TESZTELÉS A GYAKORLATBAN

A SZAKÉRTŐ TESZTELŐK LAPJA

2015/I. szám

Teszteljük a visszafejtett kódot

Kedves Olvasó!

Nem is olyan régen még csak elvétve lehetett hallani olyat, hogy egy vállalatnál a tesztelőket mennyire megbecsülik. Nem telt el olyan konferencia, rendezvény, vagy szakmai összejövetel, melyen az előadók ne szólaltak volna fel a témában. Lehet, hogy ez csak a tesztelői közösség kishitűségéből adódott, de mindenki arról beszélt, hogy a tesztelést mennyire lenézi a projekt többi szereplője.

Eltelt pár év és most ott járunk, hogy számtalan konferencián előjön a minőség és ezzel együtt a tesztelés kérdésköre. Maguk a fejlesztők, projektvezetők is rengetegszer előadnak a témában. Látszik, hogy mennyire fontos a minőség és ezáltal az Ügyfél elégedettsége.

Azt lehet észrevenni, hogy már a kisebb fejlesztő cégek is elkezdtek dedikált tesztelőket keresni, tesztelési folyamatot kiépíteni, eszközöket beszerezni. Az agilis módszerek elterjedésével egyre több projektben megjelentek a tesztelői szerepkörök, amelyek a csapatok fontos részévé váltak. Ez a nagyfokú figyelem a tesztelői szakma felemelkedését, egyben a tesztelők önérzetének javulását vonja maga után.

Én mindig is annak a híve voltam, amit manapság az agilis csapatok bizonyítanak, hogy nincs különbség az emberek munkája között. Egyik sem fontosabb a másiknál. Ha valamelyik szerepkört kivesszük a projektből – legyen az tesztelő, fejlesztő, szervező - máris elérhetetlen távolba kerül a siker. Vegyük észre, nem az egyének viszik a siker felé a projektet, csapatként kell leküzdenünk a kihívásokat!

Köszönöm!

Pongrácz János

IMPRESSZUM

Kiadja:
Acrobyte Informatikai Kft.

Szerkesztőség:
Alapító főszerkesztő
Pongrácz János
info@tesztelesagyakorlatban.hu

Kiadványszerkesztés, grafikai munkák:
Mogyoró Győző
Graphic Designer
gyozo.mogyoro@gmail.com

Internet:
www.tesztelesagyakorlatban.hu

Kapcsolat:
1149 Budapest,
Mexikói út 11/a
Telefon/Fax: 1/ 789 2525

Publikáció:
publikacio@tesztelesagyakorlatban.hu

Hirdetésfelvétel:
hirdetes@tesztelesagyakorlatban.hu

TARTALOMJEGYZÉK

A legjobb tesztelővé válás 4 egyszerű lépése

Ha a cikkben leírtakat betartjuk lehet, hogy nem a legjobb, de mindenképpen jobb tesztelővé válunk. Néhány egyszerű dolog kell csak ahhoz, hogy a szakmánkban fejlődjünk és előrébb jussunk. Amennyiben elkötelezettek vagyunk a fejlődésben, fogadjuk meg az itt leírtakat.

Manapság, hála a YouTube hirdetési szabályzatának, valahányszor videót vagy zenét akarok hallgatni, el kell tűrnöm, valami rettenetes reklámot pl.:„Helló, így tudod az életedet teljesen megváltoztatni 5 lépésben”-t. Borsózik a hátam az ilyenektől, mint „Mágikus titkok, a milliomossá válás felé...”, vagy „Még van 100 hely az előadásomra, tízezer dollárért!”. El kell ismernem, nekem is vannak ilyen jellegű gondolataim. Ebben a pillanatban azon gondolkodom, hogy én is tudnék ilyet a tesztelésben! Az egyetlen bökkenő az, hogy a teszteléssel kapcsolatos tanulmányok már tele vannak ilyenekkel és még egy hozzáadása nem sokat segít a tesztelői közösségen.

Felhagytam a napi gyakorlattal, hogy minden nap beszélgetek Gunával, a junior tesztelő kollégámmal, aki olyan jó tesztelő akar lenni, amilyen csak lehet. Csodálom az energiáját, mivel mára az egyik legaktívabb a közösségben amellett, hogy a projektjében is kulcsszerepe van. Mindig büszkeséggel tölt el, amikor Gunához hasonló tesztelő szakemberekre lelek. A Gunával történt beszélgetések közben azon gondolkodtam, mi lehet a folyamat, amitől valaki a legjobb tesztelővé válhat? Egy héten keresztül vázlatosan összeszedtem a lépéseket, ami listaként pofon egyszerűnek látszott. Párszor úgy látszott, hogy sikerül valami új elemet még hozzáadnom ehhez a listához, de mindig kiderült, hogy végül csak egy már meglévő elemhez jutottam vissza.

Nos, a következő négy egyszerű lépést szedtem össze, a legjobb tesztelővé váláshoz. Ezek nagyon egyszerű koncepciók, de a projekt komplexitásától függően ezek is bonyolultakká válhatnak. Ha minden nap ezen értékek szerint élünk, az sem garantálja, hogy briliáns „szupertesztelőkké” válunk, de azt ígérem, hogy ezek alapján jobb tesztelők leszünk, mint amilyenek tegnap voltunk.

Bővítsd a tudásod!

Az tény, hogy ma van valamilyen szintű tesztelői tapasztalatod, tudásod. Keress egy módszert arra, hogy ezt kiterjeszd! A legtöbb embernek ez rögtön azt jelenti, hogy keressünk egy tréninget, ahol majd megtanítanak, mert a tréningek nagyon hasznosak. Azonban vannak más módszerek is. Például keress egy teszteléssel kapcsolatos könyvet, olvass tesztelő blogokat, vagy magazinokat. Emellett ne feledkezz el a szocializálódásról sem! Az EDS-nél gyakran elmentünk közösen a tesztelőkkel szendvicsezni és közben a fociról, vagy a tegnapi TV műsorról beszélgettünk ebédidőben. Legtöbbször a teszteléssel kapcsolatos tapasztalatainkat is megosztottuk egymással ilyenkor: mit csináltunk, hogyan teszteltünk, milyen a tesztkörnyezet, stb. Ezek az információmorzsák 2 év alatt kézzelfogható eredményeket hoztak és természetesen a csapat nagy részével még a mai napig is tartom a kapcsolatot. Amikor teszteléssel kapcsolatos kérdéssel szembesülök, mindig az eszembe jutnak. Nagyon sok városban találkozhatsz úgynevezett „meetup” –okkal, ahol tesztelőkkel ismerkedhetsz meg és kiterjesztheted a kapcsolatrendszeredet, valamint néhány új és hasznos dolgot is tanulhatsz. Emellett nagyon sok hasznos dolgot találsz a TED-en vagy a YouTube-on is. Tedd fel magadnak a kérdést a munkahelyeden, vagy egy rendezvényen, hogy értelmes, értékes beszélgetést folytattál-e a tesztelésről éppen akkor!

Próbálkozz az újdonságokkal!

Ahelyett, hogy mindig azt hajtogatnád, hogy ezt így szoktuk, keress új ötleteket a megoldáshoz. Ez nem azt jelenti, hogy mostantól úgy dolgozz, ahogy előtte sohasem, és hogy olyan megoldásokat szállíts, amik kritikusak lehetnek a projektre nézve! Találd meg a módját és próbálj ki új dolgokat úgy, hogy ez ne okozzon problémát a projektben. Még abban az esetben sem, ha véletlenül nem úgy menne, ahogy azt eredetileg gondoltad volna.

Legyél felkészülve arra, hogy ha módosítani kell az eredeti elgondolásokat, akkor át tudd ugrani a felmerülő akadályokat. Az eddigi tapasztalataid, valamint a tesztelő kollégáid tapasztalatai alapján kérdezd meg magadtól, hogy miben más ez a módszer a jelenleg használt módszernél? Miben segíti jobban a munkádat? Hogyan fogod majd elmondani a többieknek, hogy miként működik? Ez vezet el a következő ponthoz…

Figyelj a visszajelzésekre!

Ha újdonságokat használsz, előfordulhatnak váratlan dolgok – amit előre nem vártunk, nem számítottunk rá, éppen ezért váratlanok. Sokszor az elgondolások alapvetően jók szoktak lenni, nem kell mást tenned, csak vissza kell lépni egyet, hogy lásd az egész képet egyben. Így észrevehetsz a folyamatban apróbb hibákat, amit kijavíthatsz. Ne csak a saját megfigyeléseidre támaszkodj, kérdezd meg a munkatársaidat, a csapattagokat is! Remélhetőleg a jelenlegi csapatodban olyan emberekkel dolgozol, akikben megbízol (amenyben nem így van, tedd fel a kérdést, hogyan tudnád kiépíteni ezt a bizalmat).

Kérj visszajelzést a projektben a többiektől a feladatokkal kapcsolatban. Talán nem tudnak túl sokat a tesztelésről? Ez egy jó alap lehet, hogy röviden elmeséld, miről is szól a tesztelés, mik a feladataid pontosan és hogyan illenek be a feladataid a projektbe. A gyakorlat és az elmélet gyakran különbözik, nem az történik, mint amit előre elgondolunk. Ne félj felírni ezt a tapasztalataid közé, mert lehet, hogy most látod csak meg a régi megoldásban lévő értékeket, amelyeket eddig nem is vettél észre! Lehet, hogy az elgondolásod a következő projektben már működni fog. Nehéz így eredményeket elérni, de rajtad múlik, hogy elég kitartó vagy-e!

Tesztelj, amit csak tudsz!

Ötvözd a filozófiát és a kritikus gondolkodást, amely az alapja a tesztelői mesterségnek. A mindennapi életünkben egy láthatatlan, feltételezésekből épített hajóban ülünk, amelyet sokszor észre sem veszünk. Néha-néha mégis nyisd ki a szemedet a körülötted lévő világra, és mint egy mentális feladat, kérdezd meg magadtól, hogy „Hogyan tudom igazolni azt, hogy jó irányban haladok a sok kétely ellenére?”. Érdemes talán azt is igazolnod, hogy „a Föld forog”, „a rózsák vörösek”, vagy „két különböző ember felismeri az édes ízt ugyanazon étel elfogyasztásakor”.

A szoftver és a tesztelés a feltételezések hajóján épült, ugyanazon feltételezések alapján, mint hogy a „Föld forog”, és mi csak elfogadjuk ezeket a kérdés feltétele nélkül: Milyen közvetlen bizonyítékot láttam valóban rá? Néha azonban mentálisan biztosnak kell lennünk, mert nem minden feltételezés elfogadható. Rengeteg projekt bukott már bele ilyen feltételezésbe! A kritikus gondolkodás elősegíti a feltételezések elfogadását (vagy elutasítását) a sikertelenségek, vakvágányok elkerülése érdekében. Elengedhetetlenül szükséges ez a fajta kritikus felfogás az élet más területein is, mivel a világban olyan sok a téves információ! ■

Szerző: ***Mike Talks***

Forrás:
http://www.testingcircus.com/four-simple-steps-to-becoming-the-best-tester-you-can-be/

Mike Talks

Bár eredetileg Mike Talks nem új-zélandi, mégis úgy érzi, hogy Wellington az egyik legizgalmasabb hely a világon, ahol szoftvertesztelés lehet foglalkozni. Mike elment a Hobbit film válogatására is, de azt mondták neki , hogy túl magas, hogy egy hobbit szerepét el tudja vállalni (6’ 2”), pedig még le is hajolt. Ő a szerzője a Leanpub tesztelői könyvnek, melynek címe „A szoftver aknamező”.

Együtt a manuális és az automata tesztelés

Sok vállalatnál hallottam már azt a kérdést, hogy „Automatizált vagy manuális teszteléssel oldjuk meg a feladatot?” A másik amit még szívesen kérdeznek, hogy „Az automata teszteléssel mennyi manuális tesztelői erőforrást tudunk felszabadítani?” Így nem csoda, ha rengeteg erőfeszítésbe telik, amíg a manuális és automata tesztelés együttes használatáról meg lehet győzni az embereket.

A szoftvertesztelés történelmére visszatekintve az automata tesztelés nem egy új keletű dolog. Tény, James Bach fogalmazta meg, hogy az automatizált tesztelés megelőzte a dedikált tesztelő koncepcióját.

„Az automatizálás nem teljesen új dolog. Összehasonlítva, ami viszonylag új, az a dedikált tesztelő. A negyvenes években a dedikált tesztelő gyakorlatilag ismeretlen fogalom volt. A szoftvereket a fejlesztők tesztelték. A hatvanas években a tesztelésről szóló dokumentációk legtöbb esetben arról számoltak be, hogy a programozók tesztelik az általuk fejlesztett alkalmazásokat, ami nem igazán volt megkülönböztethető a debuggolástól. A nagyobb és komplexebb rendszerek megjelenésével jelentek meg a dedikált tesztelői kompetenciák, a dedikált tesztelők.”

Extrém gondolkodás

És mégis, annak ellenére, hogy az automatizált tesztelés nem egy új dolog, nagyon fontos témává vált a fejlesztői stratégiákban és metódusokban, mint az Agile és az ATDD/TDD.

A párbeszéd egyre kiélezettebb. Azok, akik az automatizált teszteléssel szemben állnak, azt hangoztatják, hogy az automatizált tesztelés a régi, jól bevált manuális tesztelés halálát jelentik. Sőt, a tesztelők és a tesztelés általában válik ettől feleslegessé. Azok, akik az automatizálás mellett kardoskodnak, a felderítő tesztelés létjogosultságát kérdőjelezik meg, szerintük az csak „játék”.

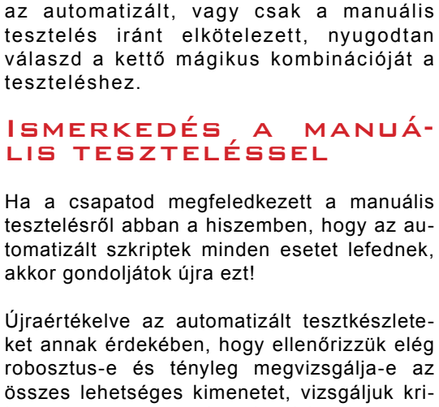
A kétfajta megközelítés hozta létre azt a felfogást, hogy a manuális és automata tesztelés nem létezik egymás nélkül, nincs automata tesztelés manuális tesztelés nélkül és fordítva. Nem mondhatjuk ki, hogy egyik, vagy a másik.

“És” a “vagy” helyett

Tehát „VAGY” helyett, ha igazán számít az alkalmazás minősége, ezentúl használjuk az „ÉS”-t.

Ne csak az egyiket értsük a tesztelés alatt. A manuális és automatizált tesztelés együttes használata sokkal értékesebb és eredményesebb mint az egyik vagy a másik külön-külön. Íme két példa:

- Az automatizált tesztelés a tesztelési körök sebességének a növelését segíti elő, de vegyük észre, hogy az automata tesztek azon ellenőrzéseket végzik el, amelyeket definiáltunk nekik, hogy ellenőrizzenek. A gyorsabb tesztelés rendben van, de a körültekintő tesztelés hiánya nagyon nincs rendben.



- Manuális tesztelés – különösen a felfedező teszt – nagyon sok kérdést megvilágít a felhasználói esetektől kezdve rengeteg olyan hibára rámutat, amelyre előtte nem is gondoltunk. A manuális tesztelés magas színvonalon való végzése igen időigényes és nagyon nagy odafigyelést igényel. Ha mindent meg akarsz csinálni manuálisan, sohasem fogsz végezni a teszteléssel.

A manuális és az automatizált tesztelés nem tudnak egymás nélkül létezni, ezek egymás kiegészítői.

Csináld magadnak

Ha a cég, ahol tesztelsz jelenleg csak az automatizált, vagy csak a manuális tesztelés iránt elkötelezett, nyugodtan válaszd a kettő mágikus kombinációját a teszteléshez.

Ismerkedés a manuális teszteléssel

Ha a csapatod megfeledkezett a manuális tesztelésről abban a hiszemben, hogy az automatizált szkriptek minden esetet lefednek, akkor gondoljátok újra ezt!

Újraértékelve az automatizált tesztkészleteket annak érdekében, hogy ellenőrizzük elég robosztus-e és tényleg megvizsgálja-e az összes lehetséges kimenetet, vizsgáljuk kritikus szemmel az automatizált tesztek eredményeit. Ne feledd, hogy a 100%-os arány nem jelent teljes hibamentességet! Azt is jelentheti, hogy az automatizált tesztek nincsenek jól megírva, vagy nem teljesen azt csinálják, amit kellene. Frissítsük az automata szkripteket úgy, hogy a legközelebb legyünk az elvárásokhoz.

Ha biztos vagy abban, hogy az automatizált készlet teljesen rendben van, itt az ideje a manuális tesztelésre koncentrálni. Végezzen a csapat olyan teszteléseket, amelyeket az automata tesztelés nem fed le és adjunk elég időt a tesztelőknek a felfedező tesztelésre is. Ütemezve, a fontossági sorrendet betartva rendben lesz a tesztelés.

Nagyon fontos, hogy a manuális teszteléssel a mixben, képesek leszünk funkcionálisan teljesen lefedni az alkalmazást.

Ismerkedés az automata teszteléssel

Amennyiben a csapat csak manuális tesztelést végez, hozzuk be az automata tesztelést a képbe!

Első lépésként ki kell választani a megfelelő automata teszteszközt, amely legjobban alkalmazkodik a tesztelendő alkalmazáshoz és a csapattagokhoz (tesztelők és fejlesztők) akik majd az automata teszteket készítik. Számtalan eszköz van a világon, így nagyon nehéz a legmegfelelőbb kiválasztása. A teszteszközök listája (mint pl. ez: uTest’s list of highest rated tools) egy jó kiinduló pont lehet. Amikor az eszköz megvan és az is megvan, hogy kik fogják majd létrehozni az automata szkripteket, biztosnak kell lennünk abban, hogy a csapatnak lesz ideje a szkriptek megírására.

Nyilvánvaló, hogy az automata szkriptek nem készülnek el egy éjszaka alatt, de elemzéssel figyelmesen létrehozva néhány automatizált szkriptet minden egyes sprintben, fokozatosan el lehet jutni a kívánt eredményhez.

Amikor az elkészült szkriptek rendszeresen futnak, rácsodálkozhatunk, hogy mennyire segíti néhány csapattag munkáját és hogy mennyivel körültekintőbben tesztelt alkalmazásunk lesz.

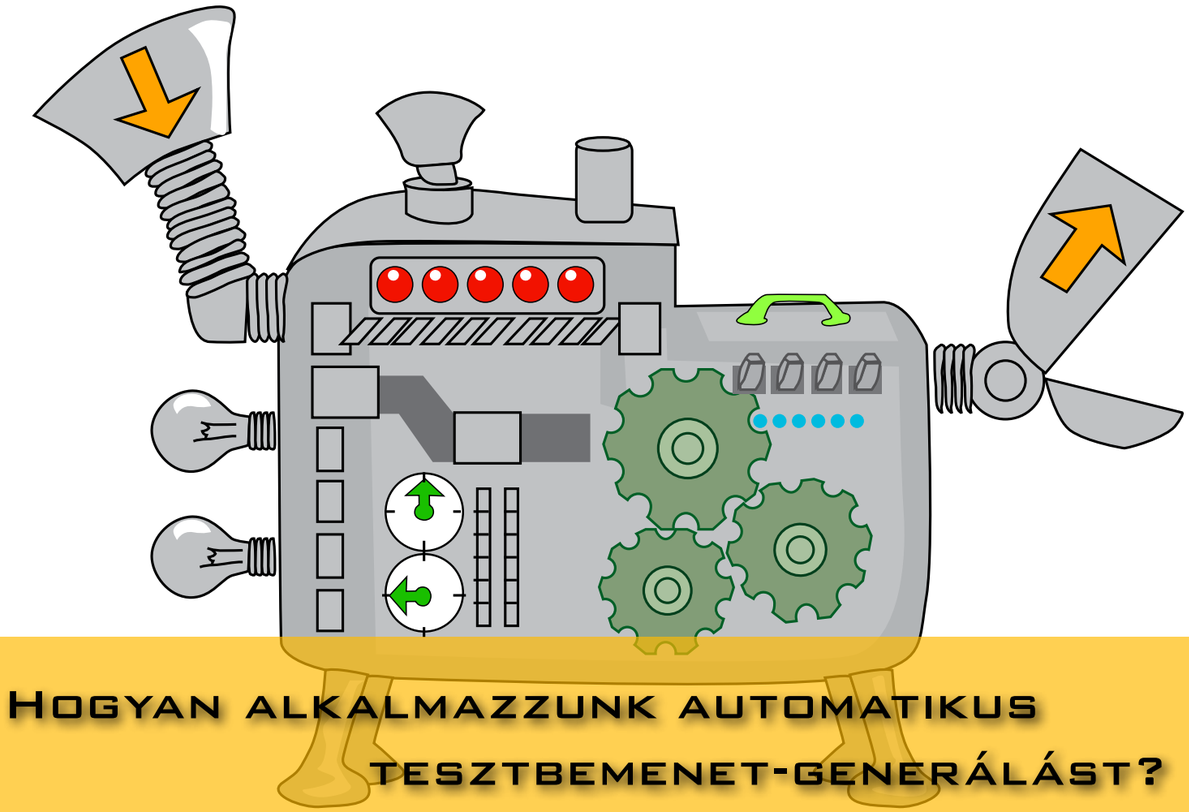
Hagyjunk fel a „VAGY” gondolattal, használjuk a manuális és az automata tesztelést is az alkalmazásunk minél magasabb színvonala és minősége érdekében! ■

Szerző: ***Cullyn Thomson***

Forrás:
http://qablog.practitest.com/2015/01/bringing-manual-and-automated-testing-together/

Cullyn Thomson

Cullyn Thomson önkéntes szerkesztőként kezdte meg munkáját 2005-ben. Az újság, amit szerkesztett az „USC Today Magazin” volt. 2008-tól 1 évet és 4 hónapot tevékenykedett ügyfél kapcsolattartóként. 2011-től 2 hónapig gyakornok volt a „Leech Tishman Fuscaldo & Lampl, LLC”-nél. Gyakornoki tevékenységei után tanulmányokat folytatott a Denison Egyetemen, ahol 3 évet és 10 hónapot töltött el. Az egyetem után 2 évet (2012-2014) technikai íróként és marketing munkatársként dolgozott a „Garnt Street Group”-nál. Cullyn jelenleg Projekt Menedzser a Telluriumnál.



Hogyan alkalmazzunk automatikus tesztbemenet-generálást?

Napjainkban egyre nagyobb népszerűségnek örvendenek az automatizált tesztelési módszerek mind az ipari felhasználás, mind az akadémiai kutatások esetén. A két terület között azonban láthatóan eltérő az automatizálási törekvés: míg akadémiai területen sok kutatás a tesztek előállításának automatizálását célozza meg, addig az ipari alkalmazások döntő részében az automatizálás a tesztek menedzselésére, futtatására törekszik.

Napjainkban egyre nagyobb népszerűségnek örvendenek az automatizált tesztelési módszerek mind az ipari felhasználás, mind az akadémiai kutatások esetén. A két terület között azonban láthatóan eltérő az automatizálási törekvés: míg akadémiai területen sok kutatás a tesztek előállításának automatizálását célozza meg, addig az ipari alkalmazások döntő részében az automatizálás a tesztek menedzselésére, futtatására törekszik. Jelen cikkben a két szféra közötti szakadék áthidalására történt erőfeszítéseim során nyert tapasztalatokat mutatom be.

Az akadémiai területet tekintve aktív kutatások folynak többek között az egységtesztelés terén is, hiszen a szoftverminőséget alapvetően meghatározó tesztelési szintről van szó. Az fejlesztési irányok azonban az egységtesztek előállításának vizsgálata felé mutatnak, ami magába foglalja a struktúra alapján történő származtatásukat is. Ebben az esetben a tesztelt szoftver forráskódja alapján történik a tesztek generálása, természetesen szem előtt tartva a fedettségi statisztikákat. Fontos kiemelni azonban, hogy a forráskód struktúrájának alapján nehézkes teljes teszteseteket származtatni, hiszen valójában nem vagy csak nehezen lehet megfogalmazni valós, elvárt kimeneteket.

A tesztbemenetek generálásáról

Az ide kapcsolódó kutatások tehát valójában tesztbemenet-generálással foglalkoznak, amelyekből előálló bemeneteket kiegészítve ugyanakkor tesztesetté is előléptethetünk. Bemenetek forráskód-struktúra alapján történő generálására több technika is létezik, amelyek közül az egyik legismertebb a szimbolikus végrehajtás. A módszer a kód végrehajtását egy magasabb absztrakciós szinten, szimbolikus változók segítségével végzi, a végrehajtás eredményeként pedig indirekten konkrét tesztbemenetek állnak elő.

A szimbolikus végrehajtás a működési elvéből fakadóan viszonylag sok kihívással küzd. A teljesség igénye nélkül ezek például a ciklusok kezelése, a mély bejárási tér, többszálúság ellenőrzése, környezeti interakció. Megoldásuk a legtöbb kapcsolódó kutatás témájaként szolgál jelenleg is, de a sok biztató eredmény ellenére vannak bizonyos tényezők, melyek maradéktalanul nem oldhatók meg.

A szimbolikus végrehajtást technikájának implementációjaként több eszköz is készült, habár az egyes eszközök képességei között meglehetősen nagy különbség lehet. A legismertebbek között említhető a NASA által fejlesztett Symbolic PathFinder, a Microsoft kutatórészlege által készített Pex, és a nyílt forráskódú projektként létező KLEE is. Lassan harmadik éve, hogy a Pex eszköz vizsgálatával foglalkozom, ugyanis a megvizsgált eszközök közül a Pex-et találtam a legfejlettebbnek a képességeiket összemérve. Ezt a feltételezést a későbbi tapasztalataim is alátámasztották.

A Microsoft Pex a szimbolikus végrehajtást parametrizált egységteszteken (PUT – Parameterized Unit Test) keresztül alkalmazza a tesztelni kívánt kódegységen. A PUT lehetőséget ad előfeltételek, elvárt kimenetek, izoláció és más tesztelési logika megvalósítására is. Az eszköz a PUT paraméterlistája alapján generál bemeneteket a végrehajtás során. Ezeket a bemeneteket a parametrizált tesztekbe helyettesítve konkrét tesztesetek kaphatók, melyek el is menthetők későbbi felhasználásra (1.ábra).

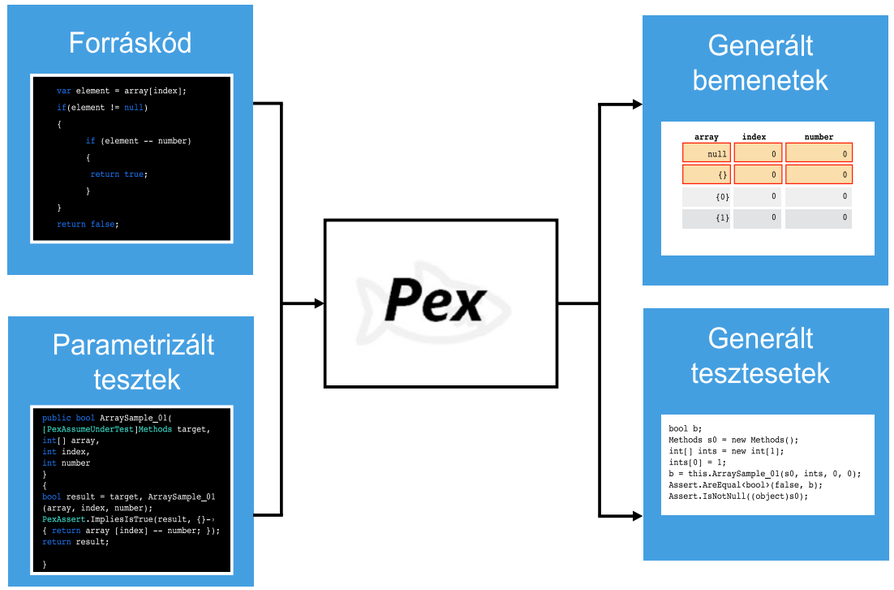
Tapasztalatok komplex rendszerekben

Az eszközt két komplex szoftverprojekten alkalmaztam tesztek előállítására a tesztbemenet-generálás segítségével. Az első egy összetett modellező eszköz volt, amelyet korai stádiumában vizsgáltam meg. A folyamat során feltérképeztem azokat a tényezőket, melyek befolyásolják a hatékony tesztgenerálást.

A Pex használatakor az első problémát az izoláció megvalósítása jelentette, ahol egységtesztelés lévén, mind a külső függőségeket, mind a fájlrendszer felé történő hívásokat le kellett választani. A komplex bejárási tér miatt azonban a hívások pontos, hely szerinti beazonosítása különösen nehéz feladat volt a bejárás átláthatatlansága miatt.

A másik probléma, amelybe beleütköztem a Pex használatának első néhány lépése során a komplex objektumok létrehozása és megfelelő állapotba történő beállítása volt. Ezt a Pex számára Factory minta segítségével lehet megadni, amelyre az egyszerű konstruktorokkal rendelkező osztályokon kívül a legtöbb esetben szükség is van. Ehhez kapcsolódóan érdemes említést tenni a dinamikusan, futási időben példányosított objektumok kezeléséről is, melyet a Pex nem tud kielégítően kezelni. A projekthez tartozó eredmények viszont bizakodásra adtak okot, hiszen az eszköz a vizsgált 136 metódushoz 371 teszteset generált, melyek körülbelül 99%-os kódblokk lefedettséget biztosítottak.

A felfedezett hibák között a következőket érdemes kiemelni: defenzív programozás hiánya (rosszindulatú bemenetek), inkonzisztens változtatások, kisebb funkcionális hiányosságok, illetve dinamikusan elérhetetlen kódrészletek. Ugyanakkor nem szabad



1.ábra

elfeledni, hogy a pusztán kódlefedettségen alapuló szoftverellenőrzés hatékonysága egyes esettanulmányok szerint megkérdőjelezhető lehet.

Az eredmények láthatóvá tették számomra a Pex és a szimbolikus végrehajtás komplex rendszerekben történő alkalmazási lehetőségeit, így egy ipari partner segítségével egy tartalomkezelő rendszer végleges változatához is készítettem teszteket. A forráskód mérete ebben az esetben már egy nagyságrenddel nagyobb volt (10.000 LOC), így az eszköz használata több előkészületet igényelt. A vizsgálat azonban felderített több olyan hiányosságot is, melyeket a meglévő tesztkészlet nem vizsgált. Közülük is érdemes említést tenni egy biztonságtechnikai problémáról, illetve egy olyan változókezelési gondról, mely az adatkezelési réteg és az üzleti logikai réteg között okozhatott kommunikációs zavart.

A két projektből nyert tapasztalatok rámutattak arra, hogy a Pex képes lehet ipari méretű alkalmazások esetén is olyan teszteket előállítani, melyek a tesztek manuális tervezése és implementációja során kimaradtak. A generált tesztbemenetek között azonban sok esetben irrelevánsak is voltak, melyek a valós működés során nem fordulhattak elő. Ezeket a szimbolikus végrehajtás számára előfeltételek segítségével lehet megfogalmazni, amellyel ezek a bemenetek kizárhatók a generálásból.

Érdemes alkalmazni? És ha igen, akkor hogyan?

Az előzőekben az eszköz használatából nyert, saját tapasztalataimat mutattam be. Felvetődik így a kérdés, hogy új felhasználók (például tesztelő mérnökök) hogyan alkalmazhatják a gyakorlatban, illetve, hogy egyáltalán érdemes-e alkalmazni? A kérdéssel a Microsoft is foglalkozik, de több másik szimbolikus végrehajtást alkalmazó eszköz esetén is már készültek kapcsolódó esettanulmányok. A saját tapasztalataim alapján a válasz egyértelműen igenlő, azonban a mérnökök általi hatékony használathoz véleményem szerint jól meghatározott folyamatra, lépésekre, illetve támogató eszközökre lehet szükség. Az igenlő választásomat az a tény is alátámasztja, hogy a Microsoft a Visual Studio 2015-ös verziójában már Smart Unit Tests néven integráltan szállítja a Pex eszközt.

Az eszköz hatékony használatához a tesztgenerálást a fejlesztéssel párhuzamosan javasolom. Az iteratív folyamat bevezetésével elkerülhetők azok a használat során felmerülő nehézségek, melyeket a tapasztalataink között is bemutattunk (pl. példányosítás, izoláció). Az alkalmazandó lépések a tesztelő mérnök számára a következők lehetnek egy agilis fejlesztési folyamat során.

1. Specifikáció: Interfészek kidolgozása a fejlesztőkkel közösen figyelembe véve a tesztelhetőségi szempontokat.

2. Definiálás: Parametrizált egységtesztek definiálása minden vizsgálandó egység számára az AAA (Arrange-Act-Assert) mintát követve, ezen kívül pedig az izolációs környezet megtervezése (kódstruktúra elemzésével).

3. Implementáció: Mockok és csonkok, illetve orákulumok létrehozása a specifikáció alapján.

4. Végrehajtás: Objektumokat előállító Factory metódusok létrehozása, majd Pex futtatása minden parametrizált egységteszten.

5. Iteráció: Eredmények visszacsatolása a fejlesztők számára, majd ismétlődően a Pex futtatása a módosított kódon és visszajelzés.

Összegzés

Általánosságban tekintve nem csak a Pex, hanem a többi szimbolikus végrehajtást implementáló eszköz is alkalmas lehet arra, hogy csökkentsük az esetlegesen előforduló hibák számát, ezáltal a forráskód minőségét magasabb szintre emeljük. Ezen kívül az eszközök által készített tesztkészlet felhasználható a későbbiekben regressziós célokra is, hiszen a generálódott tesztek elmenthetők és behelyezhetők például folytonos integráció folyamatába is. Összefoglalva tehát elmondható, hogy a tesztbemenet-generálását kísérleti szinten érdemes lehet kipróbálni, de a fejlesztési folyamatba történő integráció körültekintést és megfelelően előkészített környezeteket igényel. ■

Szerző: ***Honfi Dávid***

Honfi Dávid

Dávid a BME mérnök informatikus mesterképzésének végzős hallgatója. Immáron három éve foglalkozik szoftverek automatizált tesztelésével, azon belül pedig tesztgenerálással. Jelenlegi kutatásai során az akadémiai szférából származó eredmények mérnöki gyakorlatba történő átültetését vizsgálja.

Kezdjünk beszélgetni a teszteredmények ismeretében

A tesztcsapat és a fejlesztői csapat összehangolása, a megfelelő kommunikáció megteremtése rettentően fontos minden egyes projektben, kiváltképp igaz ez az agilis módszereket használó csapatoknál. Milyen technikák vannak arra, hogy ezt a közös csapatmunkát megteremtsük? Az együttműködésnek milyen pozitív hatásai vannak a projektre?

Vajon hányan dolgozunk Agile-ban ami nagyrészt követi az előírásokat? Vagy hányunknak van tapasztalata olyan Agile-ban, ami egy hibrid, vízesés modellhez közelít?

A legutolsó tapasztalatom egy ilyen. A projekt, amin dolgoztam, vízesés modellen alapult. Kezdetben mérföldkövekkel, ismétlődő ciklusokkal dolgozunk, de elmozdultunk az Agile felé és a leadáshoz közeledve kéthetes sprintekre tértünk át. A legnagyobb kihívás a két modell összehangolásában a tesztcsapat és a fejlesztői csapat közötti „ők és mi” mentalitás felszámolása volt. Ennek leküzdése érdekében a teszteredmények ismeretében kezdtünk egyeztetni a további munkáról és a következő futtatások elősegítéséről.

Háttér

Amikor a projekt a hibrid irányba kezdett átmenni az új megközelítés első modulján dolgoztam. Az első szakaszban a rendszerkövetelményeket tanulmányoztam a felhasználói estetek létrehozása érdekében, hogy minél világosabb képet és terveket tudjak a vezetőségnek megmutatni. Ezek segítségével létrehoztunk egy tesztelési követelményekből álló mindmap-et, amelyet (nem kevés harc után) a fejlesztői csapat számára is elérhetővé és használhatóvá tettem.

Bár fel tudtam mérni, hogy az algoritmusmotor – ami az egész projekt alapja volt – miképpen tesztelhető a fejlesztők számára, mindig gondban voltam ez egyes permutációk eredményével. A fejlesztők sosem látták, hogy mit kell igazából tesztelni, de ez a vizuális tesztelési terv segített nekik az unit tesztek elkészítésében.

A kihívás

A funkcionalitás mindig változott a fejlesztés során. A fejlesztők sosem tudtak egy pontos képet adni a készülő megoldásról. Valahányszor hibát rögzítettem, azt vagy azonnal kijavították, vagy érvénytelen hibának minősítették. Továbbra is szükségét éreztem annak, hogy visszajelezzek a fejlesztői csapatnak kiemelve azokat a hibákat amelyek javítása a modul általános minőségének javulásához vezetne még a hivatalos tesztelési időszak előtt. A változó funkcionalitást is figyelembe kellett vennem, hogy ne okozzak nagy fejtörést a fejlesztés folyamán a fejlesztőknek.

A megoldás

A felfedező megközelítést követve, valamint a homályos részek feltárását figyelembe véve egy tiszta képet alkottam a funkcionalitás állapotáról.

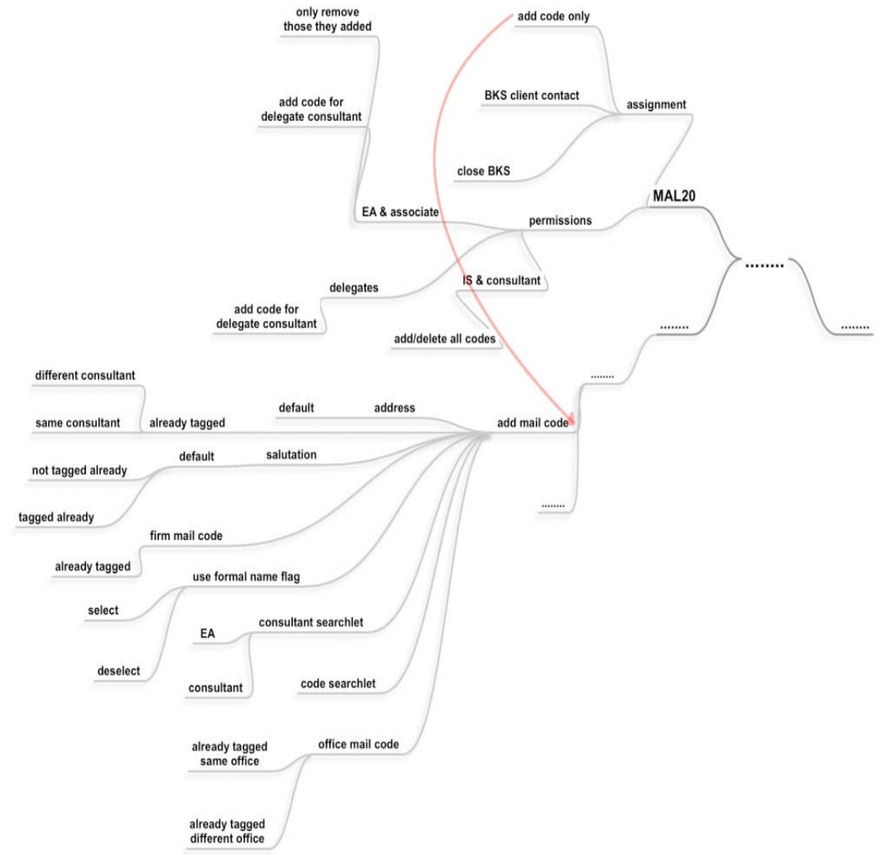
Minden scrum eseménynél meg kellett kérdeznem a fejlesztőket, hogy a legutolsó buildben vannak-e újonnan tesztelhető modulok. Amennyiben voltak, abban az esetben ezt egy JIRA bejegyzésben rögzítenem kellett. Az összes érintett funkció vizsgálatára kiterjedő tesztelés hatókörét mindmap-ben elkészítettem, majd csatoltam a JIRA bejegyzéshez. Minden további jegyzetet is a bejegyzéshez csatoltam (1. ábra).

A tesztkörök nem voltak időhatárhoz kötve, de akkor mindenképp véget értek, amikor a JIRA bejegyzésben leírt összes funkció le lett tesztelve. A tesztelés hatókörétől függően a futtatás néhány óra alatt megtörtént, vagy útközben kiderült, hogy a funkcionalitás az aktuális build-ben valamilyen hiba miatt nem tesztelhető.

A tesztelés folyamán a megjegyzéseket a JIRA-ban rögzíteni kellett és egy előtaggal jeleztük, hogy az ISSUE, vagy QUERY. Például:

- **ISSUE**: Nincs kész. Hiba a BackgroundProcessingServices.log fájlban. E-mail értesítés kiküldve, de az eredmény hibás.
- **QUERY:** A hibaüzenet nincs definiálva.

Az ISSUE egy olyan találat, ahol a felhasználói igények és a funkcionalitás nem egyezik, vagy hiba van az alkalmazás működésében. A QUERY pedig a tesztelésből adódó kérdés, ami rendszerint olyan hiányosság, ami nincs definiálva a tervdokumentumokban.



1.ábra

Iain Bright

Iain Bright több mint 10 éve teszteléssel foglalkozik. UAT csapatban kezdte meg munkáját a pénzügyi szektorban mielőtt Írországba költözött. Mióta dolgozik rengeteg projekten részt vett különböző szektorokban, ahol rendszer és felhasználói átvételi tesztekkel foglalkozott. Jelenleg Dublinban szenior tesztelő az FBD biztosítónál.

Segítünk megkeresni a hibákat!

A futtatások eredményét (illetve egy linket a JIRA bejegyzésre) a vezető fejlesztőnek el kellett küldeni, melyről természetesen az üzleti elemző és a termékfelelős kapott egy másolatot.

Ezt utána a felelős fejlesztővel közösen át kellett nézni, megbeszélni minden találatot és az ehhez kapcsolódó JIRA megjegyzést. Meg kellett tudnunk egyezni abban, hogy miket kell azonnal javítani, miket javítunk majd később és mik adódnak a folyamatban lévő fejlesztésből. A megbeszélés eredményét a JIRA-ban rögzíteni kellett.

A következő build-nél a kijavított ISSUE-kat ellenőriztük. Ha egy ISSUE továbbra is fennállt, létrehoztunk rá egy hibabejegyzést, hogy ne kerülje el a későbbiekben a figyelmünket.

Amikor az összes ISSUE kijavításra került és minden QUERY-re választ kaptunk, az aktuális ide kapcsolódó JIRA bejegyzést lezártuk. Ezt a folyamatot minden sprintben újra megismételtük a végső verzióig, a formális teszt fázis előtt.

Mit tanultunk?

Mi, mint scrum csapat, a sprinteken keresztül folyamatosan tanultunk. A következő felfedezéseket tettük:

1. Nincs bejáratott út az ISSUE-k átnézésére a fejlesztőkkel.
2. Nincs idő hozzárendelve a sprintekhez.
3. Nincs semmilyen időkorlát arra, hogy az ISSUE-kat és a kérdéseket mikorra vezessük át hibaként.
4. Néhány ISSUE bejegyzés elveszik és csak később, a funkcionális tesztelésnél jelentkezik hibaként.

Fejlesztések

Amikor egy modul elkészült vagyis le lett fejlesztve és tesztelve, akkor át kell nézni minden érintettet bevonva. Különösen a fejlesztő-tesztelő viszonyát elemeztük, hogy kellően jó volt-e a sprintekben a megfelelő teszteredmények elérése érdekében. Ezeket jegyeztem fel:

1. Hozzuk létre a JIRA bejegyzéseket és határozzuk meg a hatókörét úgy, hogy mindmap-be összeszedjük a tesztelendő funkciókat, vagy csak egyszerűen szövegesen leírjuk őket.

2. A tesztek szorosan a fejlesztéshez kapcsolódjanak.

3. Üljünk össze a fejlesztőkkel, hogy egyetértsünk milyen funkcionalitást kell szállítaniuk.

4. A JIRA bejegyzést használjuk arra, hogy megvilágítsuk vele az általunk elterve-zett tesztelési feladatokat a fejlesztők számára.

5. A funkcionális tesztek eredményét és az ISSUE-kat a közvetlenül a JIRA bejegyzéshez kell csatolni. Legjobb ezt az esti release-ek után reggel megcsinálni.

6. Az új ISSUE-kat a fejlesztővel a délután folyamán át kell nézni. Meg kell beszélni, hogy melyik lesz kijavítva, melyikkel nem foglalkoznak és mik a tényleges hibák.

7. A kijavított ISSUE-k újbóli tesztje, és az eredmények rögzítése a JIRA-ban.

8. Hibák bejegyzése – ahogy előre egyeztettük, vagy olyan ISSUE-k jelentése, amelyek nem javultak ki a következő esti release-ben.

9. A JIRA bejegyzés lezárása.

Ezzel azt is meg tudom mutatni, hogy pontosan mely sprintben miket teszteltünk a felhasználói tesztelési fázis előtt. Valamint a felhasználói teszten is sokkal kevesebb hiba várható majd.

Sokkal fontosabb a talált hibák számának a csapaton belüli kapcsolatokra gyakorolt hatása. Két csoportból egy csoporttá váltunk, ennek eredményeként sokkal rövidebb határidőket tudtunk tartani. A fejlesztők tiszta képet kaptak arról, hogyan dolgozunk és miket nézünk a tesztelés során.

Képesek voltunk kiterjeszteni ezt a gondolkodást a hibák javításakor, az újratesztelések idején és egyéb területeken is ugyanezzel a csapattal. Amikor egy fontos hibát találtunk, akkor kiegészítettük azt egy regressziós teszttel, amit jeleztünk a fejlesztőknek is, miként fogunk meggyőződni arról, hogy az adott hiba kijavult. Ezt a fejlesztők a saját munkájuk ellenőrzésekor is használták. A szorosabb együttműködés a projektek felgyorsításában is nagy szerepet játszott. ■

Szerző: ***Iain Bright***

Forrás:
http://www.ministryoftesting.com/2014/12/using-test-output-start-conversation/

Teszteljük a visszafejtett kódot

Mit tudunk kezdeni egy APK fájllal? Fejtsük vissza és teszteljük le, még akkor is, ha nem vagyunk biztonsági szakemberek! A visszafejtett kód nagyon sok információt elárul az alkalmazásról, segít az alkalmazás jobb megértésében, a funkciók átlátásában és az alkalmazás teljesítményének mérésében. Ezek tudatában szélesíteni, javítani tudjuk tesztkészletünket, ezáltal a tesztelés nagyobb lefedettséget érhet el.

Mi az APK fájl?

Az android application package (APK) egy alkalmazáscsomag, amivel a Google Android platformjára lehet alkalmazásokat installálni. Hasonlít az installerekre, ahol az állományok egyben vannak összecsomagolva. Egy ilyen APK fájl programkódot (*.dex fájlok), manifeszt fájlokat, forrásokat és certifikátokat tartalmaz.

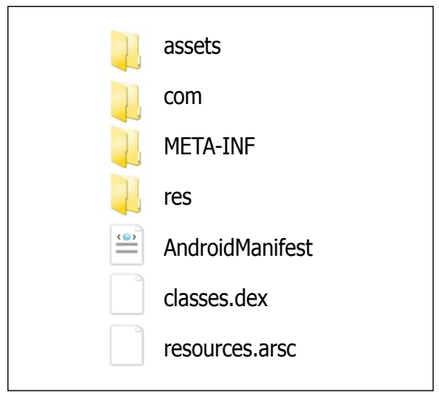
Az APK egyszerűen egy archív fájl, osztályokkal és forrásokkal. A Dalvik számára futtatható állományok a classes.dex-ben lesznek elérhetőek. A Classes.dex tartalmazza a lefordított futtatható kódot, osztályokat, metódusokat, amelyek az üzleti logika megoldását szolgálják. Az egyik képes a classes.dex fájlt (ami a gép számára olvasható formátumban van) átfordítani JAR fájllá (ami az ember számára olvasható formátumban van). (A segédállományok listája cikk végén a függelékben található meg.)

Az APK fájl struktúrája

A speciális fájlok, karakterkészletek, 3rd party SDK-k és az integrációjuk a com könyvtárban találhatóak (1. ábra).

Manifest fájl

Egy Android alkalmazásnak rendelkeznie kell egy “AndroidManfiest.xml”állománnyal az APK főkönyvtárában, ahogy a fenti struktúrában látszik. Ez tartalmazza azokat a szükséges engedélyeket, melyek az installálás során szükségesek, valamint egyéb részleteket is. Vannak olyan Android alkalmazások, amelyekkel a manifeszt és az installált alkalmazás megtekinthető. A manifeszt ugyanúgy olvasható az APK-ből is a megfelelő eszközök segítségével. (Lásd a függeléket.)



1.ábra

Dex (Dalvik Executable)

Az Android alkalmazások Java-ban íródnak, és a DVM-ben futtathatóak (Dalvik Virtual Machine). Ez eltér a JVM-től (Java Virtual Machine). A DVM az Android operációs rendszer számára lett kiépítve. A Java kód DVM kóddá transzformálódik és a .dex, valamint a .odex (Optimized Dalvik Executable) kiterjesztésű állományokban van tárolva. Az odex és a deodex kifejezések a Dalvik műveletekkel vannak összekapcsolva.

Amikor a Java fordító létrehozza a JVM kódot, a Dalvik letörli a class fájlokat és újrafordítja azokat. Ezek után a Dalvik mindet egy .dex fájlba csomagolja össze. Ezalatt az idő alatt a fordítás, újraépítés megtörténik az osztályokban.

Van egy határ, hogy egy .dex fájl pontosan mit és pontosan mennyit tartalmazhat. Ennek megfelelően az APK egy vagy több .dex állományt is tartalmazhat.

Erről a limitről a JAR fájlban található bővebb információ, ha a .dex-ről JAR-ra való konverzió után a JAR fájl megtekintjük. Ez árulkodik az alkalmazás összetettségéről és innen szerezhetünk információt a tesztelésre vonatkozóan is, hogy hogyan tudjuk minél körültekintőbben tesztelni az alkalmazást. Az Android 5.0 ART, dex fájl OAT fájllá konvertálódik így: è Java -> Class à Dexà OAT. Ennek a tanulmányozása hasznos lehet, hogy jobb teszteket tudjunk kidolgozni.

JAR visszafejtés

A JAR fájlok a Java decomplierrel tekinthetők meg. Rendszerint a visszafejtett kódok össze vannak kavarva (hála a fejlesztői torzításnak), viszont rálátást adnak az osztályokra és a metódusokra. Bár a visszafejtett kód nem tartalmazza az egész forrást, de tartalmaz olyan fontos információkat, amiből az üzleti logika kiolvasható. A tesztelők a forráskód áttekintése után pontosan látják majd, hogy mit és hogyan kell tesztelni, valamint fontos javaslatokat tehetnek a fejlesztőknek a fejlesztésre vonatkozóan. Ha a kód nem teljesen kusza, abban az esetben a termékfelelősöknek és a programozóknak is hasznos lehet. Az olvasást megkönnyítik különböző célszoftverek, mint a ProGuard és a DexGuard.

A visszafejtett JAR fájl fontos tesztjei

1. Olvasható üzleti logika

Az illetéktelen hozzáférések elkerülése érdekében a fejlesztők különböző biztonsági mechanizmusokat alkalmaznak. Ha a kód egyszerű szöveg formátumban van, az annyit jelent, hogy a kód olyan mértékben olvasható és érthető, hogy áttekinthetővé teszi a lefordított függvényeket és osztályokat. A kód egyes részei szabadalmaztatva lehetnek, vagy szabadalmaztatás előtt állhatnak. Az is elképzelhető, hogy pont ebben a kódrészben található az üzleti logika nagy része. Az ilyen formán nyíltan és olvashatóan tárolt kódok nagy veszélyt jelentenek a szervezet hírnevére, valamint a cég bevételére is.

Megoldás

A kódnak minden esetben torzítottnak kell lennie. A torzítás egy olyan folyamat, amely során a szervezet úgy rejti el a kódot, hogy az üzleti logika ne lehessen kiolvasható, a kód ne legyen másolható, reprodukálható. A ProGuard az egyik ilyen eszköz, amivel általában a programozók a kódokat torzítani szokták.

2. Manifest fájl

Az engedélyek mindenki számára elérhető, aki látja a manifest fájlt, ennélfogva teljes képet ad arról, hogy az adott alkalmazásnak milyen engedélyekre van szüksége a futáshoz. Ez egy újabb biztonsági kérdés.

Megoldás

Tanácsos a manifeszteket is titkosítva tárolni, ami megnehezíti a hackerek dolgát, hogy visszafejtsék és megnézzék a fájlt. A webes technológiákban van néhány titkosítási eljárás, ellenben a mobilos világban utána kell járni ezeknek a lehetőségeknek. Az Android áruház megköveteli az alkalmazások forrásának a titkosítását, és az engedélyeket a felhasználók számára elérhetővé kell tenni. A tesztelőn múlik, hogy ezeket az engedélyeket megvizsgálja és azonosítsa a potenciális kockázatokat.

3. getDeviceID() használata

Úgy tisztességes, ha bárki nyomon követheti az eszközén az egyes installálásokat. Valószínű, hogy a TelephonyManager.getDeviceId() meghívásával kinyerhető a kívánt adat és beazonosítható a telepítés. Vannak azonban problémák ezzel. Először is, nem mindig működik. Másodszor pedig ha mégis működik, akkor az értékek túlélhetik még a gyári állapot visszaállítását is, így csúnya hibát okozhatnak amikor a felhasználó törli az eszközét és átadja azt egy másik felhasználónak.

Megoldás

Nagyon sok jó megoldás van arra, hogy azonosítsunk egy eszközt. Azoknak, akik használni akarják, ajánlom az ANDROID_ID-t.

Nagyon ajánlott annak a tanulmányozása, hogy az alkalmazások hogyan használják az engedélyeket. Ha ezek nincsenek jól kezelve, az alkalmazás a személyes adatainkhoz is hozzáférhet bizonyos esetekben. A visszafejtett JAR lehet ebben a segítségünkre.

4. A forrásfájlok hozzáférhetősége

A forrás XML fájlok, fontos képek és egyéb állományok egyszerűen letölthetők és hozzáférhetőek. Nagyon fontos ezen állományok titkosítása.

Megoldás
A forrásállományokat ne lehess visszafejteni az APK-ból.

Összefoglalás

Az egy közhely, hogy a JAR visszafejtés a biztonsági teszteléshez kapcsolódik. De ez nem igaz. A visszafejtés az alkalmazás jobb megértését segíti elő. Segíti a funkciók működésének átlátásában, az alkalmazás teljesítményének mérésében és a túlzott memóriahasználat kiküszöbölésében. Legközelebb, egy alkalmazás visszafejtésénél figyeljünk ezekre a kincsekre és fedezzük fel a számtalan lehetőséget.

A csapatom és jómagam kidolgoztunk néhány kulcs tesztet azokhoz az Androidos alkalmazásokhoz amiket teszteltünk. Ezek a kulcstesztek jó szolgálatot tettek a fejlesztőknek az alkalmazást illetően és abban, hogyan fog majd az alkalmazás a Google világba illeszkedni. Azok számára, akik nem tudták, mennyi veszély leselkedik egy APK fájlban, nekik is jó gyakorlat volt. Továbbá a tesztcsapatnak is segített a fejlesztőkkel való együttműködésben.

Köszönetnyilvánítás

Ravisuriya-nak, aki lektorálta a cikket és folyamatosan támogatott. Köszönöm Ravi!

Függelék

Mit kell egy Java kódban mindenképp ellenőrizni?

1. Az alkalmazás általános kondíciói (forráskód, bizalmas adatok)
2. Biztonsági auditok (a kód nincs titkosítva)
   a. Érzékeny adatok
   b. Hiányos hozzáférési szabályok
   c. Kód sebezhetőségek
   d. Jogosultságvizsgálatok
3. Third-party protokollok, API-k vizsgálata

APK Védelem

Az APK fájl védelme egy külön fejezet az ingyenes Internet világában. A fejlesztőknek állandóan figyelniük kell erre az APK fájlok készítésekor. Rengeteg megoldás van az APK védelmére, valamint rengeteg cikk is foglalkozik a témával. A legfőbb témákat listába szedtem, amikről bővebben is tájékozódhatunk az Android-os fejlesztői oldalakon.

1. Biztonságos Java forráskód
2. Class fájlok titkosítása
3. Kódok torzítása

Eszközök a JAR visszafejtéséhez

1. Dex2jar – Eszköz az Android *.dex és java *.class állományokhoz
2. JAD – Java nyelv visszafejtő
3. JD-GUI – Egy grafikus segítség a *.class állományok visszafejtéséhez
4. APKtool – 3rd party reverse-engine eszköz androidos applikációkhoz
5. Winzip / WinRAR – tömörítés
6. Proguard - Ingyenes Java class fájlok visszafejtéséhez
7. APK Protect – APK védelem ■

Szerző: ***Parimala Hariprasad***

Referenciák:
http://en.wikipedia.org/wiki/Android_application_package

http://www.decompileandroid.com/

Forrás:
http://curioustester.blogspot.hu/2015/01/key-tests-on-decompiled-jar-of-dex-file.html

Parimala Hariprasad

Parimala kilenc évnyi tapasztalattal rendelkezik tesztelés, vezetés és szoftvertesztelő csapatok trenírozásában. Dolgozott már CRM, biztonsági, kereskedelmi és támogatásautomatizálási területeken. A tesztelés mellett – amit nagy szenvedéllyel űz – nagyon kedveli a tesztelők trenírozását is. Gyakran ír tapasztalatairól a http://curioustester.blogspot.com oldalon. Emellett még számos cikket publikált tapasztalatairól olyan magazinokban, mint a Better Software, Testing Circus és Testing Planet. Parimala aktív résztvevője a területhez kapcsolódó konferenciáknak és találkozóknak. Mélyen hisz a csapatmunkában, és segíti a csapatokat, hogy közösen dolgozzanak a végső cél elérésében. Ha épp nem tesztel, szívesen játszik két tündéri gyermekével, könyveket, magazinokat, cikkeket olvas. Jelenleg tesztmenedzserként dolgozik a Moolya Software Testing Pvt Ltd-nél, Bangalore-ban. Elérhető a parimala@moolya.com címen vagy twitteren @CuriousTester néven.

Adattárház tesztelés (II. rész)

Adattárház tesztelés ellenőrző listával bővítve

Az adattárházak kapcsán az egyik legfontosabb kérdés, a forrásadatok minősége. Ez határozza meg, hogy milyen kinyerési folyamatot alkalmazunk, milyen transzformációkat kell elvégeznünk, míg az adatok az adattárházba kerülnek. A tesztelés sem az adattárház létrehozása után, hanem már a forrásadatok szintjén elkezdődik. Milyen lépéseket tegyünk, hogy megfelelően ellenőrzött folyamataink legyenek?

Valószínűleg a napi munka folyamán mindenki használ ellenőrző listákat. Ha az adattárház teszteléshez nem használunk ellenőrző listákat, akkor nagyon sokat vesztünk az eredmény minőségéből és a profizmusból is.

A procedurális adattárház ellenőrző lista tartalmazza az összes elvégzendő feladatot és a feladatok sorrendiségét is. Ez egy olyan segítség, amely áthidalja az emberi memória és a figyelem korlátait, így csökkentve a hibázás lehetőségét.

A cikk első részében, rámutattam a teljes körű adattárház tesztelés fontosságára és kiemeltem a folyamat kulcsfázisait. A második részben javaslatokat teszek az ellenőrző listákhoz, amelyek abban segítenek, hogy ne kerüljék el a figyelmünket a kulcsfontosságú lépések a bonyolult adattárház tesztelés tervezésekor, és a tesztek futtatásakor. Az ellenőrző listák a teljes tesztelési stratégia fejlesztésekor is segítenek a fókuszálásban és a tesztesetek priorizálásban, valamint hibák sikeres feltárásában.

Unit teszt ellenőrző lista

A legtöbb fejlesztő enyhén szólva sem képzett tesztelő. Programoznak, majd telepítik az elkészített részeket és elkezdik a soron következő feladat kódolását anélkül, hogy unit teszteket futtattak volna. Egy ellenőrző lista segítheti az adatbázis fejlesztőket a kód szisztematikus tesztelésében, mielőtt még az a QA csapathoz kerülne.

- A mezőmegfeleltetések ellenőrzése a kritikus adatok vonatkozásában a staging (előkészítő) és az éles szinteken.
- A szekvencia generátor által előállított adatok duplikációjának a vizsgálata.
- A másodlagos kulcsok egyediségének ellenőrzése.
- Az adattípus kényszerek ellenőrzése a staging és az éles szinteken.
- Az adatbetöltés ellenőrzése és a hibaüzenetek elemzése minden ETL után. (ETL – extraction, transformation, load – kinyerés, átalakítás, betöltés)
- A rosszul jobbról vagy balról levágott szövegek megkeresése.
- Annak az ellenőrzése, hogy minden tábla és mező betöltődött-e a staging-ről.
- Annak az ellenőrzése, hogy a nem NULL értékek biztosan betöltődtek.
- Ellenőrizzük, hogy nincsenek adatcsonkolások.
- Adattípusok és formátumok ellenőrzése.
- Győződjünk meg róla, hogy nincsenek duplikált rekordok a tény táblákban.
- A transzformációk pontosan az üzleti igények szerint hajtódtak-e végre?
- A numerikus mezők pontosan betöltődtek?
- Megvizsgálni, hogy az összes ETL csak a előre tervezett figyelmeztetésekkel futott le.
- Az adattisztítások és hibakezelések vizsgálata.
- Tárolt eljárások és adatleképezések ellenőrzése

Az integrációs teszt ellenőrző listája

Egy integrációs teszt ellenőrzőlista abban segít, hogy meggyőződjünk róla, hogy az ETL folyamatok a tervek szerint a megfelelő függőségekkel hajtódtak-e végre.

- Ellenőrizzük az adatbetöltés sikeres végrehajtását.
- Ellenőrizzük, hogy a táblák jó adatokkal töltődtek-e fel és nincs-e adatvesztés.
- Minden adatbetöltéshez kapcsolódó függést, feltételt vizsgáljunk meg – beleértve az összes szakaszt. (forrás táblák-> staging táblák; staging táblák -> adattárház táblák)
- Minden ETL hibát és log-ot ellenőrizzünk a kijavíthatóság érdekében.
- Ellenőrizzük, hogy az adatbetöltés akkor kezdődött és fejeződött be, amikor azt terveztük.

Teljesítmény teszt ellenőrző lista

A nagy tömegű adatnövekedés miatt az adattárházban arra kell számítani, hogy az ETL futtatási idők megnyúlnak és a lekérdezések hosszabb időt vesznek igénybe. Ezek a változások egy jól átgondolt architektúrával és ETL-el kezelhetőek. A performancia tesztelés az adattárházban az ETL tervezés gyengeségeinek a feltárására hivatott, mint állományok újbóli kiolvasása, felesleges állományok létrehozása és tárolása. A teljesítményteszt ellenőrzőlista segít a performancia kérdések felderítésében.

- Adatbetöltés csúcsra járatása annak érdekében, hogy az ETL végrehajtása a kért időintervallumon belül megtörténik-e.
- Az ETL betöltési idejének összehasonlítása viszonylag kis adatmennyiséggel a skálázhatóság ellenőrzése érdekében. Az ETL futások idejének az ellenőrzése minden komponens esetében az esetleges gyengeségek felderítése miatt.
- A visszautasított folyamatok monitorozása és annak a megfigyelése, hogy mennyi visszautasított adat kezelhető egy időben.
- Az egyszerű és összetett join-ok vizsgálata a nagy volumenű lekérdezések futtatása érdekében. Együttműködés az üzlettel olyan lekérdezések kidolgozásában, amelyeknek a válaszideje elfogadható.

Rendszer teszt ellenőrző lista

Az adattárház tesztelés egyik fő oka annak az ellenőrzése, hogy az üzleti igények megfelelően lettek-e implementálva. Ez a szakasz tartalmaz adatellenőrzést, amely azt ellenőrzi, hogy az adatok a megfelelő módon vannak-e a cél táblákba transzformálva. A rendszerteszt ellenőrzőlista a következőképpen segít:

- A rendszer funkcionalitása megfelel-e az üzlet által megfogalmazott igényeknek.
- A forrás és cél rekordok számosságának a vizsgálata, a visszautasított vagy hibás adatok figyelembevételével.
- Minden integráció és kapcsolat ellenőrzése (pl. a hardver és hálózati beállítások vizsgálata)
- Minden tranzakció, adatbázis frissítés és adatáramlás pontosságának vizsgálata.
- Az üzleti jelentések megfelelőségének a vizsgálata.

Technikai teszt ellenőrző lista

A széleskörű és sokszínű adatforrások integrációjának a bonyolultsága miatt néhány tipikus kezdeti problémára felkészülhetünk. A technikai teszt ezért az összes egyéb (rendszer, performancia, UAT) előtt van.

- A hardver a helyén van és megfelelően lett inicializálva, beleértve az ETL eszközöket, forráshoz való kapcsolódást és az üzleti objektumokat.
- Minden szoftver megfelelően lett migrálva a tesztkörnyezetbe.
- Minden kapcsolat működik a rendszerek között.
- Minden tranzakció lefut, nincs lefagyás.

Összefoglalás

Az összes itt említett ellenőrzőlista - habár nem teljesen részletes – de abban bízok, hogy értékes lesz az adattárház tesztelési projektekben. Az ellenőrzőlisták segítenek a magasabb minőség elérésében a memóriánk adta lehetőségek kompenzálásával. Segítenek a bonyolult adattárház tesztelések megtervezésében és futtatásában, amely kulcsfontosságú az adattárház tesztelési munkában és annak a sikerességében. ■

Szerző: ***Wayne Yaddow***

Forrás:
http://ibmdatamag.com/2013/07/data-warehouse-testing-part-2/

Wayne Yaddow

Wayne informatikai tanácsadóként, vezető QA szakemberként dolgozik. Nagy tapasztalata van az adatmigrációk és az adatok integrációja területén. Különösen a pénzügyi ágazatban nagy adattárház projekteken tevékenykedik. Ő az egyik társszerzője a „Testing the Data Warehouse: Assuring Data Content, Data Structures and Quality” című könyvnek.

A következő e-mail címen tudsz kapcsolatba kerülni vele: wyaddow@gmail.com

Szinergikus tesztelés

Nagyon sokan a kereskedelmi, pénzügyi, banki, vagy telekommunikációs területen dolgoznak tesztelőként, esetleg egy szoftverfejlesztő vállalatnál látnak rá ezekre a területekre. De vajon milyen lehet játéktesztelőként dolgozni? Milyen kihívások vannak ezen a területen? Egyáltalán különbözik-e bármiben is a játéktesztelés az üzleti élet egyéb területein végzett teszteléstől?

“Ó, akkor Te egész nap csak kaszinózol!”

Fantasztikus életem van.

Ez az igazság... van egy nagyszerű magánéletem és a munkám is kitűnő!

Biztos vagyok abban, hogy nem sok tesztelőtől hallottál már hasonlókat.

Rengeteg rémes cikket olvastam tesztelői sorsokról, olyan emberekről akiket egyszerűen nem vesznek emberszámba a kollégáik és magát a tesztelést sem tartják fontosnak. Arra jutottam, hogy ez nincs így jól és ideje lenne felállnom és elmondanom, hogy ennek nem így kellene lennie.

Ha kíváncsi vagy rá, brit vagyok, Svédországban játék tesztmérnökként dolgozom a díjnyertes online kaszinó játékokkal foglalkozó NetEnt-nél.

Ahogy egyre jobban belemélyedek a tesztelői közösségbe, azt látom, hogy van egy hierarchia a tesztelők között. A sok éves tapasztalat nagyobb elismertséget és dicsőséged ad, ezt követi, ha valaki automata tesztszakemberként dolgozik. Megduplázódik a rangunk, ha banki vagy telekommunikációs területen dolgozunk. Ha valaki mobiltesztelésen dolgozik, annak még nagyobb az elismertsége manapság.

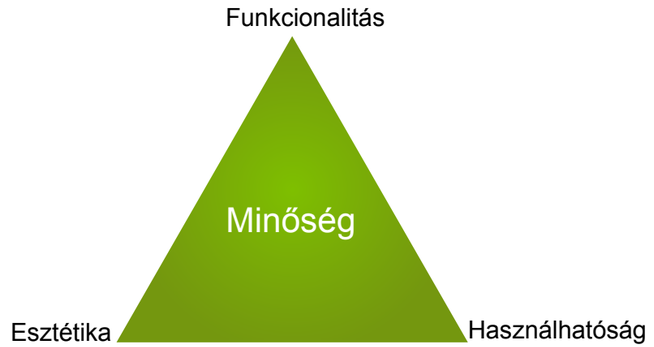
Az egésznek az alján a játéktesztelők állnak. A legalján pedig az online kaszinó játékokat tesztelő mérnökök vannak.

Amikor egy szemináriumra, vagy előadásra megyek, előbb utóbb jön a kérdés: „Szóval, ki merre dolgozik?” A válaszom azoknak akik nem járatosak az online kaszinók területén ismeretlenül hangzik, mert még nem hallottak a NetEnt-ről. Amikor leesik nekik, jön a felismerés: „Ja, Te csak egy játéktesztelő vagy!” Ilyenkor mosolyogni szoktam a megvető pillantásokon abban a tudatban, hogy túl sok energiámba telne elmagyarázni, hogy én ugyanolyan jó tesztelő vagyok, mint ők!

A legnagyobb fejfájásunk egyébként akkor van, amikor tesztelőt próbálunk felvenni a céghez. Egy friss diplomás egyszer azt válaszolta nekem, hogy nem érdekli az ajánlatom, mert ő egy sokkal komolyabb tesztelői állásra vágyik. Komolyan?

Nos, most itt van a lehetőségem hogy tisztázzam, mit csinálunk mi, online kaszinó játék tesztelők, hogyan dolgozunk és milyen tesztelési gyakorlatot alkalmazunk.

Általánosságban mindenki azt gondolja, hogy a tőzsdén és a bankokban vannak a legnagyobb számú pénzügyi tranzakciók (talán azért, mert az ottani tesztelők elitnek érzik



magukat). Tudtad, hogy a NetEnt 1.7 milliárd játékkört kezel minden hónapban? És minden hónapban több, mint 2.4 milliárd EUR folyik át a rendszerünkön. És ezek a számok egyre csak növekednek! El tudod képzelni, hogy egy hibás, vagy egy rosszul működő játék micsoda galibát okozhat? Akkor el tudod azt is képzelni, hogy a tesztelés jelentősége mennyire fontos egy ilyen vállalatnál, mint a NetEnt.

Mi, online kaszinó játék tesztelők legalább annyira komolyan vesszük a tesztelést, mint amennyire a többi ágazatban is komolyan veszik. Mégis mi a különbség köztünk és közted, mint más ágazatban dolgozó tesztelő szakember között? Az, hogy a mi értékeinket elismerik és tiszteletben tartanak minket a vállalatnál. Mi sosem ülünk és várjuk, hogy a fejlesztési lánc végén kipottyanjon a legújabb „majdnem kész” verzió. Mi a magunk módján fejlesztők vagyunk.

Mint a többi játék tesztelő az Agile-ban, már a fejlesztés legelső pillanatában részt veszek a munkákban. Nemcsak, mint a fejlesztő kispajtása a kódot és a grafikus kinézetet ellenőrzöm, hanem az egészet rendszert vizsgálom, egyben az egészet. A funkcionalitást, a grafikai megoldásokat, az animációkat és az egész logika megvalósítását, amely lehetővé teszi majd a játékot. Megítélésem szerint egy nagyon értékes pozícióban vagyok. Én vagyok a játék tesztelője, akinek számít a szava az egész alkalmazás létrehozásában (a tervezéstől a kivitelezésig). Nálunk a tesztelőnek a szava rendkívül fontos. Személyes tapasztalataim alapján képes vagyok egy új játék készítésénél akár a számításokra is hatással lenni, illetve olyan vadhajtásokat távolítok el, amelyek nehezítik a játszhatóságot. Azért, mert a szemem mindig a minőségen tartom. Pontosabban a NetEnt minőségen.

Mi az a NetEnt minőség?

Néhány hónappal ezelőtt összetrombitáltak minket, tesztelőket, hogy definiáljuk a NetEnt minőséget és a következővel jöttünk elő:

’A szoftverminőség mérhető felhasználói elégedettség a funkcionalitásban és a használhatóságban.’

Nem olyan régen egy prezentáción dolgoztam és egy újabb definíciót kerestem a minőségre és rádöbbentem, hogy egy nagyon fontos elemet kihagytunk a kirakós játékból, mert túlzottan a szoftverminőségre koncentráltunk, ahelyett, hogy a NetEnt minőségre figyeltünk volna. Ez a hiányzó elem az esztétika volt.

A játékok a szórakoztatásért vannak és véleményem szerint a legjobb minőség elérése érdekében meg kell találni az egyensúlyt a funkcionalitás a használhatóság és az esztétika között.

Mit értek esztétika alatt? Ha azt feltételezzük, hogy minden energiánkat a legjobb funkcionalitás elérésére fordítjuk, áldozva kinézhet és a hangzás oltárán, akkor nagy valószínűséggel a játékkal senki sem fog majd játszani. Megfordítva, ha csillog a felület és eszméletlen a zene, de a funkcionalitás katasztrofális, szintén ugyanaz lesz a helyzet.

A használhatóság alapvető kérdés, de sok esetben túl van dimenzionálva. Van egy szabályrendszerem a használhatósággal kapcsolatban:

Mint játékos ne kelljen gondolkoznom. Ha a játék nem intuitív, vagy nem egyszerűen tanulható, akkor bizony elveszítjük a játékosokat. Mint átlagos intelligenciával rendelkező egyén legtöbbször unok egy játékot, ha az

nem intuitív, vagy nem tudom 30 másodpercen belül használni.

Ez a minőségről szóló egyenlő oldalú háromszög kulcsfontosságú a tesztelésben. Minden csúcshoz különböző tesztelési metodika tartozik. Nem lehet például automata teszteléssel használhatóságot tesztelni, továbbá a legtöbb esztétikai kérdésre manuális teszteléssel adhatunk választ, míg a funkcionalitás egy része automata teszteléssel lefedhető.
Számtalan tanulmány készült már az agilis teszteléssel kapcsolatosan, és gyanítom, hogy ahány tesztelőt megkérdezel erről, annyiféle agile környezetről fog beszámolni. Azt fogod észrevenni, hogy mindenki egy saját maga által kialakított hibrid módszert alkalmaz. Egyik jobb, mint a másik? Természetesen nem, hiszen minden tesztelés és minden fejlesztés különböző, nincs jó vagy rossz megoldás, csak hatékony, használható vagy fájdalmas út van. Ez pediglen attól függ, hogy melyik vállalatnál dolgozol és milyen fejlesztési folyamatot használtok. Nem azt mondom, hogy az automata tesztelés semmit sem ér az esztétika érdekében, én csak annyit mondok, hogy nem az automata a legjobb megoldás erre.

A manuális tesztelés gyengébben teljesít az automatizált tesztelésnél abban az esetben, ha rossz célra használjuk. Klasszikus példa az online játékban a zsetonok 26 féle valutanemben való ellenőrzése manuálisan (napokat venne igénybe), amit egy automata teszt percek alatt elvégez és a létrehozása is csak néhány órát vesz igénybe.

Tesztvezetővé válni az olyan sikeres játékoknál, mint a Starburst™, South Park™ vagy Aliens™, csak úgy lehet, ha olyan teszt stratégia kialakítására vagyunk képesek, amely a legkisebb mértékben növeli a fejlesztési életciklust és biztosítja a legmagasabb szoftverminőséget. Ezen játékok mindegyikét bonyolult volt megtervezni és kifejleszteni, de ez nem jelenti azt, hogy a tesztelésnek is ugyanilyen bonyolultnak kell lennie. A tesztelési metodikákat minden esetben a fejlesztésekhez alakítom - kódellenőrzés, felfedező tesztelés, automatizálás, használhatósági és regressziós tesztek. Az aranyszabály, hogy a tesztvezetőnek ismernie kell az alkalmazást töviről-hegyire.

Nem viccelek. Ha már a játékon dolgozol minden nap, úgy megismered azt, hogy minden apró változtatás a teljesítményben, animációban, hangban vakítóan nyilvánvaló lesz még akkor is, ha meg se néztük a kódváltoztatások listáját.

A kulcs

Ki kell mondani, hogy a tesztelés hatékonysága a kommunikáción múlik. A kollégák közötti jó viszony kialakítása alapvető, mert ha tudjuk, hogy melyik fejlesztő pontosan mivel foglalkozik, akkor nem fogunk területek felett elsiklani és a lehető legtöbb rész tesztelve lesz. Még a nagyobb csapatokra is érvényes ez.

A legtöbbször, amikor valami furcsát találok a játékban, akkor nem rögzítem hibaként egyből, hanem megkérdezem a fejlesztőt, akihez tartozik (minden kódváltoztatáshoz, modulhoz rendelve van egy felelős). Legtöbbször olyan válasz kapok, hogy „bocsánat, elrontottam a kódot”, vagy „éppen most ezen dolgozom”, vagy az, hogy „holnap már tesztelhető”.

Ahelyett, hogy arra koncentrálnánk, hogy melyik tesztet kellene lefuttatni, koncentráljunk arra, hogy mi van kész a kódból. Így megkímélhetjük magunkat a felesleges hiba felvételektől. Továbbá csökkenti a stresszt a közelgő határidőknél a csapaton belül, mert kevesebb hibán kell átverekedni magukat. A legnagyobb előnye, hogy egyúttal növeli a szoftverminőséget is. Hogyan mérhetjük a szoftverminőséget? A játék sikerességével és a hibák számával.

Úgy hívom ezt, hogy szinergikus tesztelés, mert a valós idejű fejlesztés alapján tesztelünk egy előre eltervezett út helyett. Az én szemszögemből ez a legagilisebb tesztelés, ami csak létezik.

A NetEnt-nél nem létezik „Mi” vagy „Ők”. Ha fejlesztünk akkor mi együtt vagyunk, egy csapat. Kódoló, dokumentáció író, tesztelő, grafikus, animátorok és szerver guruk. És mindannyiunk ugyanazon célért, a jobb játékokért dolgozunk.

Nagyon szuper állásom van! ■

Szerző: ***Dawn Möller***

Forrás:
http://www.ministryoftesting.com/2014/12/synergetic-testing/

Dawn Möller

Dawn Möller jelenleg Játék tesztelő mérnök a NetEnt-nél. Több, mint 26 év tapasztalattal rendelkezik az IT iparban, ezen belül is az adatkommunikáció a szakterülete. Az elmúlt 5 évben kizárólag a tesztelésre, QA arenára, kihívást jelentő munkahelyi gyakorlatokra és a teszt módszertan paradigmákra fókuszált. A tesztelési praktikái egyre népszerűbbé váltak a NetEnt-nél, és az utolsó pár játék átadásánál jelentős fejlesztéseket mutatott be a munka hatékonyságát illetően anélkül hogy a minőség rovására menjen.
Dawn jelenleg asztali és mobil játékokat tesztel agilis szoftverfejlesztési környezetben. Dawn a vállalatnál elismert személy és a NetEnt Akadémia aktív embereként beszédeket tart a minőségről. Dawn ezen felül elnöke a Játék Tesztelő Csapatnak.

Mobil tesztautomatizálási kihívások?

A mindennapi élet szerves részeit képezik az okostelefonok, melyekre ezerféle applikációt tölthetünk le. Ezeknek az applikációknak a nagy részét nemcsak manuálisan, hanem automata módszerekkel is tesztelik. A tesztek automatizálása gyakran sok erőfeszítésbe telik. Még mielőtt nekikezdenél az automatizálásnak körültekintően olvasd el a leírtakat, hogy minél kevesebb fejfájásod legyen.

Amikor a TenKodnál kezdtem dolgozni automatizálási megoldásokon, a mobilok már a napi élet részei voltak és csilli-villi mobil alkalmazások jelentek meg napról napra. Arra kerestük a választ, hogyan segíthetünk a vállalatoknak a robusztus automatizálási projektek minél egyszerűbb és gyorsabb megoldásában. Azt láttuk, hogy a mobil teszt automatizálás implementálása gyakran sokkal nagyobb erőfeszítéssel jár, mint maga az alkalmazás kifejlesztése. Akarod tudni, hogy mi ennek az oka? Ha igen, akkor jó helyen jársz! Mivel elég sok ilyen kihívással találkoztunk már, azt gondoljuk, hogy olyan tapasztalatokat tudunk megosztani, amik igenis értékesek lehetnek a számodra.

Válaszd ki a megfelelő eszközt

Az első kihívás, amivel valószínűleg találkozni fogsz, hogy megtaláld a megfelelő eszközt a teszteléshez. A rengeteg mobilkészülék, képernyőfelbontás és operációs rendszer miatt annyira nem triviális a feladat. Az automatizált tesztek futtatása az összes típusú mobilon ugyanazzal a megoldással gyakorlatilag lehetetlen. Rá kell szánnod egy kis időt, hogy meghatározd, kik (kik lesznek) a felhasználók és ők milyen készülékeket használnak (fognak használni) leggyakrabban. Az ökölszabály az, hogy válasszuk ki az 5 leggyakrabban használt mobilkészüléket, a többit pedig szimulátoron keresztül teszteljük.

Ennek az infrastruktúrának a karbantartása igen költséges lehet. A készülékek megvásárlása költség és természetesen költség ezek karbantartása is, hogy mindig az aktuális legfrissebb verziójú operációs rendszer legyen az összes készüléken. Ha nem akarunk erre költeni, akkor ott vannak a felhőszolgáltatók, akiktől bármilyen készülékre bérelhetünk megoldásokat.

Válasszunk szoftvert

Most következik a kód megírása, amely a következő fejtörést okozza majd – melyik szoftvert használjuk az automatizált megoldás megírásához? Rengeteg nyílt forráskódú és fizetős megoldás elérhető a piacon és a legtöbb nyílt forráskódú (mint például az Appium, Selendroid Robotium) rendelkezik automatizáló keretrendszerrel, ami alkalmas a tesztek implementálására. De sajnos egyelőre nincsenek integrált fejlesztői környezetek, amik lefedik a teljes tesztelési életciklust (tervezés, fejlesztés, futtatás, karbantartás). Így tehát az összes ilyen megoldásnak megvan a maga kelepcéje, ami az igazán nagy erőfeszítést jelentheti, mire megtaláljuk az aktuális konfigurációs beállításokat, amikre szükségünk lesz.

A fizetős megoldások legtöbbje már fel van készítve erre, de ott következik az újabb fejtörés, a „vendor-lock”. Más szóval egy fizetős megoldás nehezen, vagy egyáltalán nem integrálható a meglévő rendszerekhez, illetve csak a szállító által meghatározott szoftverekkel működik együtt és ad teljes funkcionalitást. Ezek a szállítók minden esetben csodát ígérnek, de csendben azt kérik, hogy „finomhangold” az alkalmazást hozzájuk, vagy jailbreak-eld a készüléket amelyen tesztelsz.

A „finomhangolás” azt jelenti, hogy egy bizonyos előre megadott kódsort tegyünk be a tesztelt alkalmazásba a tesztelhetőség és kompatibilitás biztosításához, a jailbreak pedig megsérti az eszköz szolgáltatási feltételeit (terms of service). Mindkét megoldás erősen ellenjavallt és a legtöbb esetben ezt a vállalatok egyszerűen nem is vállalják be.

Rögzítsd és ne játszd vissza

A modern tesztelési megoldások lehetőséget biztosítanak a felhasználói műveletek rögzítésére és belőlük automatizált tesztek gyártására. Ezek a lehetőségek bár igen csábítóak, de minden esetben alaposan meg kell vizsgálni az általuk generált kódok minőségét. A legtöbbször ezek egyáltalán nem fejlesztőbarát megoldások és a későbbiekben igen nehéz változtatni rajtuk, amennyiben az üzleti logika változik. Ilyen esetekben általában nem marad más megoldás, mint a tesztek újbóli nulláról való elkészítése. Ez néhány tesztesetnél még kezelhető, de robusztus automatizált megoldásoknál igen időigényes és elveszi az időt a további fontos tesztek megírásától. A legjobb megoldás az volna, ha ezekkel a megoldásokkal létre lehetne hozni olyan teszteket, amelyeknek a kódjai olvashatók és karbantarthatók. Ez lehetőséget adna arra, ha az alkalmazás változik, néhány sor hozzáadásával vagy módosításával a tesztet hozzá lehetne igazítani és karban lehetne tartani.

Ha a tesztek elkészültek, futtatni kell őket az elvárt gyakorisággal a regressziós esetek minél korábbi feltárására. A 21. század emberének elvárása, hogy ezek a tesztek automatizáltak legyenek. Itt jönnek képbe a folyamatos integrációt támogató eszközök (Jenkins, TeamCity) ezeket a megoldásokat lehet úgy paraméterezni, hogy az automatizált tesztek minden változtatás után, vagy minden release után ütemezetten lefussanak.

A hétköznapi munkában a hibák és a regressziós kérdések mindennaposak, szóval ezek a tesztek biztosan le fognak állni. A hibák feltárásában nagy segítség a teszt riport, amely a lehető legalacsonyabb szinten adja meg, hogy mely tesztek futottak le sikeresen és melyek álltak le. Álmodhatunk képernyő mentésekről a tesztfuttatások során, valamint automatikus videofelvételekről is. A modern folyamatos integrációt támogató eszközök az egyszerű riportokat alapból támogatják és elég flexibilisek ahhoz, hogy a kívánt többlet kimeneteket könnyen lehessen paraméterezni, vagy kódolni a keretrendszerükbe.

Konklúzióként a mobil alkalmazások automatizált tesztelésének még nagyon sokat kell fejlődnie, mire eléri az asztali- és webes alkalmazások automatizált szintjét. A fejlődés ebben a szakaszában még azt javasoljuk, hogy igen körültekintően kezdjünk neki a mobil applikációk automatizált tesztelésének.

Körültekintő eszközválasztás

Úgy válasszunk eszközt, hogy az a legmesszemenőbbekig támogassa az aktuális mobilalkalmazás automatizált tesztelését, ne legyenek benne elkötelezettségek a gyártó felé és egyéb trükkös kötöttségek.

Az automatizált tesztek rögzítéséhez olyan nyílt forráskódú vagy fizetős eszközt válasszunk, amelyben a rögzített automata aktivitások kódjai könnyen olvashatók, változtathatók és karbantarthatók.

Győződjünk meg arról, hogy a kiválasztott megoldás integrálható-e a meglévő környezetbe és könnyen alkalmazkodik-e a mobil alkalmazás folyamatos változásához.

A legtöbb, mobil alkalmazásokkal foglalkozó vállalat komolyan veszi és invesztál az mobil automatizálásba, ezért mindenkit arra buzdítok, hogy kezdje el beépíteni a hétköznapi munkájába az itt olvasott ajánlásokat! ■

Szerző: ***Emil Simeonov és Danail Branekov***

Forrás:
http://www.ministryoftesting.com/2014/10/got-mobile-test-automation-challenges/

Danail Branekov

Emil Simeonov

Danail Branekov fejlesztő mérnök a TenKod-nál. Mélyreható szoftverfejlesztési és technológiai ismeretei segítették ahhoz, hogy a legmodernebb mobil tesztautomatizációs technológiát beleépítse a TenKod fejlesztési folyamatába. Nagy rajongója az agilis fejlesztési módszertanoknak. Az általa bevezetett rendszerek magas minőségére törekszik. Danail a TenKod-nál több SAP terméken is dolgozik szenior fejlesztőként és a csapat mérnökeként. Általában Eclipse keretrendszert használ és részt vett sok Eclipse projekten.

Emil Simeonov 2003-ban kezdett el dolgozni az IT iparban szoftverfejlesztőként. Azóta termék menedzsmenttel, szoftver architektúrákkal, projektmenedzsmenttel foglalkozott, valamint szakmai tanulmányokat készített és előadásokat tart.

Emil részt vesz a vállalat szociális tevékenységeiben, cikkeket ír az IT magazinoknak és számos bolgár egyetemen tart előadásokat.

Fény az alagút végén a regressziós tesztelésben

Mihez lehet kezdeni, ha a regressziós tesztkészletünk átláthatatlanná válik? Hogyan lehet rendszerezni, kezelni a több száz manuális és automatikus tesztet? Relatív könnyen felismeri az ember, hogy mikor kell változtatni a dolgokon és elkezdeni átalakítani a regressziós tesztek halmazát. De hogyan kezdjünk hozzá?

Mire gondolsz, amikor valaki a regressziós tesztre utal? Az attól függ, hogy Te a tapasztalataid alapján mit nevezel annak. Ha például egy új izgalmas projekten dolgozol ahol folyamatos szállítással (Continuous Delivery) működtök, amelyre jellemző a rendszeres automatikus ellenőrzésekkel támogatott kiadás, valószínűleg azt gondolod, hogy a regressziós tesztelés kényelmes időtöltés. Másrészt ha – mint én – egy olyan szoftveren dolgozol amely kb. egy évtizede folyamatosan fejlődik, a hidegrázás kerülget, ha a sok manuális regressziós tesztkészlet az eszedbe jut.
Jóllehet a fejlesztési módszerek változnak, néhány tesztelési folyamat megmarad. Rengeteg projektben dolgoztam már. Olyanban is, ahol a vízesésből az agile-ra váltottak, de még mindig voltak „regressziós sprintek”, vagy „stabilizáló körök” a release-ek előtt. Ezek természetesen nem oldják meg a problémát, csak szőnyeg alá söprik azt.

A vállalatok egy része azt gondolja, hogy a regressziós teszt fázis elhagyható. Másrészt minket tesztelőket azzal vádolnak, hogy bomlasztjuk és lassítjuk az új release kiadását. Biztos, hogy a regressziós fázis kihagyása jó ötlet? A regresszió általában a kód megváltozásához, vagy a komponensek megváltozásához kapcsolódik a fejekben.

Amennyiben a tesztelés feladata az, hogy megmondjuk az üzletnek, hol tart a fejlesztés, akkor biztos, hogy értékes munkát végzünk a regressziós tesztekkel? A tapasztalataim szerint a regressziós találatok ilyen esetekben igen ritkák, és a regressziós teszt gyakorlatilag alkalmatlan az ilyenfajta feladatvégzésre. Ilyen esetekben, az új funkciók fejlesztését figyelembe véve inkább azon kéne gondolkodni, hogy milyen tevékenységet végezzünk az értékes munka érdekében a regressziós tesztek futtatása helyett.

Egy regressziós teszteléssel foglalkozó előadás vezérgondolata az volt, hogy milyen eredményekkel járhat egy regressziós teszt. Például annak a megválaszolása, hogy az új release megtartotta-e a minőségét az előző release-hez képest, vagy - csak ugyanolyan, mint egy felfedező teszt, - megnézzük az alkalmazást, hogy van-e változás ott, ahol nem kellene. Láthatjuk, hogy a fókusz egészen máshol van, mint önmagában a tesztelésnél.

A változások nehezek is lehetnek. Mint ahogy Grace Hopper híres mondata: „Az ember allergiás a változásokra, imádja azt mondani, hogy mi mindig így szoktuk csinálni”. Relatív könnyű felismerni, hogy jobb és értékesebb regressziós tesztelés is létezik, mint egy tesztrendszer ezernyi manuális regressziós tesztesettel. Végelát-hatatlan folyamat! Mégis hogy kezdjünk a megváltoztatáshoz?

Lássuk, miből élünk! Nem kell egyből minden könyvet elégetni!

Mint ahogy nagyon sok projektnél láttam, a tesztkészlet általában olyan, mint egy mini termék Biblia, míg a dokumentációkon gyakorlatilag áll a por és a wikik már rég idejét múltak. Így a tesztelők a szoftvert nézik és ez alapján állítják elő a tesztkészletet, amely valós és pontos képet ad a rendszerről. Ugye?

Sajnos nem. A legjobb tesztkészlet is csak egy pillanatkép a termékről a tesztelők akkori legjobb tudása szerint. A legelső implementációhoz képest a rendszer nagyot változhatott és a felhasználók, valamint az üzlet elképzelései is egész mások lehetnek most, mint az első verzióban voltak. Például egy projektben a konfigurációs beállítások hat sorát teszteltem, amikor kis nyomozás után kiderült, hogy a dokumentációban csak két sor szerepel és a regressziós tesztek is csak ezt a két sort fedik le. Vagyis az esetek 1/3-át teszteltük, míg a maradék 2/3-ra esély sem volt, hogy valaha egyáltalán vizsgáltuk volna.

Kritikus szemmel nézve a folyamatot a tesztkészletek a „túlélésre játszanak”. Ha egy tesztelő kolléga egyszer megírta a teszteseteket, aztán munkahelyet váltott, akkor a munka nagy része az lesz, hogy azon törjük a fejünket, mit akarhatott az aktuális tesztesettel. Ez egyrészt időt rabló, másrészt a minőség rovására is megy.

Sokszor nagy a kísértés, hogy dobjunk ki mindent és kezdjük elölről az egészet. Lehetnek olyan esetek, amikor ez működhet (például, amikor nagyon kevés teszteset-ről van szó, vagy ha az áttekintés alapján azt látjuk, hogy a tesztkészlet nagyon gyenge lefedettséget eredményez), de legtöbbször ezek az utak tele vannak csapdákkal.

Először is a meglévő tesztek tartalmazhatnak hasznos metaadatokat, mint az előző futtatások eredményei, hibaszámok, vagy hibajegyek referenciái (megmutatva, hogy mely hibák ismertek a felhasználók számára, vagy mely hibák a többszörösen visszaesők). Ahol lehet, ott ragaszkodjunk a meglévő metaadatokhoz, mert befolyásolhatják a jövőbeni teszteket.
Másodszor pedig, valamit újra elölről elkezdeni, nagyon nem lesz népszerű a menedzsment számára. Ha egy release közepén arra kérsz időt, hogy alaposan átvizsgálhasd a teszteket, a vezetők nehezen fogják elfogadni a hosszú távú előnyöket a rövid távú fejfájásokkal ellentétben. „Ha eddig jó volt így, akkor jó lesz ezután is! Miért nem jók azok? Mibe fog kerülni ez nekünk?” A kevésbé drasztikus megközelítés elkerüli ezeket a kellemetlenségeket.

Fokozatosan javítsd a folyamatot a jobb eredmény elérése érdekében

Értem már el sikereket a fokozatos átszervezéssel, és a tesztesetek számának csökkentésével. Amely funkciókon a fejlesztők dolgoztak – és ahol úgy találtuk, hogy nagyon szükséges – a teszteket újraírtuk az új funkcionalitásra alapozva. Ez fenntartható ütemű, fokozatos javulást eredményezett.

Eredményes lehet a tesztek természetének a megváltoztatása is. Lehetőség van a több ezer tesztesetet kisszámú end-to-end teszté transzformálni, amelyek a termék nagyobb darabjait vizsgálják. Megkönnyítheti továbbá a dolgot a folyamatvezérelt gondolkodás, ami jobb tesztriportokhoz vezet.

Ha újra átnézed a teszteket, valószínűleg fel fogod ismerni, melyek azok az esetek, amelyek automatizálhatók, ezzel is megmutatva, hogy a fekete-fehér látásmód helyett az elvárt legjobb eredményre koncentrálsz. Ha látszik, hogy a jövőben ezeket az eseteket futtatni kell, akkor érdemes elgondolkodni az automatizált keretrendszeren. A tesztelők pedig az értékesebb képességeiket használják inkább a jövőben!

Az automatizálás csökkenti a manuális tesztelés hibalehetőségeit. Az automatizálás túlzott használata viszont eltereli a figyelmet az alkalmazásról, ami potenciális veszélyeket hordoz magában. Ahogy Richard Bradshaw is megerősít ebben: ne essünk bele abba a tévhitbe, hogy az automatizált tesztelés rengeteg időt szabadít fel és kevesebb tesztelőre lesz szükségünk!

Legyen a minőség a csapatunk felelőssége

Az automatizálás egy igen értékes eredmény, de nem jár kevesebb költséggel, vagy karbantartással. Az ilyen ellenőrzések csak egy részét jelentik a tesztelési feladatoknak – és ha regresszióról beszélünk – jelentős erőfeszítésbe kerül a regressziós hibákat

elkapni nem is szólva a megoldás költségességéről. Sokkal inkább eredményre vezet, ha a regressziós hibák okait próbáljuk megtalálni.

Az aktuális pozíciómban a Towers Watson-nál a csapatunknak nagy erőfeszítésébe került amíg „tesztelőkből" „test specialistákká" promotáltuk magunkat. Ez egy felismerés, miszerint mi a minőségbiztosítás részei vagyunk, és a tesztelési feladat nem csak a mi dolgunk. A tesztelésre fókuszálunk, de emellett a projekt első pillanatától kezdve, különböző módon segítünk a fejlesztőknek minőséget építeni a termékbe:

- Folyamatot hozunk létre, amivel megbizonyosodunk arról, hogy a kódellenőrzés minden esetben megtörténik a tesztelés előtt;
- Átnézzük a unit teszteket a fejlesztőkkel és javaslatot teszünk újabb tesztekre;
- Felhívjuk a figyelmet az alkalmazásban lévő beállítások jelentőségére, ami legtöbbször elkerüli a fejlesztők figyelmét az új funkciók kialakításánál;
- Együtt vizsgáljuk meg a felhasználók által bejelentett hibákat, hogy megértsük, hogyan használják a rendszert, mi az amire mi nem gondoltunk a tesztjeink során.

A fejlesztői kódvizsgálat célja, hogy segítse a tesztelési folyamatot. A hibakezelő rendszerünkben van egy kötelezően megadandó „Tippek a teszteléshez" szekció, olyan adatokkal feltöltve, mint a változás hatóköre, és várható hatása. Pl: "Ez a dolog a tárolt eljárásokat is érintheti, amelyek erről és erről a helyről is meghívhatóak, és ez a speciális mód csak akkor elérhető, ha az első paraméter NULL". Ez segít a regressziós teszteknél a változások esetében. Természetesen kell, hogy a megjegyzéseknél tovább tudjunk gondolkodni - mint ahogy a fejlesztők is azt mondják el, amit tudnak - és a regressziót oda is kiterjesszük, ahol az esetleges hiányosságokat érezzük.

Mindent elértünk azzal, hogy csapatként működtünk és a közös munkát támogattuk. Nyíltan és elfogadóan álltunk egymáshoz, mint fejlesztő és tesztelő. Ha ez nem látszik most az aktuális csapatodban, akkor kezdjetek el gondolkodni ebben az irányban a siker érdekében. Ha azt látod, hogy nem tudtok kilépni a „tesztelés a tesztelő problémája" mantrából, itt az ideje, hogy lépéseket tegyetek a változásért!

Mindig van lehetőség a fejlődésre

Ezeket az útmutatásokat követve, amiket itt megemlítettem, nagyon sok szervezetnek segítettem a regressziós teszteknél az idő és energia hatékonyabb felhasználásában akár a végső, release előtti stádiumban is. A legtöbb alkalmazás release-ben a csapatom számos regressziós tesztet azonosított, de ezek azokban a hónapokban készültek, amikor a fejlesztés folyt, és nem pedig az utolsó pillanatban, a kiadás előtti pánikhangulatban. A regressziós tesztek kialakításával a fejlesztés alatt azt értük el, hogy sokkal több időnk volt azokra a dolgokra koncentrálni, ami az üzlet számára igazán fontos és értékes.

Még mindig van hová fejlődni – nem könnyű mindent megoldani egy éjszaka alatt – de ha elfogadod a regressziós tesztelést, mint a projekt fontos alkotóelemét, meg fogsz döbbenni, hogy mennyire könnyű és egyszerű lesz használni. A tesztelőknek pedig lesz ideje a minőségre koncentrálni. ■

Szerző: ***Neil Studd***

Forrás:
http://www.ministryoftesting.com/2014/10/light-end-regression-testing-tunnel/

Neil Studd

Neil Studd különböző vállalati és kereskedelmi szoftvereket tesztelt az elmúlt tíz évben. Számos cégnél megfordult kezdve a Last.fm-től egészen az Oracle-ig. Jelenleg a Towers Watson-nál tesztel egy kockázatelemző szoftvert. Ő az egyik fő segítője a Weekend Testing Europe-nak, amely egy havi rendszerességű meetup, ahol a tesztelők együtt lehetnek, megvitathatják a különböző tesztelési technikákat, és számos élő tesztelési kihívással találhatják szembe magukat.

NE a végén fedezze fel a hibákat!

Passed Informatikai Kft.

A hibák többsége az alkalmazás elkészítése alatt kiszűrhető, ezzel nagy mértékben csökkenthető a fejlesztési folyamat költsége!

Szoftvertesztelési szaktudásunkkal támogatjuk, hogy ügyfeleink kritikus üzleti alkalmazásai hatékonyan, megbízhatóan működjenek minden körülmény között.

Passed
Informatikai Kft.

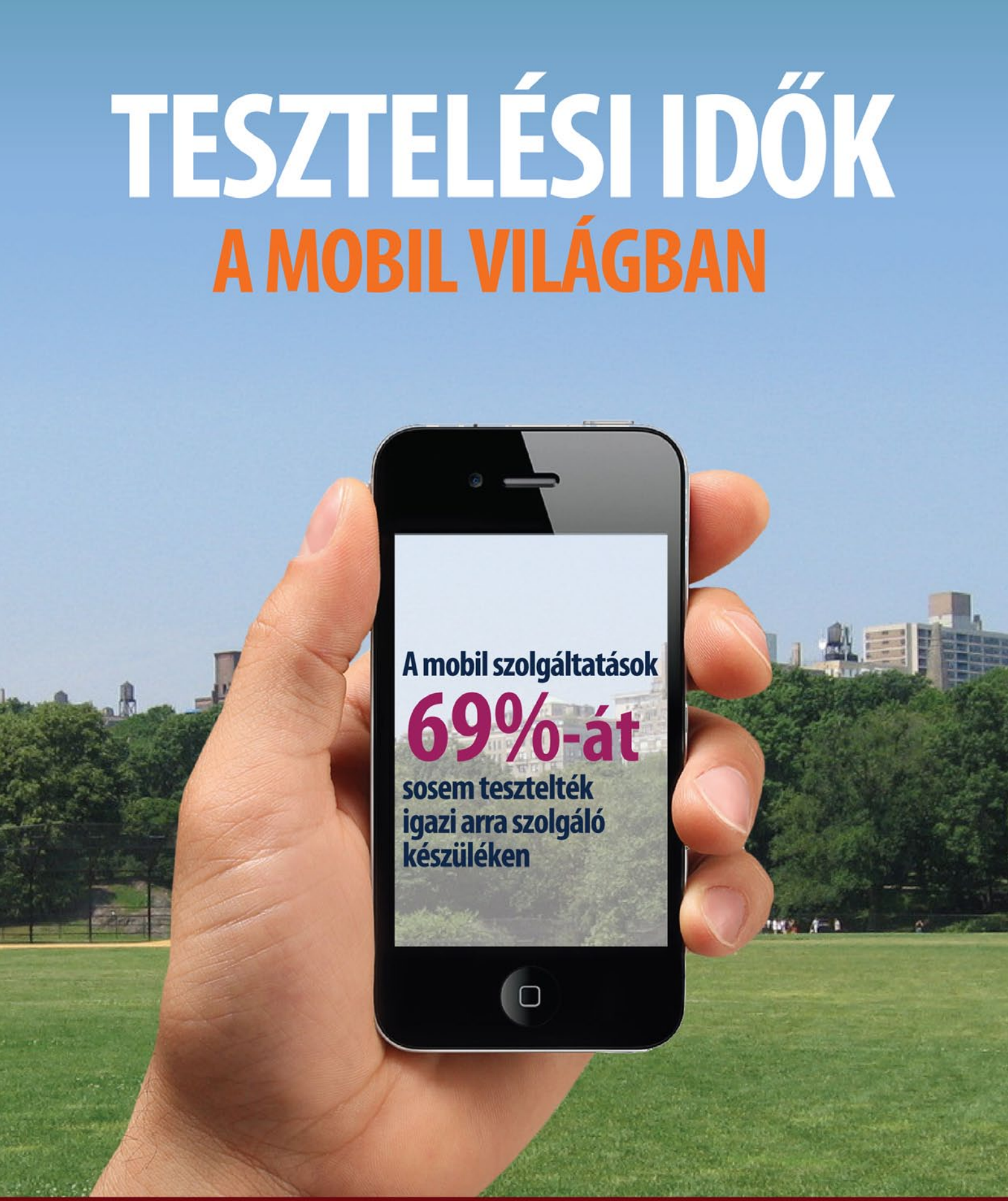
A megbízható tesztcsapat!

www.passed.hu

Publikálj nálunk!

TESZTELÉS A GYAKORLATBAN
A SZAKÉRTŐ TESZTELŐK LAPJA

TESZTELÉSI IDŐK
A MOBIL VILÁGBAN



Borland®
www.borland.hu