گزارش‌کار تمرین کامپیوتری دوم آزمون نرم افزار

محمدهادی بابالو – ۸۱۰۱۹۹۳۸۰ سنا ساری نوایی – ۸۱۰۱۹۹۴۳۵

شناسه آخرین کامیت

https://github.com/Hadi-loo/Software-Testing-Course/commit/bf3e89c10285d719d7a6d4c964e007105173c637

سوال ۱

Dependency injection الگویی جهت تزریق وابستگی‌های خارجی یک کلاس به آن، به جای استفاده مستقیم از آن وابستگی‌ها در درون کلاس می‌باشد. ابتدا به تعریف مختصر از dependency injection ها می‌پردازیم:

- Dependency injection by constructor: این روش این امکان را به ما می‌دهد که mock objectها، stubs و سایر test double ها را به عنوان وابستگی به یک کلاسی که آن را تست می‌کنیم، بدهیم. یعنی به جای آنکه وابستگی‌های موردنظرمان را در کلاس ایجاد کنیم، این وابستگی‌ها را هنگام ایجاد نمونه‌ای از کلاسی که می‌خواهیم آن را تست کنیم، به constructor آن کلاس منتقل می‌کنیم.
- Dependency injection by setter: در این روش، وابستگی‌ها از طریق متدهای setter ارائه می‌شود. این روش اغلب زمانی استفاده می‌شود که نیاز به تغییر یا به روز رسانی وابستگی‌ها در طول عمر یک object داریم اما در روش قبل، تنها در زمان ایجاد آن می‌توانیم این کار را انجام دهیم. در این حالت انعطاف پذیری بیشتری داریم اما اطمینان از اینکه همه وابستگی‌ها قبل از استفاده کاربر تزریق شده و معتبر هستند را دشوار می‌کند.
- Dependency injection by field: در این روش وابستگی‌ها مستقیما به فیلدها یا ویژگی‌های یک کلاس تزریق می‌شوند و این کار اغلب بدون استفاده از متدهای setter یا constructor انجام می‌شود.

برای حالاتی که dependency های ضروری داشته باشیم، بهتر است از dependency injection by constructor استفاده کنیم؛ زیرا هر object ای که در خود constructor دارد، بدون داشتن آن نمی‌تواند ساخته شود. به عبارت دیگر می‌توانیم اطمینان حاصل کنیم که کلاس مورد نظر تنها زمانی ساخته می‌شود که وابستگی‌هایش تأمین شده‌اند. اما در بقیه حالات و برای سایر وابستگی‌ها به طور کلی استفاده از setter پیشنهاد می‌شود.

محمدهادی بابالو – ۸۱۰۱۹۹۳۸۰ سنا ساری نوایی – ۸۱۰۱۹۹۴۳۵

سوال ۲

الف) یکی از دلایل این است که آن‌ها رفتار object های واقعی را جعل می‌کنند اما در واقع رفتار دقیق و واقعی آن‌ها را انجام نمی‌دهند. دلیل دیگر این است که از آنها به جای object واقعی استفاده می‌شود، یعنی test doubles به جای وابستگی‌های واقعی جایگزین می‌شوند و به همین دلایل به آنها imposters گفته می‌شود.

ب) test double ها اجزایی هستند که به جای پیاده‌سازی واقعی برای تسهیل تست استفاده می‌شوند و به جداسازی کد تست از وابستگی‌های آن کمک می‌کنند. انواع test double ها به شرح زیر می‌باشد:

- Dummy Object: این object ها به عنوان پارامتر پاس می‌شوند ولی درواقع در تست استفاده نمی‌شوند و فقط برای اینکه امضای توابع صحیح باشد کاربرد دارند. یعنی مقدار واقعی پارامتر به تست مربوط نیست و هیچ تاثیری بر رفتار یا نتیجه تست ندارد. با استفاده از این نوع از خطاهای کامپایل هم جلوگیری می‌شود.
- Test Stub: برای ارائه کد تست شده با indirect input استفاده می‌شود. آنها به گونه ای طراحی شده‌اند که پاسخ‌های از پیش تعریف شده‌ای را به فراخوانی‌های متدهای انجام شده در طول تست ارائه می‌دهند و به هیچ چیز خارج از رفتار برنامه ریزی شده برای تست پاسخ نمی‌دهند. این روش با اطمینان از اینکه تعاملات با وابستگی‌ها کنترل شده و قابل پیش بینی است، به جداسازی کد تحت آزمایش کمک می‌کند.
- Test Spy: برای ضبط و تایید indirect output یا تعاملات بین کد تست و وابستگی‌های آن استفاده می‌شود. Test spy را می‌توان در موقعیت‌هایی استفاده کرد که می‌خواهیم تعاملات را ضبط، نظارت و تایید کنیم؛ بدون آنکه آن‌ها را از قبل مشخص کنیم. اغلب برای اطمینان از اینکه کد تست به درستی با وابستگی‌های آن تعامل دارد استفاده می‌شود.
- Mock Object: برای تایید indirect output در کد تست استفاده می‌شود؛ با این فرض که انتظاراتمان را قبل از اجرای کد تست تعریف کنیم. منظور از انتظارات این است که مشخص شود چه روش‌هایی باید بر روی وابستگی‌ها و با چه استدلالی فراخوانی شوند.در واقع ما انتظارات خود از چگونگی فراخوانی mock object ها تعیین می‌کنیم و تست بررسی می‌کند که این انتظارات برآورده می‌شوند یا نه.
- Fake Object: به عنوان یک پیاده‌سازی ساده‌تر استفاده می‌شود. معمولا برای ارائه جایگزینی ایجاد می‌شوند که پیچیدگی کمتر و سبکتر از اجرای واقعی داشته باشند. یکی از کاربردهای رایج

Fake object ها جایگزینی پایگاه داده واقعی با پایگاه‌های داده درون حافظه در طول تست است. این کار اجازه می‌دهد تست های سریع‌تری را بدون نیاز به دسترسی به پایگاه داده فیزیکی انجام دهیم.

سوال ۳

تفاوت‌های classical و mockist:

1. تست classical بر تست behavior و state تاکید دارد اما تست mockist بر تعامل بین unit و وابستگی‌های آن تاکید دارد.
2. تست classical اولویتش بررسی وضعیت یا نتایج فراخوانی متد می‌باشد، در حالی که تست mockist به نحوه تعامل متد با وابستگی های آن مربوط می شود.
3. تست classical از mocks به مقدار کم و در درجه اول برای وابستگی های خارجی استفاده می‌کند، در حالی که تست mockist برای کنترل تعاملات به شدت به mock object ها متکی است.
4. تست های mockist ممکن است در تعیین نحوه تعامل object ها واضح تر باشد، که می تواند خوانایی تست را افزایش دهد. با این حال، آنها همچنین می توانند تست‌ها را پیچیده تر کنند و با جزئیات بیشتری پیاده سازی کنند.

مزایای classical:

1. برای درک و پیاده‌سازی راحت‌تر می‌باشد. درک مفهوم تست نیز برای توسعه دهندگان آسان‌تر است.
2. تست های classical معمولا مستقل هستند و این کار، نوشتن unit test را ساده‌تر می‌کند.
3. این نوع تست‌ها، به دلیل اینکه کمتر تحت تاثیر جزئیات پیاده‌سازی قرار می‌گیرند، پایدارتر هستند.
4. تست classical بر آزمایش رفتار واقعی کد تاکید دارد که می تواند منجر به تست های قوی‌تر و قابل اعتمادتر شود.
5. این نوع تست سربار کمتری نسبت به mocking دارد زیرا نیاز به setup کردن mock ها جهت جایگزینی وابستگی‌ها ندارد و این منجر به افزایش سرعت تست‌ها می‌شود.

معایب classical:

1. از آنجایی که تست classical استفاده از mock ها را به حداقل می‌رساند، ممکن است برای زمانی که نیاز به تعامل با سرویس‌های نیاز به شبیه‌سازی داشته باشیم، مناسب نباشد.
2. این نوع از تست می‌تواند حس امنیت کاذبی را ایجاد کند، زیرا بر تعامل بین object ها تمرکز نمی‌کند و برخی از مسائل یکپارچه‌سازی ممکن است نادیده گرفته شود.
3. تست classical برای سیستمی با تعاملات پیچیده ممکن است ناکافی باشد و نوع دیگری از تست را برای این حالت در نظر بگیریم.
4. همانطور که میدانیم، تمرکز این نوع تست بر state و behavior است. تمرکز بر این حالات به تنهایی ممکن است منجر به تست‌های با خوانایی کمتر شود.

مزایای mockist:

1. Mockist test این امکان را به ما می‌دهد که واحد خاصی از کدی که در حال تست آن هستیم را با جایگزین کردن وابستگی های آن mock object جدا کنیم. این کار می‌تواند تشخیص هرگونه مشکل یا خرابی را آسان‌تر کند.
2. تست mockist می تواند به شناسایی مشکلات در مراحل اولیه توسعه کمک کند. اگر مشکلی در تعامل بین واحد و وابستگی‌های آن وجود داشته باشد، در طول آزمایش مشخص می‌شود و رفع آن آسان‌تر می‌شود.
3. جایگزینی با mock object ها باعث افزایش سرعت تست می‌شود که این موضوع در یک محیط یکپارچه سازی بسیار مهم است.
4. Mocking test یک روش behavior-driven است. به این معنی که بر رفتار مورد انتظار واحد مورد آزمایش تمرکز دارد و این می‌تواند به موارد آزمایشی معنادارتر منجر شود.

معایب mockist:

1. نوشتن تست برای برنامه‌هایی با وابستگی‌های زیاد سخت‌تر است. همچنین در این حالت خواندن و نگهداری تست سخت‌تر می‌شود.
2. استفاده از mock object می‌تواند تست را کمتر واقع‌بینانه کند زیرا تعاملات دنیای واقعی به طور کامل و دقیق تکرار نمی‌شود.
3. استفاده بیش از حد از mock object ها میتواند منجر به تست‌هایی شود که به اندازه کافی رفتار

واقعی سیستم را تایید نمی‌کند.

4. Mockist test ها می‌توانند شکننده باشند. تغییرات در پیاده‌سازی داخلی ممکن است به به‌روزرسانی‌های زیاد در تست‌ها منجر شود.