آزمون نرم‌افزار یکی از فعالیت‌های کلیدی در چرخه توسعه نرم‌افزار است. امروزه زندگی ما به سیستم‌های نرم‌افزاری وابسته شده است. این سیستم‌ها متنوع هستند و شامل سیستم‌های تحت شبکه، سامانه‌های بانکی، حمل و نقل و بهداشت می‌شوند. کاربرانی که تحت تأثیر این سیستم‌ها قرار می‌گیرند، انتظار دارند که سیستم مطابق با انتظارات آن‌ها عمل کند. آزمون نرم‌افزار روش اصلی صنعت برای ارزیابی نرم‌افزار در کل فرآیند توسعه نرم‌افزار است. برای درک بهتر آزمون، ابتدا لازم است با چند مفهوم مرتبط آشنا شویم.

مفاهیم پایه

خطا: اشتباهی که در کد نرم‌افزار وجود دارد و باعث می‌شود نرم‌افزار مطابق با نیازمندی‌های تعریف شده عمل نکند، خطا (fault) نامیده می‌شود.

برای درک بهتر خطا، به کد زیر توجه کنید. تابع نوشته شده قرار است تعداد اعداد نامنفی را در یک لیست برگرداند. اما این برنامه دارای خطا است. در خط شماره ۴، اگر عنصر element برابر صفر باشد، به اشتباه به عنوان عدد نامنفی محاسبه نمی‌شود. بنابراین، انتظار می‌رود که در خط شماره ۴، عبارت if element >= 0 را ببینیم، اما عبارت if element > 0 ظاهر شده است.

```
1def non_negative(lst):
2    count = 0
3    for element in range(lst):
4        if element > 0:
5            count += 1
6    return count
```

برنامه‌نویسان معمولاً در قالب کامیت‌هایی سعی در رفع خطا موجود در نرم‌افزار می‌کنند. به طور مثال، این لینک، کامیت رفع خطا در پروژه سوئیفت را نشان می‌دهد.

خرابی: به رفتار قابل مشاهده از نرم‌افزار که با انتظارات ما مطابقت نداشته باشد، خرابی (failure) یا شکست می‌گویند.

همان مثال فوق را در نظر بگیرید. اگر مقدار lst به عنوان ورودی تابع [3,11,7,-5,-2] باشد، با وجود نقص در برنامه، خروجی برنامه صحیح است و شکستی مشاهده نمی‌کنیم. حال اگر مقدار lst برابر [2,3,-5,0] باشد، به خاطر وجود عدد صفر در لیست، خروجی برنامه مطابق انتظار نیست و یک خرابی را شاهد هستیم.

معمولاً سیستم‌های گزارش باگ، امکان گزارش وقوع خرابی را فراهم می‌کند. به طور مثال، این لینک، یک نمونه خرابی در پروژه سوئیفت است.

اشتباه: به وضعیت نادرستی در حالت داخلی برنامه که به خاطر وجود خطا ایجاد شده است، اشتباه (error) می‌گویند.

برای درک بهتر مفهوم اشتباه، به این مثال توجه کنید. فرض کنید بیماری به خاطر حالت تهوع به پزشک مراجعه کرده است. پزشک برای درک بهتر شرایط، یک سری آزمایش انجام می‌دهد، مانند بررسی میزان فشار خون و ضربان قلب نامنظم. در این جا، حالت تهوع همان خرابی است که ممکن است به خاطر مسمومیت باشد. مسمومیت معادل خطا است. فشار خون بالا یا ضربان قلب نامنظم همان حالت داخلی نادرست است که به آن اشتباه می‌گویند.

تفاوت آزمون با روش‌های صوری

باید تأکید کنیم که داشتن آزمون‌های جامع و کامل که با موفقیت اجرا می‌شوند، به معنای نبود خطا در برنامه نیست. ممکن است خطایی وجود داشته باشد که هنوز کشف نکرده‌ایم. عموماً با کمک آزمون نمی‌توانیم بگوییم که نرم‌افزار به صورت ۱۰۰ درصدی درست کار می‌کند، بلکه تنها اطمینان خود را نسبت به کیفیت بالای آن افزایش می‌دهیم. برای اطمینان از صحت کامل برنامه که وجود خطا را به طور کامل نفی کند، نیازمند روش‌های صوری هستیم. متأسفانه، روش‌های صوری به خاطر ماهیت ریاضی آن حتی برای برنامه‌های ساده با چند خط کد هم وقت‌گیر است. به همین خاطر، از روش‌های صوری تنها برای پروژه‌های با درجه اهمیت بحرانی که با جان انسان‌ها سروکار دارد استفاده می‌کنیم. سیستم

کنترل‌کننده ترمز یک ماشین را در نظر بگیرید. منطق چنین برنامه‌ای پیچیده نیست، اما لازم است از درستی ۱۰۰ درصدی آن اطمینان حاصل کنیم زیرا هر گونه اشتباه در برنامه می‌تواند سبب از بین رفتن جان یک انسان بیگناه شود. اینجاست که روش‌های صوری ظاهر می‌شود.

اعتبارسنجی و صحت‌سنجی

منظور از صحت‌سنجی (verification)، اطمینان از منطبق بودن محصول یک مرحله از چرخه ایجاد نرم‌افزار با نیازمندی‌های مرحله قبل از آن است. اعتبارسنجی (validation)، از سوی دیگر، اطمینان از ساخت محصولی است که کاربر می‌خواهد. به زبان ساده، صحت‌سنجی یعنی آیا محصول را به شیوه درست می‌سازیم و اعتبارسنجی یعنی آیا محصول مناسب را می‌سازیم. برای تضمین صحت‌سنجی، از انواع تکنیک‌های کنترل کیفیت استفاده می‌کنیم: آزمون نرم‌افزار، مرور کد، اجبار سبک‌کد زنی، یکپارچه‌سازی مستمر/تحویل مستمر. برای تضمین اعتبارسنجی، کاربر نهایی را درگیر می‌کنیم: استفاده از نقش مالک محصول در اسکرام، آزمون آلفا و بتا.

انواع آزمون

آزمون‌های نرم‌افزار را می‌توان از منظر کارکرد، سطح درشت‌دانگی و هدف دسته‌بندی کرد. این دسته‌بندی‌ها برای سازماندهی کد آزمون و درک بهتر آن‌ها انجام می‌شود. برای فهم بهتر متن پیش رو، هر جا که لازم باشد، از پروژه سوئیفت به عنوان مثال استفاده می‌کنیم. سوئیفت یک سرویس ذخیره‌ساز اشیاء است و یکی از پروژه‌های اوپن‌استک است.

آزمون واحد

در آزمون واحد، کوچکترین قطعه کد دارای عملکرد را مورد آزمون قرار می‌دهیم. به طور مثال، یک تابع یا کلاس می‌تواند در آزمون واحد به عنوان هدف انتخاب شود. می‌خواهیم بدانیم که آیا تابع یا کلاس با تغییر ورودی‌ها، خروجی مورد انتظار را تولید می‌کنند یا خیر. در آزمون واحد، تا جای ممکن قطعه کدی که مورد آزمایش قرار می‌گیرد باید از باقی اجزای سیستم ایزوله باشد. از آنجایی که کلاس‌ها و توابع با یکدیگر در ارتباط هستند، گاهی این ایزوله کردن ممکن نیست. در این صورت دو راهبرد وجود دارد:

- هر جا که لازم باشد، کلاس‌ها یا توابع را فراخوانی کنیم تا پیش‌شرط آزمون محقق شود.
- از ماک (mock) استفاده کنیم.

نکته: توجه داشته باشید که استفاده از راهبرد ماک باید به صورت حداقلی باشد. در حالت کلی، یک برنامه‌نویس هنگام نوشتن کد باید به آزمون‌پذیری قطعه کد توجه کند. استفاده مکرر و بیجا از ماک سبب می‌شود که کیفیت آزمون نوشته‌شده کاهش یابد.

این نکته را با یک مثال شرح می‌دهیم. فرض کنید تابعی به نام total_value در ماژولی به نام core.py داریم. این تابع مجموع قیمت صورت‌حساب خرید را محاسبه می‌کند.

```
1# core.py
2from db import db_read
3def total_value(email, invoice_id, discount=0.0):
4    items = db_read(email, inv_no=invoice_id)
5    return sum(items) + (1.0 - discount)
```

تابع db_read آدرس رایانامه خریدار و شناسه صورت‌حساب را دریافت می‌کند و با خواندن از پایگاه داده، اقلام آن را به عنوان خروجی باز می‌گرداند. خواندن از پایگاه داده می‌تواند زمانبر باشد. بنابراین، برنامه‌نویس تصمیم می‌گیرد که برای آزمایش تابع، از ماک استفاده کند.

```
1# test_core.py
2from unittest.mock import patch, call
3from core import total_value
```

```
4def test_total_value:
5    with path("db.db_read") as mock_read:
6        mock_read.return_value = [100, 200]
7        assert 300. == total_value("example@example.com", 92)
8        assert [call("example@example.com", inv_no=92)] == mock_read.calls
```

در ظاهر، این مثال ساده که تنها از یک ماک استفاده می‌کند، مشکلی ندارد و آزمون هم با موفقیت پاس می‌شود. اما کد آزمون از منظر رعایت کردن اصول کد تمیز، مشکلاتی دارد. مهمترین مشکل کد فوق این است که تابع test_total_value قرار است تنها total_value را مورد آزمون قرار دهد، اما کد آن به db_read هم وابستگی دارد. به عنوان مثال، فرض کنید در آینده تصمیم گرفتیم که نام تابع db_read به read_from_database تغییر کند، آنگاه مجبور خواهیم بود که خط شماره ۵ کد آزمون را هم تغییر دهیم. یا اگر به هر دلیل خروجی تابع db_read به جای لیستی از اقلام به تاپل تغییر کند که عنصر اول آن لیستی از اقلام و عنصر دوم آن مقدار true/false باشد (نشان‌دهنده موفقیت‌آمیز بودن عملیات پایگاه داده)، در این صورت مجبور خواهیم شد که خط شماره ۶ کد آزمون را تغییر دهیم.

در حالت ایده‌آل، تابع test_total_value تنها به صورت مستقیم باید به total_value وابستگی داشته باشد و نسبت به تابع db_read دیدی نداشته باشد. چگونه می‌توانیم با اجتناب از ماک، مسئله فوق را حل کنیم؟ در این مثال خاص، می‌توانیم یک داده ساختار در ماژول db تعریف کنیم که نقش پایگاه داده را به صورت فیک ایفا کند. از آنجایی که داده ساختار در حافظه جای می‌گیرد، مسئله کند بودن برطرف می‌شود و همچنین کد آزمون هم نسبت به db_read اطلاعی نخواهد داشت.

نکته: توابع، کلاس‌ها، داده‌ساختارها و کد مدل‌های نگاشت شده به پایگاه داده از جمله مواردی هستند که در آزمون واحد به عنوان قطعه کد هدف انتخاب می‌شوند.

نکته: آزمون واحد در خط‌لوله یکپارچه‌سازی مستمر/تحویل مستمر، در بخش یکپارچه‌سازی مستمر و بعد از فعالیت‌های ساخت و تحلیل ایستا قرار می‌گیرد.

آزمون یکپارچگی

در آزمون یکپارچگی، بر خلاف آزمون واحد، همکاری و تعاملات بخش‌های مختلف کد برنامه (یا حتی سرویس‌های خارجی که تحت کنترل ما نیست) را مورد آزمایش قرار می‌دهیم. آزمون‌های یکپارچگی معمولاً بر اساس یک سناریو طراحی می‌شوند. به عنوان نمونه، این سناریو را در نظر بگیرید: در سوئیفت، یک شیء ایجاد می‌کنیم. سپس حسابی را که این شیء به آن تعلق دارد، حذف می‌کنیم. بعد از حذف حساب، باید شیء و تکرارهای آن سرانجام حذف شوند. در سوئیفت، آزمون‌های یکپارچگی با نام probe شناخته می‌شوند.

نکته: آزمون یکپارچگی در خط‌لوله یکپارچه‌سازی مستمر/تحویل مستمر، در بخش یکپارچه‌سازی مستمر و بعد از فعالیت آزمون واحد قرار می‌گیرد.

آزمون سیستمی

در آزمون سیستمی، تمام سیستم به صورت یک تکه (از دید برنامه‌نویس) مورد آزمایش قرار می‌گیرد. این آزمون معمولاً در سطح واسط برنامه‌نویسی کاربردی صورت می‌گیرد. آزمون سیستمی را با نام‌های آزمون فانکشنال و آزمون اِی.پی.آی نیز می‌شناسند. منظور از دید برنامه‌نویس این است که هنگام نوشتن آزمون، به جزئیات فنی قطعه کد توجه داریم. اگر واسط برنامه‌نویسی کاربردی صرفاً در اختیار تیم توسعه قرار دارد و برای کاربر نهایی قابل استفاده نیست، باز هم در آزمون سیستمی مورد آزمایش قرار می‌گیرد. همچنین فرض می‌کنیم که سیستم می‌تواند با سرویس‌های خارجی که تحت کنترل تیم برنامه‌نویسی نباشد، تعامل داشته باشد..

نکته: آزمون سیستمی در خط‌لوله یکپارچه‌سازی مستمر/تحویل مستمر، در بخش یکپارچه‌سازی مستمر و بعد از فعالیت آزمون یکپارچگی قرار می‌گیرد.

آزمون پذیرش

در بالاترین سطح درشت‌دانگی، آزمون پذیرش قرار دارد. در اینجا، از دید کاربر نهایی، سیستم را مورد آزمایش قرار می‌دهیم. در این آزمون، سیستم در محیط واقعی محک می‌خورد. به طور مثال، مرورگر باز می‌شود، آدرس مورد نظر در مرورگر وارد می‌شود و کاربر نتیجه درخواست خود را مشاهده می‌کند. این نوع از آزمون‌ها از روی داستان‌های کاربری استخراج می‌شوند. به عنوان نمونه، این داستان کاربری را در نظر بگیرید: به عنوان کاربر،

می‌خواهم زمانی که یک فایل را در سیستم آپلود می‌کنم، اگر فایل از قبل موجود باشد، آن را با نسخه جدید جایگزین کند. برای خودکارسازی این نوع از آزمون‌ها، ابزارهایی مانند سِلِنیوم ممکن است به کار گرفته شوند.

نکته: در عمل، هر چه از آزمون واحد به سمت آزمون پذیرش حرکت می‌کنیم، پیچیدگی آزمون افزایش می‌یابد. همچنین، به عنوان یک قاعده کلی، حداقل ۸۰ درصد موارد آزمون باید توسط آزمون واحد تشکیل شود و هر چه آزمون درشت‌دانه‌تر می‌شود، تعداد کمتری از آن خواهیم داشت.

شکل ۱. هرم آزمون

نکته: آزمون پذیرش، که بخشی از آن می‌تواند به صورت خودکار وارد خط‌لوله یکپارچه‌سازی مستمر/تحویل مستمر شود، در قسمت تحویل مستمر و بعد از آزمون سلامت و استقرار محصول قرار می‌گیرد.

آزمون سلامت

آزمون سلامت، یک سری آزمون مقدماتی است که بعد از بالا آوردن سیستم انجام می‌شود تا اطمینان حاصل کنیم که سیستم در حال کار کردن است. اطمینان از اجرا شدن برنامه، بالا بودن سرویس‌ها و باز بودن پورت‌ها، و اجرای برخی از مهمترین قابلیت‌های سیستم در این آزمون بررسی می‌شود. هدف از این آزمون آن است که در خط‌لوله یکپارچه‌سازی مستمر/تحویل مستمر، اگر سیستم این آزمون را با موفقیت پشت سر نگذاشت، آزمون‌های پذیرش اجرا نشوند تا هزینه محاسباتی اجرای خط لوله کاهش پیدا کند. این آزمون با نام آزمون دود هم شناخته می‌شود.

آزمون‌های گفته شده تا به اینجا، انتظارات ما از آنچه که سیستم انجام می‌دهد را محک می‌زند. از طرف دیگر، آزمون‌های غیرعملکردی نیز وجود دارد که به بررسی اینکه چقدر خوب این انتظارات برآورده می‌شوند، می‌پردازند. در ادامه، به توضیح این نوع آزمون‌ها خواهیم پرداخت.

نکته: آزمون سلامت در خط‌لوله یکپارچه‌سازی مستمر/تحویل مستمر، در بخش تحویل مستمر و بعد از استقرار محصول صورت می‌گیرد.

آزمون بار

آزمون بار، سیستم را در بازه پیش‌بینی‌مان از تعداد کاربران و درخواست‌ها آزمایش می‌کند. به عنوان مثال، اگر انتظار داریم که در هر روز ۵۰۰۰ درخواست خواندن شیء به سوئیفت ارسال شود، وضعیت سیستم را (زمان پاسخ، میزان مصرف حافظه، میزان مصرف پردازش و ...) در طیف صفر تا ۵۰۰۰ درخواست بررسی می‌کند. آزمون بار یکی از انواع آزمون‌های عملکردی محسوب می‌شود.

آزمون استرس

آزمون استرس، سیستم را در ورای انتظارات ما از تعداد کاربران و درخواست‌ها آزمایش می‌کند. به عنوان مثال، اگر انتظار داریم که در هر روز ۵۰۰۰ درخواست خواندن شیء به سوئیفت ارسال شود، وضعیت سیستم را برای ۵۰۰۰ درخواست به بالا بررسی می‌کند. می‌خواهیم بدانیم که با افزایش درخواست‌ها که بیش از حد توان سیستم است، عملکرد به چه میزان افت می‌کند. آزمون استرس هم یکی از انواع آزمون‌های عملکردی محسوب می‌شود.

نکته: چارچوب Locust یک ابزار آزمون بار و آزمون استرس است که با زبان برنامه‌نویسی پایتون نوشته شده است. آزمون‌های نوشته‌شده در این ابزار تماماً به زبان پایتون هستند و می‌توانند به راحتی با سیستم‌ها یکپارچه شوند.

آزمون والیوم

آزمون والیوم، پایگاه داده سیستم را با داده پر می‌کند (حجم داده می‌تواند زیاد باشد) و سپس عملکرد سیستم را با داشتن این حجم از داده آزمایش می‌کند. انتظار می‌رود که هر چقدر مقدار داده در پایگاه داده افزایش پیدا کند، از یکجایی به بعد کارایی سیستم کاهش پیدا کند. آزمون حجم نیز یکی از انواع آزمون‌های عملکردی محسوب می‌شود. همچنین، در آزمون حجم می‌توان وضعیت عملکرد سیستم را با تغییر ابعاد داده ورودی بررسی کرد.

آزمون پیکربندی

آزمون پیکربندی، وضعیت سیستم را تحت پیکربندی‌های گوناگون نرم‌افزاری، سخت‌افزاری و شبکه آزمایش می‌کند. آزمون پیکربندی هم یکی از انواع آزمون‌های عملکردی محسوب می‌شود.

نکته: بر اساس مطالب فوق، آزمون عملکردی (performance) شامل چهار نوع آزمون است:

- آزمون بار
- آزمون استرس
- آزمون حجم
- آزمون پیکربندی

نکته: برخی از آزمون‌های عملکردی که امکان قرارگیری در خط لوله یکپارچه‌سازی مستمر/تحویل مستمر را دارند، در انتهای خط‌لوله تحویل مستمر قرار می‌گیرند.

نکته: لازم به ذکر است که در برخی از منابع و اسناد، تعریف‌های متفاوتی برای آزمون‌های عملکردی، بار و استرس ارائه شده است، نسبت به آنچه در اینجا بیان شده است. به عنوان مثال، خوانندگان می‌توانند به اسناد نماتن مراجعه کنند تا با تعریف دیگری از این اصطلاحات آشنا شوند.

آزمون امنیت

در بخش امنیت، به دنبال آسیب‌پذیری‌ها، یافتن مشکلات امنیتی نرم‌افزار، سرویس‌ها و شبکه هستیم. به عنوان نمونه، اطلاعات حساس مانند رمز عبور نباید در کد مخزن قرار بگیرد. آزمون نفوذپذیری یکی از روش‌های بررسی موارد امنیتی است. این آزمون با شبیه‌سازی حمله هکرها به دنبال رخنه به سیستم هدف است.

نکته: آزمون‌های امنیت به دو دسته ایستا و پویا تقسیم می‌شوند. بخش ایستا در خط لوله یکپارچه‌سازی مستمر/تحویل مستمر در بخش یکپارچه‌سازی مستمر، بعد از فعالیت ساخت و قبل از آزمون واحد صورت می‌گیرد. بخش پویا در انتهای تحویل مستمر و بعد از آزمون‌های نظیر آزمون عملکردی قرار می‌گیرد.