به نام خدا

درس تست نرم‌افزار

پروژه 2

محمد سعید صدیقی(810100179) / نیما تاجیک(810100104)

آخرین هش: eb3fe71382e7947c251c6d02632ee2baa4822f10 آدرس گیتهاب: https://github.com/nimaCod/Software-Test-Course-projects/tree/CA02

1. در state verification ما state های UUT مورد نظر را بررسی می کنیم و معمولا مستقل از دیگر moduleها و dependency ها می باشد. در واقع در این روش خروجی نهایی یا تغییر در متغیر ها که UUT باید انجام دهد بررسی می شود که همان تغییرات و یا خروجی مورد انتظار باشد. در این حالت از روش Stub استفاده می شود تا آن قسمت از کد که در حال تست آن هستیم را از باقی قسمت ها جدا کنیم و وابستگی که داریم را حذف کنیم. در behavior verification رفتار UUT مورد بررسی قرار می گیرد یعنی چه توابع، ماژول ها و یا سرویس هایی در حین اجرا مورد استفاده قرار گرفته‌اند و با چه آرگومان هایی صدا زده شده‌اند. در واقع در این حالت ما رفتار UUT را نسبت به ماژول ها و وابستگی های دیگر آن بررسی میکنیم که حتما تابع یا سرویس مورد نظر با ورودی های مورد نظر صدا زده شود و ... . به این منظور از Mock استفاده می کنیم که ابزار خوبی است برای بررسی صدا زدن سرویس ها و میتوانیم بررسی کنیم که آیا این تابع با این ورودی ها صدا زده شده است یا نه.

2. مانند mock روش test spy هم برای بررسی behavior verification مفید است اما برخلاف Mock ها در spy ها ما انتظارات از رفتار UUT ها در ابتدا ست نمی کنیم و بعد از اجرا آن ها بررسی می کنیم. ما در حالت کلی دو نوع spy داریم یکی partial spy و دیگری full spy. در partial spy ما فقط قسمت خاصی از رفتار های یک شئ را بررسی میکنیم و این شئ در باقی قسمت ها رفتار عادی خود را دارد. این روش برای بررسی ماژول ها یا سرویس های بزرگ که ما به دنبال تست قسمتی از آن ها هستیم کاربردی است. اما در full spy تمام methodها و توابع در

آن شئ مورد بررسی قرار می گیرند که شباهت زیادی به mock دارد با این تفاوت که ما رفتار آن شئ را بعد از انجام شدن کار بررسی میکنیم. برای مثال تعداد یوزر ها را از یک سرویس دریافت می کنیم و سپس چک می کنیم که آن سرویس حتما از متد کوئری خود استفاده کرده باشد.

3. الف) از shared fixture زمانی استفاده می‌کنیم که ایجاد مجدد fixture برای هر آزمون هزینه‌بر یا زمان‌بر باشد، یا وقتی که داده ها ثابت و تغییر ناپذیر هستند و تمامی آزمون ها می‌توانند بدون تاثیرگذاری روی یکدیگر از آن استفاده کنند. این روش مناسب است وقتی که آزمون ها فقط به داده های اولیه نیاز دارند و هیچ کدام از آزمون ها تغییراتی در داده ها ایجاد نمی‌کنند که آزمون‌های دیگر را تحت تاثیر قرار دهد.

ب) در مقابل، fresh fixture برای زمانی مناسب است که هر آزمون نیاز به یک نسخه جدید و جداگانه ازfixture دارد. این رویکرد برای مواقعی مناسب است که هر آزمون قرار است تغییراتی روی داده ها اعمال کند و این تغییرات نباید روی سایر آزمون ها تاثیر بگذارد.

استفاده از lazy setup:

مزایا: افزایش سرعت اجرای تست ها به دلیل ایجاد داده ها در صورت نیاز، کاهش مصرف منابع در زمان تست به دلیل اینکه داده ها در صورت نیاز ساخته می شوند.
معایب: ایجاد پیچیدگی در تست ها و ناخوانا و بلند شدن هر تست به دلیل ایجاد داده مورد نیاز. سخت شدن یافتن bug در تست ها.

استفاده از suite fixture setup:
مزایا: کاهش پیچیدگی در تست ها زیرا داده ها یک جا تولید می شوند.
معایب: ممکن است به دلیل استفاده از داده های مشترک، آزمون ها به هم وابسته شوند و در صورت بروز مشکل، اشکال یابی سخت‌تر باشد.

ج)
برای مثال می توانیم از fresh fixture استفاده کنیم. یعنی هر برای هر تست یه fixture جدید بسازیم تا روی هم تغییرات ایجاد نکنند. می توانیم در پایان هر تست تغییرات را بازگردانیم (برای هر تست این مدیریت شود). راه حل دیگر این است که مثلا برای دیتابیس از دیستابیس in-memory استفاده کنیم که پس از اتمام کار و اتمام کد از بین رود و اثری بر سایر داده ها نگذارد.