تست نرم‌افزار؛ مفاهیم، ابزارها و روش‌ها

محمدجواد غلامی
کارشناس کنترل کیفیت مرکز تحقیقات کامپیوتری علوم اسلامی
mjgholami@noornet.net

چکیده
هدف از این نوشتار، دریافت نگرش کلی از فرایند تست و مشاهده ابزارها و روش‌هایی است که با آنها می‌توانیم برای تست نرم‌افزار، برنامه‌ریزی و اقدام کنیم؛ ضمن اینکه اهمیت، مراحل و انواع تست نرم‌افزار نیز در این مقاله مورد توجه قرار گرفته و به شرح و توضیح آن پرداخته‌ایم.

از دیگر موضوعات مهم در این بحث، پاسخ‌دادن به این سؤال است که تست نرم‌افزار توسط چه کسانی و به‌وسیله چه ابزارهایی تحقق می‌یابد؟ اصولاً انجام تست نرم‌افزار توسط کارشناسان، به کمک ابزارهایی است که در مراحل مختلف کار، به کمک می‌آیند تا روند تست به‌شایستگی انجام شود؛ که از آن جمله می‌توان به تست واحد و تست رابطه کاربر اشاره نمود.

کلیدواژگان: تست، خودکارسازی تست، تست جعبه سیاه، تست جعبه سفید، تست‌های تکرارپذیر، ابزارهای تست خودکار، ابزارهای تست نرم‌افزار.

مقدمه مهندسی نرم‌افزار

مقوله مهندسی در تولید نرم‌افزار، امری حیاتی است. در مهندسی نرم‌افزار، اندیشه آدمی مهم است و چون انسان‌ها بر اساس تجربه‌های گوناگون به راه حل‌های مختلفی می‌رسند، لازم است کار به صورت گروهی باشد و اینکه به طور سیستمی به راه حل برسیم. مهندسی نرم‌افزار، خود، یک سبک است.

اصل مهندسی نرم‌افزار می‌گوید: کار باید به طور مهندسی انجام شود و وابسته به فناوری نباشد. بنابراین، با PLC وابسته به فناوری نخواهیم بود. دو عامل مهم در مهندسی نرم‌افزار، هزینه و زمان می‌باشد. چرخه تولید محصول (PLC)، عبارت است از قالب کاری و مجموعه‌ای از راهکارها برای سازماندهی و پیاده‌سازی برنامه‌های توسعه محصولات. این چرخه، تمام مراحلی را که در انجام یک عمل طی می‌شود، به صورت استاندارد مستند می‌کند. همچنین، پروسه تولید محصول، شامل تعاریف قوانین سازماندهی فعالیت‌ها و روابط بینابینی سازمانی، در واقع، اجزای کلیدی و نقاط تحویل کار بین فازهای مختلف و تست و ارزیابی را مستند می‌نماید.

مفهوم تست نرم‌افزار

تست، مجموعه‌ای از فعالیت‌هاست که می‌تواند به طور پیشرفته، برنامه‌ریزی‌شده و به طور سیستمی هدایت شود. برای این منظور، باید برای فرآیند تست نرم‌افزار، یعنی مجموعه گام‌های خاص و test-caseها که بتوانیم فنون طراحی روش‌های تست را جایگزین کنیم، الگویی تعریف شود.

دلایل اهمیت تست نرم‌افزار

به طور معمول، تست، بیشترین تلاش را نسبت به دیگر فعالیت‌های مهندسان، نیاز دارد. اگر فرایند تست بدون دقت هدایت شود، باعث هدر رفتن زمان، تلاش غیرضروری، بدتر شدن اوضاع و عدم تشخیص خطاهای پنهان می‌شود. بنابراین، برپاکردن یک راهبرد سیستمی برای تست نرم‌افزار، معقول به نظر می‌رسد.

برای انجام تست مؤثر، باید مقالات فنی اثربخش را دنبال کنید. با انجام این کار، بسیاری از خطاها قبل از شروع تست، حذف خواهند شد. تست در سطح تست واحد شروع می‌شود و در جهت خارج، به سوی یکپارچه‌سازی کل سیستم مبتنی بر رایانه، کار می‌کند. فنون تست برای دیدگاه‌های متفاوت مهندسی نرم‌افزار و در زمان‌های متفاوت، می‌توانند مناسب باشند. فرایند تست توسط توسعه‌دهنده نرم‌افزار و برای پروژه‌های بزرگ، از سوی یک گروه تست مستقل هدایت می‌شود.

برای انجام تست مؤثر، باید مقالات فنی اثربخش را دنبال کنید. با انجام این کار، بسیاری از خطاها قبل از شروع تست، حذف خواهند شد. تست در سطح تست واحد شروع می‌شود و در جهت خارج، به سوی یکپارچه‌سازی کل سیستم مبتنی بر رایانه، کار می‌کند. فنون تست برای دیدگاه‌های متفاوت مهندسی نرم‌افزار و در زمان‌های متفاوت، می‌توانند مناسب باشند

تست و اشکال‌زدایی، فعالیت‌های متفاوتی هستند؛ اما اشکال‌زدایی باید در هر فرایند تست جا داده شود.

چرا باید تست نوشت؟

وقتی برنامه شما یک بار درست کار می‌کند و بار دیگر نه، چه باید کرد؟ زمانی که برای کدی تست ننوشته‌ایم، به احتمال زیاد، این مشکل برایش پیش خواهد آمد؛ چرا؟ دلیلش ساده است. وقتی کدی را می‌نویسیم، ممکن است به همه نکات آن توجه نکنیم؛ البته این موضوع با توجه به تجربه برنامه‌نویس متفاوت است؛ ولی هنگامی که شما برای کدتان تست می‌نویسید، مجبور می‌شوید به خیلی از جزئیاتی که در حالت معمول به آن توجه نمی‌کنید، توجه داشته باشید.

تست نوشتن، باعث بهبود کیفیت کد می‌شود؛ زیرا برای تست نوشتن مجبوریم یکسری قواعد را رعایت کنیم. مدتی پیش، مجبور بودم متدی را تست کنم که در آن، یک دستور به سیستم استریسک (سیستم تلفنی تحت لینوکس) فرستاده می‌شد و من نمی‌توانستم به طور مستقیم آن را تست کنم؛ زیرا در محیط تست، استریسکی در کار نبود. بنابراین، تنها راهی که وجود داشت، یین بود که از dependency injection استفاده کنم. یک کلاس ایجاد کردم که کار ارتباط با استریسک را در آن انجام داده بودم و آن را به کلاسی که قصد تست آن را داشتم، تزریق کردم. این موضوع، باعث شد که وابستگی کد به سیستم دیگر، مدیریت شود.

تست نوشتن، باعث مشخص‌شدن کدهای مرده می‌شود. اگر مدت‌هاست که به کلاسی سر نزده‌اید و حال قصد کار بر روی آن را دارید، با تست نوشتن می‌توانید کدهایی را که هرگز استفاده نشده‌اند، پیدا کنید. تست نوشتن، باعث آشکارشدن کدهای تکراری می‌شود؛ ضمن اینکه باگ‌های پنهان را آشکار می‌کند. زمانی که شما برای کدی که از قبل نوشته شده است، تست می‌نویسید، مجبورید دقت بیشتری به خرج دهید و در نتیجه، باگ‌های آن کشف خواهد شد.

نوع تست، ممکن است با توجه به فناوری تغییر کند؛ ولی چیزی که مهم است، این است که ما تست را جزئی از کد و برنامه بدانیم.

مراحل تست نرم‌افزار

تست اوّلیه، بر یک مؤلفه واحد یا گروه کوچکی از مؤلفه‌ها تمرکز دارد و تست‌هایی را برای آشکارکردن خطاهای داده‌ای و پردازش منطقی که توسط مؤلفه‌ها کپسوله شده، اعمال می‌کند. پس از اینکه مؤلفه‌ها

تست اوّلیه، بر یک مؤلفه واحد یا گروه کوچکی از مؤلفه‌ها تمرکز دارد و تست‌هایی را برای آشکارکردن خطاهای داده‌ای و پردازش منطقی که توسط مؤلفه‌ها کپسوله شده، اعمال می‌کند. پس از اینکه مؤلفه‌ها تست شدند، باید یکپارچه شوند تا سیستم کامل را ایجاد کنند. در این مرحله، یکسری تست‌های مرتبه بالا برای آشکارکردن خطاهای مربوط به نیازمندی‌های کاربر انجام می‌شود

تست شدند، باید یکپارچه شوند تا سیستم کامل را ایجاد کنند. در این مرحله، یکسری تست‌های مرتبه بالا برای آشکارکردن خطاهای مربوط به نیازمندی‌های کاربر انجام می‌شود. بعد از بروز خطاها، باید با بستفاده از فرایند تشخیص داده‌شده و (De-bugging) اشکال‌زدایی، تصحیح شوند.

سازماندهی تست نرم‌افزار

توسعه‌دهنده نرم‌افزار، موظف است که واحدهای منفرد برنامه (مؤلفه‌ها) را تست کند تا مطمئن شود هریک رفتاری را بروز می‌دهند که برای آن طراحی شده‌اند. در بسیاری حالات، توسعه‌دهنده، تست یکپارچه‌سازی نیز انجام می‌دهد؛ یک مرحله تست که منجر به ساخت و تست معماری نرم‌افزار کامل می‌شود.

پس از تست یکپارچه‌سازی، یک گروه تست مستقل (ITG)(۱) نیز نقش ایفا می‌کنند. نقش گروه تست مستقل، این است که مشکلات مربوط به تست سازنده را که به بخش‌هایی اجازه ساخته‌شدن می‌دهد، حذف می‌کند. ITG پس از همه این مراحل، گروه متعهد هستند که خطاها را پیدا کنند. ITG به طور تنگاتنگ با هم کار می‌کنند تا تضمین نمایند که تست‌های توسعه‌دهنده به‌درستی انجام شده است. هنگامی که تست انجام شد، توسعه‌دهنده باید حضور داشته باشد تا خطاهای آشکارشده را تصحیح نماید.

مراحل تست

تست واحد(۲): در این نوع تست، قسمت‌های کوچک برنامه را تست می‌کنیم؛ مثلاً توابع یا کلاس‌ها در زبان‌های object oriented. در این تست، با کل برنامه کاری نداریم و هدف ما، اطمینان از کارکرد قسمت‌های کوچک برنامه است. تست واحد، در سطح کد است و یک ابزار مناسب برای آن JUnit است. این ابزار، تست خودکار انجام می‌دهد که در مرکز تحقیقات کامپیوتری علوم اسلامی (نور) زبان‌هایی مثل: C ++، C # و Java وجود دارد که برنامه‌نویسان تست‌های واحد را در برنامه‌های خود می‌نویسند و در آینده، همه کدهای موجود در مرکز تست‌های واحد خواهند داشت.

تست جامعیت(۳): این تست برای این است که مطمئن شویم کامپونت‌های مختلف برنامه با هم درست کار می‌کنند. تست جامعیت نیز در معاونت فنی نور توسط برنامه‌نویسان، و در معاونت پژوهش نور توسط بخش‌های بررسی نرم‌افزارهای موبایل، دسکتاپ و وب در حال کنترل است.

- Acceptance test /Functional test: برای این است که مطمئن شویم آیا نرم‌افزار کامل هست و به‌درستی کار می‌کند؟ این نوع تست، یک تست سطح بالا محسوب می‌شود و ابزار این تست، Fitnesse می‌باشد و روند آن توسط همکارمان در بخش بررسی نرم‌افزار در مرکز نور انجام می‌شود و قسمتی از آن نیز توسط کاربران نهایی انجام خواهد شد.

ـ Performance test: در این نوع تست، کارایی کامپونت‌های برنامه مورد بررسی قرار می‌گیرد. تست کارایی در معاونت فنی نور توسط برنامه‌نویسان، و در معاونت پژوهش توسط قسمت بررسی نرم‌افزار انجام می‌شود.

ـ System test: در این نوع تست، تمام سیستم تست می‌شود و حتی خود نرم‌افزار، سخت‌افزار و ارتباطات بین کامپونت‌ها مورد توجه قرار می‌گیرد. خود تست سیستم، شامل: تست گرافیک رابط کاربری(۴)، Smoke test، تست‌هایی از قبیل تست کارایی(۵) و تست نصب(۶) می‌باشد و همه تست‌های سیستمی، توسط کارشناسان نور در بخش بررسی نرم‌افزار مرکز انجام می‌شود.

انواع تست

انواع تست، عبارت‌اند از:

ـ تست نصب: این نوع تست به ما این اطمینان خاطر را می‌دهد که برنامه به‌درستی در سیستم‌های مشتریان درست نصب خواهد شد. در مرکز توسط بررسی نرم‌افزار انجام می‌شود.

ـ تست سازگاری(۷): در این نوع تست، باید بررسی کنیم که برنامه با محیطی که قرار است در آن اجرا شود و همچنین برنامه‌های دیگری که کنار آن وجود دارند، سازگار باشد؛ مثلاً ممکن است که یک برنامه‌نویس برنامه‌اش را برای یک نسخه خاص سیستم‌عامل بنویسد. بنابراین، به دلیل نداشتن backward compatibility، این نرم‌افزار برای همه کاربران قابل استفاده نیست. به طور کلی، تست سازگاری، به همت برنامه‌نویسان و همکارانمان در بخش بررسی نرم‌افزار انجام می‌شود.

-تست smoke و **sanity:** تست san-ity به ما می‌گوید که آیا منطقی است که به انجام تست ادامه بدهیم یا خیر؛ یعنی قبل از اینکه تستی را شروع کنیم، ممکن است با این تست بتوانیم بفهمیم که برنامه از نظر منطقی درست پیاده‌سازی نشده. بنابراین، آن را به توسعه‌دهندگان ارجاع می‌دهیم، به جا آنکه بیشتر بر روی تست وقت بگذاریم.

تست smoke: در این تست، شامل حداقل تلاش برای اجرای نرم‌افزار است. این تست از این نظر مفید است که ما متوجه شویم که آیا هیچ مشکل حداقلی برای اجرای نرم‌افزار داریم یا نه؛ یعنی بعد از این تست ما می‌فهمیم که این نرم‌افزار، حداقل، کار می‌کند. در واقع، می‌توان تست verifi-cation هم نام‌گذاری شود.

توسعه‌دهنده نرم‌افزار، موظف است که واحدهای منفرد برنامه (مؤلفه‌ها) را تست کند تا مطمئن شود هریک رفتاری را بروز می‌دهند که برای آن طراحی شده‌اند. در بسیاری حالات، توسعه‌دهنده، تست یکپارچه‌سازی نیز انجام می‌دهد؛ یک مرحله تست که منجر به ساخت و تست معماری نرم‌افزار کامل می‌شود

ـ تست رگرسیون(۸): تست فوق، روی یافتن باگ بعد از یک تغییر اساسی در کد پروژه تمرکز دارد؛ به‌عنوان مثال، وقتی یک فیچر به پروژه اضافه می‌گردد، با این تست، باگ‌های احتمالی را مشاهده می‌کنیم. جایی که یک کد عمده یا یک کتابخانه به پروژه افزوده می‌شود، معمولاً باگ‌های زیادی مشاهده می‌شود. شاید به دلیل conflict با کد قبلی یا هر دلیل دیگری، در این قسمت باید این تست مربوطه را انجام داد. معمولاً قدم اوّل تست regression، این است که تست کیس‌های پیشین را دوباره تکرار کنیم و ببینیم که این تغییر کد، موجب ظهور باگ‌های قبلی نشده باشد. عمق تست، به ریسک فیچر اضافه‌شده بستگی دارد. این نوع تست، در شرکت‌های تجاری بیشتر مورد توجه قرار می‌گیرد.

ـ تست پذیرش(۹): این نوع تست، می‌تواند دو مفهوم داشته باشد؛ اوّلی همان تست smoke که مطمئن می‌شویم اگر برنامه با حداقل منابع کار می‌کند. بنابراین، احتمالاً در بیشتر شرایط نیز کار می‌کند. دومین مفهوم، مربوط به سمت مشتری می‌شود که مثلاً آنها هم یک آزمایشگاه داشته باشند و روی سخت‌افزار خودشان برنامه را تست کنند که به این تست، user acceptance test یا UAT نیز گفته می‌شود. این نوع تست، می‌تواند در بین فازهای پروژه هم معنا پیدا کند؛ به این شکل که در صورت موفقیت در این تست، برنامه به فاز بعدی می‌رود.

ـ تست آلفا(۱۰): تست آلفا، نوعی شبیه‌سازی از بازار واقعی است که مثلاً به دست یک تیم تست جداگانه سپرده شود یا به کاربران احتمالی برنامه داده شود.

ـ تست بتا(۱۱): این تست، بعد از تست آلفاست که در واقع، یک تست UAT خارجی هم محسوب می‌شود. این نسخه از نرم‌افزار با نام نسخه بتا شناسایی می‌شود که به یک تیم تست خارج از تیم برنامه‌نویسی ارسال می‌شود که به آنها beta testers

برخی شرکت‌ها برای تست نرم‌افزار، برنامه‌هایی را ارائه داده‌اند که می‌توان از جمله به: رونرکس، تست کامپلیت و تستینگ انی‌ور اشاره کرد. این سه نرم‌افزار، روی محصولات وب، موبایل و دسکتاپ قابل استفاده هستند. در این نرم‌افزارها، انجام تست خودکار و دستی امکان‌پذیر است

می‌گویند. برنامه به تعداد کمی از کاربران داده می‌شود؛ تا زمانی که مطمئن شوند باگ‌های چندانی ندارد.

ـ تست Functional و Non-functional: تست functional برای ما کارکرد (function) خاصی را تست می‌کند. در واقع، سناریوهای پیاده‌سازی‌شده را تست می‌نماید؛ مثلاً برنامه قرار است بتواند یک عکس را نمایش بدهد. این تست، مسئله مزبور را بررسی می‌کند که آیا برنامه می‌تواند عکس نمایش دهد یا خیر؟ در مقابل، تست Non-functional مربوط به آن قسمت برنامه است که به کاربر ارتباط مستقیم ندارد؛ مثل کارایی سیستم یا قابلیت گسترش برنامه، رفتار برنامه در یک شرایط خاص و یا امنیت برنامه. این نوع تست، نقطه شکست برنامه را می‌یابد. تست Non-functional در واقع، کیفیت برنامه را آزمایش می‌کند.

ـ تست مخرّب(۱۲): این نوع تست تلاش می‌کند که برنامه یا یک زیربرنامه موفق به عملیات نشود. این نوع تست بررسی می‌کند که برنامه با دادن ورودی‌های ناجور یا غیرقابل پیش‌بینی و مختلف چگونه رفتار می‌کند و نتیجه این تست موجب می‌شود که روتین‌هایی برای مدیریت error و اعتبارسنجی ورودی نوشته شودکه در نتیجه، موجب قدرتمند شدن برنامه می‌شود.

ـ تست کاربری(۱۳): این تست، بررسی می‌کند که آیا کارکردن با رابط کاربری برنامه راحت است، یا نه.

ـ تست امنیت(۱٤): در واقع، تست متدهای مختلف امنیت در برنامه است و تستی مهم به شمار می‌رود؛ به‌خصوص در برنامه‌هایی که حریم خصوصی افراد در آن تعریف می‌شود و باید با نفوذ هکرها مقابله کرد.

ـ تست توسعه(۱۵): این نوع تست، راهبردهای‌های مقابله با باگ را مطرح می‌کند که با پیروی از آنها، آسیب‌های توسعه برنامه، هزینه و زمان آن به حداقل می‌رسد.

ـ تست A/B: در واقع، مقایسه دو خروجی است. پروسه تست، این‌گونه است که وقتی یک متغیر تغییر می‌کند، تست را اجرا می‌کنیم و با دوباره تغییر دادن آن، تست را تکرار نموده، با مقایسه خروجی‌ها، متوجه درستی عملکرد برنامه می‌شویم.

ـ تست Concurrent: بر کارایی برنامه در شرایطی که ما تست را به صورت مداوم انجام می‌دهیم، تمرکز دارد. این نوع تست، در مقابل تست فشار یا استرس است که در مدت‌زمان کوتاه فشار بسیاری به برنامه می‌آوریم.

ـ تست conformance یا typing: این نوع تست، بررسی می‌کند که برنامه بر اساس استانداردهای تعریف‌شده کار می‌کند یا نه؛ مثلاً در تست کامپایلرها، این تست بدین هدف انجام می‌شود که بدانیم آیا کامپایلر مورد نظر بر اساس استانداردهای لازم آن زبان خاص، درست کار می‌کند؟

ـ تست ad-hoc testing: به تست‌های فی‌البداهه‌ای گفته می‌شود که بدون هیچ برنامه یا مستنداتی اجرا می‌شوند و تنها یک بار صورت می‌گیرد؛ مگر اینکه یک خطا و یا به اصطلاح تست نرم‌افزاری، یک defect وجود داشته باشد که برطرف نشده باشد. در این نوع تست، test case تولید نمی‌شود و تنها نکته قدرت آن، در سرعت پیدا کردن defectها می‌باشد و بنا به ابتکار عمل و تشخیص tester، مورد تست صورت می‌گیرد. این فرایند، در مرکز توسط بخش بررسی نرم‌افزار انجام می‌شود.

رویکردهای تست

رویکردهای مربوط به تست عبارت‌اند از:

۱. تست ایستا(۱٦): نوعی تست است که در آن، از نرم‌افزار استفاده نمی‌شود. در این تست، وارد جزییات نمی‌شویم و مثلاً در آن منطق برنامه، الگوریتم و داکیومنت‌های آن بررسی می‌شوند؛ در این نوع تست، کد را می‌خوانیم و به دنبال اشکال‌ها می‌گردیم که معمولاً خود توسعه‌دهنده برنامه، این کارها را انجام می‌دهد.

۲. تست پویا(۱٦): رفتار نرم‌افزار را به ورودی‌هایی که به مرور زمان تغییر می‌کنند، نشان می‌دهد؛ یعنی تستی که در آن برنامه اجرا می‌شود و با ابزاری دیگر به آن ورودی‌های مختلف داده می‌شود و رفتار یا خروجی آن را می‌بینند و تحلیل می‌کند.

۳. تست جعبه سفید(۱۸): در این تست،

کارکرد ساختارهای داخلی برنامه بررسی می‌شود و اینکه منطقاً به کد احتیاج هست تا مثلاً متدهای مختلف بررسی شوند. فنونی که در این نوع تست استفاده می‌شود، به این شرح است:

ـ **تست API:** در تست جعبه سفید، هر API را به صورت جدا گانه تست می‌کنیم.

ـ **بررسی سطربه‌سطر کد(۱۹):** یعنی بررسی خطوط کد، به‌صورت خطبه‌خط و جزءبه‌جزء؛ به صورتی‌که خطوط کد و مسیرهای مستقل داخل یک پیمانه حداقل یک بار اجرا و تست شوند.

ـ **متدهای Fault injection:** طی این متد، یک کد خطا در کد تزریق می‌کنیم تا خطایی رخ دهد و بتوانیم موردی را تست کنیم.

ـ **متدهای mutaion test:** برای تست نرم‌افزار، خودمان یک خطا را در کد ایجاد می‌نماییم.

٤. تست جعبه سیاه(۲۰): در این نوع تست، به اجزای داخلی نرم‌افزار کاری نداریم و به آن به‌عنوان یک جعبه سیاه نگریسته، رفتار برنامه را با دادن ورودی بررسی می‌کنیم.

۵. تست جعبه خاکستری(۲۱): در این نوع تست، در مورد ساختار داخلی نرم‌افزار و الگوریتم آن اطلاعاتی داریم و از آن برای طراحی تست استفاده می‌کنیم؛ درحالی‌که در تست‌هایی که طراحی می‌کنیم، هیچ دسترسی به ساختار داخلی نرم‌افزار نداریم و مانند یک جعبه سیاه به آن می‌نگریم.

چه کسانی تست نرم‌افزار را انجام می‌دهند؟

متخصصان و یا ابزارهایی که روند تست نرم‌افزار را انجام می‌دهند، عبارت‌اند از:

ـ **مدیر پروژه:** شاید بیشتر مدیر پروژه تست‌هایی را برای تأیید یک فاز پروژه و رفتن به فازهای بعدی، انجام می‌دهد. تست مدیر پروژه، بعد از بررسی‌های دقیق کارشناسان تست و تست برنامه‌نویسان است.

ـ **مهندسان نرم‌افزار:** مهندسان تست، برای هر قسمت از برنامه، کد تست می‌نویسند که از این نوع تست‌ها، به‌عنوان تست‌های واحد تعبیر می‌شود. یا هنگامی که چند واحد برنامه را کنار هم قرار می‌دهند، برای تست اینکه متوجه بشوند چند واحدِ یکپارچه‌شده در کنار همدیگر درست کار می‌کنند یا نه، بررسی‌هایی انجام می‌دهند و تست‌هایی تعریف می‌نمایند که به این نوع تست، تست یکپارچگی گفته می‌شود.

ـ **متخصصان تست:** افرادی هستند که در تمام مراحل تولید برنامه، روش‌هایی برای تست تعریف می‌کنند و سعی دارند این تست‌ها را انجام دهند؛ مثلاً در قسمت کد، از

اینکه برنامه‌نویس یک سیستم، نرم‌افزار را تست کند، لازم و ضروری است؛ ولی به هیچ عنوان کافی نیست. این تصمیم که محصول تولیدی، توسط تیم جداگانه‌ای تست شود، شاید در مرحله اوّل خوب به نظر نیاید، ولی در واقع، بسیار مفید است

برنامه‌نویس می‌خواهند که تست‌های واحد را انجام دهند و اینکه چه جاهایی را بیشتر تست نمایند. بعد از ورژنیگ، تست رگرسیون را انجام می‌دهند؛ یعنی اینکه چه جاهایی از سیستم تغییر کرده است. برای این منظور، تست‌های پیشین را یک بار دیگر انجام می‌دهند. وقتی رابط کاربری آماده شد، برای کاربردپذیری و اینکه اجزای واسط کاربری درست کار کنند، اقدام به انجام تست‌های دستی می‌نمایند.

در رابط کاربری، برخی تست‌ها را به صورت ماشینی انجام می‌دهند؛ همان‌طورکه در کد برنامه، برخی تست‌ها را به صورت ماشینی انجام دادند. برای انجام تست و آلفا و بتا، روش‌هایی را اتخاذ می‌کنند که کمترین هزینه را داشته باشد و در کمترین زمان ممکن سامان یابد.

انواع برنامه‌های تست نرم‌افزار

برخی شرکت‌ها برای تست نرم‌افزار

برنامه‌هایی را ارائه داده‌اند که می‌توان از جمله به: رونرکس، تست کامپلیت و تستینگ انی‌ور اشاره کرد. این سه نرم‌افزار، روی محصولات وب، موبایل و دسکتاپ قابل استفاده هستند. در این نرم‌افزارها، انجام تست خودکار و دستی امکان‌پذیر است. همچنین، این نرم‌افزارها قابل زمان‌بندی و برنامه‌نویسی هستند؛ مثلاً می‌توان یک تست، مانند ورود به یک وب‌گاه را انجام داد. همچنین، این تست را می‌توان به این صورت نوشت که هر هفته یک بار این تست انجام شود و اگر تست موفقیت‌آمیز باشد، یک ایمیل با این عبارت به فرد مورد نظر بزند: «تست مورد نظر درست انجام شد.» اما اگر فیلد شد، یعنی نتوانست وارد وب‌گاه شود، به کارشناس تست ایمیل بزند: «تست مورد نظر درست انجام نشد.» و همچنین، دلایلی را که نتوانسته وارد وب‌گاه مربوطه شود، به کارشناس تست گزارش دهد.

گفتنی است، این سه نرم‌افزار یک نسخه آزمایشی یک ماهه دارند که پس از پایان یک ماه، باید نسخه تجاری آن را تهیه کرد؛ ولی یک راهکار برای استفاده از این نرم‌افزارها بدون خرید آنها وجود دارد و آن این است که روی یک رایانه مجازی این نرم‌افزار را نصب کنیم و حالت اوّلیه آن را ذخیره کنیم و هر ماه به حالت اوّلیه برگردیم.

افزون بر سه نرم‌افزار یادشده، نرم‌افزارها و وب‌گاه‌های بسیاری در انجام تست نرم‌افزار در سه نوع موبایل، وب و دسکتاپ وجود دارد؛ مثل نرم‌افزار مانکی که مخصوص تست نرم‌افزارهای موبایلی است.

برخی ابزارهای تست

ـ Appium: نرم‌افزار تست خودکار.

ـ Calabash: نرم‌افزار تست خودکار.

ـ xamarin test cloud: با این شرکت، می‌توان یک تست خودکار روی تعداد بسیاری گوشی به طور هم‌زمان انجام داد.

ـ Testdriod: تست خودکار نرم‌افزار موبایل روی گوشی‌های واقعی اندروید و iOS.

ـ prefecto mobile: تست خودکار و functional برنامه. استفاده از مدل continuous quality این نرم‌افزار، commercial است.

ـ SOASTA touch test: تست خودکار نرم‌افزار موبایل، به صورت functional.

ـ Testin: این نرم‌افزار، این قابلیت را فراهم می‌سازد که نرم‌افزارهای خود را روی بیشتر از ۳۰۰ گوشی تست کنیم. حالت cload آن، امکانات بیشتری مانند automated compatibility و performance testing دارد.

ـ Ubertesters: این برنامه کمک می‌کند که روند توسعه نرم‌افزار بر اساس پروسه QA، سازمان یافته‌تر باشد.

ـ crashlytics: این برنامه، یک نرم‌افزار آزاد برای iOS و اندروید است.

ـ Ranorex: این برنامه، یک نرم‌افزار تست موبایل است که به وسیله آن یک تست را رکورد می‌کنیم و می‌توانیم آن را روی deviceهای بسیاری اجرا نماییم.

ـ Experitest: نرم‌افزار تست خودکار و تست functional. با این نرم‌افزار می‌توان یک تست رکورد کرد و آن را به دفعات زیاد روی یک دیوایس اجرا نمود. می‌توان با زبان‌های python و C# یک تست را نوشت و روی گوشی‌های مختلف اجرا کرد.

ـ Android Lint: یک ابزار است که به eclipse اضافه می‌شود و کدهای پروژه را به منظور یافتن اِشکال چک می‌کند و همچنین، از نظر امنتی و کارایی، عملکرد نرم‌افزار را بهبود می‌بخشد.

ـ FindBugs: یک نرم‌افزار تست کد استاتیک است که می‌توان از آن برای اصلاح برنامه‌های جاوا استفاده کرد.

ـ Maveryx: یک نرم‌افزار تست برای تست functional، تست خودکار و تست GUI در نرم‌افزارهای اندروید است.

ـ clang static analyzer: یک ابزار اپن‌سورس برای iPhone است که تست static انجام می‌دهد و نرم‌افزار را آنالیز می‌نماید.

به نظر می‌رسد، مشتری انتظار دیدن باگ در سیستم را ندارد و با دیدن هر باگ، از سیستم ناامید و ناامیدتر می‌شود. باگ را در هر کدی می‌توان یافت و فقط به برنامه‌نویسان ضعیف اختصاص ندارد. تست توسط تیم جدا می‌تواند به‌راحتی از هزینه‌های اضافی و نارضایتی مشتری جلوگیری نماید

ـ Analyze code for Xcode: یک افزونه برای Xcode است که در زمان compile، کد را آنالیز کرده، اِشکال‌ها و باگ‌ها را گزارش می‌دهد.

ـ Monkeytalk: یک نرم‌افزار آزاد است که تست خودکار و functional را با روش‌های بسیاری مثل smoke test انجام می‌دهد.

ـ Caliper: یک نرم‌افزار اپن‌سورسِ ساخته‌شده توسط گوگل است که نتایج javamicrobenchmarks را روی اپلیکیشن ما نشان می‌دهد. این روش، در واقع، یک تست کارایی با benchmarkهای موجود است.

ـ EMMA: این برنامه، یک ابزار مناسب برای اندازه‌گیری code coverage مربوط به یک برنامه جاواست.

ـ Robotium: یک فریم‌ورک تست خودکار است که بدون نیاز به دانش زیاد در مورد اپلیکیشنِ مورد تست، می‌توان آن را آزمایش کرد. نوع تست آن، balck-box است؛ یعنی نیازی به این نیست که کد برنامه را در اختیار داشته باشیم.

ـ Robolectric: یک فریم‌ورک unit test است که کلاس‌ها را دنبال می‌کند و قابلیت این را دارد که بدنه متدها را دوباره بنویسد؛ به‌گونه‌ای‌که به صورت default مقادیر را به Null یا ۰ برگرداند.

ـ Monkeyrunner: به IDE اضافه می‌شود و به ما اجازه می‌دهد که تست functional بگیریم. این نوع تست، به سورس‌کد برنامه احتیاجی ندارد و بر روی دیوایس و emulator اجرا می‌شود و در نتیجه، می‌توان با python برنامه نوشت.

ـ Apthwack: یک API آزاد برای تست نرم‌افزار است که به کمک آن، برنامه را آپلود کرده، تست می‌کنیم که اشکال‌ها را به ما می‌گوید. این متد، روی تعداد بسیاری دیوایس فرایند تست را انجام می‌دهد و مصرف منابع را به ما گزارش می‌دهد.

ـ HP: یک راه حل تست شرکت hp با قابلیت‌های تست موبایل، مانیتورینگ نرم‌افزار موبایل، امنیت نرم‌افزار موبایل و شبیه‌سازی شبکه با کارایی بالاست.

ـ mobilelabs: یک راه حل جامع برای تست نرم‌افزار موبایل در کلود (فضای ابری) است.

ابزارهای تست GUI(۲۲)

نام نرم‌افزار	قیمت
Squish	Commercial
Ranorex	Commercial
TestComplete	Commercial
Test Studio	Commercial
eggPlant	Commercial
Testing anywhere	Commercial

ابزارهای محبوب خودکارسازی تست وب

نام نرم‌افزار	رجیستر
Selenium	Open Source
Watir	Open Source
Windmill	Open Source
Ranorex	Commercial
SoapUI	Open Source
Sahi	Open Source
Tellurium	Commercial

ابزارهای تست واحد(۲۳)

نرم‌افزار
NUnit
xUnit.net
PyUnit / unittest
JUnit
TestNG
PHPUnit
Symfony Lime
Test::Unit
RSpec

ابزارهای تست نرم‌افزار موبایل

ابزارهای تست موبایل، ممکن است به کد برنامه احتیاج داشته باشند یا تنها لازم باشد که فایل محصول را به آن داد تا آزمایش مورد نظر را انجام دهد.

ابزارهای تست کد

در این قسمت، برنامه‌هایی را معرفی می‌کنیم که برای استفاده از آنها احتیاج به کد است و گاهی نیز حین توسعه نرم‌افزار کارایی دارند. در این نوع، ابزارهای جداگانه‌ای برای مشاهده وجود ندارد و عموماً با نصب SDK می‌توان از این ابزارها استفاده کرد.

ـ Monkey: این ابزار به صورت اتفاقی دکمه‌های مختلف را تست می‌نماید؛ یعنی eventهای مختلف را call می‌کند تا

شرکت‌های تولید محصولات نرم‌افزاری، اگر بیشتر روی یونیت تست‌ها سرمایه‌گذاری کنند و تمامی کدهای تست خود را بنویسند، ضمن اینکه بخشی از کد خود را می‌توانند به صورت ماشینی خودکار نمایند، در قسمت رابطه کاربری هم می‌توانند با توجه به کدهای تست در لایه کد، برخی از تست‌های رابطه کاربری را به صورت ماشینی انجام دهند

برنامه در صورت وجود مشکل، کرش کند. ابزار فوق، برای stress test، برنامه مناسبی است.

ـ Fitnesse: این یک فریم‌ورک است که به افراد پروژه امکان ارتباط بیشتر را با کسانی که در یک پروژه در حال همکاری هستند، می‌دهد؛ ولی در عین حال، ابزارهای تست هم دارد. این ابزار، برای مدل تست acceptance test و Integrated test استفاده می‌شود.

ـ Hamcrest: یک لایبرای تست است که به کد هم احتیاج دارد. با استفاده از متدهایی که دارد، می‌تواند Unit تست انجام دهد. در صورتی که احتیاج باشد برنامه را به صورت ایزوله تست کند، می‌تواند با امکانی به اسم parser test کامپوننت‌های برنامه را به صورت ایزوله تست نماید؛ حتی اگر آنها به یک API، JSON یا XMLهایی خارج از برنامه احتیاج داشته باشند. همچنین، می‌توان با آن Memory leak برنامه را تشخیص داد.

Mobile Web Emulators & Testing

نام نرم‌افزار
mobiReady
BrowserStack
CrossBrowserTesting
Screenfly
Mobile phone emulator
Responsive
ProtoFluid

Automated Mobile Testing Tools

نام نرم‌افزارها	دسترسی
DeviceAny-where	Commercial
Ranorex	Commercial
eggPlant	Commercial
Silk Mobile	Commercial
SeeTest	Commercial
MonkeyTalk	Open Source
NativeDriver	Open Source

Android Testing Tools

نام ابزار تست آندروید
Testdroid
robotium
scirocco
Monkey

حجم برنامه‌های کاربردی، روزبه‌روز در حال افزایش است. از نظر مهندسان نرم‌افزار، نوشتن کدهای تست، به‌خودی‌خود، مثل توسعه خود محصول، وقت‌گیر و گران است. خودکارسازی تست، زمانی مؤثر است که شرایط نرم‌افزار تحت تست مناسب باشد. باید توجه داشت که تست خودکار، به این معنا نیست که کل فرآیند تست نرم‌افزار به صورت خودکار انجام می‌شود؛ بلکه به معنای تست نرم‌افزار با کمک رایانه است. استفاده از تست خودکار، پُرهزینه است و در واقع، تست خودکار، مکملی برای فرآیند تست دستی می‌باشد.

لزوم تست نرم‌افزار توسط تیم جدا

آیا بخش تست شما جداست؟ این، یعنی همان دهمین آیتم از تست جوئل. اگر در تیم شما افرادی که وقتشان اختصاصاً برای تست‌کردن باشد ـ حداقل یک نفر برای هر دو یا سه برنامه‌نویس ـ وجود نداشته باشد، شما محصولات خود را باگ‌دار تحویل خواهید داد.

اینکه برنامه‌نویس یک سیستم، نرم‌افزار را تست کند، لازم و ضروری است؛ ولی به هیچ عنوان کافی نیست. این تصمیم که محصول تولیدی، توسط تیم جداگانه‌ای تست شود، شاید در مرحله اوّل خوب به نظر نیاید، ولی در واقع، بسیار مفید است. دلایل اشتباهی که ممکن است فکر کنیم به شخص یا ابزار تست‌کننده نیاز نداشته باشیم، عبارت‌اند از:

ـ باگ‌ها توسط برنامه‌نویسان ضعیف تولید می‌شوند؛

ـ برنامه من، یک برنامه وب است و سریع می‌توانم مشکل آن را برطرف کنم؛

ـ مشتری من، سیستم را برای من تست می‌کند؛

ـ من از عهده هزینه تستر برنمی‌آیم.

به نظر می‌رسد، مشتری انتظار دیدن باگ در سیستم را ندارد و با دیدن هر باگ، از سیستم ناامید و ناامیدتر می‌شود. باگ را در هر کدی می‌توان یافت و فقط به برنامه‌نویسان ضعیف اختصاص ندارد. تست توسط تیم جدا می‌تواند به‌راحتی از هزینه‌های اضافی و نارضایتی مشتری جلوگیری نماید.

چشم‌انداز

حجم برنامه‌های کاربردی، روزبه‌روز در حال

افزایش است. از نظر مهندسان نرم‌افزار، نوشتن کدهای تست، به‌خودی‌خود، مثل توسعه خود محصول، وقت‌گیر و گران است. خودکارسازی تست، زمانی مؤثر است که شرایط نرم‌افزار تحت تست مناسب باشد. باید توجه داشت که تست خودکار، به این معنا نیست که کل فرآیند تست نرم‌افزار به صورت خودکار انجام می‌شود؛ بلکه به معنای تست نرم‌افزار با کمک رایانه است. استفاده از تست خودکار، پُرهزینه است و در واقع، تست خودکار، مکملی برای فرآیند تست دستی می‌باشد.

نتیجه‌گیری

در این بین، جای یک چارچوب تست خودکار جامع که در آن مجموعه‌ای از ابزارهای خوب و کارآمد موجود باشد، خالی است و نیاز به چنین چارچوبی به‌وضوح قابل درک است. در مجموع، می‌توان گفت روش آزمون در این چارچوب، روش جعبه خاکستری است؛ زیرا گرچه نیازی به داشتن اطلاعات دقیق از کد منبع نرم‌افزار و ارتباطات داخلی آن برای انجام آزمون نداریم، اما دسترسی به یک مدل کلی از نرم‌افزار و یا دسترسی به خود کد منبع برای به دست آوردن یک مدل، به ما در تهیه موارد تست جامع‌تر کمک می‌نماید.

شایان توجه است که در قسمت ابزار تست، با توجه به آزمون‌های تست که از طریق ابزارهایی مانند: تست کامپلیت، رونرکس و تستینگ انی‌ور داشته‌ایم، به این نتیجه رسیدیم که این ابزارها تا حدودی می‌توانند نیازهای شرکت‌های تولید نرم‌افزار را پوشش بدهند؛ اما با توجه به اینکه این نرم‌افزارها کامرشیال است، مقرون به صرفه نیست که شرکت‌های تولید نرم‌افزار به سوی این نرم‌افزارها بروند و از آنها استفاده کنند. این نرم‌افزارها، ایده‌های خوبی برای خودکارسازی تست می‌دهند و می‌توان از این محصولات جهت تولید ابزار تست به صورت مهندسی معکوس عمل کرد.

به‌علاوه، شرکت‌های تولید محصولات نرم‌افزاری، اگر بیشتر روی یونیت تست‌ها سرمایه‌گذاری کنند و تمامی کدهای تست خود را بنویسند، ضمن اینکه بخشی از کد خود را می‌توانند به صورت ماشینی خودکار نمایند، در قسمت رابطه کاربری هم می‌توانند با توجه به کدهای تست در لایه کد، برخی از تست‌های رابطه کاربری را به صورت ماشینی انجام دهند.

پی‌نوشت‌ها:

1. ITG: Independen Test Groupt.
2. Unit test.
3. Integration test.
4. GUI test.
5. Performance test.
6. Installation test.
7. compatibility.
8. regression.
9. acceptance.
10. alpha.
11. beta.
12. Destructive.
13. usability.
14. security.
15. Development.
16. static.
17. dynamic.
18. white-box.
19. Code coverage.
20. black-box.
21. grey-box.
22. Automated GUI Testing Tools.
23. Unit Testing Frameworks.

منابع:

1.http://www.testingtools.com/test-automation

2. https://en.wikipedia.org/wiki/List_of_web_testing_tools

3. https://en.wikipedia.org/wiki/List_of_GUI_testing_tools

4. http://ageeknotes.com

5. Automatic Test Factoring for Java – David Saff, Shay Artzi, Jeff H.Perkins, Michael D. Ernst

6. Selective Capture and Replay of Program Executions – Alessandro Orso and Bryan Kennedy

7. Eclat: Automatic Generation and Classification of Test Inputs – Carlos Pacheco, Michael D. Ernst

8. Orstra: Augmenting Automatically Generated Unit-Test Suites with Regression Oracle Checking

9. Substra: A Framework for Automatic Generation of Integration Tests – Hai Yuan, Tao Xie

10. Carving Differential Unit Test Cases from System Test Cases – Sebastian Elbaum, Hui Nee Chin, Matthew B. Dwyer, Jonathan Dokulil

11. Rostra: A Framework for Detecting Redundant Object-Oriented Unit Tests – Tao Xie, Darko Marinov, David Notkin

12. Symstra: A Framework for Generating Object-Oriented Unit Tests Using Symbolic Execution – Tao Xie, Darko Marinov, Wolfram Schulte, David Notkin

13. An Empirical Comparison of Automated Generation and Classification Techniques for Object Oriented Unit Testing – Marcelo d'Amorim, Carlos Pacheco, Tao Xie, Darko Marinov, Michael D. Ernst.■