

## ریپو گیت هاب

**Repository:** <https://github.com/elahekhodaverdi/SWT-Fall103>

**Commit Hash:** b552309ec365ba38b141438ea0b57889ec28b855

## باگ های حل شده در کد

### سوال اول

#### State Verification

در این رویکرد، هدف بررسی وضعیت نهایی سیستم پس از اجرای یک عملیات است. به طور خاص، تمرکز بر روی این است که آیا مقادیر متغیرها، اشیاء، یا داده های ذخیره شده پس از اجرای عملیات به درستی به روزرسانی شده اند یا خیر. برای مثال این که آیا یک آبجکت به لیست اضافه شده است یا خیر یا آیا داده جدیدی به دیتابیس اضافه شده است یا خیر.

در این نوع اعتبارسنجی، فرآیند و مراحل که برای رسیدن به این وضعیت طی می شود، اهمیت چندانی ندارد؛ بلکه نتیجه نهایی مهم است.

در اجرای State Verification می توانیم از Stub استفاده کنیم. Stub ها آبجکت های ساده هستند که رفتارهای از پیش تعریف شده را تقلید می کنند تا به ما کمک کنند تمرکز خود را بر روی وضعیت نهایی سیستم معطوف کنیم.

#### Behavior Verification

در این رویکرد، هدف ما بررسی رفتار سیستم در حین اجرای عملیات است. این بدان معنی است که ما می خواهیم اطمینان حاصل کنیم که آیا سیستم کارهای مشخصی را در حین فرآیند انجام می دهد یا خیر. برای مثال آیا متدی با پارامترهای مشخصی فراخوانی می شود یا خیر یا اینکه مطمئن شویم متدی ابداً فراخوانی نمی شود و کارهایی از این قبیل. در این حالت، به جای بررسی نتیجه نهایی، تمرکز بر چگونگی اجرای عملیات است.

برای اجرای Behavior Verification می توانیم از Mock استفاده کنیم. Mock ها علاوه بر تقلید رفتار آبجکت ها، می توانند فراخوانی متدها را track کنند و بررسی کنند که آیا متدهای مورد انتظار در جریان عملیات به درستی فراخوانی شده اند یا خیر.

## سوال دوم

ابزار test-spy در unit-testing به منظور بررسی و مانیتور کردن رفتار توابع به کار گرفته می‌شوند. این ابزار به ما این امکان را می‌دهد تا بررسی کنیم که آیا یک سری اکشن‌های مدنظر ما اجرا شده یا نه و این کار بدون بازرسی روی خروجی فانکشن به کمک این ابزار انجام پذیر است. با این ابزار می‌توان بررسی کرد که یک تابع مثلاً چند بار اجرا شده و هر بار با چه آرگومان‌هایی اجرا شده است. دلایل استفاده از این ابزار به شرح زیر است:

- Behavior Verification
- Dependency isolation
- Increased Test Coverage

انواع test-spies:

1. Spy: در این مدل بررسی می‌شود که آیا یک تابع فراخوانی شده و اگر بله چندین بار و با چه ورودی‌هایی
2. Stub: این مدل به ما کمک می‌کند تا برای هر تابع رفتار خاصی را تعریف کنیم بدین صورت که حین فراخوانی چه مقادیری را بازگرداند یا چه خطایی را بیندازد. به صورت کلی ما می‌توانیم با این روش سناریوهای مختلف را بدون فراخوانی تابع شبیه سازی کنیم.
3. Mock: این مدل یک spy است که روی آن دسته‌ای از expectation ها ست شده است. در این مدل بنا بر این است که بررسی شود که آیا یک تابع با یک سری آرگومان مشخص فراخوانی می‌شود یا نه. از طرفی این مدل به نحوی کارساز است که مثلاً ما در رویه تست خود نیاز به فراخوانی تابعی برای خواندن دیتابیس داریم ولی چون این عمل زمان‌بر است از mock برای این منظور در تست‌ها بهره می‌بریم.

## سوال سوم

### (الف)

دو دلیل اصلی برای استفاده از Shared fixture به جای fresh fixture وجود دارد. دلیل اول سرعت است. گاهی می‌خواهیم فرایند تست سریع‌تر شود و استفاده از fresh fixture زمان زیادی برای راه‌اندازی fixtureها می‌گیرد، زیرا برای هر تست این فرایند دوباره انجام می‌شود. در این شرایط از Shared fixture استفاده می‌کنیم. هرچند پیچیدگی افزایش می‌یابد و تست‌ها به یکدیگر وابسته می‌شوند، که یافتن مشکل را دشوارتر می‌کند.

دلیل دوم این است که گاهی یک توالی طولانی از فعالیت‌ها داریم که هرکدام به قبلی وابسته هستند. اگر بخواهیم کل فرایند را تست کنیم، باید یک تست داشته باشیم که هر فعالیت را به ترتیب فراخوانی کند. هرچند، این روش همان‌طور که به نظر می‌رسد برای تست مناسب نیست، زیرا عیب‌یابی آن نیز دشوار است. در این حالت، از یک shared fixture استفاده می‌کنیم و هر فعالیت را به عنوان یک تست جداگانه در نظر می‌گیریم که به ترتیب باید اجرا شوند. در نتیجه، یک زنجیره‌ای از تست‌ها خواهیم داشت.

دلایل دیگری نیز وجود دارد، مانند این‌که ممکن است تست‌ها در محیطی انجام شوند که بین تست‌ها تغییری نمی‌کند یا تست‌ها تاثیری بر محیط نمی‌گذارند و به طور کلی تست‌ها تاثیری بر یکدیگر ندارند.

## (ب)

### مزیت‌ها:

در Lazy Setup منابع فقط زمانی تعریف می‌شوند که به آن‌ها نیاز باشد. بنابراین، اگر یک تست به منابع خاصی نیازی نداشته باشد، آن‌ها تعریف نمی‌شوند. در مقابل، در Suite Fixture Setup ممکن است منابعی در ابتدا ایجاد شوند که در بسیاری از تست‌ها استفاده نمی‌شوند، اما هزینه آن‌ها پرداخت می‌شود. به همین دلیل Lazy Setup بهینه‌تر است و بازدهی بیشتری دارد. همچنین، به دلیل این که منابع تنها در صورت نیاز تعریف می‌شوند، Lazy Setup از حافظه کمتری در مقایسه با Suite Fixture Setup استفاده می‌کند. از طرفی، در Lazy Setup این امکان وجود دارد که برای هر تست، منابع متفاوتی به صورت اختصاصی تعریف شوند. در حالی که در Suite Fixture Setup تصور می‌شود که منابع ثابتی داریم که باید در تمام تست‌ها استفاده شوند، که این باعث ایجاد محدودیت‌هایی در تعریف منابع برای همه تست‌ها می‌شود.

### معایب:

در Lazy Setup، چون منابع در هر تست جداگانه تعریف می‌شوند، بسیاری از تست‌ها ممکن است به منابع یکسانی نیاز داشته باشند و این منجر به تکرار کد خواهد شد. در مقابل، در Suite Fixture Setup منابع یکبار برای تمام تست‌ها تعریف می‌شوند و از تکرار کد جلوگیری می‌شود. در Lazy Setup، باید در هر تست منابع مورد نیاز مشخص شوند که این کار باعث افزایش پیچیدگی می‌شود. در حالی که در Suite Fixture Setup، منابع یکبار و به صورت مرکزی تعریف می‌شوند و این کار مدیریت آن‌ها را ساده‌تر می‌کند. اگر تست‌های زیادی نیاز به تعریف منابع یکسان داشته باشند، در مقایسه با Suite Fixture Setup زمان بیشتری نیاز است، زیرا هر بار منابع برای هر تست به صورت جداگانه تعریف می‌شوند.

## (ج)

به صورت کلی باید شرایطی را فراهم کرد که در صورت تغییر بتوان به استتیت اولیه برگشت. می‌توان از روش‌های زیر کمک گرفت:

- استفاده از ساختار داده های immutable  
این داده ها به ما کمک می کنند که حتی در صورت تغییر در آنها یک کپی از آنها تهیه شده و داده اصلی immune باقی بماند
  - استفاده از قابلیت transaction rollback در دیتابیس ها  
با استفاده از این قابلیت می توان تمامی تغییرات لازم در یک تست را در یک تراکنش قرار داده و آن را در انتها rollback کنیم.
  - استفاده از snapshot در دیتابیس ها  
با تهیه snapshot از دیتابیس در ابتدای تست و بازگردانی آن در انتها می توان از داده ها محافظت کرد.
- توجه شود این ویژگی برای سایر متغیر ها نیز قابل اجراست.
- استفاده از setup/teardown hooks
  - تبدیل کردن داده ها به تایپ read-only
- در این مدل باید توجه شود که اجازه هرگونه تغییر در داده ها از تست سلب می شود.