آزمون نرم‌افزار

توسعه نرم‌افزار
فعالیت‌های اصلی در مهندسی نرم‌افزار
فرایند • نیازمندی‌ها • طراحی • ساخت • آزمون • اشکال‌زدایی • استقرار • نگهداری
پارادایم‌ها و مدل‌ها
چابک • Cleanroom • افزایشی • پروتوتایپ • مدل مارپیچی • آبشاری
متدولوژی‌ها و چارچوب‌ها
دواپس • PSP • MSF • MDD • Lean SD • Kanban • IID • FDD • DSDM • توسعه سریع نرم‌افزار• SAFe • RAD XP • مدل وی • Scrum • SEMAT • TSP • RUP
رشته‌های مورد حمایت
رشته‌های مورد حمایت مدیریت پیکربندی • مستندسازی • تضمین کیفیت (SQA) • مدیریت پروژه • تجربه کاربری
کاربردهای عملی
توسعه آزمون محور • TDD • Stand-up • PP • DDD • CD • CI • CCO • BDD
ابزار
کامپایلر • اشکال‌زدا • رخ‌نمانگار • GUI designer • مقایسه ابزارهای زبان مدل‌سازی یکپارچه • محیط توسعه یکپارچه • آزمون • Build automation • Release automation • Infrastructure as Code
استانداردها و بدنه‌های دانش
IEEE standards • CMMI • ایزو ۹۰۰۰ • SWEBOK • ISO/IEC standards • پم‌باک • BABOK
واژه‌نامه‌ها
Artificial intelligence • Computer science • Electrical and electronics engineering
رئوس مطالب
Outline of software development

آزمون نرم‌افزار یا **تست نرم‌افزار** (به انگلیسی: software testing) به فرایند ارزیابی نرم‌افزار به منظور اطمینان از عملکرد صحیح آن در رویدادهایی مختلفی که ممکن است در دوره استفاده از نرم‌افزار با آن مواجه شود می‌باشد و به عبارت دیگر پیدا کردن خطاهایی احتمالی یک نرم‌افزار برای عملکرد درست، صحیح و بهینه آن در طول استفاده از آن است. هر چقدر نرم‌افزار بتواند با رویدادها مختلف به صورت مطلوب تر و قابل پذیرش تری چه از نظر عملکرد و چه از راحتی کاربر داشته باشد می‌توان انتظار داشت نرم‌افزار دارای عملکرد بهتری می‌باشد.

آزمون برنامه شامل اجرای بخش هایی (کامپوننت هایی) از برنامه یا بخش هایی از سیستم میشود تا مشخصات موردنظر سیستم را ارزیابی کند. بصورت کلی این مشخصات مشخص میکنند که هرکدام از بخش های برنامه ویژگی های زیر را تحت عملیات آزمون کردن دارند :

- به نیازمندی هایی که توسعه و طراحی نرم افزار را جهت دهی کرده اند رسیده است؟
- به انواع ورودی ها پاسخ مناسبی میدهد؟
- عملکرد خود را در زمان قابل قبولی انجام میدهد؟
- به اندازه کافی کارآمد است؟

• آیا میتوان آن را روی محیطی که برای آن برنامه ریزی انجام گرفته است نصب و اجرا کرد؟

• به نتیجه کلی که مطلوب سرمایه گذاران است دست پیدا کرده است؟[۱]

همانطور که تعداد تست های ممکن حتی برای مولفه های برنامه های ساده اغلب نامحدود هستند، تمامی تست کننده های برنامه از روشی استفاده میکنند که تست هایی ساده و در عین حال مناسب برای زمان و منابع سیستم را انجام دهند. در نتیجه، تست کردن برنامه عموما در تلاش است تا برنامه یا اپلیکیشن را با رویکرد یافتن حفره های برنامه اجرا کند.فرایند تست کردن، پروسه ای همراه با تکرار است یعنی که هنگامیکه یکی از حفره های برنامه درست شد فرایند تست کردن باید مجددا انجام پذیرد زیرا درست کردن این حفره میتواند حفره های دیگر، عمیق تر یا حتی جدیدی را نمایان کند. آزمون نرم افزار میتواند اطلاعات حیاتی و مستقلی درباره کیفیت برنامه و میزان ریسک شکست یا عدم موفقیت آن در مقابل استفاده کنندگان یا اسپانسرهای سیستم را ارائه دهد. آزمون نرم افزاری میتواند در زمانیکه برنامه بصورت کامل یا حتی قسمتی از آن در دسترس بود انجام پذیرد.فرایند توسعه نرم افزار عموما مشخص میکند چه زمانی و به چه صورتی آزمون نرم افزرا صورت پذیرد. برای مثال در فرایند فاز بندی شده، بیشتر تست کردن ها زمانی انجام میگیرد که پیش نیاز های سیستم پیدا شده و سپس در برنامه ای قابل آزمون پیاده سازی شده اند.البته در رویکرد Agile نیازمندی ها، برنامه نویسی و آزمون نرم افزاری عموما بصورت همزمان انجام میپذیرند.[۱]

در سال‌های اخیر آمارهای شگفت‌آوری از سوی مؤسسه NIST(National Institute of Standards and) تست نرم‌افزارTechnologyدربارهٔ شکست سیستم‌های نرم‌افزاری ارائه شده‌است. در کشور ایالات متحده، این شکستها سالیانه حدود ۵۹٫۵ میلیارد دلار به اقتصاد این کشور صدمه می‌زند. طبق بررسیهای انجام شده با بکارگیری تست در تمام فازهای تولید نرم‌افزار ۲۲٫۲ میلیارد دلار از این خسارت را می‌توان کاهش داد. طبق آمارهای ارائه شده از سوی مؤسسه IDC(International Data Corporation)، چهل درصد از بودجه نرم‌افزارها صرف تست آن می‌گردد. در کشور ما نیز، با توجه به رشد فن آوری اطلاعات و ارتباطات در طی چند سال گذشته و تولید بومی بسیاری از نرم‌افزارهای مورد نیاز، نیاز به این فرایند بیش از پیش احساس شده و در صورت عدم توجه به آن، کاهش کیفیت سیستم‌های ارائه شده، عدم رضایت مشتری و در نهایت از دست دادن بازار را به همراه خواهد داشت.[۲] تست خوب: احتمال پیدا کردن خطاهای کشف نشده توسط ارزیابی زیاد است. تست موفق: که حداقل یک خطای کشف نشده را بیابد تست فقط وجود خطا را نشان می‌دهد و نه عدم وجود آن را. پیدا نشدن خطا در تست به معنای بدون خطا بودن برنامه نیست. اصول تست با توجه به نیازمندیهای کاربر برنامه‌ریزی قبل از اجرا (test plan) نوشتن برنامه تست قانون پارتو ۸۰% خطاهای کشف نشده در ۲۰٪ کد است تست باید از اجزای کوچک شروع شود ممکن نیست (exhaustive) تست کامل برای مؤثر بودن باید توسط شخص ثالث بی‌طرف انجام شود معیارهای تست پذیر بودن نرم‌افزار:

۱. قابلیت اجرا Operability – هرچه نرم‌افزار بهتر کار کند و در محیط‌های بیشتری قابل اجرا باشد، n بهتر قابل ارزیابی است

۲. مشاهده‌پذیری Observability – قابلیت مشاهده نتایج ارزیابی

۳. کنترل‌پذیری Controlability – قابلیت اجرای تستهای خودکار (مثل امکان اجرای خودکار تست‌های واحد توسط jUnit برای زبان جاوا)

۴. تجزیه‌پذیری Decomposability – ارزیابی می‌تواند هدفمند تر شود

۵. سادگی Simplicity – کاهش پیچیدگی معماری و منطق برنامه

۶. پایداری Stability – برای ارزیابی تغییرات کمی بخواهد

۷. درک‌پذیری Understandability – قابلیت درک طراحی و وابستگیهای بین اجزا

سطوح مختلف آزمون:

- آزمون واحد (Unit testing)
- آزمون یکپارچه‌سازی افزایشی
- آزمون یکپارچه‌سازی (Integration testing)
- آزمون سیستم (System testing)

- آزمون پذیرش (Acceptance testing)

۱. آزمون آلفا

۲. آزمون بتا

محتویات

آزمون واحد

تست واحد یا micro level پایین‌ترین سطح تست است. هر کد تست واحد، یک قطعه کد یا یک تابع (متد) خاص را تست می‌کند. این تست نیاز به دانش در مورد طراحی و نحوه عملکرد داخلی تابع یا قطعه کد دارد. توسط برنامه‌نویس (و نه تست‌کننده) انجام می‌شود.

آزمون یکپارچه‌سازی افزایشی

تست یکپارچه‌سازی افزایشی با افزوده شدن قابلیت جدید به نرم‌افزار، مجدداً نرم‌افزار تست می‌شود. هدف این تست، بررسی درستی نرم‌افزار پس از افزوده شدن امکان جدید است. امکانات نرم‌افزار باید از هم استقلال داشته باشند تا بتوان پیش از تکمیل کل نرم‌افزار و به صورت افزایشی نرم‌افزار را تست کرد. توسط برنامه‌نویس یا تیم تست انجام می‌شود.

تست یکپارچه‌سازی

تست یکپارچه‌سازی تست نرم‌افزار حاصل از کنار هم قرار گرفتن قطعات مختلف آن به منظور بررسی درستی عملکرد نرم‌افزار یکپارچه شده قطعات مختلف شامل قطعه کدها (ماژول‌هایی از کد برنامه‌های مجزا که در کنار هم برنامه اصلی را تشکیل می‌دهند برنامه‌های مشتری-کارگزار عمل‌کننده در یک شبکه پس از تست واحد انجام می‌شود.

آزمون سامانه

آزمون سامانه به منظور بررسی عملکرد نرم‌افزار بر روی پلتفرم‌های مختلف انجام می‌شود و نرم‌افزارهای OS پلتفرم: سخت‌افزار + نرم‌افزار (شامل کاربردی مورد نیاز برنامه) به منظور اطمینان از اینکه برنامه با مؤلفه‌های دیگر محیط اجرایش به خوبی کار می‌کندبه منظور اطمینان از اینکه نرم‌افزار ارائه شده در محیط مورد نظر قابل استفاده است. مثالی

از مشکل حاصل از انجام ندادن تست سیستم نرم‌افزار بازی شیر شاه دیزنی Disney’s Lion King Game در پاییز سال ۱۹۹۴ شرکت دیزنی اولین CD بازی خود تحت عنوان شیر شاه Lion King که بر اساس کارتونی به همین نام ساخته شده بود را وارد بازار کرد. بسیاری از شرکت‌های دیگر تا آن زمان اقدام به ساخت بازی‌های رایانه‌ای کرده بودند اما این اولین بار بود که شرکت دیزنی وارد این تجارت شده بود. دیزنی برای فروش این بازی دست به تبلیغات گسترده‌ای زد و در نتیجه این محصول با فروش بسیار بالایی مواجه شد. اما اتفاقات پس از آن تبدیل به کابوسی برای این شرکت شد. در ۲۶ دسامبر، روز پس از کریسمس تلفن‌های بخش پشتیبانی مشتریان شرکت دیزنی شروع کرد به زنگ زدن و زنگ زدن و زنگ زدن! متصدیان پاسخگویی به تماس‌ها با خیل عظیمی از والدین عصبانی با بچه‌های گریان مواجه شدند که ادعا می کرند نرم‌افزار مزبور کار نمی‌کند. این خبر به سرعت در مطبوعات و تلویزیون نیز پخش شد و کریسمس آن سال را برای بسیاری از پرسنل دیزنی تلخ کرد. تست علت چه بود؟ پس از بررسی مشخص شد که دیزنی نرم‌افزار خود را بر روی بسیاری از مدل‌های PCتست نکرده بود و در نتیجه تنها بر روی سیستم‌هایی کار می‌کرد که برنامه نویسان دیزنی روی آن سیستم‌ها نرم‌افزار خود را توسعه داده بودند و نه دستگاه‌های متداولی که عموم مردم از آن استفاده می‌کردند.

آزمون پذیرش

آزمون پذیرش به منظور بررسی اینکه نرم‌افزار نیازهای مشتری را برآورده می‌کند، انجام می‌شود بعد از تست سیستم انجام می‌شود؛ که *شامل*: ۱- تست آلفا: تست آلفا در سایت توسعه دهنده نرم‌افزار و در اغلب موارد n توسط کارمندان داخلی و در بعضی از موارد توسط مشتری تعدادی از کاربرانش که به محل دعوت می‌شوند انجام می‌گیرد. ۲- تست بتا: در تست بتا نسخه‌هایی از نرم‌افزار در اختیار تعدادی از کاربران قرار می‌گیرد تا در بازه‌ای با آن کار کنند و خطاها را گزارش دهند.

روشهای ارزیابی روش جعبه سفید دانستن نحوه کار داخلی برنامه امکان تأیید نحوه عمل هر تکه کد و مسیر اجرا مراحل اولیه ارزیابی روش جعبه سیاه دانستن عمل مورد انتظار و مطلوب امکان تأیید کاری که سیستم باید انجام دهد مراحل انتهایی ارزیابی استرتژی تست استراتژی تست نرم‌افزار یک توصیف رسمی از این است که نرم‌افزار چگونه تست خواهد شد. هدف استراتژی تست تعریف همه مراحل برای فرایند تست نرم‌افزار است که شامل برنامه‌ریزی آزمایش، طراحی ابزار آرمایش، اجرای آزمایش و جمع‌آوری و ارزیابی داده‌های بدست آمده باشد.

استراتژی جعبه سیاه

این آزمایش جایگزین آزمایش جعبه سفید نمی‌باشد بلکه مکمل آن است. وخطاهایی متفاوت باآن راتست می‌کند. شما نرم‌افزاری را که به آن نیاز داشتید را تهیه می‌کنید و بر روی سیستم خود نصب می‌کنید، شما در اکثر موارد بعد از نصب برنامه فقط یک نسخه اجرایی آن را در سیستم خود خواهید داشت، و هیچ دسترسی به سورس کد و منابع دیگر برنامه نخواهید داشت. سیستم نرم‌افزاری موجود برای شما مانند یک جعبه سیاه است که شما نمی‌توانید دورن آن را مشاهده کنید و به آن دسترسی داشته باشید. استراتژی جعبه سیاه دقیقاً از این دیدگاه برنامه را مورد آزمایش قرار می‌دهد، یعنی با این پیش‌فرض که شما هیچ اطلاعاتی از کد و طراحی داخلی برنامه ندارید. حالا هیچ اطلاعاتی از کد و طراحی برنامه در اختیار ما نیست، پس چگونه می‌توان به صحت کارکرد برنامه پی برد؟ جواب خیلی ساده است، با تمرکز بر ورودی‌ها و خروجی‌ها، برای اینکار آزمایش‌کننده نرم‌افزار به مستندات نرم‌افزار مراجعه می‌کند تا مشخص کند که سیستم در مقابل یک عمل خاص چه پاسخی را باید بدهد. سپس داده‌ها را برای هر کدام از عملیات ها انتخاب می‌کند و رفتار سیستم را در مقابل آن داده‌ها با رفتار واقعی سیستم که در مستندات وجود دارد مقایسه و بررسی می‌کند.

در یک استراتژی آزمایش جعبه سیاه ما عموماً موارد زیر را مورد بررسی و آزمایش قرار می‌دهیم:

۱. بررسی اینکه سیستم نیازمندهای عملیاتی و غیر عملیاتی را تأمین می‌کند یا نه.

۲. اعتبارسنجی ورودیها

۳. بررسی مقادیر مرزی برای متغیرها: به یک متغیر مقداری کمتر از حداقل مقداری که می‌تواند قبول کند یا بیشتر از حداکثر مقداری که می‌تواند قبول کند می‌دهیم و سیستم را در این شرایط تست می‌کنیم.

۴. بررسی خروجهای سیستم: یک مجموعه از ورودهای صحیح با خروجهای مربوط به آن را تهیه می‌کنیم و سپس ورودها را به سیستم وارد می‌کنیم و خروجهای که توسط سیستم داده می‌شود را با خروجی‌های واقعی مقایسه می‌کنیم.

۵. بررسی رفتار سیستم در برابر پردازش ورودها و پرس و جوهای بزرگ و سنگین

برای موارد بالا و مواردی دیگری که ذکر نشد روش‌های مختلف تست در استراتژی جعبه سیاه وجود دارد که عبارتند از:

en:Functional testing

en:Stress testing

en:Recovery Testing

en:Volume testing

en:User Acceptance testinge

en:System testing

en:Sanity or Smoke testing

en:Load testing

en:Usability testing

en:Exploratory testing

en:Ad-hoc Testing

en:Alpha testing

en:Beta testing

آیا می‌توان در استراتژی جعبه سیاه، از این مطمئن شد که سیستم به‌طور کامل تست شده‌است؟ خیر، هرگز در این استراتژی نمی‌توان مطمئن شد که سیستم به‌طور کامل تست شده‌است. برای نمونه با چه احتمالی کاربر می‌تواند ورودهای را انتخاب کند تا شرط زیر چک شود.

```
if
("name== "Lee" && employeeNumber =="1234" && employmentStatus=="RecentlyTerminatedForCause) {
 send Lee a check for $۱٬۰۰۰٬۰۰۰;
}
```

پس همیشه در این استراتژی مسیرها خواهند بود که تست نمی‌شوند و همیشه سیستم با داده‌های ورودی محدود می‌توانند تست شوند.

استراتژی جعبه سفید

حال تصور کنید که شما خود یک توسعه دهنده نرم‌افزار هستید، پس شما می‌توانید به سورس، طراحی و منابع دیگر نرم‌افزار دسترسی داشته باشید. در این حالت سیستم را می‌توان به یک جعبه شیشه‌ای (جعبه سفید) تشبیه کرد که شما می‌توانید براحتی محتویات داخل و نحوه عملکرد آن را مشاهده کنید. آزمایش جعبه سفید نیز دقیقاً از دیدگاه توسعه دهنده نرم‌افزار را مورد آزمایش قرار می‌دهد یعنی با این فرض که شما به منطق داخلی و ساختار کد برنامه دسترسی و احاطه دارید و می‌دانید که سیستم چگونه پیاده‌سازی شده‌است. شما با دانستن این موارد می‌توانید مشخص کنید که آیا اعمال داخلی بر طبق مشخصه‌ها انجام می‌شود یا نه.

در یک استراتژی آزمایش جعبه سفید ما عموماً موارد زیر را مورد توجه و بررسی قرار می‌دهیم:

۱. بررسی سطر به سطر کد (Code coverage): در این حالت باید سیستم را به گونه‌ای اجراء و بررسی کنیم که مطمئن شویم سطر به سطر کد برنامه حداقل یکبار اجراء شده‌است.

۲. بررسی همه انشعاب‌ها در کد برنامه (branch): در کد برنامه باید تمام عبارت‌های شرطی (if elseها و Switch caseها) را تک به تک مورد بررسی قرار داد. به این صورت که در یک عبارت if else هم قسمت if و هم قسمت else هر کدام به صورت مجزا یکبار اجراء شوند.

۳. بررسی همه حلقه‌ها در برنامه: حلقه‌ها در نرم‌افزار نقش اساسی دارند، چون می توننند با اشتباه جزئی مقدار زیادی از منابع را مصرف کرد برای مثال شرط خروج از حلقه به اشتباه هیچ وقت True نشود. برای نمونه حلقه‌ها را با ورودی بزرگتر از شرط خروج حلقه چک کنید یعنی حلقه اصلاً اجر نشود. تستی طراحی کنید که حلقه دقیقاً یکبار اجراء شود، تستی طراحی کنید که حلقه در یک بازه خاص اجراء شود و ...

۴. مدیریت خطای مطلوب: بررسی اینکه اگر به یک متد یک ورودی نامعتبر وارد شود، نحوه آگاه‌سازی و نمایش مطلوب خطا برای کاربر چگونه باشد؟

۵. بررسی امنیت: سیستم را از این جهت که چگونه در برابر دسترسی‌های غیرمجاز، هک، کرک و هر چیز دیگر که می‌تواند به آن آسیب برساند مورد بررسی قرار می‌دهد. در اینجا ما باید مکانهای از کد را که داده‌ها را اعتبارسنجی و مدیریت می‌کنند، دسترسی به منابع یا عملیات مهم و حیاتی را انجام می‌دهند را بررسی کنیم.

۶. برای موارد بالا و مواردی دیگری که ذکر نشد روش‌های مختلف تست در استراتژی جعبه سفید وجود دارد که عبارتند از:

Basis Path Testing, Equivalence Partitioning/Boundary Value Analysis, Method Coverage, Statement Coverage, Branch Coverage, Condition Coverage, Data Flow Testing, Flow Graphs Revisited, ...

جستارهای وابسته

- تحلیل دینامیک برنامه
- درستی‌یابی صوری

منابع

۱. [۱] (https://en.wikipedia.org/wiki/Software_testing#Testing_approach)
۲. «نسخه آرشیو شده» (https://web.archive.org/web/20170113153706/https://www.mohandespishegan.com/education-courses/papers/163-software-testing). بایگانی‌شده از اصلی (http://www.mohandespishegan.com/education-courses/papers/163-software-testing) در ۱۳ ژانویه ۲۰۱۷. دریافت‌شده در ۱۲ ژانویه ۲۰۱۷.

- (فارسی) https://web.archive.org/web/20161220215758/https://www.mohandespishegan.com/
- (انگلیسی) http://www.kaner.com/pdfs/ETatQAI.pdf

- سیاست محرمانگی
- دربارهٔ ویکی‌پدیا
- تکذیب‌نامه‌ها
-
- توسعه‌دهندگان
- آمار
- اظهارنامهٔ کوکی