تست مبتنی برمدل در بستر اندروید

الناز ذوالفقاری[۱]، دکترعباس رضایی*[۲]

۱.کارشناسی ارشد ،کامپیوتر نرم افزار، دانشگاه آزادکرمان ، ایران،
elnazzolfaghari8838@ gmail.com

۲.عضو هیات علمی، دانشگاه آزادکرمان، ایران،
abbas_rezaiee@yahoo.com

چکیده

هدف از تست نرم‌افزار، ارزیابی آن در رسیدن به اهداف از قبل تعیین شده و همچنین مشخص کردن هر گونه خطا در عملکرد نرم‌افزار است. در استراتژی جعبه سیاه با تمرکز بر ورودی‌ها و خروجی‌ها، برای این کار تستر به مستندات نرم‌افزار مراجعه می‌کند تا مشخص کند که سیستم در مقابل یک عمل خاص چه پاسخی را باید بدهد. سپس داده‌هایی را برای هر کدام از عملیات انتخاب می‌کند و رفتار سیستم را در مقابل آن داده‌ها با رفتار واقعی سیستم که در مستندات وجود دارد مقایسه و بررسی می‌کند و می‌توان با استفاده از تست جعبه سفید کد برنامه را بررسی کرد تا این‌که عامل خطا در داخل کد برنامه مشخص شود لذا باید برای کاهش موارد ازمون جهت تست مبتنی بر مدل در بستر اندروید به ارزیابی فرایند پرداخت. آزمون نرم‌افزار 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 می باشد .

کلید واژه: تست موارد آزمون، اندروید، ماشین حالت، نمودار حالت.

۱– مقدمه

همانطور که شاهد هستیم بازار فروش نرم افزار رفته رفته رشد پیدا می کند، بنابراین کیفیت و قابلیت اطمینان محصول یکی از اساسی ترین موضوعات فرایند تولید نرم افزار محسوب می شود. در سال ۲۰۰۴ یکی از شرکت های تولید کننده دستگاه های همراه به دلیل وجود خطا در نرم افزار دستگاه، خدمات پس از فروش اضافه ای را ارائه داد. در سال ۲۰۰۸ نیز در بازار بورس توکیو مشکل دیگری به وجود امد و به خاطر همین مشکل عملیات سیستم ۱۲ روز معلق ماند. دلیل گزارش شده برای خطا، عدم مقدار دهی اولیه در حافظه سرور بود. تحت شرایط خاصی حافظه مقدار دهی اولیه نمی شد (دیفرانسکو و اسپوسیت،[۱] ۲۰۱۲: ۲۹) بنابراین واضح است که برای ارائه محصول درست و قابل اطمینان به کاربر، نرم افزار باید به طور کامل تست شود. تمام خطاها و مشکلاتی که باعث می شوند نرم افزار غیرقابل استفاده باشد باید شناسایی و رفع شوند.

[1] Di Francesco. and Esposit

تست نرم افزار یکی از فازهای اصلی و گران در چرخه حیات نرم افزار محسوب می‌شود (ملک‌زاده و همکاران، ۱۳۹۲: ۳۵) محیط تست نرم افزار غنی از تکنیک‌هایی است که می‌توانند در طول فرآیند توسعه به کار گرفته شوند تا فواصل نرم افزار را تشخیص دهند (دیکلوا و تکشی،[۲] ۲۰۱۵: ۳) در میان آنها، روش برای تولید خودکار موارد آزمون با استفاده از یک مدل رفتار یا ساختاری، که به عنوان مدل آزمون یک سیستم تحت آزمون شناخته می‌شود تعریف شده است و این رویکرد به عنوان تست بر اساسی مدل شناخته می‌شود(دیکلوا و تکشی، ۲۰۱۵: ۱۹)، که به آن تست مبتنی بر مدل گفته می‌شود. تست مبتنی بر مدل یک رشته تحقیق و توسعه در زمینه دانشگاهی است که توجه بسیاری از صنایع به آن جذب شده است (علی و همتی،[۳] ۲۰۱۴: ۳۵۳) هدف از تست مبتنی بر مدل افزایش قابلیت اعتماد نرم افزار و کامش هزینه‌ها از طریق خودکارسازی نمونه‌های آزمون از رفتارهای رسمی مدل سیستم می‌باشد(ملک‌زاده و همکاران، ۱۳۹۲: ۳۴) رویکرد کامل و روشمند تست مبتنی بر مدل ، تست خودکار را با هدف بهبود کیفیت سیستم‌های نرم افزاری تسهیل می‌کند(علی و همتی،۲۰۱۴: ۳۶۰) هر برنامه کاربردی در تست مبتنی بر مدل دارای چالش‌های متنوعی است که گزارش این چالش‌ها و راه حل‌های آنها و درسی‌هایی که از آن‌ها آموخته می‌شود، دانش فراوانی را فراهم می‌آورد که متخصصان تست مبتنی بر مدل را برای برنامه‌های آینده تست مبتنی بر مدل خود هدایت می‌کند(علی و همتی،۲۰۱۴: ۳۵۹) از این رو ارائه یک چارچوب کارامد به منظور کاهش موارد آزمون جهت تست مبتنی برمدل در بستر اندروید بسیار لازم و ضروری است.

مبانی نظری

تست نرم افزار

سازمان‌ها یا شرکت‌هایی که نرم افزارها را توسعه می‌دهند، محصولی به نام نرم افزار تولید می‌کنند. ولی چه عامل یا عواملی باعث می‌شوند که یک نرم افزار از نرم افزار مشابه دیگر متمایز و برجسته شود؟ عوامل متعددی را می‌توان نام برد که باعث این برتری و تمایز شود اما یکی از این عوامل می‌تواند کیفیت محصول نهایی باشد که به بازار عرضه خواهد شد. اما برای رسیدن به این نقطه تمایز و برتری باید چگونه عمل کرد و اندیشید؟ یک پاسخ به این سوال می‌تواند تست نرم افزار و نحوه انجام آن باشد. حتی تست را می‌توان یکی از زیرمجموعه‌های مبحث کیفیت نرم افزار با نام "تضمین کیفیت" درنظر گرفت. در واقع تست نرم افزار به دنبال خطایابی و عیب یابی محصول، قبل از تحویل به مشتری است. اینکه هم توسعه دهندگان و هم کاربران نهایی برروی یک نرم افزار کارآمد و قابل بکارگیری که پاسخگوی نیازمندی‌های تعریف شده باشد، هم نظر باشند (بیذر،[۴] ۱۹۹۰: ۱۳۴). تست را می‌توان به صورت‌های زیر معنا کرد:

تلاش‌هایی در جهت عیب یابی و رفع آن، نه تلاش در جهت اثبات کامل صحت نرم افزار، زیرا این قضیه با ماهیت تست تفاوت دارد .

[2] De Cleva. and Takeshi
[3] Ali, And Hemmati
[4] Beizer

تست نرم افزار یا آزمایش نرم افزار، تحقیق برروی کیفیت یک محصول یا سرویس نرم افزاری و ارایه اطلاعات ناشی ازآن به مشتری‌ها است. این تحقیق جستجوی نرم افزار مزبور برای یافتن خطاها را نیز دربرمیگیرد ولی به آن محدود نمیشود .
تست‌ها یک سری از سوال و جواب‌هایی هستند که نرم افزار را با آن امتحان می‌کنیم در حالی که‌از برنامه انتظارداریم با توجه به ورودی‌هایی که با استفاده از سوالات وارد می‌کنیم، جواب‌های صحیحی را به عنوان خروجی به دست دهد .
تست نرم افزار حیطه وسیعی از فعالیت‌های مربوط به تولید برنامه‌های کامپیوتری را در بر می‌گیرد که از تست کردن کد برنامه توسط برنامه نویس گرفته تا نشان دادن عملکرد درست یک سیستم اطلاعاتی بزرگ به مشتری وارزیابی نرم افزار در حین رانش یک شبکه مرکزی کاربردی را دربرمی‌گیرد.
تست نرم افزار شامل فرآیند اجراهای متعدد برنامه با هدف یافتن باگ‌های نرم افزاری است اما محدود به آن نمی‌باشد، تضمین اینکه تا چه حد نیازمندی‌های موجود را برآورده می‌کند و آیا با انتظارات مشتری سازگار است یا خیر نیز توسط فرآیند تست مشخص می‌شود.
تستها به دفعات تکرار خواهند شد تا مشکلات هر چه بیشتر نمایان شوند. استاندارد تست نرم افزار فرآیندهایی را در جهت آشکارسازی و برطرف کردن عیوب سیستم دنبال خواهد کرد .

چه کسی تست می‌کند؟

در صنعت[IT] ، شرکت‌های بزرگ، تیمی برای بررسی نرم افزارهای تولیدی در چارچوب الزامات و فعالیت‌های شرکت دارند که این تیم، تیم تست می‌باشد. در اغلب موارد تستر در یکی از بخش‌های زیر قرار می‌گیرد:

- بخش تست نرم افزار
- بخش توسعه نرم افزار

شرکت‌ها طراحی‌های متفاوت و وظایف و نقش‌های متفاوتی برای اشخاصی که تست نرم افزار انجام می‌دهند ،قائل می‌شوند. این نقش‌ها عبارتند از : تستر نرم افزار، تضمین کیفیت نرم افزار [QA]، تحلیلگر و...

چه زمانی تست شروع می‌شود؟

در طول چرخه عمر توسعه نرم افزارکه به آن[5] [SDLC] گفته می‌شود، تست آغاز شده و تا استقرار نرم افزار به طول می‌انجامد. با این حال تمامی این تست‌ها بستگی به مدل توسعه ای دارد که شرکت‌ها انجام می‌دهند. به طور مثال در مدل آبشاری، تست در مرحله تولید نرم افزار انجام می‌شود اما در مدل افزایشی، تست در پایان هر افزایش یا تغییر، تکرارمی‌شود و در پایان تولید نرم افزار هم دوباره تست انجام می‌شود.
در هر مرحله از [SDLC] ، تجزیه و تحلیل و تاییدیه‌های مورد نیاز نیز برای تست در نظر گرفته می‌شود. بررسی طراحی در مرحله طراحی محصول نیز به قصد بهبود طراحی در حوزه تست نیز در نظر گرفته می‌شود. انجام تست توسط یک توسعه دهنده پس از اتمام کد نیز به عنوان تست واحد [Unit Test] طبقه بندی می‌شود (کاپلند،[6] ۲۰۰۴: ۱۶).

[5] - Software Development Life Cycle
[6] Copeland

انجام تست در طول SDLC مزایای زیر را در بر دارد:

۱. کاهش زمان تولید
۲. کاهش هزینه‌ها
۳. کاهش زمان دوباره کاری‌ها
۴. کاهش خطاهای نرم افزاری
۵. افزایش بازدهی
۶. افزایش کیفیت نرم افزار
۷. تحویل به موقع پروژه به کارفرما
۸. افزایش رضایت مشتری

چه زمانی تست پایان می‌یابد؟

بر خلاف آنکه می‌دانیم چه زمانی تست را آغاز کنیم، تعیین زمان پایان تست بسیار دشوار است. تست فرآیندی بی پایان می‌باشد و تعیین زمانی برای توقف آن بسیار دشوار است و نمی‌توان با اطمینان گفت که نرم افزار تولیدی٪ 100 تست شده است. مواردی که نبایستی برای تست درنظر گرفته شود:

۱. تعیین مهلت تست
۲. اطمینان از کامل بودن تست
۳. پس از اتمام تست فانکشنال و تست پوشش کد، نبایستی نتیجه را به نقطه خاصی سوق دهیم.
۴. اگر در سطح خاصی نرخ خطا کمتر بوده و بدون باگ باشد این سطح از اولویت بالاتری شناخته شود.

محصولات نرم افزاری برای کشف و رفع باگ‌ها وهمچنین تعیین کیفیت نیاز به تست دارند. روند تست نرم افزار گاهی می‌تواند به اندازه توسعه نرم افزار انرژی برده و با اهمیت باشد. تست نرم افزار شامل فرآیند اجراهای متعدد برنامه با هدف یافتن باگ‌های نرم افزاری است اما محدود به آن نمی‌باشد، تضمین اینکه تاچه حد نیازمندی‌های موجود را برآورده می‌کند و آیا با انتظارات مشتری سازگار است یا خیر نیز توسط فرآیند تست مشخص می‌شود.

انواع رویکرد تست

ما دو رویکرد برای تست داریم. این دونوع مشخص کننده نحوه طراحی تست‌های ما هستند.

Blackbox Testing

در این رویکرد، تست تمامی مکانیسم‌های داخلی یک سیستم نادیده گرفته می‌شود و روی خروجی تولید شده تمرکز می‌شود. به این رویکرد تست functional نیز می‌گویند.

Whitebox Testing

در این رویکرد، تست ما با مکانیستم داخلی یک سیستم سرو کار داریم. به این نوع تست structural تست نیز گفته می‌شود.

اهداف تست نرم افزار

- افزایش کیفیت نرم افزار
- بررسی سیستم از لحاظ پاسخگویی به نیازها و قابلیت اطمینان
- شناسایی ضعف‌ها، خطاها و اشتباه‌های نرم افزار

چرخه تست نرم افزار

اگر چه فرایند تست در سازمان‌های مختلف متفاوت است، اما تست نرم افزار دارای چرخه زیر است:

- تحلیل نیازها: فرایند تست باید در فاز تحلیل نیازهای چرخه زندگی نرم افزار انجام پذیرد .
- تحلیل طراحی: در طی فاز طراحی، تست کنندگان با طراحان همکاری می‌کنند تا مشخص شود که کدام قسمت از طراحی قابل تست است و در هنگام تست از کدام پارامترها باید استفاده شود .
- طراحی تست: در این مرحله استراتژی تست مشخص می‌شود.
- اجرای تست: تست کنندگان نرم افزار را اجرا کرده و تست می‌کنند و هرگونه خطا را به تیم توسعه دهنده گزارش می‌کنند.
- گزارش تست: بعد از اینکه فرایند تست تکمیل شد، تست کنندگان نتایج بدست آمده را در قالب یک گزارش تهیه می‌کنند و مشخص می‌شود که آیا نرم افزار قابل استفاده است یا خیر.

انواع تست

یکی از مهمترین مراحل تولید نرم افزار، فاز تست و رفع اشکال سیستم است که در تمام متدهای تولید سیستم‌های نرم افزاری از جمله RUP برای آن روش مدونی در نظر گرفته شده است. تعریف تست یک نرم افزار را می‌توان فرآیندی جهت کشف خطاهای کشف نشده نرم افزار در جهت مرتفع نمودن آن‌ها عنوان نمود. از انواع مختلف تست نرم افزار عبارتند از تست عملکرد، تست استرس، تست اکتشافی، تست تطبیق پذیری با محیط، تست امنیت بر شمرد. در ادامه به توضیحی کوتاه در خصوص هر کدام می‌پردازیم.

تست عملکرد(Performance testing)

در این نوع تست، نرم افزار از نظر درستی عملکرد بررسی شده و کامپوننتها و فرآیندها تست می‌شوند. این استراتژی با تست کدهای نرم افزاری و ساختمان داخلی آن سر و کار دارد. در این روش، کدها باید به گونه ای اجرا و بررسی شوند که مطمئن شویم سطر به سطر کدهای برنامه حداقل یک بار اجرا شده است. می‌توان گفت که تست‌ها طوری نوشته می‌شوند تا ببینند که آیا نرم افزار همان گونه که انتظار می‌رود عمل می‌کند یا خیر؟ معمولا تست عملکرد در انتهای کار انجام می‌شود ولی می‌توان از همان ابتدای کار با تست کردن قسمت‌های کوچک مثل کامپوننتها نتیجه نهایی را ساده کرد. هم چنین تست عملکرد می‌تواند مشکلات برنامه را پیدا کند به طور مثال:

- پایگاه داده با مقدار کمی از داده‌ها اجرا نشود.
- برنامه در زمان پیک، به علت بار زیاد نتواند اجرا شود.

تست عملکرد ارزیابی مجموعه‌های بسیار متفاوتی از مهارت‌های عملکردی و یا توانایی برنامه را انجام می‌دهد. این تست در صنعت نرم افزار بسیار شایع می‌باشد و کاستی‌ها و مشکلات عملیاتی در برنامه را قبل از اجرا شدن آن نمایش می‌دهد. اگر کسب و کار با شکست مواجه شود می‌تواند مشکلاتی مانند زیر را در پی داشته باشد:

- از دست دادن درآمد
- از دست دادن مشتریان جدید
- از دست دادن مشتریان قدیمی
- تبلیغات منفی در رسانه‌ها و وب سایت‌ها

معرفی ابزار

ابزارهای تست عملکرد می‌بایستی مشخصات زیر را داشته باشند:

- بایستی بار بر روی سیستم در حال تست، تولید کنند .
- زمان پاسخ سرور را اندازه گیری کنند .
- توان اندازه گیری داشته باشند .

در این پست برای هرنوع تست یک ابزار معرفی می‌کنیم. برای تست عملکرد ابزار جی میتر را می‌گوییم

Apache Jmeter

جی میتر به صورت متن باز(Open source) بوده که برای پلت فرم‌های جاوا به کار می‌رود. عمدتا از این ابزار به عنوان تست عملکرد استفاده می‌کنند اما می‌توانید از آن در تست‌های دیگر مانند تست بار و استرس استفاده کنید. این ابزار تمامی بار برنامه را روی سرور یا شبکه وارد می‌کند تا عملکرد آن‌ها را مورد تست قرار دهد. در انتهای تست، تجزیه و تحلیل نتیجه تست با استفاده از گزارش گیری‌های متنوع ارائه می‌شود. عناصر جی میتر را می‌توان در تست عملکرد و تست فانکشنال به وسیله TestPlan یکپارچه کرد به این صورت که جی میتر به شما اجازه میدهد که با همان TestPlan فانکشنال، تست بار را هم انجام دهید. این کار انعطاف

پذیری جی میتر را نشان می‌دهد که برای پروژه‌های تست بسیار کارآمد می‌باشد. این برنامه قابل حمل بوده و نیازی به نصب نیست (نیک کی،[۷] ۲۰۰۸: ۲۶).

تست استرس (Stress Testing)

طراحی محیطی مخرب تر از محیطی که برنامه در دنیای واقعی و در شرایط نرمال با آن روبرو میشد، است.

معرفی ابزار

در این بخش هم ابزار Load UI را معرفی می‌کنیم .

Load UI

این ابزار به صورت متن باز(Open source) بوده و برای برنامه‌های تحت وب به کار میرود و در نوع خود انعطاف پذیرترین ابزار تست استرس است. این ابزار به شما اجازه می‌دهد در حین تست کردن، تست‌ها را بسازید، به روز یا پیکربندی کنید. از دیگر ویژگی‌های این ابزار گزارش‌های متنوعی است که پس از تحلیل تست، به شما ارائه می‌دهد.همچنین هنگام به روز رسانی نیازی نیست که برای اعمال تغییرات آن را بسته ودوباره باز کنید، بلکه به صورت خودکار به روز رسانی بر روی آن انجام می‌شود. امروزه با پیچیده شدن پروژه‌های نرم افزاری و رسیدن ضرب الاجل پروژه به زمان تعیین شده، بهترین راه تیم تست برای تست برنامه‌ها، ابزارهای موجود است. با استفاده از این ابزارها، تیم تست زمان بیشتری برای تمرکز بر روی جنبه‌های دیگر برنامه داشته و خطاهای بیشتری را کشف می‌کند. علاوه براین با این ابزارها می‌توان در هر زمان و به هر تعداد دفعات تست را اجرا کرد .

معیارهای تست پذیر بودن نرم افزار:

۱. قابلیت اجرا (Operability): هرچه نرم افزار بهتر کار کند و در محیطهای بیشتری قابل اجرا باشد، n بهترقابل ارزیابی است
۲. مشاهده پذیری (Observability): قابلیت مشاهده نتایج ارزیابی
۳. کنترل پذیری (Controlability): قابلیت اجرای تست‌های خودکار (مثل امکان اجرای خودکار تست‌های واحد توسط jUnit برای زبان جاوا
۴. تجزیه پذیری (Decomposability): ارزیابی می‌تواند هدفمند تر شود
۵. سادگی (Simplicity): کاهش پیچیدگی معماری و منطق برنامه
۶. پایداری (Stability): برای ارزیابی تغییرات کمی بخواهد

[7] Nikkei

۷. درک پذیری (Understandability): قابلیت درک طراحی و وابستگی‌های بین اجزا

تست خوب (goodTesting)

اما یک تست خوب باید چگونه باشد؟ در واقع چگونه باید تست‌ها را طراحی کنیم تا با کمترین مشکلات در نگهداری، پشتیبانی و توسعه برنامه مواجه شویم. در این بخش پارامترهای یک تست خوب را بیان می‌کنیم:

- **سرعت** : تست‌های کند تست‌هایی هستند که اجرای آنها زمانبر است. این تست‌ها زمان بیشتری از برنامه نویس می‌گیرند و در نتیجه برنامه نویس رغبت زیادی به اجرای آن‌ها ندارد و به ندرت اجرا می‌شوند؛ و اگر تست‌ها اجرا نشوند پس خطاها نیز کشف نخواهند شد.
- **استقلال**: تست‌ها باید از یکدیگر مستقل باشند. این بدان معناست که تست‌ها نباید به وجود یکدیگر وابسته باشند. گاهی ممکن است یک تست را در دل تست دیگری اجرا کنیم یا از کلاس تست دیگری که قبلاً تولید کردیم ارث بری کنیم؛ این کار کاملاً اشتباه است؛ زیرا اگر تست والد را تغییر دهیم، قاعدتاً تست فرزند نیزدرچار تغییر می‌شود، و این کار توسعه تستها را دچار اختلال می‌کند.
- **قابل تکرار:** تستی که تولید می‌کنیم در هر زمان و مکان و در هر موقعیتی باید قابل اجرا باشد .

برخی اوقات ممکن است تست‌هایی را تولید کنیم که تنها در شرایط خاصی قابل اجرا باشند. برای مثال در حالتی که Connection String یک سری خصوصیات خاص را در خود داشته باشد. در این حالت باید قبل از اجرای این تست Connection String را نیز تنظیم کنیم. برای حل این مشکل بهتر است یک سری متد تست را قبل و بعد از اجرای تست، اجرا کنیم. بسیاری از فریم ورک‌های تست این حالت را تعریف کرده اند. برای مثال،جاوا خصوصیات @Before و @After را برای آماده سازی و بازسازی تستها دارد .

اما گاهی اوقات ممکن است یک سری شرایط خاص را برای برنامه خود در نظر بگیریم. مثلاً اینکه می‌خواهیم بدانیم برنامه ما در حالت قطع شبکه به چه صورت کار خواهد کرد. در این صورت باید از متدهای fake و شبیه ساز استفاده کنیم. ویژوال استودیو از نسخه 2012 به خوبی این کار را انجام داده .

Self Validate : تستها باید در صورت موفقیت آمیز بودن، مقدار True و در غیر اینصورت مقدار False را بازگردانند؛ فقط همین، نه بیشتر و نه کمتر؛ متدهای تست نباید متن و گزارش بازگردانند. تنها نوع داده ای که این متدها بازگشت می‌دهند نوع داده bool است. این امر باعث می‌شود چک کردن متدها راحت تر شود. به این صورت که متدهایی که با موفقیت، و بدون مشکل اجرا می‌شوند سبز شده و متدهایی که خطایی را میابند قرمز می‌شوند.

تست‌ها باید به موقع باشند: تنها زمانی که به تست نیاز داشته باشیم آنرا تولید می‌کنیم. این اساس برنامه نویسی TDD است. هیچ وقت متد تست اضافه ای ایجاد نکنید. این امر از شلوغ کاری جلوگیری میکند.

تک وظیفه ای : تک وظیفه ای یعنی اینکه یک متد تست تنها باید یک چیز را تست کند. اگر متدها را به این صورت طراحی کنید؛ هر متدی که با شکست مواجه شود، دقیقا به مکانی اشاره خواهد کرد که مشکل درآن ایجاد شده. این کار باعث میشود پیدا کردن منبع خطا راحت تر شود (رومانلی و همکاران،[8] ۲۰۰۳: ۱۵).

تجزیه و تحلیل رویکردهای تست مبتنی بر مدل تجزیه وتحلیل کمی

نتایج آماری بر روی سطوح تست که رویکردهایی برای آنها به کار گرفته شده است، سطح خودکارسازی برای هر رویکرد و مدل‌هایی که برای توصیف رفتار نرم افزار استفاده شده است و ابزارها و مدل میانی در این بخش مورد بررسی قرار می‌گیرد. این اطلاعات ممکن است سناریوهایی را ارائه دهد که در مهندسی نرم افزار به وسیله رویکردهای مبتنی بر MBT پوشش داده نشده است.

سطح تست

این بخش محدوده کاربرد رویکردهای MBT را نشان می‌دهد (جدول۱).

جدول (۱) تجزیه و تحلیل سطح تست (کیتچنهام، ۲۰۰٤)

سطح تست	رویکردها	ابزار پشتیبانی	
		استفاده	ذکر نشده
سیستم	48	32	12
یکپارچه سازی	17	7	10
واحد/ مؤلفه	8	5	3
رگرسیون	4	2	2

عمدتاً MBT برای تست سیستم به کار برده شده است(66٪ از رویکردها)چنان که MBI به طور کلی برای تست سیستم در نظر گرفته شده است. در نتیجه رویکردهای MBT برای سایر سطوح یکپارچه سازی تست (22٪)، تست رگرسیون (5٪) و تست واحد (10٪) به کار گرفته شده است. تست واحد یک سطح انتزاع معمول برای MBT نیست. هدف از تست واحد، تست کردن ماژول‌های کوچک، توابع یا کلاس‌ها پس از پیاده سازی است. رویکردهای دیگر، به منظور پشتیبانی از تست ساختاری از منابع کد بهتر می‌باشند.

[8] Romanelli, et all

سطح خودکارسازی

نسبت مراحل خودکارسازی شده که یک رویکرد را تشکیل می‌دهد و سطح پیچیدگی مراحل غیر اتوماتیک، به تجزیه و تحلیل سطح خودکارسازی مربوط می‌شود. یک مرحله غیر اتوماتیک در رویکردهای خودکار می‌تواند نسبت به چندین مرحله دستی غیر اتوماتیک در رویکرد دیگر MBT سخت تر باشد. مراحل غیر اتوماتیک دارای سطوح پیچیدگی مختلفی است که وابسته به مدل سازی رفتار نرم افزار(هر رویکرد به اجرای این وظیفه نیاز دارد)، انتخاب معیار تست، مدل سازی مدل میانی و یا ترجمه از یک مدل به مدل دیگر است.

Low: مرحله‌ها ممکن است خودکارسازی شده و یا وظایف ساده ای شامل شوند.

Medium: به عنوان مثال مدل سازی مدل میانی و یا تعریف داده تست

High: به عنوان مثال، ترجمه دستی بین مدلها به طور معمول رویکردهای MBT بیشترین مراحل خودکار سازی دارا هستند به استثنای مرحله اول که مدل سازی رفتار نرم افزار می‌باشد.

ابزارهای پشتیبانی

نظریه‌های MBT دارای ابزارهایی به منظور پشتیبانی اجرای مراحل آنها می‌باشد(64٪) در درجه اول به برای رویکردهای تست سیستم به کار گرفته می‌شوند.(جدول۲) برآورد دقیق تعداد ابزارهای پشتیبانی فرایند تولید موارد تست سخت است زیرا با توجه به تعداد ابزارهای به کار گرفته شده، و ابزارهایی که به طور اختصاصی و بدون هزینه توسعه داده شده اند بسیار زیاد است.

جدول(۲). تعداد رویکردها برای مدل(کیتچنهام،۹ ۲۰۰٤)

رفتار مدل	رویکردهای MBT
نمودارحالت	27
نمودار کلاس	19
نمودار توالی	19
نمودار موارد استفاده	11
Ocl	11
ماشین حالت محدود (گسترش یافته)	10
نمودار فعالیت	9

[9] Kitchenham

نمودار همکاری	8
نمودار هدف	7
گراف	7
ویژگیهای z	4
...	...

مدل استفاده شده برای تولید موارد تست

ما تفاوت بین رویکردهای MBT را بیان می‌کنیم که از مدلهای UMIL و غیر UML استفاده کرده اند. نمودار وضعیتUML (27 رویکرد)، کلاس(19) و توالی(19) مدل‌هایی هستند که بیش تر مورد استفاده قرار می‌گیرد. رویکردهایی که از UML استفاده نمیکند شامل ماشین حالت محدود است(7 رویکرد) و مشخصه z (4 رویکرد) است.

هزینه و پیچیدگی به کارگیری رویکردهای تست مبتنی بر مدل

هزینه و پیچیدگی تلاش به کارگیری یک رویکرد MBT مورد تجزیه و تحلیل قرار گرفته است. سطوح پیچیدگی بر اساس ویژگی‌های زیر تعریف شده اند:

- بالا: مراحل دستی، بدون هیچ ابزار پشتیبانی، ترجمه بین مدل‌های مورد نیاز، ابزار کد، و خروجی ناقص
- متوسط: نمادهای پیچیدگی مدل، تلاش‌های اضافی به منظور مدل سازی (مدل میانی) و استفاده از ابزارهای اختصاص
- پایین: استفاده از مدل‌های شناخته شده(UML،B ,FSM)، کاملاً خودکار، ابزارهای در دسترس، خروجی‌های تولید شده با فرمت‌های شناخته شده، وارزیابی تجربی تجزیه و تحلیل دقیق، نمودارها و صفحات گسترده ی تمام بررسی‌های رویکردهای MBT در وجود می‌باشد (پاریسس واوبادلسام،[10] ۱۹۹۶: ۱۲۷).

محدودیت‌های راه حل‌های تست مبتنی بر مدل

دامنه کاربرد، فرمت‌های ورودی، خروجی تولید شده و زبان مدل سازی ناحیه ی استفاده از رویکردهای MBT محدود می‌کند. پیش از به کار گیری یک رویکرد MBT برای یک پروژه نرم افزاری، نیازمندی‌های آن، یعنی اطلاعاتی مورد نیاز برای استفاده از آن چیست و آیا توسط پروژه نرم افزار تأمین می‌شود، باید به صورت قطعی و مسلم باشد. محدودیت‌های دیگر شامل

[10] Parissis, F. Ouabdesselam,

ویژگی‌های کیفی نرم افزار می‌شود که این رویکرد قادر به ارزیابی آن است یا سطح مهارت برای استفاده از آن مورد نیاز است.

ویژگی‌های کیفی نرم افزار

یک برنامه کاربردی مستلزم تست هر دو ویژگی عملکردی و کیفیت یا نیازمندی‌های غیر عملکرد(NFR) نظیر کنترل دسترسی، کارایی، ایمنی، قابلیت استفاده، و قابلیت اطمینان می‌باشد:

- کاربران از یک الگوی خاصی یکسان در زمان اجرای برنامه کاربردی استفاده نمی‌کند. رفتار متناقض برای مدل به منظور استفاده برای تولید موارد تست مشکل است.
- مشخص نیست که چگونه خصوصیات کیفی نگاشت نشده توسط رویکردهای MBT مثل: مانند جریان شبکه، قابلیت استفاده نرم افزار، نیازمندی‌های امنیتی را مدل کرد؟

سطح مهارت مورد نیاز برای استفاده از یک MBT

دانش نمادهای مدل سازی نرم افزار، معیار تست، متریک‌های تست و یا زبان‌ها برای تولید اسکریپت‌های تست، الزامات مهارتی یک تست کننده می‌باشند. رویکردهای مختلف نیاز به مهارت‌های مختلف داشته و آنها پیش از استفاده مورد نیاز می‌باشند.

مسئله اصلی این است که چگونه می‌توان مهارت انسانی مورد نیاز را به حداقل رساندن و به طور همزمان سطح خودکارسازی را به حداکثر رساند.

ارزیابی تجربی راه حلهای تست مبتنی بر مدل

در رویکردهای MBT فقدان نتایج تجربی به چشم می‌خورد. در (جریستو و همکاران،[11] ۲۰۰۴: ۳۶) نویسنده سطح بلوغ دانش رویکردهای تست نرم افزار را از مطالعات تجربی صورت گرفته از سال 1979 تا 2004 را مورد تجزیه و تحلیل قرار داده است. نتایج به دست آمده یک فاصله تکنولوژیکی را نشان می‌دهد، و تکنیک‌های تست خودکار به ندرت به صنعت انتقال یافته اند.

مطالعات تجربی بایدخصوصیات زیر را نشان دهد:

- خطرات و اثرات انتقال یک رویکرد از محیط دانشگاهی به محیط‌های صنعتی؛
- اثر بخشی نتایج آن و تجزیه و تحلیل پوشش آن

[11] Juristo et all

- چگونگی سهولت در به کار گیری درون یک پروژه نرم افزاری واقعی
- تلاش، زمان و هزینه استفاده از آن
- در چه زمینه ای می‌توانند استفاده شوند و محدودیت‌های آن چیست.

سیستم عامل اندروید

در چند سال اخیر، ما شاهد پیشرفت فوق العاده سریع تکنولوژی اپلیکیشن‌های موبایل بودیم. اپلیکیشنها، مانند همه نرم افزارها باید تستهای کافی را برای اطمینان از سالم بودنشان پشت سر بگذارند.

به چند دلیل اکثر تلاش محققان در این زمینه، برای دستگاه‌های اندرویدی بوده است. اولین و مهمترین دلیل اینکه اندروید در حال حاضر گسترده ترین سیستم عامل موبایل در سطح جهان محسوب می‌شود و همین نکته محققان را وادار می‌کند که اکثر تحقیقات خود را پیرامون این سیستم عامل انجام دهند. دوم، اپن سورس بودن سیستم عامل اندروید و تکنولوژی‌های مربوط به آن، این سیستم عامل را به هدفی مناسب تر برای محققان تبدیل می‌کند، زیرا آنها می‌توانند به راحتی هم به اپلیکیشن و هم به سیستم عاملی که برنامه بر آن اجرا می‌شود، دسترسی داشته باشند. از طرفی دیگر، اپلیکیشن‌های اندروید به زبان جاوا نوشته می‌شوند. حتی اگر این برنامه‌ها توسط کامپایلر دالویک که تفاوت‌هایی اساسی با کامپایلر جاوا دارد، کامپایل شود، چارچوب‌هایی وجود دارند که می توانند این کدهای کامپایل شده توسط دالویک را به فرمت‌هایی قابل فهم انتقال دهند. همه این دلایل باعث شدند که اکثر تحقیقات انجام شده پیرامون آزمایش برنامه‌های اندرویدی انجام شود (اُکتئوا و همکاران،[۱۲] ۲۰۱۲: ۱۱).

مواد و روش‌های سیستم عامل اندروید

درابتدا توضیحی در مورد سیستم عامل اندروید می‌دهیم. اپلیکیشن‌های اندروید، بر روی سه لایه نرم افزاری دیگر اجرا می‌شوند. درلایه اول که سیستم عامل است شرایطی فراهم شده است که اپلیکیشن‌ها بدون درگیر شدن با جزئیات سطح پایین سیستم عامل، می‌توانند از امکانات آن بهره ببرند. در بخش زمان اجرا به عنوان لایه دوم نرم افزاری،Zygote daemon برای هر یک از برنامه‌های در حال اجرا، یک ماشین مجازی دالویک در نظر می‌گیرد، در واقع ماشین مجازی دالویک یک ماشین مجازی بر پایه رجیستر است که می‌تواند بایت کدهای دالویک را ترجمه کند. در پایین ترین سطح نرم افزاری اندروید، هسته لینوکس 3 قرار دارید که اصلی ترین قابلیت سیستم را فراهم می‌کند. مجموعه ای از کتابخانه" کدهای بومی "مانند WebKit و SSL که مستقیما با کرنل در ارتباط است و نیازهای اساسی سخت افزارِ لایه زمان اجرا را پاسخ می‌دهد.

[12] Octeau et all

اپلیکیشن‌های اندروید معمولا در جاوا نوشته می‌شوند، کدهای جاوا ابتدا به بایت کدهای جاوا کامپایل شده و سپس به بایت کدهای دالویک ترجمه می‌شوند. در نهایت در یک فایل که قابلیت اجرا بر روی ماشین مجازی را دارد، با فرمت dex. ذخیره می‌شوند. اپلیکیشن‌ها در نهایت با فرمت apk منتشر می‌شوند که از فشرده سازی پوشه‌هایی شامل فایل‌های dex .کدهای بومیو منابع اپلیکیشن به وجود آمده است. اپلیکیشنهای

اندروید، در فایل AndroidManifest.xml اجزای اصلی خود را تعریف می‌کنند، که بر چهار قسم هستند:

۱. فعالیت‌ها شامل اجزای رابط کاربری می‌باشند. هر فعالیت پنجره ای است شامل بخش‌های مختلف رابط کاربری، مانند دکمه‌ها و تکست باکس‌ها. برنامه نویسان می‌توانند رفتار هر فعالیت را با اجرای مناسب توابع هر فاز کنترل کنند(فازها شامل :ساخته شدن، وقفه، از کار انداختن و نابود شدن) فعالیت‌ها در مقابل هر حرکت کاربر(مثل کلیک کردن) عکس العمل نشان می‌دهند که همین نکته باعث می‌شود به هدف اصلی برای آزمایشات اندروید، تبدیل شوند.
۲. خدمات اجزایی هستند که می‌توانند فرآیندهایی که زمان طولانی برای اجرا نیاز دارند را در پس زمینه نرم افزار اجرا کنند. بر خلاف فعالیت‌ها، آنها ارتباط زیادی با کاربر ندارند و بنابراین معمولا جزء اهداف آزمایشی اپلیکیشن‌ها نیستند، اگرچه در برخی نقاط به طور غیر مستقیم مورد آزمایش قرار می‌گیرند.
۳. دریافت کننده‌ها و خواسته‌های برادکست به برنامه اجازه می‌دهد به صورت همروند کارها را انجام دهد و در یک لحظه به چند کاربر پاسخ بدهد. برنامه‌ها با ثبت دریافت کننده‌های برادکست، می‌توانند از رویدادهای خاص سیستم مطلع شوند. برنامه همچنین می‌توانند برای مثال هنگام دریافت یک پیام کوتاه جدید، یا وصل شدن به یک مودم توسط وای فای و یا برقرار شدن تماس تلفنی، عکس العمل نشان دهند .برای درست تست شدن یک برنامه، ابزارهای آزمایش باید قادر باشند که دریافت کننده‌های مرتبط را بشناسند و بر آن اساس خواسته‌های درست را انتخاب کنند.
٤. تولید کنندگان محتوا مانند رابط‌های ساخت یافته رفتار می‌کنند تا فضاهایی برای ذخیره اطلاعات ارائه دهند، مانند پایگاه داده مخاطبین و یا تقویم .برنامه‌ها ممکن است تولید کنندگان محتوای مخصوص به خود را داشته باشند و ممکن است آن را در معرض استفاده دیگر برنامه‌ها نیز قرار بدهند .مانند همه نرم افزارها، رفتار یک اپلیکیشن رابطه مستقیمی با حالت تولید کننده محتوای آن دارد .در نتیجه، ابزارهای تست باید با کپی برداری از تولید کنندگان محتوا، پوشش قوی تری به رفتارهای احتمالی یک اپلیکیشن بدهند (چودهری و همکاران،[۱۳] ۲۰۱۵: ۱۳).

[13] Choudhary et all

ابزار Monkey یکی از پر استفاده ترین روش‌های تست اپلیکیشن‌های اندروید است. این ابزار، اپلیکیشن تحت آزمایش را یک جعبه سیاه در نظر می‌گیرد و تنها می‌تواند رویدادهای رابط کاربری را تولید کند. کاربر باید تعداد رویدادهایی که می‌خواهد Monkey تولید کند را مشخص نماید. هنگامی به این تعداد مشخص برسد Monkey متوقف می‌شود(چودهری و همکاران، ۲۰۱۵: ۱۳).

جدول (۳) شامل همه این ابزار می‌شود، این جدول گزارش می‌دهد که آیا یک ابزار به طور عمومی در دسترس است یا خیر چه رویدادهایی را تحت پوشش قرار می‌دهد، رویدادهای سیستمی یا رویدادهای رابط کاربری؟ آیا این ابزار به سورس کد برنامه در حال تست نیاز دارد یا خیر؟ ودرنهایت استراتژی تست برای این ابزار چیست؟

جدول (۳) نگاهی کلی بر ابزارهای تولید ورودی تست برای اپلیکیشن‌های اندرویدی (چودهری و همکاران، ۲۰۱۵).

نام ابزار	دسترس پذیری	رویداد		نیاز به سورس کد	استراتژی تست
		سیستمی	رابط کاربر		
1.Monkey	✓	×	✓	×	جعبه سیاه
2. Dynodroid	✓	✓	✓	×	جعبه سیاه
3.GUIRipper	✓	×	✓	×	جعبه سیاه
4.A3E	✓	×	✓	×	جعبه سیاه
5.SwiftHand	✓	×	✓	×	جعبه سیاه
6.ACTEve	✓	✓	✓	✓	جعبه سیاه

جدول (۴) راحتی در استفاده و سازگاری ابزارها با محبوب ترین نسخه های اندروید(چودهری و همکاران، ۲۰۱۵).

نام ابزار	راحتی در استفاده	سازگاری با نسخه اندروید
Monkey	بدون دخالت کاربر در استفاده	همه ورژنها
Dynodroid	بدون دخالت کاربر در استفاده	ورژن 3.2
GUIRipper	دخالت زیاد کاربردر استفاده	همه ورژنها
A3E	دخالت کم کاربردر استفاده	همه ورژنها
SwiftHand	دخالت زیاد کاربردر استفاده	ورژن 1.4 به بالا
ACTEve	دخالت زیاد کاربردر استفاده	ورژن 3.2

از نظر پوشش کد به طور میانگین، Monkey و Dynodroid بهتر از سایر ابزارها عمل کرده اند و ACTEve نیز پس از آنها قرار دارد .سه ابزار دیگر سطح پوشش کدی پایین از خود به جا گذاشتند. هدف اصلی ابزارهای ورودی تست، پیدا کردن خطاهای موجود در اپلیکیشن تحت آزمایش است .بنابراین، سَوای پوشش کد، خطاهایی که هر یک از ابزارها، در طول یک ساعت از هر اپلیکیشن پیدا کنند را نیز درنظر می‌گیریم.هیچ یک از این ابزارها نمی‌توانند خطاهایی غیر از خطاهای زمان اجرا را پیدا کنند. Swift Hand در این ارزیابی، بدترین ابزار شناخته شد.

نتیجه گیری

افزایش نیاز به محصولات نرم افزاری باعث می شود تا موضوعات کیفیت و قابلیت اطمینان نرم افزارها نیز روز به روز اهمیت زیادی پیدا کنند. منظور از تست نرم افزار نیز این است که بدانیم آیا محصول تولید شده درست و مطابق با نیازهای کاربر است یا خیر. تست نرم افزار با اشکال زدایی نرم افزار متفاوت است. اشکال زدایی، اشکالات نرم افزار را رفع می کند در حالی که تست، اشکالات نرم افزار را شناسایی می کند. چرخه تست نرم افزار شامل فرایندهای تحلیل نیازها، تحلیل طراحی، طراحی تست ، اجرای تست، گزارش تست است. تست نرم افزار به دو صورت ایستا و پویا انجام می شود. در روش تست پوبا برنامه را اجرا کرده و سپس نتایج ان ها تحلیل می شوند. تست ایستا تحلیل متن برنامه بدون اجرای ان است. مزیت تست ایستا این است که به تیم برنامه نویسی بازخورد دارد. چون عمل تست در وسط فرآیند توسعه اتفاق می افتد.

روش های مختلفی برای تست یک نرم افزار وجود دارد که برخی از ان ها عبارتند از تست Black Box و تست White Box. در تست Black Box تست کننده به داخل سیستم تست شونده توجهی ندارد و هدف فقط دادن یکسری داده به عنوان ورودی و دریافت یکسری اطلاعات به عنوان خروجی است. در صورتی که این خروجی ها در رابطه با ورودی ها مورد انتظار باشند ان گاه نرم افزار صحیح است. اما تست White Box به کد منبع برنامه نیز دسترسی دارد و می تواند داده های تست را متناسب به کد برنامه تولید کند. تست White Box برخلاف تست Black Box نیاز به داشتن دانش برنامه نویسی دارد.

منابع

۱. ملک زاده، م، و شم سی، م و مبکی، م، و بالغی، ا..(۱۳۹۲). برر سی رویکرد تستهای مبتنی بر مدل و ارائه رو شی برای ت ست نرم افزارها، اولین همایش ملی برق و کامپیوتر جنوب ایران.

[2].Ali, Sh. And Hemmati, H. 2014. **Model-based Testing of Video Conferencing Systems**: Challenges, Lessons Learnt, and Results, IEEE International Conference on Software Testing Verification and Validation, 4,353-362.

[3].B. Beizer, **Software Testing Techniques. London**: InternationalThompson Computer Press,1990.

[4].B. Kitchenham, **"Procedures for Performing Systematic Review"**, Joint Technical Report Software Engineering Group, Department of Computer Science Keele University, UK, and Empirical Software Engineering, National ICT Australia Ltd, 2004

[5].D. Octeau, S. Jha, and P. McDaniel, **"Retargeting Android Applications to Java Bytecode,"** in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 6:1–6:11.

[6].De Cleva Farto, G. and Takeshi Endo, A., 2015. **Evaluating the Model-Based Testing Approach in the Context of Mobile Applications**, Electronic Notes in Theoretical Computer Science. 314.3-21.

[7].Di Francesco, A. and Esposit, P. 2012. **Software Testing of Mobile Applications Challenges and Future Research Directions**, IEEEAST 25, 29-35.

[8].Parissis, F. Ouabdesselam**, "Specification-based testing of synchronous software",** In: Symposium on the Foundations of Software Engineering (21), pp.127 – 134, 1996 .

[9].L. Copeland, **A Practitioner's Guide to Software Test Design Boston**: Artech House Publishers, 2004.

[10].N. Juristo, A.M. Moreno, S. Vegas, **"Reviewing 25 years of testing technique experiments"**. Empirical Software Engineering: An International Journal, 9, March 2004.

[11].Nikkei ITpro, **Tokyo Stock Exchange New Derivative Selling**/Buying System Restored, due to initialization error of the memory:http://itpro.nikkeibp.com .jp/article/NEWS/200802 12/293526.

[12].Romanelli, F.; Vaccari, F; Panza, G. F.; **Realistic Modeling of the Seismic Input: Site Effects and Parametric Studies**, Journal of Seismology and Earthquake Engineering, Fall 2003 Vol. 5, No. 3, Tehran, I. R. Iran.

[13].S. Choudhary, A. Gorla, A. Orso. Automated Test Input Generation for Android: Are We There Yet? ASE '15 Proceedings of the 2015 30th IEEE. 2015 November 09. 13.