مفاهیم پرکاربرد در تست نرم‌افزار

ویراست 0.5

T

- **Top-Down Testing**: توی Integration Testing ماژول‌های سطح بالا (High-Level) رو اول تست می‌کنیم و بعد کم‌کم می‌ریم سراغ ماژول‌های پایین‌تر (Low-Level). چون توی این روش ممکنه بعضی از ماژول‌های سطح پایین هنوز آماده نباشن، از Stubs (کدهای موقتی که رفتار اون ماژول‌ها رو شبیه‌سازی می‌کنن) استفاده می‌شه.

U

- **Unit Testing**: یکی از پایه‌ای‌ترین و مهم‌ترین تست‌های نرم‌افزاره که توش هر بخش کوچک از کد (معمولاً به تابع یا متد) به‌صورت جداگانه تست می‌شه.
- **Usability Testing**: یکی از تست‌های Non-Functional که بررسی می‌کنه یه نرم‌افزار یا وب‌سایت چقدر برای کاربران راحت، قابل‌فهم و کاربرپسنده است. این تست معمولاً با کمک کاربران واقعی انجام می‌شه.
- **User Acceptance Testing (UAT)**: یکی از آخرین مراحل تست نرم‌افزار که بررسی می‌کنه آیا محصول واقعاً مطابق نیازهای کسب‌وکار و انتظارات کاربر نهایی کار می‌کنه یا نه. این تست معمولاً توسط کاربران واقعی یا نماینده‌های کسب‌وکار انجام می‌شه تا تأیید کنن که نرم‌افزار آمادی انتشار هست.

V

- **Volume Testing**: بررسی می‌کنه سیستم تحت حجم بالای داده چطور رفتار می‌کنه. این تست کمک می‌کنه بفهمیم که وقتی دیتابیس یا سرور با مقدار زیادی داده پر بشه، آیا نرم‌افزار همچنان درست کار می‌کنه یا کند و ناپایدار می‌شه.

W

- **White Box Testing**: یکی از روش‌های تست که ساختار داخلی کد، منطق، و جریان داده‌ها بررسی می‌شه. برخلاف Black Box که تستر فقط خروجی رو بررسی می‌کنه، توی White Box تستر باید کد رو ببینه و تست‌ها رو بر اساس جریان منطقی اون بنویسه.

S

- **Sanity Testing**: یه نوع تست سریع و سطحی که بعد از یه تغییر کوچیک یا Bug Fix اجرا می‌شه تا ببینیم آیا مشکل حل شده یا نه، بدون اینکه تست‌های عمیق انجام بدیم. این تست مثل یه چک اولیه قبل از اجرای تست‌های کامل‌تر مثل Regression Testing عمل می‌کنه.
- **Smoke Testing**: یه تست سریع و اولیه که بررسی می‌کنه آیا بیلد جدید ارزش تست‌های عمیق‌تر رو داره یا نه. این تست معمولاً روی ویژگی‌های اصلی و حیاتی نرم‌افزار اجرا می‌شه و اگه رد بشه، یعنی کل بیلد مشکل داره و نباید وقت بیشتری روی تست‌های بعدی بذاریم!
- **Static Testing**: یکی از دو نوع اصلی تست نرم‌افزار (در کنار Dynamic Testing) هست که بدون اجرای کد انجام می‌شه، و تمرکزش روی بررسی اسناد، کد، و ساختار نرم‌افزار قبل از اجراست. این تست کمک می‌کنه که مشکلات زودتر و قبل از اجرا کشف بشن.
- **Stress Testing**: تست عملکردی (Performance Testing) که بررسی می‌کنه سیستم تحت شرایط فوق‌العاده سخت و غیرعادی چطور رفتار می‌کنه. این تست کمک می‌کنه ببینیم نرم‌افزار توی شرایطی مثل حجم بالای درخواست، کمبود منابع، یا قطعی سرویس‌ها چقدر مقاومه.
- **Stub**: یکی از مفاهیم Integration Testing و Unit Testing که ماژول یا متد ساده جایگزین یه وابستگی واقعی توی تست می‌شه. Stub معمولاً یه مقدار ثابت برمی‌گردونه و منطق پیچیده‌ای نداره.

  📌 تفاوت Stub و Mock:
  - Stub فقط داده‌های تستی برمی‌گردونه.
  - Mock رفتار و تعاملات یه کلاس رو شبیه‌سازی می‌کنه.
- **System Testing**: کل نرم‌افزار رو به‌صورت یکپارچه بررسی می‌کنیم تا ببینیم آیا همه‌ی بخش‌ها «با هم» درست کار می‌کنن یا نه. این تست بعد از Integration Testing و قبل از Acceptance Testing انجام می‌شه.

T

- **Test Case**: یه یک سناریوی دقیق و از پیش تعریف‌شده برای آزمدن یک قابلیت خاص گفته می‌شه. هر Test Case مشخص می‌کنه که چی رو باید تست کنیم، با چه ورودی‌هایی، و انتظار داریم چه خروجی‌ای بگیریم.
- **Test-Driven Development (TDD)**: یعنی قبل از نوشتن کد اصلی، اول تست‌ها رو می‌نویسیم! این باعث می‌شه که کد تمیزتر، تست‌پذیرتر و بدون باگ‌های ناگهانی باشه.

  📌 چرخه‌ی TDD اینجوریه:
  1. نوشتن یه تست که اول Fail می‌شه ❌ (چون هنوز کدی نداریم!)
  2. نوشتن کمترین مقدار کد ممکن برای پاس شدن تست ✅
  3. بهبود و ریفکتور کردن کد 🔄
  4. اجرای دوباره تست‌ها برای اطمینان از صحت کد ✅
- **Test Environment**: به محیطی شبیه به محیط Production ولی مخصوص تست کردن نرم‌افزار گفته می‌شه. این محیط شامل دیتابیس تست، سرور تست، APIهای تستی، و ابزارهای مانیتورینگ هست تا تست‌ها بدون تأثیر روی کاربران واقعی انجام بشن.

  📌 مثلا برای یه اپلیکیشن بانکی، یه Test Environment می‌تونه شامل اینا باشه:
  - ✅ یه سرور جداگانه که توش نسخه‌ی آزمایشی اپ اجرا می‌شه 🖥️
  - ✅ یه دیتابیس تستی که با اطلاعات فیک پر شده 🗄️
  - ✅ یه API پرداخت تستی که پول واقعی، از حساب کم نکنه! 💸🚫
- **Test Harness**: توی تست خودکار به مجموعه‌ای از ابزارها، اسکریپت‌ها و فریم‌ورک‌هایی گفته می‌شه که برای اجرای تست‌ها، بررسی خروجی‌ها و گزارش نتایج استفاده می‌شن. این ابزارها کمک می‌کنن که تست‌های پیچیده رو بدون نیاز به اجرای دستی انجام بدیم و حتی سناریوهای خاص مثل ورودی‌های مختلف، تست‌های استرس، و تعامل بین ماژول‌ها رو شبیه‌سازی کنیم.
- **Test Plan**: استراتژی، اهداف، منابع، زمان‌بندی و سناریوهای تست رو مشخص می‌کنه. این سند تعیین می‌کنه که چه چیزی تست بشه، چطور تست بشه، چه کسی تست‌ها رو انجام بده و چه ابزارهایی استفاده بشن.
- **Test Script**: به یه کد یا اسکریپت نوشته‌شده برای اجرای خودکار یه تست خاص گفته می‌شه. این اسکریپت‌ها معمولاً با ابزارهایی مثل Selenium، JUnit، xUnit یا Cypress نوشته می‌شن و به جای اجرای دستی، تست‌ها رو خودکار اجرا و بررسی می‌کنن.
- **Test Suite**: چندین تست‌کیس یا تست اسکریپت که برای بررسی یه بخش خاص از نرم‌افزار یا کل سیستم کنار هم قرار داده شدن. می‌تونن دستی (Manual) یا خودکار (Automated) باشن و معمولاً با هدمیکه اجرا می‌شن تا یه تست کامل روی یه ویژگی خاص انجام بشه.

P

- **Pair Testing**: وقتی دو نفر با همدیگه (معمولاً یه تستر و یه توسعه‌دهنده) روی یه تست کار کنن تا سریع‌تر باگ‌ها رو پیدا کنن و مشکلات رو درجا حل کنن. این روش باعث می‌شه هم تست مؤثرتر باشه، هم درک بهتری از سیستم به دست بیاد.

  📌 مثلا یه QA و یه برنامه‌نویس کنار هم می‌شینن، تست‌های دستی یا خودکار رو اجرا می‌کنن و اگه باگی پیدا شد، همون لحظه برنامه‌نویس کد رو چک می‌کنه و مشکل رو برطرف می‌کنه! 🕵️‍♂️🤝🐞
- **Penetration Testing یا PenTest**: تست نفوذ؛ وقتی یه حمله‌ی واقعی به سیستم برای پیدا کردن نقاط ضعف امنیتی رو شبیه‌سازی می‌کنیم، اون هم قبل از اینکه هکرها پیداش کنن! این تست معمولاً توسط متخصص‌های امنیتی (Ethical Hackers) انجام می‌شه تا ببینن آیا می‌شه به داده‌های حساس دسترسی پیدا کرد یا نه.
- **Performance Testing**: بررسی نرم‌افزار از نظر سرعت، پایداری و مقیاس‌پذیری، وقتی که تحت فشار قرار می‌گیره. این تست معمولاً روی سرعت پاسخ‌دهی، مصرف منابع، و رفتار سیستم در شرایط سنگین تمرکز داره.
- **Production Testing**: تست کردن نرم‌افزار مستقیماً روی محیط واقعی (Production) بدون اینکه روی کاربران واقعی تأثیر منفی بذاره. این کار معمولاً برای بررسی عملکرد، ثبات، و مشکلات احتمالی بعد از انتشار انجام می‌شه.

  📌 مثلا یه Feature Flag برای یه قابلیت جدید فعال می‌کنیم فقط برای ۵٪ از کاربران، تست می‌کنیم که درست کار می‌کنه، بعد کم‌کم برای بقیه هم فعالش می‌کنیم! 🔥🚀✅

Q

- **Quality Assurance (QA)**: مجموعه‌ای از فرآیندها و استانداردها که کمک می‌کنن یه نرم‌افزار با کمترین باگ، کیفیت بالا و مطابق با نیازهای کاربر تولید بشه. QA فقط تست کردن نیست! بلکه شامل برنامه‌ریزی، پیشگیری از باگ، بهبود فرآیند توسعه و تضمین کیفیت کل هم می‌شه.

  📌 مثلا توی یه تیم نرم‌افزاری، QA Engineer فقط تست نمی‌کنه، بلکه کدها رو بررسی می‌کنه، استانداردهای تست رو تعریف می‌کنه، فرآیند CI/CD رو بهینه می‌کنه و از اولِ کار، تیم رو به سمت یه محصول باکیفیت هدایت می‌کنه!
- **Quality Control (QC)**: تست کردن و بررسی نرم‌افزار بعد از توسعه، برای اطمینان از اینکه محصول واقعاً با نیازمندی‌ها و استانداردهای کیفی مطابقت داره. برخلاف QA که روی فرآیند تمرکز داره، QC بیشتر روی پیدا کردن مشکلات و ایرادهای محصول نهایی تمرکز می‌کنه.

  📌 مثلا وقتی به اپلیکیشن جدید توسعه داده می‌شه، QC تیم تست می‌کنه که آیا همه‌ی قابلیت‌ها درست کار می‌کنن، رابط کاربری مشکل نداره، و خروجی‌ها مطابق انتظار هستن یا نه!

R

- **Regression Testing**: بعد از هر تغییر یا آپدیت، دوباره تست کنیم که آیا قابلیت‌های قدیمی هنوز درست کار می‌کنن یا نه. این تست کمک می‌کنه مطمئن بشیم که یه فیکس یا قابلیت جدید، چیزی رو خراب نکرده!

  📌 مثلا یه باگ رو توی سیستم پرداخت فیکس کردیم، ولی Regression Testing اجرا می‌کنیم تا ببینیم نکنه این تغییر باعث شده تخفیف‌های خرید دیگه اعمال نشن! 🔄✅
- **Reliability Testing**: تست میزان پایداری نرم‌افزار، مخصوصاً توی یه بازه زمانی طولانی یا تحت شرایط خاص. این تست کمک می‌کنه بفهمیم سیستم چقدر می‌تونه بدون کرش، خطا یا افت کیفیت اجرا بشه.
- **Requirements Testing**: بررسی اینکه نرم‌افزار واقعاً مطابق با نیازمندی‌های تعریف‌شده پیاده‌سازی شده یا نه. این تست کمک می‌کنه مطمئن بشیم که همه‌ی فیچرهای مورد انتظار، درست کار می‌کنن و چیزی از قلم نیفتاده.
- **Risk-Based Testing**: تمرکز تست‌ها روی بخش‌هایی از نرم‌افزار که بیشترین ریسک خرابی یا تأثیر منفی روی کاربر رو دارن. به جای تست همه‌چیز با یک میزان دقت، تست‌های مهم‌تر رو اولویت‌بندی می‌کنیم.

  📌 مثلا توی یه اپلیکیشن بانکی، اگه یه باگ توی صفحه تنظیمات پروفایل باشه، مشکلی نیست ولی اگه توی پرداخت آنلاین باشه، فاجعه می‌شه! پس توی Risk-Based Testing، بیشتر روی بخش‌های حساس مثل پرداخت و امنیت تمرکز می‌کنیم. ⚠️🔍

I

- **Impact Analysis**: یعنی بررسی کنیم که یه تغییر یا آپدیت توی نرم‌افزار، چه بخش‌های دیگه‌ای رو ممکنه تحت تأثیر قرار بده و خراب کنه! این کار کمک می‌کنه قبل از هر تغییری بدونیم کجاها رو باید دوباره تست کنیم.
- **Integration Testing**: تستی که درست کار کردن بخش‌های مختلف نرم‌افزار رو وقتی کنار هم قرار می‌گیرن بررسی می‌کنه. این تست میاد تعامل بین ماژول‌ها، سرویس‌ها، یا کامپونت‌های مختلف رو بررسی می‌کنه تا مطمئن بشه داده‌ها و پردازش‌ها به درستی رد و بدل می‌شن.
- **Interface Testing**: یعنی بررسی کنیم که چطوری دو بخش مختلف از نرم‌افزار (مثلاً دو سرویس، یه API و یه فرانت‌اند، یا یه ماژول دیتابیس و یک‌اند) با هم ارتباط برقرار می‌کنن. این تست تضمین می‌کنه که ورودی و خروجی بین این بخش‌ها درست منتقل می‌شه.

K

- **Keyword-Driven Testing**: تست‌هایی که بر اساس کلمات کلیدی (Keywords) و قراردادها اجرا می‌شن، بدون نیاز به کدنویسی مستقیم. این روش برای تست‌های خودکار خوبه، چون تسترها می‌تونن بدون مهارت برنامه‌نویسی، تست‌های پیچیده بسازن و اجرا کنن.

  📌 مثلا توی یه ابزار تست، به جای نوشتن ("click("LoginButton"، می‌گیم "Click Login" و ابزار خودش می‌فهمه که باید روی دکمه لاگین کلیک کنه! این روش توی ابزارهایی مثل Selenium + Robot Framework خیلی رایجه. 🤖✅

L

- **Load Testing**: بررسی می‌کنیم که یه سیستم تحت فشار معمول (یا کمی بیشتر از حد عادی) چطور کار می‌کنه. این تست کمک می‌کنه بفهمیم سرور، دیتابیس، یا اپلیکیشن توی شرایط پرترافیک، هنوز جواب می‌ده یا از کار می‌افته!

  📌 مثلا یه سایت فروش بلیت داریم و قراره توی یه ساعت، ۱هزار نفر همزمان بیان بلیت بخرن. توی Load Testing این شرایط رو شبیه‌سازی می‌کنیم تا ببینیم سایت چند تا درخواست رو تحمل می‌کنه قبل از اینکه کند بشه یا کرش کنه! 🐌🔥
- **Localization Testing**: تست اینکه یه نرم‌افزار برای یه منطقه یا زبان خاص درست بومی‌سازی شده یا نه. آیا همه‌چیز، از ترجمه‌ها گرفته تا تاریخ، ارز، و جهت نمایش، مطابق استاندارد اون منطقه هست.

  📌 مثلا اگه یه اپلیکیشن رو برای ایران آماده می‌کنیم، توی Localization Testing چک می‌کنیم که تقویم شمسی درسته، راست‌چین بودن صفحه‌ها مشکلی نداره، قیمت‌ها به ریال نمایش داده می‌شن، و ترجمه‌ها هم طبیعی هستن!

M

- **Maintainability Testing**: اینکه آیا یه نرم‌افزار رو می‌شه راحت تغییر داد، آپدیت کرد، و دیباگ کرد یا نه؟! این تست روی ساختار کد، مستندات، و معماری تمرکز داره تا ببینیم چقدر نگهداری و توسعه‌ی بعدی سیستم آسونه.

  📌 مثلا اگه یه برنامه داریم که کدش تمیز، ماژولار، و خوب کامنت‌گذاری شده باشه، توی Maintainability Testing نمره‌ی خوبی می‌گیره! ولی اگه همه‌چیز درهم‌برهم و بدون مستندات باشه، هر تغییر کوچیک، یه دردسر بزرگ می‌شه! 🛠️🔍
- **Manual Testing**: تست نرم‌افزار بدون استفاده از ابزارهای خودکار، خودمون مثل یه کاربر واقعی بخش‌های مختلف رو بررسی می‌کنیم تا ببینیم باگ داره یا نه. این روش مقیاس‌پذیری و تکرارپذیری کمی داره (نداره 😅).
- **Mocking**: ایجاد نسخه‌ی الکی از وابستگی‌های کد که فقط نقش بازی می‌کنن، ولی کار واقعی انجام نمی‌دن! این کار توی تست‌های واحد (Unit Test) خیلی رایجه، چون نمی‌خوایم تست‌هامون به چیزای بیرونی مثل دیتابیس، API، یا سرویس‌های دیگه وابسته باشه. اگر اینکار رو بد انجام بدیم، تست خطاهای دنیای واقعی رو نشون نمی‌ده!

  📌 مثلا اگه یه متد ()SendEmail داریم، توی تست، یه Mock ارزش می‌سازیم که فقط چک کنه "ایمیل صدا زده شد یا نه"، ولی واقعاً ایمیل نفرسته!
- **Model-Based Testing (MBT)**: طراحی و اجرای تست‌ها بر اساس یه مدل ریاضی یا دیاگرام که رفتار سیستم رو توصیف می‌کنه، این مدل می‌تونه یه نمودار حالت (State Machine)، فلوچارت، یا حتی یه مدل داده‌ای باشه که سناریوهای تست از روش تولید می‌شن.

N

- **Negative Testing**: وقتی عمداً ورودی‌های نامعتبر، غیرمنتظره یا عجیب رو به سیستم بدیم تا ببینیم چجوری واکنش نشون می‌ده. هدف اینه که مطمئن بشیم نرم‌افزار توی شرایط غیرعادی هم کرش نمی‌کنه و خطای درست می‌ده.

  📌 مثلا توی یه فرم ثبت‌نام، به جای اسم واقعی، ورودی‌هایی مثل عدد، !@#$%^&*، یا یه متن خیلی طولانی رو تست می‌کنیم تا ببینیم سیستم درست پیام خطا می‌ده یا می‌پاشه! 💥🔍
- **Non-Functional Testing**: تست کردن بخش‌هایی از نرم‌افزار که به چیستی سیستم مربوط نیست، بلکه به چجوری کار کردنش ربط داره، مثل عملکرد، امنیت، تجربه کاربری و پایداری. این تست‌ها نشون می‌دن که سیستم فقط "درست کار می‌کنه" یا "به خوبی کار می‌کنه"!

  📌 مثلا توی یه سایت خرید آنلاین، Functional Testing چک می‌کنه که سفارش ثبت بشه، ولی Non-Functional Testing بررسی می‌کنه که چقدر سریع سفارش ثبت می‌شه، با چند هزار کاربر همزمان کار می‌کنه، و آیا امن هست یا نه! 🌐✅

A

- **Acceptance Testing**: تستی که بررسی می‌کنه سیستم نیازهای کسب‌وکار را برآورده می‌کنه و برای کاربران نهایی قابل قبوله.
  (عموما مرحله قبل از ریلیز نرم‌افزار انجام می‌شه)
- **Accessibility Testing**: تستی برای اطمینان از اینکه نرم‌افزار برای همه‌ی کاربران، از جمله افراد توان‌یاب (مثل نابینایی، کم‌بینایی، ناشنوایی، یا مشکلات حرکتی)، قابل استفاده باشه.
  (مثلا تطابق با استانداردهای دسترس‌پذیری (مثل WCAG))
- **Ad Hoc Testing**: تست نرم‌افزار بدون هیچ برنامه‌ریزی یا سناریوی از قبل نوشته‌شده! این تست کاملاً غیررسمی و بداهه انجام می‌شه تا با دستکاری و کنجکاوی، باگ‌های غیرمنتظره رو پیدا کنیم.
- **Alpha Testing**: تستی که توسط تیم‌های داخلی قبل از انتشار رسمی انجام می‌شه.
- **API Testing**: تست مستقیم API (خودکار یا دستی)، ورودی‌های صحیح و غلط رو درست پردازش می‌کنه، و خروجی‌هایی که انتظار داریم رو می‌ده یا نه!
- **Assertion**: نتیجه/رفتاری که انتظار می‌ره از کد، صدق کنه یا نه.
  (مثلا: (Assert.equals(4, calculator.add(2, 2))
- **Automated Testing**: استفاده از اسکریپت یا ابزارهای نرم‌افزاری برای اجرای خودکار تست‌ها. و طبیعتا باعث می‌شه تست‌ها سریع‌تر، دقیق‌تر و بدون خطای انسانی انجام بشن.

  📌 مثلا توی یه فروشگاه اینترنتی، به جای اینکه هربار دستی لاگین کنیم و خرید انجام بدیم، با Selenium یا playwright یه تست خودکار می‌نویسیم که خودش فرم لاگین رو پر کنه، محصول انتخاب کنه، پرداخت انجام بده و نتیجه رو بررسی کنه! 🤖✅

B

- **Baseline**: مقدار یا وضعیت استاندارد اولیه که بعداً تست‌های بعدی با اون مقایسه بشن. این کار کمک می‌کنه ببینیم آیا یه تغییر باعث بهبود شده یا نه!
- **Beta Testing**: تست کردن نرم‌افزار توسط کاربران واقعی (انتخاب‌شده یا داوطلب) در دنیای واقعی، اما قبل از انتشار رسمی!
- **Black Box Testing**: تستی که بدون آگاهی از ساختار داخلی، کد یا اطلاعات اضافه انجام می‌شه، فقط بر اساس ورودی و خروجی! توی این روش، تستر مثل یه کاربر عادی کار می‌کنه و بررسی می‌کنه که سیستم طبق انتظار جواب می‌ده یا نه.
  (در مورد تست امنیتی خیلی پرکاربرده)
- **Bottom-Up Testing**: تست رو اول از ماژول‌های پایین‌تر (مثلاً فانکشن‌ها یا کلاس‌های مستقل) شروع کنیم و بعد کم‌کم تست‌ها رو به سمت سطوح بالاتر (ماژول‌های پیچیده‌تر و کل سیستم) ببریم. برای تست ماژول‌هایی که هنوز آماده نیستن، معمولاً از Driver استفاده می‌کنیم تا جای اونا رو پر کنه.

  📌 مثلا توی یه اپ بانکی، اول ماژول محاسبه‌ی سود وام رو تست می‌کنیم، بعد که مطمئن شدیم درسته، میایم ماژول ثبت وام در دیتابیس رو تست می‌کنیم و بعد کل فرآیند درخواست وام رو یکپارچه بررسی می‌کنیم! 🏦✅
- **Boundary Value Analysis**: تست نرم‌افزار با مقادیری که در لبه‌های دامنه‌های ورودی‌ها باشه.

  📌 مثلا آزمایش ورودی با مقادیر 0، 1، 99، 100 و 101 برای فیلدی که باید بین 1 تا 100 باشد
- **Build Verification Test (BVT)**: شامل یه سری تست سریع و حیاتی روی قابلیت‌های اصلی سیستمه، اگه این تست‌ها رد بشن، دیگه نیازی نیست وقت تیم روی تست‌های عمیق‌تر هدر بره، چون بیلد از پایه مشکل داره!

C

- **Code Coverage**: معیاری که نشون می‌ده چند درصد از کد یا تست‌ها پوشش داده شده. هرچی این درصد بالاتر باشه، یعنی تست‌های بیشتری روی بخش‌های مختلف کد اجرا شدن. اما فقط درصد بالا مهم نیست! بعضی وقتا تستا اجرا می‌شن ولی منطق درست چک نمی‌شه، پس کیفیت تست‌ها هم مهمه.

  📌 مثلا پوشش 80٪ یعنی 80٪ از کل کد در طی تست‌ها اجرا شده، مثلا متدها یا شرط‌هایی که توی تست پوشش داده نشده از این عدد کم می‌کنه
- **Compatibility Testing**: یکی از تست‌های مهم غیرعملکردی (Non-Functional Testing) هست که بررسی می‌کنه نرم‌افزار توی پلتفرم‌های مختلف، مرورگرهای مختلف، دستگاه‌های مختلف و تنظیمات متنوع درست کار می‌کنه یا نه. این تست تضمین می‌کنه که کاربرها با هر سیستمی که دارن، تجربه‌ی یکسان و بدون مشکل داشته باشن.
- **Concurrency Testing**: یکی از تست‌های Performance Testing که بررسی می‌کنه وقتی چندین کاربر یا پردازش به‌صورت همزمان از سیستم استفاده می‌کنن، آیا سیستم درست کار می‌کنه یا نه.

C

- **Configuration Testing**: یکی نوع تست غیرعملکردی (Non-Functional Testing) که بررسی می‌کنه نرم‌افزار توی تنظیمات و پیکربندی‌های مختلف سیستم چطور اجرا می‌شه و آیا هم‌چیز درست کار می‌کنه یا نه. این تست تضمین می‌کنه که اپلیکیشن با ترکیب‌های مختلفی از سخت‌افزار، سیستم‌عامل، درایورها، و تنظیمات سرور هم سازگار باشه.
- **Continuous Integration (CI)**: کدهای به‌صورت مداوم توی مخزن کد ادغام می‌شن و هر تغییر به‌صورت خودکار تست و بررسی می‌شه. با CI، هر بار که یه تغییر جدید به کد اضافه می‌شه، کد بیلد می‌شه، تست‌های خودکار اجرا می‌شن و اگه مشکلی باشه، سریع مشخص می‌شه، پس مشکلات زودتر پیدا و حل می‌شن.

  📌 مثلا استفاده از Jenkins یا GitLab Runner یا GitHub Actions برای اجرای خودکار تست‌ها پس از هر تغییر در مخزن کد
- **Cross-Browser Testing**: تست برنامه‌های وب در مرورگرهای مختلف برای اطمینان از نمایش و رفتار یکسان توی مرورگرهای مورد پشتیبانی.

D

- **Data-Driven Testing**: تست‌هایی که ورودی‌هاشون از منابع داده می‌آیند.

  📌 مثلا اجرای یک آزمون ورود به سیستم با صدها ترکیب نام کاربری و رمز عبور از یک فایل CSV)
- **Decision Table Testing**: ابتدا همه‌ی ترکیب‌های ممکن از ورودی‌ها و شرایط مختلف رو بررسی کنیم و مطمئن بشیم که سیستم درست کار می‌کنه. این روش مخصوصاً برای سیستم‌هایی با چندین شرط و قانون خیلی مفیده.
- **Dependency Testing**: یکی از بخش‌های مهم Integration Testing هست که بررسی می‌کنه آیا ماژل‌های مختلف که نرم‌افزار که به هم وابسته هستن، درست و بدون مشکل با هم کار می‌کنن یا نه.
- **Destructive Testing**: تست نرم‌افزار که عمداً سیستم رو تحت شرایط غیرعادی، حاد یا مخرب قرار می‌دیم تا ببینیم چقدر مقاومه و چجوری به این شرایط واکنش نشون می‌ده.
- این تست برای بررسی پایداری (Stability) و امنیت (Security) نرم‌افزار خیلی مهمه.
- Dynamic Testing: یکی از دو نوع اصلی تست نرم‌افزار (در کنار Static Testing) هست که کد رو در حال اجرا بررسی می‌کنه تا ببینیم خروجی واقعی مطابق انتظار هست یا نه. این تست معمولاً بعد از نوشتن کد انجام می‌شه و می‌تونه هم دستی (Manual) باشه و هم خودکار (Automated).

E

- **End-to-End Testing (E2E Testing)**: تست که سناریوهای واقعی رو از اول تا آخر تست می‌کنه تا مطمئن بشیم کل سیستم با همه اجزاش درست کار می‌کنه. (عموما اجزاء مختلفی طی تست درگیر می‌شن و مثل unit test یک بخش کوچک رو شامل نمی‌شه)

  📌 مثلا تست فرآیند کامل خرید توی فروشگاه آنلاین از جستجوی محصول تا تکمیل سفارش
- **Equivalence Partitioning**: ورودی‌های ممکن برای تست رو به چند گروه (پارتیشن) تقسیم می‌کنیم تا به جای تست کردن همه‌ی مقادیر، فقط یک گروه رو بررسی کنیم. اینطوری تست‌ها کم‌تر ولی هوشمندانه‌تر می‌شن.
- **Exploratory Testing**: بدون اینکه از قبل سناریوی مشخصی بنویسیم، سیستم رو دستکاری می‌کنیم تا باگ‌ها و مشکلات مخفی رو پیدا کنیم. این تست بیشتر بر اساس تجربه و حس کنجکاوی انجام می‌شه تا تست‌های از پیش تعریف‌شده و سیستماتیک.

F

- **Functional Testing**: بررسی می‌کنیم که هر قابلیت نرم‌افزار دقیقاً همون‌جوری که باید کار کنه، بدون اینکه به جزئیات داخلی کد کاری داشته باشیم. این تست معمولا ورودی می‌گیره، خروجی رو چک می‌کنه و مقایسه می‌کنه که همون چیزی باشه که انتظار داریم. یعنی فقط قابلیت، نه کیفیت.
- **Fuzzing**: یه نوع تست اتوماتیک و بی‌رحمانه که کلی ورودی تصادفی، عجیب و غریب، یا مخرب می‌فرسته تا ببینه برنامه کرش می‌کنه یا نه! بیشتر برای تست امنیت و پایداری استفاده می‌شه.

G

- **Gray Box Testing**: ترکیبی از Black Box و White Box، یعنی تستی که هم از بیرون مثل یه کاربر معمولی سیستم رو بررسی می‌کنیم، هم «یه مقدار» اطلاعات از ساختار داخلیش داریم. این روش کمک می‌کنه تست‌ها هم واقع‌بینانه باشن، هم دقیق‌تر!

H

- **Heuristic Testing**: تست کردن سیستم بر اساس تجربه، حدس‌های منطقی، و بهترین روش‌های شناخته‌شده، بدون اینکه همه‌ی سناریوهای ممکن رو از قبل برنامه‌ریزی کنیم. این روش مخصوص وقتاییه که زمان محدوده یا مستندات کافی نداریم.