایجاد می‌شود و در طول تست‌ها به اشتراک گذاشته می‌شود. این کار باعث صرفه‌جویی در زمان و بهبود کارایی آزمون‌ها می‌شود. اما اگر هر آزمون نیاز به حالتی تمیز و بدون تغییر داشته باشد که توسط دیگر آزمون‌ها تحت تأثیر قرار نگیرد، باید از **Fixture Fresh** استفاده کرد، چرا که این روش برای هر آزمون یک Fixture جدید ایجاد می‌کند.

ب) **Setup Lazy** مزیت‌هایی دارد از جمله اینکه فقط زمانی اجرا می‌شود که واقعاً به آن نیاز باشد، که این کار از اجرای غیرضروری تست‌ها جلوگیری می‌کند و سرعت اجرای تست‌ها را در مواقعی که برخی قسمت‌های سیستم نیاز به setup ندارند، افزایش می‌دهد. اما عیب آن این است که تا زمانی که آزمونی از setup استفاده نکند، پیاده‌سازی انجام نمی‌شود؛ بنابراین، ممکن است در مواقعی که وابستگی‌ها پیچیده باشند، ردیابی و اشکال‌یابی مشکلات سخت‌تر شود.

از سوی دیگر، **Setup Fixture Suite** تمام setupهای موردنیاز را از ابتدا آماده می‌کند که این کار باعث می‌شود همه‌ی تست‌ها از ابتدا در یک حالت مشخص و ثابت اجرا شوند و از بروز وابستگی‌ها و تداخل‌های ناخواسته بین تست‌ها جلوگیری می‌کند. این روش قابلیت اطمینان بیشتری دارد، اما به دلیل اجرای کامل setup، زمان بیشتری نیاز دارد و منابع بیشتری مصرف می‌کند، حتی اگر تمام تست‌ها به همه قسمت‌های setup نیازی نداشته باشند.

ج) برای اطمینان از اینکه تست‌ها در برابر تغییرات ناخواسته به یک Fixture مشترک مقاوم هستند، چند رویکرد مناسب وجود دارد:

1. **Fixture Fresh**: یکی از بهترین رویکردها استفاده از Fixture Fresh است؛ یعنی برای هر تست یک Fixture جدید ساخته می‌شود که پس از اجرای تست حذف می‌شود. این روش استقلال کامل تست‌ها را تضمین کرده و از تأثیر تغییرات ایجاد شده در سایر تست‌ها جلوگیری می‌کند.

2. **دیتابیس در حافظه (In-Memory Database)**: استفاده از دیتابیس در حافظه به ما امکان می‌دهد تا با هر بار اجرای تست، داده‌ها از ابتدا بارگذاری شوند. این باعث می‌شود هر تست در محیطی کاملاً تازه اجرا شود.

3. **بازگردانی وضعیت اولیه دیتابیس**: اگر از دیتابیس واقعی استفاده می‌کنید، پس از اجرای هر تست می‌توان وضعیت دیتابیس را به حالت اولیه بازگرداند تا تست‌ها از هم مستقل باشند و اجرای تست‌ها تحت تأثیر تغییرات تست‌های قبلی قرار نگیرد.

4. **بارگذاری داده‌های اولیه ثابت**: برای هر تست، داده‌های اولیه و مستقل بارگذاری کنید که ثابت و بدون تغییر در هر اجرا باقی بمانند و از تأثیرگذاری تست‌ها روی هم جلوگیری شود.

5. **استفاده از Mock یا Stub**: به جای تکیه بر داده‌های واقعی، استفاده از Mock یا Stub باعث می‌شود تست‌ها بدون وابستگی به داده‌های واقعی و تغییرات محیطی اجرا شوند، در نتیجه اجرای تست‌ها قابل کنترل‌تر و قابل اعتمادتر خواهد بود.

این روش‌ها کمک می‌کنند تا تست‌ها در برابر تغییرات ناخواسته مقاوم باقی بمانند و هر تست در محیطی مستقل و بدون تأثیر از سایر تست‌ها اجرا شود.