TESZTELÉS A GYAKORLATBAN

A SZAKÉRTŐ TESZTELŐK LAPJA

2018/III. szám

Szoftvertesztelés és gyártásoptimalizálás

Tökéletesítenéd tesztelési folyamataidat?

Minden cég arra törekszik, hogy a **szoftvertesztelésben maximalizálja** a minőségi és mennyiségi eredményeit. A piacon számtalan lehetőségből választhatsz.

Azonban, ha a **legjobb eredményt** szeretnéd elérni, csak egy megoldás létezik.

A **SpiraTest®** használatával egyszerűen, költséghatékonyan, egyedülálló módon javíthatod tesztelési folyamatodat.

Bővebb információért keresd a **SpiraTest®** hivatalos magyarországi forgalmazóját.

Passed Informatikai Kft.
www.passed.hu
+36/1/789-2525

© Copyright 2006-2008, Inflectra Corporation
SpiraTest and Inflectra are registered trademarks of Inflectra Corporation

Kedves Olvasó!

A jelenlegi magyarországi helyzetet nevezhetjük akár munkaerőpiaci válságnak is, habár nem az elnevezés számít. Nem célom az okokat keresni, inkább nézzük meg milyen folyamatok zajlanak a szakmában. A kivándorlás nagyrészt lezajlott, manapság egyre kevesebb hírt hallani olyan emberekről, akik tapasztalt tesztelőként külföldre igyekeznek. A megüresedett helyekre nagyon sok kezdő tesztelő jelentkezett az elmúlt időszakban. Ez a folyamat ma sem ért véget, mai napig sokan próbálják ezt a szakmát választani többkevesebb sikerrel.

Azok akik régről maradtak és tesztelési tapasztalattal bírnak, olyan tesztelési technikákat kezdtek el megtanulni / használni, amelyek piacképesek. A piac átalakult, manapság nem elég általános tesztelőnek lenni, specializálni kell magunkat. Számukra van pár automatizálásról szóló cikk a jelenlegi magazinban, de éppúgy találhatnak módszertani újdonságokat is.

A nemrég kezdő, vagy leendő tesztelőknek is tudunk érdekességgel szolgálni, hiszen nem egy cikk szól arról, hogyan válljunk tesztelővé, milyen készségek kellenek hozzá és milyen ismeretekkel kell rendelkeznünk ha tesztelők akarunk lenni.
Így tehát nyugodt szívvel ajánlom a mostani számot a tapasztalat és a teljesen kezdő (leendő) kollégáknak is!

Köszönöm!

Szőke Ármin

IMPRESSZUM

Kiadja:
Passed Informatikai Kft.

Szerkesztőség:
Főszerkesztő
Szőke Ármin
info@tesztelesagyakorlatban.hu

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

Internet:
www.tesztelesagyakorlatban.hu

Twitter:
@tesztAgy

Kapcsolat:
1095 Budapest,
Tinódi utca 1-3. C 1/9.
info@tesztelesagyakorlatban.hu

Szerzői jogok

Azzal, hogy belép a www.tesztelesagyakorlatban.hu oldalára, elfogadja az alábbi feltételeket, még abban az esetben is, ha nem regisztrált felhasználója, előfizetője a rendszer egyik szolgáltatásának sem:
A **www.tesztelesagyakorlatban.hu** webszájton („lap") található tartalom a SZERZŐ(K) és a („kiadó") szellemi tulajdona.
Az **Passed Informatikai Kft.** fenntart minden, a lap bármely részének bármilyen módszerrel, technikával történő másolásával és terjesztésével kapcsolatos jogot. A laphoz tartozó oldalak tartalmát és kialakítását nemzetközi és magyar törvények védik.
A kiadó előzetes írásos hozzájárulása nélkül tilos a lap egészének vagy részeinek (szöveg, grafika, fotó, audio- vagy videoanyag, adatszerkezet, struktúra, eljárás, program stb.) feldolgozása és értékesítése. A lap tartalmának egyes részeit - kizárólag saját felhasználás céljából - merevlemezére mentheti vagy kinyomtathatja, de ebben az esetben sem jogosult a lap így többszörözött részének további felhasználására, terjesztésére, adatbázisban történő tárolására, letölthetővé tételére, kereskedelmi forgalomba hozatalára.
A Tesztelés a Gyakorlatban Magazin oldalai teljes egészében, a reklámokkal, hirdetésekkel együtt szerzői jogvédelem alatt állnak, azokból bármely részt kivágni, a megcsonkított részt pedig nyilvánossághoz bármely módon újraközvetíteni tilos. Tilos továbbá a kiadó előzetes írásbeli engedélye nélkül a lap tartalmát tükrözni, azaz technikai művelet segítségével nyilvánossághoz újraközvetíteni, még változatlan formában is.
A jogosulatlan felhasználás büntető- és polgári jogi következményeket von maga után. Az Tesztelés a Gyakorlatban Magazin követelheti a jogsértés abbahagyását és kárának megtérítését.
A laptól értesüléseket átvenni csak a lapra való hivatkozással lehet, azzal a feltétellel, hogy az átvevő
a) nem módosítja az eredeti információt,
b) a lapra utaló egyértelmű hivatkozást minden közlésnél feltünteti.
Az Passed Informatikai Kft pontos és hiteles információk közlésére törekszik, de a tájékoztatásból fakadó esetleges károkért felelősséget nem vállal.

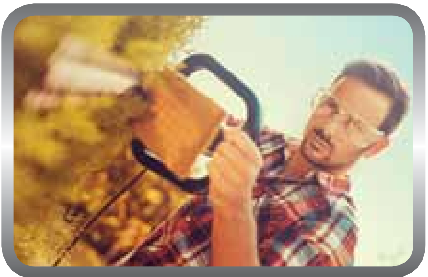
TARTALOMJEGYZÉK

6 AUTOMATA
Ali Khalid

Az automatizálás tényleg költséghatékony?
Úgy érzem a "költségcsökkentés" jelen lehet az automatizálásban, de nem abban az értelemben ahogy a legtöbben gondoljuk, és ez megváltoztathatja az automatizálásról alkotott képünket. Az egyik probléma az, hogy az automata tesztek futásának idejét egy manuális tesztelő munkaóráihoz hasonlítjuk. A másik pedig az, hogy alapvetően kevés a szoftvertesztelői erőforrásra egy projektnél.

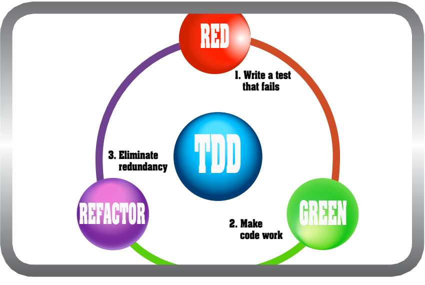
8 AUTOMATA
Fritzus Michael

Gyomláld ki az automatizációd!
Olvastam egy cikket nemrég arról, hogy milyen fontos az automatizációd napra készen tartása. A tesztek, mint az étel, úgy a legjobbak, ha frissek.



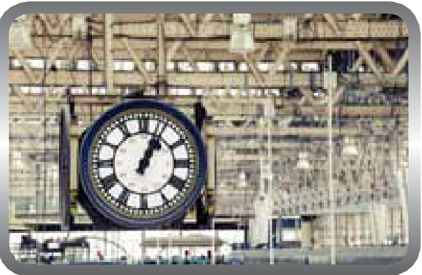
12 AUTOMATA
David Bernstein

3 módszer, hogy elsajátítsuk a teszt-vezérelt fejlesztést
A professzionális szoftverfejlesztők évtizedes tanításával felfedeztem, hogy a TDD elsajátításának három fő összetevője van: megérteni, hogy mi is ez valójában; a kódot megbízhatóan tesztelni; és gyakorlatot szerezni a TDD-ben.



14 MÓDSZERTAN
Schaffhauser Balázs

Szoftvertesztelés és gyártásoptimalizálás
Elsőre kicsit távol állnak egymástól. Holott céljaikban sok hasonlóság van, egymást remekül kiegészítik. Sok, változatos és érdekes út létezik a minőségbiztosítási szakmában; alábbiakban saját nem mindig tudatos döntéseken alapuló szubjektív tapasztalataimat szeretném megosztani.



18 MOBIL
Kristin Jackvony

A mobil 12 kihívása
Még csak egy évtized telt el az első iPhone megjelenése óta, de már olyan korban élünk, ahol az okostelefonok mindenhol jelen vannak. Mobiljaink olyanok, mint egy jó svájci bicska, rengeteg funkcióval szolgálnak. Milyen kihívásai vannak egy alapos mobil-tesztelésnek?

The testing company

TARTALOMJEGYZÉK

22 Módszertan

Matthew Heusser

Következő generációs gyakorlatok szoftvertesztelőknek

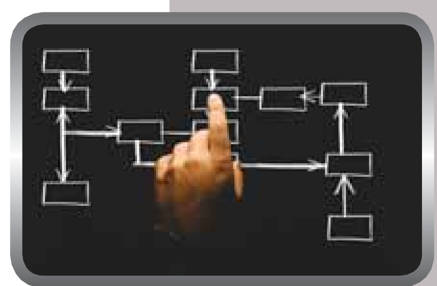
Az Internet előtt is léteztek tesztelési feladatok, manapság viszont más készségekkel kell rendelkeznie egy tesztelőnek. A cikkben találsz olyan szoftvert, amelyen gyakorolhatsz. A program segít azon készségeinket is használni, amik ahhoz szükségesek, hogy igazi tesztelővé váljunk.

24 Módszertan

Viktor Slavchev

A manuális tesztelésnek se múltja, se jövője. Soha nem is létezett!

Ha esetleg a blogolvasók számára nem lenne világos, mert erről általában konferenciákon beszélek – abszolút nem értek egyet a 'manuális tesztelés' kifejezés létjogosultságával és szükségességével.



26 Módszertan

Michael Stahl

Tesztelői döntéshelyzet: duplikált hibajegyek

Mikor lehet eldönteni, hogy egy hibát fölösleges rögzíteni mert már szerepel a hibajegykezelő rendszerben? Ha megkérdezed a fejlesztőktől és két hiba ugyanazért a hibás kód miatt eredményez helytelen működést, ezeket a hibákat elég csak egyszer rögzíteni. Michael Stahl azonban jó érvekkel támasztja alá, hogy tesztelő szemszögéből miért jobb inkább több hibát rögzíteni.

28 Teszteszköz

Szőke Ármin

AutoHotKey

A napi munka legkevésbé kedvelt részei, mikor ugyanazon műveleteket kell végrehajtani nap, mint nap: bejelentkezés alkalmazásokba, azonos karaktersorozatok gépelése, ugyanazon űrlapok kitöltése. A monotonitás az alkotómunka elől veszi el a levegőt. A jó hír az, hogy létezik segítség.

30 Oktatás

Thanh Huynh

7 általános szoftvertesztelési tevékenység, amit ismerned kellene kezdés előtt

A szoftvertesztelési ipar olyan ütemben nő, mint azelőtt soha. Nem meglepő, hogy emiatt egyre több és több ember szeretne tesztelővé válni. Milyen is maga a szoftvertesztelés? Milyen tesztelési feladatok várnak rád?

The testing company

Az automatizálás tényleg költséghatékony?

Úgy érzem a "költségcsökkentés" jelen lehet az automatizálásban, de nem abban az értelemben ahogy a legtöbben gondoljuk, és ez megváltoztathatja az automatizálásról alkotott képünket. Az egyik probléma az, hogy az automata tesztek futásának idejét egy manuális tesztelő munkaóráihoz hasonlítjuk. A másik pedig az, hogy alapvetően kevés a szoftvertesztelői erőforrásra egy projektnél.

Mint sok más ember régebben én is úgy számítottam ki az automatizálás ROI-ját, hogy megmértem mennyi munkaórát takarítunk meg, ha az ember helyett a gép végzi el a teszteket.
Talán emiatt alakult ki a jelenlegi piaci trend és ez biztosította, hogy a szállítók eladhassák termékük, szolgáltatásaikat. Miután évek óta ebben az iparágban dolgozom és figyelemmel kísérem más vezetők és emberek közösségen belül megosztott véleményeit, úgy érzem a "költségcsökkentés" jelen lehet az automatizálásban, de nem abban az értelemben ahogy a legtöbben gondoljuk, és ez megváltoztathatja az automatizálásról alkotott képünket.
Hogy egy cseppet egyértelműbbé tegyem, Te mit mondanál egy átlag embernek mennyi egy zongora ROI-ja? Nem egyszerű számszerűsítseni egy 'eszköz' befektetésének megtérülését, de ez nem azt jelenti, hogy ez kevésbé fontos.

A csodafegyver

Évtizedeken keresztül az automata teszteszközöket és szolgáltatásokat úgy értékesítették, mintha azok a költségcsökkentés módszerei lennének. Ez elméletileg logikusnak tűnik, habár évek óta az iparban dolgozva nem ismerek senkit (beleértve magamat is), aki valóban érzékelte volna ezt a költséghatékonyságot. Elemezzük csak a költségcsökkentés számítást, és próbáljuk meghatározni az okokat.

A formula

Az egyenlet nagyvonalakban ilyesmi szokott lenni:

```
Tesztciklusonkénti megtakarítás = Automatikus tesztek száma * Manuális végrehajtási idő (emberóra) tesztenként
```

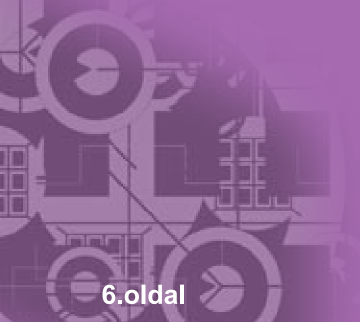
És akkor kiszámítjuk a megtérülési pontot, amikor is a megtakarításunk eléri a kezdeti beruházás költségek értékét, amibe az automata teszthalmaz kialakítása és az egyéb költségek tartoznak. Egy könyvelőnek ez tökéletes lenne kivéve, hogy a "Manuális végrehajtási idő tesztenként" költség nem létezik! Hadd magyarázzam meg!

Automatikus ellenőrzés vs. tesztelés

Az első probléma az, hogy az automata tesztek futásának idejét egy manuális tesztelő munkaóráihoz hasonlítjuk. A számítógép szkript futtatásának módja nem egyezik meg azzal, ahogyan egy tesztelő tesztelné ugyanazt a funkciót. Ezt a koncepciót rengeteg dolog támasztja alá, ha ismerned az RST-t (Rapid Software Testing) és a hozzá kapcsolódó fogalmakat. Amennyiben mégsem vagy járatos benne, akkor megpróbálom gyorsan összefoglalni ezeket:
A "Tesztelés" ige "Gondolkodás"-on és "Kommunikáció"-n alapuló cselekvés jelent, amely arról szól, hogyan kell megvizsgálni egy adott funkciót. Miután a tesztelő eldöntötte mit teszteljen, akkor végrehajtja a forgatókönyveket. A gép nem tud tesztelni, mivel nem képes gondolkodni és kommunikálni mint egy ember. Csak utasításokat hajt végre. (Köszönet az RST közösségnek, James Bach-nak, Michael Bolton-nak és a többieknek az egyértelmű megfogalmazásért.)

A hiányzó erőforrás

Vegyünk egy példaalkalmazást, amely rendszernek a teljes tesztelése megközelítőleg 1000 emberórát igényelne (gondolom sok termékre illik ez a leírás). Hány tesztelőre lenne szükségünk, ha a teljes alkalmazáson



2 héten alatt szeretnénk regressziós tesztelést végrehajtani? Megközelítőleg 13 teljes munkaidős tesztelőre. Szerinted általában egy csapatban van 13 tesztelő? Többnyire nincs, a csapatoknak kevesebb emberük szokott lenni, mint amennyire szükségük lenne és annyit tesztelnek, amennyit a rendelkezésre álló tesztelők tudnak.
Most a "Tesztelés" fele volt "Gondolkodó", amit egy gép nem tud megtenni (egyesek szerint és szerintem is, sokkal több mint a fele). A másik felét feltételezhetően akkor a "Végrehajtás" volt, amire ténylegesen csak egy kis százalékot költöttünk, mivel a csapat mérete MINDIG kisebb a szükségesnél.
Így lehet, hogy van néhány emberóra megtakarításunk, de gyakorlatilag ez semmi, mivel a legtöbb csapat nem a fenti ROI számítás szerinti feltételezések alapján működik.

Akkor miért automatizálunk?

Növeljük a tesztlefedettséget

Az előző példában nem voltunk képesek a teljes alkalmazást letesztelni. Saját gyakorlati tapasztalataim alapján rengeteg az ilyen "alultesztelt" termék. Amelyeknél általában egy tucatnyi tesztelő hozzáadása sem oldja meg a helyzetet.
Hogy nagyobb lefedettséget érjünk el, a tesztelők készíthetnek egy buta végrehajtó gépet, amely az egyszerű, unalmas, sokszor végrehajtandó teszteseteket futtatja minden egyes release kijövetelénél. Ezáltal a felszabadult idejükben intelligens munkát végezhetnek, mivel az ismétlődő feladatokat a gép végzi el.

A lényeges területekre koncentrálhatunk

Az előző bekezdés ismételhetőnek tűnik, de van itt egy apró ám fontos dolog. A tesztelők nem csak felszabadítják az idejüket, de egyben kihasználhatják a gépet, hogy csak a "Gondolkodó" részre koncentrálnak és a "Végrehajtó" részt minél nagyobb arányban a gépre bízzák.
Nem biztos, hogy nagy százalékban tudják a végrehajtási feladatokat delegálni, de ha ügyesen használják az automatizálást, akkor a tesztelés minősége jelentősen javulni fog, mivel több időt fordítanak gondolkodásra, mint az ismétlődő feladatok végrehajtására. Ha többet akarsz tudni azokról a tesztesetekről amelyek ideálisan automatizálhatóak, akkor nézd meg ezt a videót.
(https://youtu.be/nEGvb7ZMCj8)

Találd meg a problémát hamarabb

Hányszor fordult már elő, hogy egy hiba javítása után egy másik fontos funkció leáll, természetesen az utolsó pillanatokban, amikor a hiba felderítésére és kijavítására már nincs elég idő.
Éppen ezért nagyon sok haszna van a gyors visszajelzésnek. A fejlesztési folyamat különböző szakaszaiban zajló különböző ellenőrzések biztosítják ezt a fajta gyors visszajelzést. Egy jó példa erre, amikor unit teszteket és magas szintű funkcionális teszteket futtatunk a dev környezeten, teljes regressziós tesztet és részletes funkcionalitást ellenőrzünk a QA környezeten, felhasználói és átadás/átvételi teszteket a production-ön.

Nagy lépés a Continuous Delivery felé

A legsikeresebbek azok a fejlesztő cégek, akik a tervasztalon felvázolt ötletekből a lehető leggyorsabban gyártanak termékeket és adják azokat az ügyfélek kezébe. Itt jön képbe a Continuous Delivery. A piacra kerülési idő minimalizálására irányuló verseny győztesei óriási előnyre tehetnek szert. Ez a videó részletesen leírja, hogy az automatizálás hogyan könnyíti ezt meg.
(https://youtu.be/JdddbPlpmbE)

Megnövekedett bizalom a termékben

A felfedező tesztelés hátránya és egyben előnye is, hogy emberek végzik. A tesztelő egy funkciót mindig máshogyan tesztel le, sőt néha hajlamos elfelejteni ellenőrizni azt. Az automatizált ellenőrzésekkel biztosak lehetünk abban, hogy a funkciókat megvizsgáltuk és hogy azok működnek, vagy sem.
Az automatizáció megkönnyíti a döntést egy release kiadásáról mivel egy mennyiségi mérőszámot ad a kezünkbe, amelyre a döntésünket alapozhatjuk. Habár ez önmagában nem elegendő a döntés meghozatalához, de alapos felfedező teszteléssel párosítva már nagymértékben támaszkodhatunk rá.
Nem csak a csapat, de az ügyfelek is elégedettek lehetnek azzal, hogy tudják bizonyos tesztek automatizálva vannak, mivel ezzel biztosítjuk, hogy a funkcionalitás nagy valószínűséggel végigment az ellenőrzési folyamatunkon.

A minőség iránti elkötelezettség

A fenti példában láthattuk és a saját tapasztalataim alapján is kijelenthetem, hogy a legtöbb csapatnak nincs elég tesztelői munkatársa teljes egészében ellenőrizni az alkalmazást, amikor abban valamilyen változás történik. Sokan azt állítják ez amúgy is kivitelezhetetlen. Az automatizálás használata az egyes területeken, megmutatja, mennyire elkötelezetten biztosítjuk a rendszer különböző pontjain, hogy az a kiadás előtt ellenőrizve, vagy tesztelve legyen.
Ez az ahonnan a "Minőségi gondolkodásmód" kifejezés származik. Ahoz, hogy magunkat és termékünk is magas színvonalon tartsuk előbb-utóbb szükséges elmélyednünk valamilyen automatizálási folyamatban, mivel a legmodernebb alakalmazásokat nem lehet megfelelően tesztelni egy "költséghatékony" méretű csapattal.

Léteznek megtakarítások

Az emberórák és a gépi órák egymással szembe állítása nem a legjobb formula az automatizálási projekt beruházási költségeinek megtérülésének a meghatározásához. A megtakarítás nem az emberórákban keletkeznek, hanem más formákban amely nem azt jelenti, hogy ezek kevésbé fontosak lennének, csak kevésbé nyilvánvalóak.
A valódi érték a megnövekedett tesztlefedettségből származik, amely lehetővé teszi a tesztelőknek, hogy a számítógépre delegálják a feladatokat, miközben gyorsabb visszajelzést kapnak az alapvető funkciókról, ami fontos mérföldkő a piacra kerülési idő csökkentése szempontjából, valamint növeli az ügyfél és a csapat termék minőségébe vetett bizalmát. Mindez a minőség iránti elkötelezettséget mutatja, mely hatással van a végfelhasználóra. ■

Szerző: ***Ali Khalid***

Forrás: http://quality-spectrum.com/does-automation-save-money/

Ali Khalid

Szeretné a szoftvertesztelési szakmát előremozdítani az automatizációra specializáltan. Gyakran podcast-okon és meetup-okon előadóként jelenik meg, embereket képez automata keretrendszerek fejlesztésére és különböző újságokba, tesztelési magazinokba ír cikkeket. Egy évtizednyi tapasztalattal bír a szoftvertesztelési iparban, ahol több mint 20 féle terméket tesztelt, köztük nagy volumenű vállalatirányítási rendszereket (ERP), SaaS webes applikációkat és beágyazott eszközöket, miközben különféle menedzsmenti, tesztautomatizációs és tesztelési kihívásokat oldott meg.

Gyomláld ki az automatizációd!

Olvastam egy cikket nemrég arról, hogy milyen fontos az automatizációd napra készen tartása. A tesztek, mint az étel, úgy a legjobbak, ha frissek.

Mikor megpróbálsz rájönni arra, hogyan vond az automatizálási sort az irányításod alá, – megszabadítva azt az elavultságától – én a folyamatot egy sövényhez tudnám hasonlítani. Míg a megromlott ételt általában kidobjuk, addig a sövényt időről időre megnyírjuk, így őrizvén meg szépségét.

Továbbá, a cikkem írásához ezt a háttérképet találtam és már akkor tudtam mi lesz a címe, így maradok ennél a témánál.

Az egyik legfontosabb kérdés a cikkhez az volt, hogy milyen stratégiákat javasolnának az emberek arra vonatkozóan, hogy megszabaduljunk az ehhez hasonló tesztektől?

Mik is ezek és hogyan történnek?

Először is tudnunk kell, hogy mit is jelent egyáltalán ez a fogalom. Hogy mondhatja egyáltalán bárki is azt, hogy „ezt a tesztet ki kell gyomlálni”? Milyen tesztek illenek a kritériumokba?

Másodszor, az is fontos, hogy tudjuk, hogyan is lehet megelőzni a tesztek elavulását a megelőzés érdekében. Ha meg tudod akadályozni elsősorban az automatizált tesztek elavulását, az fogadok, hogy nagymértékben megoldaná a problémád igen nagy részét.

Egy teszt akkor áll készen a gyomlálásra, ha már nem bizonyul megfelelően értékesnek a korábbi értékéhez képest.

Az alábbi példákat a mostani munkád és kódbázisod lencséjén keresztül olvasd el. Nem mindegyik fog mindenki számára működni.

Már van egy teszt, ami valami hasonlót csinál

Ok: Duplikált tesztesetek előfordulhatnak, ha több ember függetlenül ír teszteket. Vagy talán csak egy feledékeny személyről van szó.

Megelőzés: Végezzetek folyamatosan kód felülvizsgálatokat, vagy kérj felülvizsgálást időnként, hogy az emberek ne csak arra figyeljenek, hogy milyen teszteket írnak, hanem arra is, hogy azok milyen fajtájúak.

Gyomlálás: Jöjj rá arra, hogy mik a tesztek legfontosabb részei, és vagy szabadulj meg az összestől egyet kivéve, vagy rakj össze egy új tesztet. Aztán tudasd a többiekkel, hogy mit alkottál.

A teszt a rendszer egy nem használt részét vizsgálja

Ok: Amikor új elemeket implementálnak, sok minden történek egyszerre. A fejlesztők változtatnak és az implementációkat újra kitelepítik, a teszteket futtatják és újrafuttatják. Az

automatizáció segít a változtatások gyorsabb tesztelésében egy ilyen új kitelepítés esetén. De amint az aktivitás a rendszer egy másik részére mozdul át, ezek a tesztek továbbra is megmaradnak, továbbra is futnak.

Megelőzés: Vegyél egy nagy levegőt és emlékezz, hogy nemsokára ez a része a rendszernek szilárdabbá fog válni. Állapíts meg egy kisebb számú tesztesetet, amikre szükséged lesz, hogy továbbra is bizonyítani tudd, hogy a rendszer ezen része működik. Hagyatkozz a szemeidre és az agyadra a hibák megtalálásához.

Gyomlálás: Gyomláld ki visszamenően a teszteseteket, amelyek a rendszer ezen részét vizsgálják, míg a teszteknek csak egy kicsi, de továbbra is hasznos része marad meg. Teszteld ezeket az eshetőségeket alap szinten, csak hogy biztos legyél abban, hogy nem romlott-e el valami elképesztő módon időközben. Vagy, fontold meg ezen tesztek megjelölését valamilyen módon, hogy ne fussanak le folyamatosan, csak akkor, amikor szükség van rájuk.

„Ha már itt vagyunk…”

Ok: Amikor a teszt automatizálás megírása kihívásnak ígérkezik, lehet, hogy elkezdesz meglátni hasonló adatok tesztelésének olyan kombinációját, ami neked már elégséges lenne, csak azért, hogy mihamarabb túl legyél az automatizálás megírásán. Valójában, ez néha csak a lefedésről/átvilágításról/teszt szám növeléséről szól. Nincs semmi bajom a például a JUnit Theories vagy a Scenario Outlines típusú technikákkal, de ezek termékeny földet jelentenek ennek a rossz mintának a terjedéséhez.

Megelőzés: Dolgozz azon, hogy olyan tesztautomatizálásokat írj, amelyek egészen egyszerűek. Ez nemcsak hogy felgyorsítja azt az időt, ami a megíráshoz szükséges, de mentálisan is az emberek könnyebben veszik egy olyan teszt megírását, amely nem kerül óriási időbefektetésbe.

Gyomlálás: Ez hasonló a duplikált tesztek gyomlálásához. Tedd fel a kérdést: Ez a teszt tényleg fog találni valami olyasmit, amit egy másik (már meglévő) nem? Ha a válasz „nem”, akkor töröld az egyiket a kettőből. Vagy ha tényleg szükségét érzed (és érthető okból), akkor állítsd úgy át a maradék teszteket, hogy minden futtatásnál véletlenszerű adatokkal dolgozzanak.

Van egy „félelem szag” a tesztekkel kapcsolatban

Ok: Néhány ember egy csomó tesztet ír attól való félelmében, hogy valamit kihagy. És senki sem tudja igazán, hogy mit is keres, csak biztosak akarnak lenni abban, hogy semmi rossz ne történjen. Az eredmény, hogy néhány része a rendszernek igen keményen tesztelésre kerül az általános hibákra fókuszálva és a néhány terület igen kevés figyelmet kap. Ez valójában tényleg csak egy biztonsági érzetet ad nekünk, ahelyett, hogy biztosítana minket arról, hogy a kód használatra kész.

A Megelőzés (és a Gyomlálás is): Ez inkább egy szociális megoldás, mint bármi más. Az embereknek kevésbé kell attól félniük, hogy kevesebb tesztet írjanak, és arra kellene inkább a hangsúlyt fektetniük, hogy a tesztek, amiket írnak több esetleges hibafelületet is lefedjenek. Minden egyes tesztnél tedd fel a kérdést: Miért is írtam meg ezt a tesztet? Mit jelent az, ha hibára fut? Ha egy teszt nem igazán bír értékkel, akkor töröld nyugodtan.

A tesztelés egy kevésbé kockázatos részét vizsgálja a rendszernek

Ok: Néhány céges környezetben elvárt, hogy egy közel 100%-os lefedettséget sikerüljön produkálni minden téren. De ha őszinték vagyunk magunkkal szemben, nincs szükség ilyen mértékű lefedettségre. Még tovább is mehetünk azzal, hogy azt mondjuk, hogy néhány hiba… nem igazán elfogadható, azonban sokkal kevesebb kárt okoz, mint néhány más hiba, amit ki tudunk szűrni.

Megelőzés: Azonosítsd be a rendszer azon részeit, amelyek igazán rizikósnak ígérkeznek és helyezd a tesztelési hangsúlyt magasabbra ezeken a helyeken. De más esetben tedd fel a kérdést: Mi történne, ha [ez a kifejezett hiba] jelenne meg [ezen] a helyen? Ez a világ végét jelentené? Sok pénzt veszítenénk ezzel? Csak egy kisebb fejfájást okozna a probléma? Egyáltalán az ügyfél észrevenné ezt?

Gyomlálás: Vedd figyelembe az ilyen tesztek megjelölését/karanténba helyezését és nézd át őket a csapatoddal. Történetesen, jött már elő ez a hiba korábban, és ha igen, mit (ha egyáltalán bármit) szólt hozzá az ügyfél? Ha ez egy eléggé alacsony rizikó, csak töröld. Mérlegeld azt a tényt, hogy az ügyfelek megbecsülik azokat, akik gyorsan meg tudják oldani a felmerülő problémákat, szóval, ha valamilyen kevésbé fontos probléma mégis felkelti az ügyfél figyelmét és ez gyorsan kijavítható, az még jól is jön ki a te számodra.

Ez egy kozmetikai változást tesztel

Ok: Néha szükségünk van arra, hogy egy kozmetikai változtatást eszközöljünk az interfészünkön. Talán egy elem helye vagy típusa megváltozott. Egy automatizált teszt ezen a téren nem bír túl nagy értékkel, mivel ezek a fajta változtatások nem történnek meg túl gyakran. Ha ez a teszt mégis hibára fut, az inkább egy átívelőbb problémára világítana rá, mint arra, hogy „az elemnek nem megfelelő a színe”. Ehelyett a hiba oka lehetne például az, hogy „az oldal valójában azért nem töltött be, mert a szerver éppen leállt”.

Megelőzés: „Attól, hogy valamit megtehetsz, még nem biztos, hogy meg kellene tenned”. Több junior automata tesztelőt is találhatsz, akik ilyen

ÁLLÁSHIRDETÉS

A Passed Informatikai Kft. munkatársat keres

Szoftvertesztelő

munkakörbe.

FELADATAI:
- Ügyfeleink informatikai rendszereinek tesztelésében való aktív részvétel.
- A fejlesztett szoftverek tesztelési terveinek elkészítése, tesztelések dokumentálása.
- A backend és frontend szoftverek különböző teszt fázisaiban való aktív közreműködés.
- Éles üzem során felmerülő hibák kivizsgálása, javításának és termelési rendszerbe történő bevezetésének végig követése.

ELVÁRÁSOK:
- Felsőfokú iskolai végzettség
- Középszintű angol nyelvtudás szóban és írásban egyaránt
- Hasonló munkakörben szerzett minimum 1 éves tapasztalat

ELŐNYT JELENT:
- ISTQB CTFL minősítés
- Webes alkalmazások tesztelésében szerzett tapasztalat
- Automata tesztek végrehajtásában vagy tervezésében szerzett tapasztalat
- Teszteszközök ismerete

AMIT AJÁNLUNK:
- Stabil munkahely
- Hosszú távú munkalehetőség
- Folyamatos továbbképzési lehetőségek
- Karrierépítési lehetőség

MUNKAVÉGZÉS HELYE:
Budapest

Önéletrajzokat az info@passed.hu email címre várjuk!

Intelligens tesztautomatizálási keretrendszer

Felhasználóbarát felület
Szaktudás nélkül is használható
Kulcsrakész, automatizált riportok
Adatvezérelt, kulcsszóvezérelt
Moduláris felépítés

www.mortoff.hu

teszteket írnak, mert ez újszerű és ők ezt képesek erre. És igen, ez egy jó gyakorlás, azonban hosszútávon nincs sok haszna, mert ezután ezt folyamatosan karban kell tartani. Előzd meg ezen tesztek elkészítését azzal, hogy egyszerűen beszélsz arról, hogy mit kell/mit nem kell automatizálni, de figyelj arra, hogy az embereknek így is elég érdekes munkát adj, hogy a kreativitásukat, melyekkel ilyen teszteket írnának, hasznos dolgokba fektessék.

Gyomlálás: Komolyan, itt azzal, hogy alapvetően ezek a tesztek megírásra sem kerülnek, ki tudod küszöbölni ezt a fajta problémát.

A tulajdonság, amit teszteltek, nagyon érett

Ok: Amint a kód a nyilvánosság elé kerül, az ügyfelek olyanok, akár egy seregnyi öntudatlan tesztelő. Ha találnak valami hibát a termékkel kapcsolatban, biztos lehetsz benne, hogy hallani fogsz róla. A tesztek, melyek ezt az adott tulajdonságot vizsgálták, valószínűleg még mindig léteznek, még akkor is, ha ez a része a rendszernek már egy ideje nagyszerűen működik.

Megelőzés: Először is, amikor írod a tesztjeidet, jelöld meg őket valamilyen formában, hogy később elkülöníthesd őket. Amint a rendszer egy része már egy ideje a közönség előtt van anélkül, hogy panasz érkezne rá, abbahagyhatod ezen tesztek futtatását. Beszélj a csapatoddal arról, hogy erre mi lenne a megfelelő időbeli célkitűzés – egy hónap? Három hónap? Egy év? Ha az ügyfelek egy jókora csoportja - akik alapvetően tömegtesztelik a terméket - nem talált semmit, akkor eléggé valószínűtlen, hogy a további folyamatos tesztelések fognak bármit is.

Gyomlálás: Ha egy implementáció már egy ideje (és megegyezéses alapon) a tesztközönség előtt van, csak töröld azt a tesztet teljesen.

„Isten” tesztek

Ok: Néha fogsz találni olyan óriási teszteket, amik sok mindent csinálnak egyszerre. Talán ezek azért születnek, mert „nos, bele kellene tennem valamivel több funkcionalitást néhány helyre és aztán ez is működni fog” vagy, „nem akarom folyton ledönteni és újra előállítani az új esetet, úgyhogy időt spórolok azzal, ha mindent egy nagy tesztbe építek be”. Minden egyes sor, amit egy teszt kódba beleírsz, növeli az esélyét annak, hogy a teszt sikertelen lesz. És ha a teszt túl sokat tud, de már korai szinteknél hibát jelez, előfordulhat, hogy a többi része ennek az „Isten” tesztnek már talált volna valami mást is, amit így viszont kihagyott.

Megelőzés: Írj teszteket specifikus célok figyelembevételével és tegyél ezekbe leírást is. Azok a tesztek, amik sokkal többet tudnak, mint amit ezekbe a leírásokban foglaltak, intő jelei az ilyen típusú teszteknek.

Gyomlálás: Tördeld ezeket a nagyobb teszteket kisebbekbe. Gyorsabban tudnak különállóan futni (ami nagyon előnyös a felhőkezeléses és szimultán munkavégzés korában) és valószínűleg különállóan több hibát fognak találni, mint az egész együtt.

Hiba-alapú tesztek

Ok: Mikor egy hibát megtalálsz, szépen néz ki, ha az ismételt újrafuttatásnál a teszt sorozat azt mondja neked, hogy „nem ok”, amíg a hiba ki nincs javítva és így tudod, hogy mikor tudsz tovább haladni. Viszont a hibák általában meglehetősen helyhez kötöttek a kódban, és amint javításra kerülnek, valószínűleg nem fognak ugyanolyan módon elromlani. Ezen tesztek megtartása a továbbiakban kevés lefedettséget ad.

Megelőzés: Ha szükséged van az automatizálás során arra, hogy lásd mikor javítják ki a hibákat, az rendben van, viszont tartsd ezeket a teszteket elkülönítve a többitől, hogy egyszerűen leállíthasd őket (vagy csak töröld) a későbbiekben. Ideális esetben viszont valószínűleg jól jársz azzal, ha ezek a tesztek egység vagy integrációs szinten azért meg vannak írva.

Gyomlálás: Különítsd el ezeket a teszteket, majd beszélj a csapatoddal, hogy eldöntsétek mennyire rizikós a kód azon része, és hogy ez a hiba historikusan okozott-e már többször hibát. Ha a jelek kedvezőek arra, hogy törölhesd ezeket a teszteseteket, tedd meg.

...Tartsd meg a rózsákat

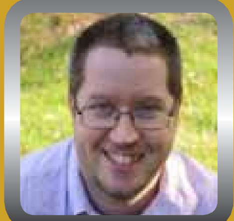
Remélhetőleg nem kezdtél el gyomlálni, miközben ezt a cikket olvastad, mert nagyon fontos, hogy „a rózsákat megtartsd” – azokat a részeit a teszteknek, amelyek érdekes kódolást vagy megoldásokat implementálnak. Tedd ezeket félre és bogozd ki őket később – a trükkök, amik ezekbe a tesztekbe implementálva vannak, lehet, hogy segítenek a későbbi tesztelésekben. ■

Szerző: ***Fritzius Michael***

Forrás: https://testzius.wordpress.com/2017/04/20/prune-your-automation/

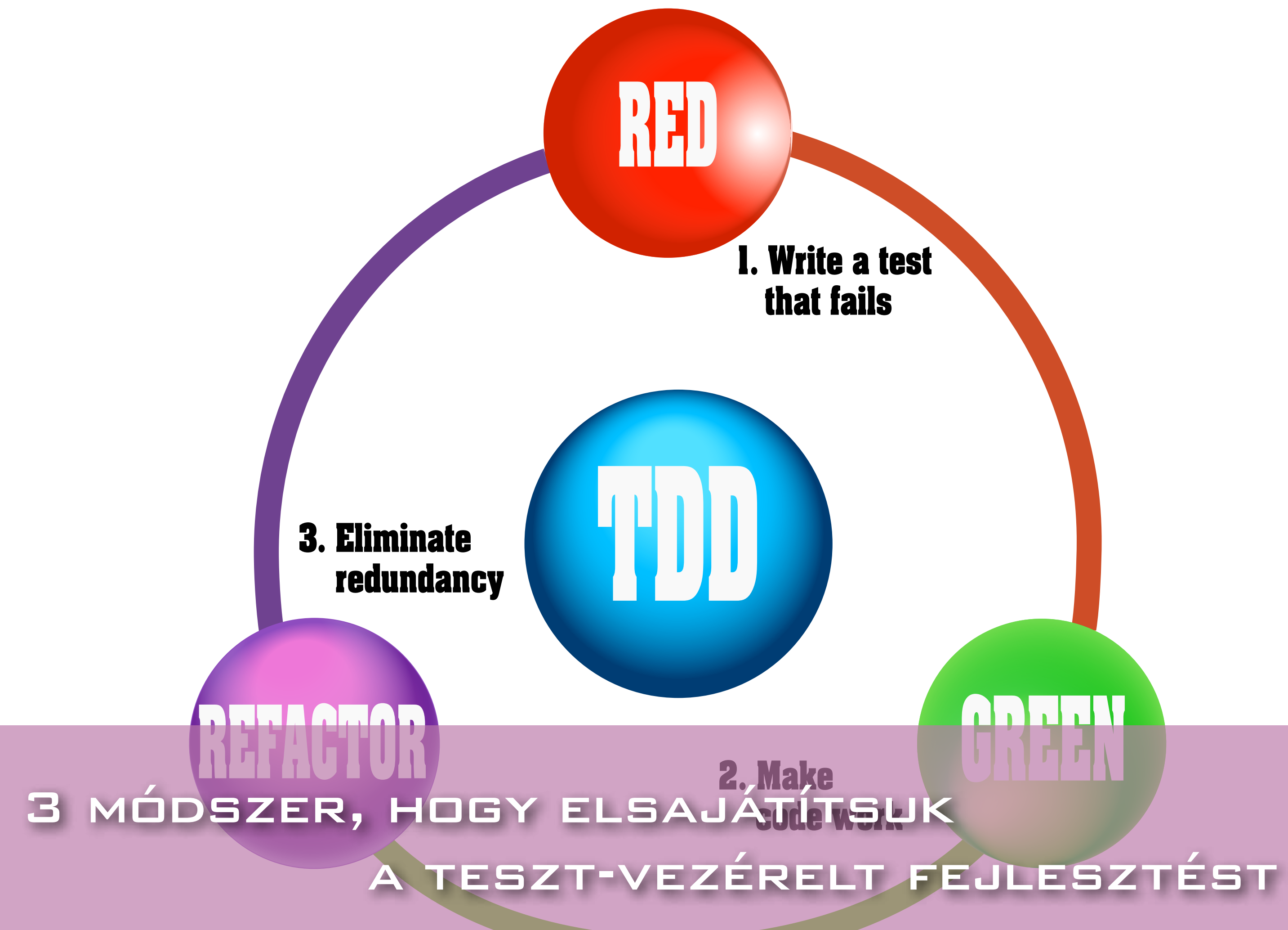
https://magenic.com/thinking/don-t-eat-stale-automation

https://testzius.files.wordpress.com/2017/04/hedge-trimmers.jpg



Fritzius Michael

Fritz egy minőségbiztosítási tesztelő és az automatizáció rajongója és az Arch DevOps vezérigazgatója, mely az USA-ban, Missouri államban, St. Louis -ban székel. Különféle nagyságú szoftvertesztelő csapatokkal dolgozik együtt, hogy az automatizáció technológiáját beemelje a tesztelési fázisba, emellett nagy volumenű keretfolyamat készítésének és karbantartásának elméletét oktatja.



3 módszer, hogy elsajátítsuk a teszt-vezérelt fejlesztést

A professzionális szoftverfejlesztők évtizedes tanításával felfedeztem, hogy a TDD elsajátításának három fő összetevője van: megérteni, hogy mi is ez valójában; a kódot megbízhatóan tesztelni; és gyakorlatot szerezni a TDD-ben.

Az elmúlt évtizedben kiváltságom volt többezer profi szoftverfejlesztőt oktatni arra, hogyan lehet hatékony a teszt-vezérelt fejlesztés (TDD). Ezekből a tapasztalatokból megtanultam, hogy három fő eleme van a teszt-vezérelt fejlesztésnek:

- megérteni, hogy mi is az valójában,
- hogy legyen a kód megbízhatóan tesztelhető
- és a gyakorlati tapasztalat

Nézzük meg mindegyiket, hogy megértsük, hogyan tudjuk a TDD-t hatékonyan használni a projektjeinél.

Megérteni mi a TDD

Az első kulcsfontosságú összetevő, hogy hatékonyan használjuk a teszt-vezérelt fejlesztést, hogy megértsük, mi is az valójában. Úgy találom, hogy sok tévhit létezik arról, hogyan kell a TDD-t megfelelően használni, és a TDD az egyik olyan gyakorlat, amit ha rosszul csinálunk, azért gyakran nagy árat fizetünk.

A TDD több, mint amit ez a rövid cikk mondhat róla. Az egyik dolog, amit észrevettem, hogy az emberek számára a legnehezebb az, hogy a TDD-t a tesztelés vagy a minőségbiztosítás egyik formájának gondolják. Úgy gondolom, hogy ez egy hibás gondolkodásmód, a TDD műveléséhez.

A QA elkötelezetten gondolkodik, hogy mi romolhat el, és hogy megtalálja a módját annak biztosítására, hogy ez ne történjen meg. A fejlesztői gondolkodásmód reményhetőleg optimistább, és arra összpontosít, hogy mi kell ahhoz, hogy a dolgok rendben menjenek.

Ahelyett, hogy arra gondolnánk, hogy a TDD-t a kód tesztelésének módjaként kell kezelni, arra kell gondolni, hogy a TDD-t a rendszer viselkedésének meghatározására használjuk. Ez arra késztet, hogy nagyon különböző fajta teszteket hozzunk létre, amelyek hajlamosabbak a jövőbeli változásokra, mivel inkább ellenőrzik a viselkedésmódokat, és nem pedig a kódot tesztelik.

Az unit test kifejezés kissé félrevezető lehet. Ha megírjuk a tesztet, mielőtt írnánk a kódot, akkor ez valójában nem teszt, mert amikor megírjuk, még nincs mit tesztelni. Kicsit furcsa, hogy ezt tesztnek nevezzük. Ehelyett szeretek ezekre hipotézisként gondolni. Amikor a kód megírása előtt írjuk meg a tesztet, feltételezzük, hogy hogyan fog viselkedni a program, mit kell átadni, és mivel fog visszatérni.

Ez hasonlít a tudományos megközelítéshez. Véletlenszerűen nem kísérletezünk. Mindig hipotézissel kezdünk: valamit bizonyítani vagy vitatni próbálunk. Ezután kísérletet dolgozhatunk ki a hipotézisünk bizonyítására, vagy megcáfolására. Gondolj a tesztre, mint hipotézisre és a kódra egy kísérletként, amit a hipotézis igazolására használsz.

De a nagyobb tévhit, hogy amikor kipróbálják a TDD-t, azt gondolják, hogy ez egy unit megközelítés. A legtöbb fejlesztő számára, amikor a „unit teszt” kifejezést hallják a „unit” kifejezés alatt egy kódegységre gondolnak, például egy metódusra vagy utasítások blokkjára, vagy akár egyetlen sorra. De ez nem az, ami ebben az összefüggésben egy „unit”-ot jelent. Úgy gondolom, hogy

a unit - egység kifejezést egy funkcionálisan független viselkedésegység hangsúlyozására fogadták el.

Ideális esetben az általunk ellenőrizni kívánt viselkedés egység közvetlenül kapcsolódik az elérni kívánt elfogadási feltételekhez. Ha az egységvizsgálatok is elfogadási tesztek, a követelmények nyomon követhetőségét és ellenőrizhetőségét ingyen megkapjuk.

A „viselkedési egység" (unit of behavior) több olyan objektumot is magában foglalhat, amelyek együtt dolgoznak. Például, ha tesztelni szeretnénk az árverésen való ajánlattételi szabályokat, előfordulhat, hogy az 'eladó' objektum egy 'aukciós' objektumot és egy 'ajánlattevő' objektumot hoz létre az adott aukción. Vannak, akik ezt integrációs tesztnek hívják, mert ez több objektum kölcsönhatását foglalja magában. Én ezt „unit test"-nek nevezem, mert egy viselkedési egységet tesztelek, az árajánlatot.

Gyakran előfordul, hogy amikor az átvételi kritériumokat teljesítő funkciók építésére koncentrálunk, olyan kódot írunk le, amely lényegesen olcsóbb a karbantartás szempontjából, mert ezt a tervet egyszerűbb megérteni és kiterjeszteni.

Tegyük tesztelhetővé a tesztelhetetlen kódot

A TDD tanulásának második kulcsfontosságú összetevője olyan technikák sokaságának elsajátítását foglalja magában, amelyek az ellenőrizhetetlen kódot tesztelik. Sok meglévő kódot nagyon nehéz tesztelni, és amikor kapcsolatba kell lépnünk ezzel a kóddal, nehézségekbe ütközhetünk.

Az egyik fő probléma, amellyel sok helyen találkoztam a kódban az az, hogy egy szolgáltatás igénybevételéhez példányosítják a klienst, majd közvetlenül hívják a szolgáltatást. Kívülről a szolgáltatás és a szolgáltatás kliense ugyanannak látszik, és nem szétválasztható. De amikor ezt újra és újra megtesszük egy rendszeren keresztül, akkor átláthatatlanul kuszává tesszük rendszerünket, melyben az egyes építőkockákat nem lehet különválasztani és tesztelni.

Ennek a problémának az egyik megoldása a függőség befecskendezés (dependency injection). A függőség befecskendezés ismerős lehet például a Spring keretrendszerből (spring.io). Ez a módszer [azonban] keretrendszer nélkül, manuálisan is használható. Ahelyett, hogy egy szolgáltatást [közvetlenül] egy objektumpéldány által hoznánk létre, a példányosítás feladatát kiszervezzük, majd referenciaként átadjuk az azt meghívó ügyfél kódnak.

Ha biztosítjuk, hogy a szolgáltatásra hivatkozó referencia a szolgáltatást igénybe vevő ügyfél kódba kerüljön, [azzal] elrejtettük az eredeti objektumot. Egyszerű koncepció, amely létfontosságú a kicsi, tesztelhető viselkedési egységek készítéséhez és a monolitikus kód feldarabolásához.

Számos elrejtési megközelítés létezik, melyeket használhatunk a függőség helyettesítése érdekében. Az egyik a kézzel készített mock létrehozása, a függőségek alosztályokba szervezésével, és a hivatkozó kódrészletek metódushívásainak felülírásával. Ahelyett, hogy a valódi függőséget hívná, a mock az objektum felülírt metódusát hívja meg, amely visszaadhat bármilyen értelmes eredményt. Ne felejtsd el, itt a célunk az, hogy kipróbáljuk a kódunk kapcsolatát a külső függőséggel, nem magának a függőségnek a tesztelése.

Tapasztalatot szerezni TDD-vel

A képesség jó tesztek készítésére és a kód írására vonatkozó készségek csak részei a TDD elsajátításának. A TDD elsajátításának harmadik és legfontosabb összetevője, hogy tapasztalatot gyűjtsünk a TDD-vel. Amikor a fejlesztők elkészülnek a teszt-vezérelt fejlesztésekkel, és meglátják, hogy a tesztek azonnal elkapják a problémákat – és mennyire jobb a kódjuk ennek eredményeképpen – elkezdenek egyre inkább TDD-t használni a projektek elkészítésekor.

Hasznos lehet a TDD megismerése egy zöldmezős projekten, mert sokkal több komplikáció akad a teszt-vezérelt fejlesztéssel egy régi kódon. Ez önmagában egy külön tanulmánynyi terület, és van néhány kiváló könyv is a témában. Úgy gondolom, hogy minden profi szoftverfejlesztőnek el kellene olvassnia Martin Fowlertől a „Refactoring: Improving the Design of Existing Code"-ot. Ha pedig régi kódon dolgozol, olvasd el Michael C. Feathers munkáját is: „Working Effectively with Legacy Code" - és ne felejtsd el megnézni a „Beyond Legacy Code: Nine Practices to Extend the Life (and Value) of Your Software" című könyvemet.

Amikor először kezdtem szoftverfejlesztőket tanítani a teszt-vezérelt fejlesztésről, előadtam nekik, hogy mi a TDD, és hogyan lehet tesztelni az ellenőrizhetetlen kódot. Tudták, mi a teendő, de mivel nem használták még projekten keresztül a gyakorlatban, nem maradt meg számukra. Hat hónappal később, amikor visszamentem, senki sem használta a TDD-t.

Amikor viszont tizenkét órás gyakorlatot tartottam a TDD-vel a képzés részeként, megfigyeltem az embereknél a szemléletváltozást, észrevették milyen előnyöket jelent számukra a TDD. Valójában ez az egyetlen módja annak, hogy megtanuljuk és új viselkedésmódot szerezzünk: tevékenykedés közben bizonyságot szerezni az új tudás értékességében. A tapasztalat nem szerezhető meg azzal, ha csak hallgatunk valakit egy témáról.

A TDD elsajátításnak három fő összetevője megérteni, hogy mi a TDD valójában, és tudni azt, hogyan lehet tesztelhetővé tenni a kódot, és TDD-t használó projekteken gyakorlati tapasztalatot szerezni. Úgy gondolom, hogy ha a fejlesztők már rendelkeznek ezzel a három összetevővel, akkor motiváltak lesznek a TDD-vel kapcsolatban, és használni is fogják azt a projektjeiken.

Szerző: David Bernstein ■

Szerző: ***David Berstein***

Forrás: https://www.agileconnection.com/article/3-keys-mastering-test-driven-development

David Bernstein

David Bernstein a "Beyond Legacy Code: Nine Practices to Extend the Life (and Value) of Your Software" könyv szerzője (http://BeyondLegacyCode.com). Több mint 8000 professzionális szoftverfejlesztőt oktatott szerte a világon. Cége a To Be Agile (http://ToBeAgile.com) segít a fejlesztőknek az extrém programozási gyakorlatokat elfogadni és használni, mint például a teszt-vezérelt fejlesztés, refactoring vagy a folyamatos integráció.

Szoftvertesztelés és gyártásoptimalizálás

Elsőre kicsit távol állnak egymástól. Holott céljaikban sok hasonlóság van, egymást remekül kiegészítik. Sok, változatos és érdekes út létezik a minőségbiztosítási szakmában; alábbiakban saját nem mindig tudatos döntéseken alapuló szubjektív tapasztalataimat szeretném megosztani.

Szoftvertesztelésben, nekem legalábbis, az első lépések arról szólnak, hogy az ember teszteli az elé tett alkalmazást. Ha van mentora, vagy ideje tanulni, akkor megfelelő módszertani háttérrel, ha nincs, akkor a józan ész által diktált rendszerben.

A szoftvertesztelés alapvető célja, hogy a termék, amelyet kiadunk kezeink közül, a felhasználóit hosszabb távú, fenntartható versenyelőnyhöz juttassa. Kezdő tesztelőként, a végfelhasználókkal egy szinten ülve, verzióról verzióra lépve a tesztelés alatt álló alkalmazással, napi szinten szembesültem azzal, hogy ez mit is jelent. Kényelmetlen menürendszer, átgondolatlan folyamatok, a túlterhelt munkatársak életét szükségtelenül lassító, instabil, funkcionalitási hibákkal terhelt alkalmazást használtak.

Ezt követően teszt menedzserként először négy majd nyolc ember munkáját felügyeltem a következő cégben. A tesztelés belső folyamatainak szervezése, finomhangolása után felszabaduló energiáimat a szoftverfejlesztő cég belső folyamatainak áttekintésére szántam. A tesztelés sikere nagyban függ attól, hogy megkapjuk-e a tervezett, igényelt időszeletet, tesztjeink megtervezésére, elkészítésére és lefuttatására. Nem optimálisan működő szervezeteknél a tesztelés az utolsó rész a kőbe vésett kiadási határidő előtt; a minket megelőző folyamatok jellemző csúszásával.

Mit is tehet ilyenkor a teszt menedzser? Természetesen, a szolgálati út betartása az első feladatunk, a projekt / szervezeti egység vezetője felé kommunikáljuk a problémát. Ám a lehető legjobb szándék sem mindig elegendő, nem várhatjuk mindig felülről a megoldást.

A teszt menedzsment szakma innovációs lehetőségei szűkek, de annál érdekesebbek. Az említett cégnél maradva, a programozók és a tesztelők közötti „átadás - átvétel” (Scrum előtti világ) egy, másfél nap volt, tesztelendő verziókként. Ennek fő oka a belső folyamataink menedzseléséhez használt rendszerek szigetszerű mivolta volt. Az akkori termékfejlesztésünkre jellemző: 6 részben különböző ügyfél, ügyfeleként 2-3 támogatott verzióval; elavult technológiával. Egy egy tesztelésre átadott csomagban több ügyfél számára javított, fejlesztett verzió is megjelenhetett. A programozók által használt forráskód kezelő rendszerből szöveges állományként kaptuk meg a változások leírását, amelynek feldolgozása és a hibajegy követő rendszerrel összevetése volt az említett, átlagosan másfél nap. Amely szükségtelenül kidobott időszak volt, nem egyszer stresszes, határidő szorításban - ennek megfelelően nagyobb hibaaránnyal.

Együttműködve a vezető programozóval, standardizáltuk a változásokat leíró szövegállomány belső struktúráját. Ennek nyomán, a hétköznapokban, programozóinknak checkin-enként 20 másodperccel több munkája lett. Ezen felül, a hibajegy kezelő rendszerhez írtunk egy feldolgozó alkalmazást, a fent említett állomány feldolgozására. A kapott infor-

mációt az eszköz összehasonlította a hibajegy adatbázissal és megtette a szükséges változtatásokat. Az „átadás-átvétel” folyamat átlagosan fél órára csökkent, sokkalta kevesebb elnézett változással, gyorsabb, pontosabb, több idővel rendelkező tesztelést eredményezve.

Kicsivel később, egy másik helyen körülbelül 50 fős fejlesztési csapatban megkaptam 12 tesztelő vezetését. A tesztelők és a programozók egymástól részben független szervezeti egységekbe voltak sorolva, a tesztelők egy nagyobb, a programozók 5 kisebb csapatban dolgozva. A hétköznapokra jellemzően a tesztelők általában kiegészítő információ nélkül megkapták az adott programozó csapattól a tesztelendő build-et, amelyet néha telepíteni sem tudtunk. A tesztelésre alkalmatlan verzióval eltöltött munka időpocsékolás - elkészítése, tesztelőknek átküldése, telepítési próbálkozás, a hibák nyomozása. Többek között ezen problémák miatt is, vezetőnk kívánságára a teszt csapatot megszüntettük, a tesztelők kiültek az egyes programozó csoportokhoz.

Az addigi „túlmunka”; használhatatlan, telepíthetetlen teszt verziók, a teszt csapat tennivalólistáján, megszűnt. A szükségtelen munka általi problémákat a programozók sokkal közelebbről érzékelték és partnerek lettek a megoldásban.

A fejlődés lépései a szervezetet és a teszt menedzsert is jellemzik. Teszt menedzserként a belső tesztelési folyamatok optimalizálása a cél, hogy a feltételezetten egységesnek tekinthető elvárt munkaszint mellett, optimálisabb teszteléssel, tovább fejlődési időt szerezzünk magunknak.

Önállóvá váló tesztcsapat mellett, a teszt menedzsernek is több ideje lesz kitekinteni - tesztelés környéki folyamatokra, azok információtartalmára, ütemezésére. Előfordul, hogy nem egyszer véglegessé válik a kitekintés és a teszt menedzser Scrum Masterként folytatja.

Scrum Master-ként is idővel elérünk a lehetőségeink határaihoz - átlagos szervezeti felépítést tekintve a Scrum Master az üzleti oldal - termékfejlesztés - támogatás felosztásban, a termékfejlesztés területen tevékenykedik. Agilis szoftverfejlesztést nehéz úgy csinálni, ha a megelőző és követő területek hozzáállása nagyban különbözik a miénktől. Következő fejlődési lépcső az agilis coach, aki oktatja a cégre jellemző módszertant, valamint elősegíti az optimális együttműködést a különböző szervezeti egységek között a problémák strukturált feltárásával, rendszerszintű elemzésével és fenntartható szervezeti, folyamatmódosítások definiálásával és bevezetésével - lásd: Tesztelés a Gyakorlatban, 2018/II, Követelménykezelés üzleti funkciókat megvalósító tesztmappákkal

A fentiek okán, teszt menedzserként több helyen is lehetőségem volt részt venni ISO szabványok bevezetésében. A „folyamatos javítás” ISO 9001 elvét nehéz működőképesen átültetni a hétköznapokba (vagy csak nekem nem volt szerencsém).

Kerestem tovább és J. Liker - The Toyota Way könyvében olvasottak alapján összeállt a kép. A könyvben részletes leírást kapunk a Toyota Production System-ről (TPS) amelynek főbb elveit alább, kivonatosan igyekszünk bemutatni - kapcsolván a szoftvertesztelés, teszt menedzsment céljaihoz, lehetőségeihez.

1., A menedzsment döntéseket hosszú távú érdekek alapján kell meghozni, nem egyszer a rövidtávú célok kárára.

A fenntartható teszt automatizálás egy fényes példa erre. Rövid távon pénzbe kerül, jól megírva viszont hosszú távú céljainkat szolgálja.

2., A helyes folyamatok helyes eredményekre fognak vezetni.

Mind a tesztelést érintő folyamatokra igaz, mind a tesztelés alatt álló termékre.
Folyamatokat érintően, mindegyikünk be tud számolni olyan élményekről, hogy az előttünk késésben lévő kollégák miatt, jelentősen kevesebb idő alatt, nem egyszer túlórákat is beleértve tudjuk letesztelni a kibocsátandó terméket, a mindenki számára kőbevésett határidőig.
Terméket érintően gondolkodnunk kell a majdani végfelhasználó folyamatainak megfelelő támogatásáról. Ha magának a specifikációnak meg is felel, de a magasabb szintű tesztelés során kényelmetlennek érezzük, akkor lehetőségeink szerint eme probléma jelzésére térjünk ki.

3., A gyártási folyamatban hátrább lévő egységek „kérjenek” maguknak munkát.

A feljebb említett példa, miszerint a tesztelők, kérés nélkül kapnak új, amúgy tesztelhetetlen buildeket, ide tartozik. Ezen felül, ki ne látott volna „kanban” (és itt erős az idézőjel - a kanban eredetileg pont arról szól, hogy a folyamatban hátrébb lévő kéri a munkát az őt megelőzőtől) board-okat, ahol a tesztelésre vár, tesztelés alatt oszlopokban található feladatok bőven meghaladták a tesztelés aktuális lehetőségeit.

4., A folyamatban résztvevő egységek munkaterhelése kiegyenlített legyen.

Célunk, hogy minimalizáljuk a hulladékidőt vagy az értéktelen résztermékét, csökkentsük az emberek és a termelésben használt eszközök túlterhelését, valamint kiegyenlített termelési szinteket hozzunk létre.
Tesztelésben is nem egyszer tapasztaljuk a tesztelési piramis hiányosságát. Az alattunk lévő szintek elégtelen minőségbiztosítása folyamán egyszerű hibákkal kell foglalkozni, nem arra szánt időben (ennek nyomán a munkaterhelés egyenlőtlenné válik) - pld kiadási határidő előtt pár nappal, egy unit vagy komponens szinten elfogható hibával.
Emberi oldalról is fontos elv: sokkalta jobban fognak reagálni a tesztelők a mostani termékkiadás

ÁLLÁSHIRDETÉS

Automata Szoftver Tesztelő

Szívesen vennél részt izgalmas nemzetközi banki IT projektekben? Érdekel a tesztelés, tesztautomatizálás? Ha igen, akkor Téged keresünk!

Feladatok:
- banki informatikai alkalmazás felhasználói, üzleti tesztelése;
- banki informatikai rendszer részletes, angol nyelvű üzleti specifikációjának megismerése és megértése;
- a specifikáció alapján a tesztesetek elkészítése és a tesztek automatizálása Robot Framework használatával;
- a tesztelések során felmerült hibák jelentése a fejlesztőknek, illetve a hibajavítások ellenőrzése;
- a tesztesetek karbantartása és szükség esetén módosítása

Elvárások:
- közép-/felsőfokú stabil angolnyelv-tudás (szóban és írásban);
- konstruktivitás, eredményesség, logikai képességek;
- belső motiváció, csapatban gondolkodás;

Előny:
- Robot Framework, vagy egyéb kulcsszó alapú tesztautomatizálási eszköz ismerete;
- Git ismeretek;
- 1-2 év tesztelői tapasztalat.

Amit nyújtunk:
- nemzetközi banki projekteken folyamatos szakmai fejlődés;
- konstruktív vállalati kultúra, egymást segítő összetartó közösség;
- versenyképes juttatási csomag; széles körűen választható cafeteria;
- előmeneteli lehetőség;

Munkavégzés Helye: Budapest

Az info@passed.hu oldalon várjuk a jelentkezéseket.

Vegye igénybe hatékony, pontos, megbízható szoftvertesztelési szolgáltatásainkat!

www.passed.hu

A megbízható tesztcsapat!

után következő feladatra, ha látják, hogy nem kell szükségtelenül időt eltölteniük olyan résztermékek tesztelésére, ahol az alapvető programozói minőségbiztosítás nem történt meg.
Jellemzően a teszt mérnök hétköznapokban ezen a problémakörön sokat nem tud segíteni, folyamatosan ismételni kell, a kiegyenlítettlen bejövő munka által kellett - rossz - hatásokról szóló beszámolót. E cél érdekében dolgozó agilisabb teszt menedzserek a fenti példához hasonlóan, Scrum Master-ként, agilis coach-ként fogják magukat találni.

5., A termék már először legyen jó minőségű. Problémamegoldó vállalati kultúra építése.

TPS-ben dolgozó üzemben a munkatársnak joga (és kötelessége) minőségi probléma esetén megállítani a folyamatot.
Tesztelésre átfordítva, ez az elv nagyban rezonál az ISTQB FL CT-ben is olvasott korai tesztelés elvével. Ha már a tesztelő az értékteremtő folyamat elején is jelen van, akkor mind a minőségtudatosságot tudja emelni a többi kollégában, mind az adott résztermékben nagyobb eséllyel találja meg korábban a hibát.

6, Standardizált feladatok és folyamatok az alapja a folyamatos fejlődésnek.

Gyorsan összetudunk rántani egy megbeszélést, hogy mit kellene másképp csinálni. Jellemzőleg nem készül róla feljegyzés, mindenki visszatér az asztalához, még talál emlékszünk a megállapodásra pár napig, majd ez szépen a ködbe vész, és a berögzült, jól, rosszul működő folyamatok veszik vissza a hétköznapokat. Pár nappal, héttel később, ismét lesz egy ilyen beszélgetés, lehet, hogy eggyel magasabb vezető kérésére, majd az „új rendszer" hasonló véget ér.
E helyett, definiálva, minden részvevővel elfogadtatva, fenntartható, csak a szükséges feladatokat előíró rendszerben kellene gondolkodnunk, amelyet iteratív módon felülvizsgálhatóvá tesszük, hogy reagáljunk a szervezetet ért változásokra. Egy ilyen világban a tesztelés is standardizált, előre becsülhető feladatokkal számolhat.

7, Gyártási folyamat felügyelet vizuális eszközökkel - a hibák jó eséllyel nem maradnak rejtve.

Leginkább elfogadott, és a hétköznapokban megjelent elv a TPS-ből – fejlesztési folyamatok, rendszer állapotok vizuális megjelenítése.
Információs monitorok a fejlesztésen, láva lámpák, megszólaló szirénák rossz buildnél, illetve ehhez hasonló eszközök, mind figyelmünkért versenyeznek, egyre erősebb ingert keltve. Egy helyen dolgozó csapatoknál nem egyszer mai napig könnyen áttekinthető, megfogható fizikai scrum / kanban táblákat használnak.
A teszt jelentésben foglalt információ tartalom struktúrája és megjelenítésének módja is átalakulóban van. CI/CD környezetre jellemző pipeline megközelítésben, a teszt jelentés eggyé vált a pipeline állomás sikerességével - ha a definiált tesztek helyesen lefutottak, a build tovább mehet. Ennek megfelelően kell gondolkodásunkat a teszt készlet - test suite fogalommal kapcsolatban átalakítani. Többé nem elég a teszt automatizálás magas aránya, az automata tesztnek gyorsnak kell lennie, igény szerint több tesztkészletbe rendezve, hogy a CI/CD környezet igényeit kielégítsük.

8., Megbízható, tesztelt technológiát kell a termékbe építeni.

Szubjektív véleményem szerint, a szoftvertermék némiképp hasonló az autóhoz. Amennyiben olyan terméken dolgozunk, amelyiknek a módosítására, gyorsjavítására nincs könnyed mód (offline, laboratóriumi eszközök, stb.) akkor a hasonlóság erőteljesebben látszik. A megvásárolt autóban, a fékrendszert nem a gyártósoron próbálták ki, hanem a kutatólaborban. Adjuk lehetőséget időben, térben, pénzügyekben a szoftverfejlesztési csapatnak arra, hogy a kutatást különválassza a fejlesztéstől.
Iterációs szoftverfejlesztésben számtalan alkalommal tapasztaljuk, hogy programozó kollégáink iteráción belül, az adott követelmény kielégítése érdekében, több különféle technikai megközelítést próbálnak ki. Majd, jobb esetben félúton ráböknek az egyik lehetséges megvalósításra, és elkészítik -amelyet az iteráció 2/3-nál lát először tesztelő. Ha egy átlagos iterációba 8 - 10 feladatot beválasztunk, és csak a feladatok felénél járunk így, már akkor katasztrófaközeli a tesztelő helyzete. Adjunk lehetőséget a kiegyenlített munkára, a hulladékidő és termék minimalizálására: jelentősen koraibb követelmény priorizálással, megvalósítási módokat kutató idővel (spike); melyek nyomán a végső, terméket előállító iteráció sokkalta inkább fog hasonlítani az autógyárban található gyártósorhoz.

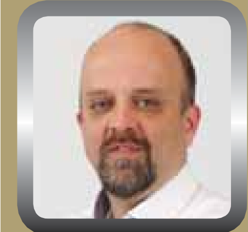
A tesztelést érintő termék és folyamat problémák megoldásához a Toyota tapasztalatait sem szabad elvetnünk. Sőt, fontos tanulság Liker 2004-es The Toyota Way könyvének publikálása után, pár évvel bekövetkezett események sorozata, mikor a Toyota jelentős visszahívásokra kényszerült; mely problémába, saját elveiket feladva, az erőltetett növekedést hajszolva kerültek. ■

Szerző: ***Schaffhauser Balázs***

Források: http://www.cssp.com/CD0310b / SuccessSowsSeedsOfFailure/

https://en.wikipedia.org/wiki/The_Toyota_Way

Jeffrey K. Liker - The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer, McGraw-Hill Education; 1 edition (January 7, 2004) ISBN-13: 978-0071392310



Schaffhauser Balázs

Balázs 15 éves tapasztalattal rendelkezik szoftver teszt menedzsment területen. Négy nagyobb tesztcsapat felépítését vezette különböző vállalatokban. Szoftvertesztelésen felül érdekelt minőségbiztosítási folyamatokban, valamint az agilis fejlesztésben. Jelenleg a Passed Informatikai Kft-ben vezető tanácsadóként dolgozik. https://www.linkedin.com/in/bschaffhauser

A mobil 12 kihívása

Még csak egy évtized telt el az első iPhone megjelenése óta, de már olyan korban élünk, ahol az okostelefonok mindenhol jelen vannak. Mobiljaink olyanok, mint egy jó svájci bicska, rengeteg funkcióval szolgálnak. Milyen kihívásai vannak egy alapos mobil-tesztelésnek?

Még csak egy évtized telt el az első iPhone megjelenése óta, de már olyan korban élünk, ahol az okostelefonok mindenhol jelen vannak. Az okostelefonjaink olyanok, mint egy jó svájci bicska – rengeteg funkcióval szolgálnak, legyen szó a térképeinkről, névjegyzékeinkről, naptárjainkról, kameráinkról, zene lejátszóinkról vagy természetesen a kommunikációs eszköz funkciójáról.

A szoftvertesztelés nem lenne teljes mobil tesztelés nélkül. Olyan sok szempont van a mobil-tesztelés során, hogy úgy döntöttem, hogy három részre osztom: most a mobil-tesztelés kihívásait fogom megvitatni, következőnek a mobilok manuális tesztelése, és legvégül pedig a mobilok automata tesztelése lesz a téma.

Először is, a kihívások! Az alábbiakban felsorolok 12 okot, miért bonyolult a tesztelés mobilon. Úgy gondolom, hogy érdekes lehet illusztrálni ezeket, egy-egy általam talált valós hiba-példával. E hibák közül néhányat már a tesztelési karrierem során felfedeztem, és néhányat a saját eszközeimen találtam, mint végfelhasználó.

1. Szolgáltató:
A mobil applikációk teljesítménye nagyban függ attól, hogy melyik szolgáltatóhoz tartozunk. Az USA-ban két nagy szolgáltató viszi a piacot, a Verizon és az AT&T, de találunk kisebb szolgáltatókat is, mint a Sprint vagy a T-Mobile. Európában a főbb szolgáltatók közül kiemelhetjük a Deutche Telekomot, Telefonica-t, Vodafone-t és az Orange-t; Ázsiában pedig a China Mobile-t, Airtel-t, NTT-t és a Softbankot. Ha mobilon tesztelsz, fontos figyelembe venni a szolgáltatót amit a végfelhasználó választani fog, és tesztelni ezekkel a szolgáltatókkal.

Hiba példa:
Teszteltem egyszer egy térkép funkciót az applikáción belül, és felfedeztem, hogy ha a térképet frissítettem a helyzetem alapján, az egyik szolgáltatót használva frissített, a másik szolgáltatót használva nem frissített. Ennek ahhoz lehetet köze, hogy a helyzetemet melyik cellás vevőtorony közvetítette.

2. Hálózat vagy Wifi:
A (eszköz)felhasználóknak lehetőségük van arra, hogy úgy használják az applikációkat, hogy közben a szolgáltatók hálózatához csatlakoznak, vagy wifihez. Még úgy is dönthetnek, hogy az app használatának kellős közepén csatlakoznak át; vagy a

The testing company

csatlakozás teljesen meg is szakadhat, ha kilépnek a hálózati tartományból. Fontos, hogy úgy is teszteljünk egy applikációt amikor a hálózathoz csatlakozunk, és úgy is amikor wifi-re és nézzük meg, hogy mi történik, ha kapcsolat megváltozik közben, vagy akár teljesen meg is szakad.

Hiba példa:
Van egy wifi extender a házamban. Amikor átkapcsolom a telefont az extender IP-jére, a Spotify azt hiszi offline vagyok. Csinálnom kell egy kényszerleállítást az alkalmazással, aztán újranyitni, hogy a Spotify felismerje, hogy online vagyok.

3. Alkalmazás típusa:
A mobilalkalmazások lehetnek webalapúak, natívak vagy e kettő keveréke (webes alkalmazásként írva, de natív alkalmazásként telepítve). A végfelhasználók dönthetnek úgy, hogy nem a natív vagy kevert megoldást választják, hanem jobban preferálják, ha böngészőjükből működtetik az alkalmazást. Különböző variációk használhatók a mobil böngészők között is, mint például a Safari, Chrome vagy az Opera. Tehát fontos meggyőződni róla, hogy a webalkalmazás jól működjön számos mobilböngészőn is.

Hiba példa:
Sokszor előfordult már, amikor ráléptem a mobilweboldalra, hogy a „mobilra optimalizált” oldal nem rendelkezett a szükséges opciókkal, amit akartam. El kellett döntenem, hogy rámegyek-e a teljes oldalra, ahol az összes szöveg apró és nehéz navigálni.

4. Operációs rendszer:
A mobil alkalmazások eltérőek lesznek operációs rendszertől függően. A két legnagyobb operációs rendszer, amit ismerünk az iOS és az Android, és vannak mások, mint például a Windows Mobile és a Blackberry. Fontos, hogy teszteljünk mindegyik operációs rendszeren, amit a végfelhasználónk használni fog, hogy biztosak legyünk abban, hogy az alkalmazás összes funkcióját minden rendszer támogatni fogja.

Hiba példa:
Ez nem egy hiba, hanem kulcsfontosságú különbség az Android és az iOS között: az Android eszközöknek van vissza gombja, míg az iOS eszközöknek nincs. Az iOS-re írt alkalmazásoknak tartalmazniuk kell a vissza gombot, hogy a felhasználóknak lehetőségük legyen visszalépni az előző oldalra.

5. Verzió:
Minden operációs rendszer, rendszeresen frissíti magát, olyan új funkciókkal amik arra ösztönzik a felhasználókat, hogy frissítsenek. De nem minden felhasználó frissíti a telefonját a legújabb és legjobb verzióra. Fontos, hogy elemzés szerint meghatározzuk, hogy mely verziókkal rendelkeznek a felhasználók, és meggyőződni arról, hogy teszteltük ezeket a verziókat. Továbbá, minden verziófrissítésben előfordulhat, hogy az alkalmazásban olyan új hibákat hoztak létre, amik korábban nem voltak ott.

Hiba példa:
Gyakran, amikor frissítem a verziót a telefonomon, nem tudom használni a kihangosító funkciót amikor telefonálok. Hallom a hangot a másik oldalon, de a hívó fél nem hall engem.

6. Gyártmány/Márka:
Bár az összes iOS készüléket az Apple gyártja, az Androidos készülékek már nem ilyen egyszerűek. A Samsung a legnagyobb Android eszközgyártó, de ezen kívül találunk még sok más márkát, mint például a Huawei, a Motorola, Asus vagy az LG. Fontos megjegyezni, hogy nem mindegyik Android felhasználó Samsungot használ, ezért teszteljünk más Androidos eszközökön is.

Hiba példa:
Egyszer teszteltem egy tabletes alkalmazást, ahol a billentyűzet funkció jól működött néhány helyzetben, de nem mindig. A billentyűzet egyszerűen nem ugrik fel ezeken az eszközökön, ezért nem tudtam bármilyen típusú mezőbe gépelni.

7. Modell:
A verzióhoz hasonlóan évente új modellek kerülnek bemutatásra. Míg egyes felhasználók évente vagy két évente cserélnek a legújabb készülékekre, addig mások egyáltalán nem. Ezenkívül egyes eszközök nem tudják frissíteni az operációs rendszer legújabb verzióját, így kétféle módon is elavultak lesznek. Ismételten: fontos, hogy képünk legyen róla, hogy a végfelhasználók melyik modelleket használják, így eldönthetjük, hogy melyik modelleket teszteljük és támogassuk.

Hiba példa:
Ez nem egy hiba, de fontos szempont volt: amikor az Apple piacra dobta az új iPad modelljét, amely lehetővé tette az aláírás ellenőrzését a felhasználóknak. A szoftver, amit akkor teszteltem már tartalmazta ezt a funkciót, de az iPad régebbi verziói nem tudták ezt támogatni, ezért az alkalmazásoknak figyelembe kellett venniük, hogy régebb verzióval rendelkező felhasználóktól ne kérjen dokumentum aláírást.

8. Tabletek és Okostelefonok:
Számos végfelhasználó fogja használni az applikációd inkább a tableten, mint okostelefonon. A natív alkalmazások gyakran különböző alkalmazás-

ÁLLÁSHIRDETÉS

Szoftvertesztelő munkatárs

CSATLAKOZZ HOZZÁNK, HA
- Szeretnél egy fiatalos szakmai csapat része lenni
- Szeretnél izgalmas projektekben részt venni

AKIT VÁRUNK:
- Aktuális kliensoldali webes technológiák ismerete (minimum Javascript/HTML5)
- Magas szintű számítástechnikai és hálózati ismeretekkel rendelkezik
- Magabiztos angol nyelvtudással rendelkezik (minimum írásban)

ELŐNYÖS:
- Linux rendszerek ismerete, virtualizáció
- Verziókövető rendszerek ismerete (SVN)
- TDD módszerek ismerete
- C/C++ ismerete
- IP kamerák és videó megfigyelő rendszerek ismerete
- Biztonságtechnikai végzettség
- Szoftverfejlesztési ismeretek
- Szakirányú végzettség

AMIT NYÚJTUNK:
- Saját technológiákat fejlesztő, nemzetközileg is sikeres cég tapasztalatát és támogatását, mely lehetőséget nyújt számodra a legújabb iparági trendek és technológiák megismerésére
- Hosszú távú biztos, kiegyensúlyozott stabil munkahelyet és munkakörülményeket
- Erkölcsi és anyagi megbecsülést

ÁLLÁS, MUNKA TERÜLETE(I):
- IT programozás, Fejlesztés
- Tesztelő, Tesztmérnök
- Teljes munkaidő

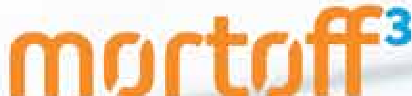
SZÜKSÉGES TAPASZTALAT:
- Mindegy, vagy nem igényel tapasztalatot

SZÜKSÉGES NYELVTUDÁS:
- Angol - Felsőfok/tárgyalóképes szint

MUNKAVÉGZÉS HELYE:
Budapest

Amennyiben felkeltette érdeklődését ez a lehetőség, kérem küldje el szakmai önéletrajzát az info@passed.hu címre.

The testing company



Légy te is elismert tesztelési szakértő!
Junior vagy szenior vagy? Építsd nálunk karriered!

Tesztautomatizáló mérnök
Manuális szoftvertesztelő
Beágyazott tesztelő

Nálunk egyéni karrier program keretein belül fejlődhetsz!

Understand. Go beyond. **Deliver.**

karrier.mortoff.hu

facebook.com/mortoff

verziókkal rendelkeznek, attól függően, hogy tabletre vagy telefonra készültek eredetileg. Az okostelefonra tervezett applikációkat gyakran le lehet tölteni a tabletre, de fordítva ezt már nem tudod megtenni. Ha elkezdesz használni egy webalkalmazást, fontos emlékezni rá, hogy a tabletek és okostelefonos más funkciókat tartalmaznak. Teszteld az alkalmazásod tableten és telefonon is egyaránt.

Hiba példa:
Olyan alkalmazásokat teszteltem, amelyek teljesen jól működött okostelefonon, viszont tableten egyszerűen egy üres képernyőt dobott be.

9. Képernyő méret:
A mobil eszközök nagyon-nagyon sok féle méretben kaphatók. Bár az iOS eszközök csak néhány méretezési standerdet követnek, addig az Androidok tucatnyi méretben jönnek ki. Bár lehetetlen minden képernyőméret tesztelése, fontos tesztelni a kisebb, közepesebb, nagyon vagy extra nagy méretet is, hogy biztosak lehessünk abban, hogy az alkalmazásunk helyesen rajzolódik ki minden felbontásban.

Hiba példa:
Olyan alkalmazásokat teszteltem kis méretű telefonokon, amelyeken az oldalelemek egymást fedték, ami nehezíti a szövegmezők megjelenítését vagy a gombok kattintását.

10. Álló vagy fekvő mód:
Az okostelefonokon történő tesztelés során könnyű elfelejteni a fekvő mód tesztelését, mert gyakran portré módban tartjuk a telefont. De néha az okostelefon felhasználók látni akarják majd az alkalmazást fekvő módban is, és ez még inkább igaz a tablet használókra. Fontos. hogy ne csak álló és fekvő módban teszteljünk, hanem ügyeljünk arra is, hogy a használat közbeni oda vissza kapcsolgatás is működjön az alkalmazásban.

Hiba példa:
Tesztelten egy alkalmazást, ami nagyon jól nézett ki tableten, álló helyzetben, de amikor elfordítottam a képernyőt az összes mező eltűnt róla.

11. Alkalmazásközi integráció:
Az egyik legfontosabb dolga a mobilalkalmazásoknak, hogy képesek legyenek együttműködni más funkciókkal, mint például a kamera vagy a mikrofon. Ezen kívül kapcsolódniuk kell más alkalmazásokhoz is, mint például a Facebook vagy a Twitter. Bármilyen integrációt is támogat az alkalmazás, mindenképpen teszteljük alaposan.

Hiba példa:
Teszteltem egy alkalmazást, ami a felhasználó számára lehetővé tette, hogy készítsen egy képet egy otthoni berendezésről, majd hozzáadhatta házi leltárához. Amikor kiválasztottam a kép készítését, és lefotóztam utána nem léptetett vissza az alkalmazásba.

12. Az alkalmazás integrációján kívül:
Még ha az alkalmazást nem is úgy tervezték, hogy más alkalmazásokkal vagy funkciókkal működjön, akkor is lehetséges, hogy vannak hibák ezen a területen. Mi történik, ha a felhasználó kap egy hívást, üzenetet, vagy épp alacsony töltöttségfigyelmeztetést miközben használja az alkalmazást? Fontos ezt is kitalálni.

Hiba példa:
Egy ideig hibás volt a telefonom, mert amikor az időzítő funkció leállt és épp hívásban voltam, nem tudtam leállítani, csak ha kikapcsoltam az egész telefont.

Remélem, hogy a fenti leírások és példák megmutatták milyen nehéz egy mobilalkalmazás tesztelése. Első látásra lehet nyomasztónak tűnik, de az elkövetkező két írásomban elmagyarázom, hogyan lehet egyszerűbbé tenni a tesztelést. A következő cikkemben megvizsgáljuk a mobil tesztelés tervezését és összeállítunk egy portfóliót a fizikai eszközök teszteléshez. ■

Szerző: ***Kristin Jackvony***

Forrás: http://thethinkingtester.blogspot.com/2018/07/mobile-testing-part-i-twelve-challenges.html

Kristin Jackvony

A vonzódásomat a szoftvertesztelés irányába nagyjából két évtizednyi zeneoktatás után fedeztem fel. Voltam már minőségbiztosítási tesztelő mérnök, menedzser, és az elmúlt 8 évben (jelenleg is) minőségbiztosítási tesztelési vezetőként dolgozom a Paylocity-nél. Egy hetenként jelentkező blogot írok, melynek címe: „Gondolkodj úgy, mint egy tesztelő" https://thinkingtester.com/, mely kihangsúlyozza a fontosságát a szoftvertesztelés alapjainak, így segítve a szoftvertesztelőket.

The testing company

Következő generációs gyakorlatok szoftvertesztelőknek

Az Internet előtt is léteztek tesztelési feladatok, manapság viszont más készségekkel kell rendelkeznie egy tesztelőnek. A cikkben találsz olyan szoftvert, amelyen gyakorolhatsz. A program segít azon készségeinket is használni, amik ahhoz szükségesek, hogy igazi tesztelővé váljunk.

Rengeteg cikket lehet olvasni a tesztelői gondolkodásról, a tesztelés elméletéről és a teszteszközökről. De milyen gyakran beszélünk arról, hogy mit csinálunk éppen akkor, amikor tesztelünk?

Ha a weben tesztelési gyakorlatokat keresel, valószínűleg találsz ősrégi grafikai programokat amelyek egy háromszöget rajzolnak ki, vagy olyat ahol szavakat kell megszámolni, vagy egy régi parkolási díjkalkulátort. A legtöbb ilyen program korábbi az Internet megszületésénél. Ezek közül néhány, olyan Windows-ra készített program, amelyet régebbi, kisfelbontású monitorra terveztek, a mai képernyők egyik sarkába illeszkedik. Sokat ezek közül papírra terveztek. A webre tervezettek is a mobil korszak, a reszponzív dizájn előtti korszakból valók.

Két évvel ezelőtt elhatároztam, hogy előrelépek ebben az ügyben. Teszttervezéssel foglalkozó workshop-ot tartottam Ohio-ban, ahol ígéretet tettem néhány teszttervezést támogató gyakorló feladat kifejlesztésére és publikálására.

Kigondoltunk és megvalósítottunk egy „oda-vissza ugyanaz” szójátékot és felraktuk a weboldalunkra. Fantasztikus, gondoltam, de valójában önző voltam: cégünk interjú feladatként használja. Itt hát ideje betartani az ígéretet és mindenkinek elérhetővé tenni.

A palindrom gyakorlat

Frissen alkalmazott tesztelő vagy. A cég szeretne estére egy szoftvert, ami a palindromokat teszteli olyan szavakat, amik ugyanazt jelentik oda-vissza olvasva (Például „gőg” egy palindrom szó, de „gőz” nem) Írj be valami szöveget, kattints a beküldésre (Submit), és a szoftver megmondja, hogy a szöveg palindrom-e - enynyire egyszerű. A vezető fejlesztő beteg, de van egy junior fejlesztő, aki ki tudja javítani hibákat, amiket találsz. Természetesen időre van szükség a javításra és az újratesztelésre. A termékgazda (product owner) nem szakértő, de tud válaszolni a kérdésekre.

Ez elegendő részlet a szoftver teszteléséhez. Itt a weboldal: http://xndev.com/palindrome Nézd meg és kommentálj a kedvenc programhibáidról. De mielőtt ezt megtennéd, álljunk meg egy pillanatra, és gondoljuk át a helyzetet. A fenti gyakorlat valódi ereje az, hogy ha valaki más játsza

The testing company

a termékgazda szerepet. Alább álljon pár példa ahol ez érdekes lehet:

- Melyik böngészőben teszteled? Melyik mobil eszközön? Mikor gondolod, hogy eleget teszteltél?
- Mennyi időt vesz igénybe a tesztelés?
- Milyen helyzetet, melyik problémát tekintjük hibának? Melyiket nem?
- A termékgazda aggódik az API teljesítménye miatt, amit a Submit gomb lenyomása hív meg. El tudod különíteni az API-t? Hogyan tesztelnéd a teljesítményt?
- Találtál biztonsági problémát az oldalon? Esetleg hozzáférhetőséggel vagy többnyelvűséggel kapcsolatban?
- Tudjuk szállítani a szoftvert? (Ez gyakran teljesen értelmetlen vitához vezet a tesztelő szerepével kapcsolatban, amelyben ez a kérdés mindenképp előkerül: „Legalább javasolni tudsz valamit?")

A fentiekhez hasonló kihívások elé állított tesztelők általában rosszul szerepelnek. Viszont számos egyéb készséggel rendelkeznek. Meg fognak küzdeni a válaszért, ráveszik a junior fejlesztőt a válaszok keresésére, vagy felhívják a szeniort a kórházi ágyán. Felütik szamárfüles példányaikat a „Hogyan szerezzünk barátokat, és hogyan befolyásoljuk embereket" ill. „Rendkívül hatékony emberek 7 szokása" című könyvekből. Vagy nem. Néhány tesztelő állást foglal a fenti kérdésekben, miszerint, a fenti kérdések a fejlesztők számára szólnak, nem tesztelői feladat. Néhány, teszt automatizálással foglalkozó kolléga válasza: „Csak add oda a teszt eseteket és automatizálom őket".

Csak néhány tesztelő képes modellezni a kockázatokat, elvégezni a technikai tudást igénylő háttérvizsgálatokat és az ezekről jellemzőleg mit sem tudó ügyféllel kommunikálni. Ez a fajta beszélgetés új képességeket igényel: azt a fajtát, amelyet egyre inkább keresnek ügyfeleink.

Készségek a modern szoftverteszteléshez

A 90-es években, a szoftver lemezen jött, a maga fizikai valóságában –drága volt minden egyes kiadás. Ma, a cégek gyakrabban próbálnak meg szállítani, az első kiadás kódminőségét javítandó, gyorsabban érkeznek a javító csomagok. A programozók unit teszteket írnak, integrációs teszteket készítenek, felhasználói felületen futtatott teszt automatizálást is készítenek és mindezt bekötve a CI rendszerbe. Egyértelmű hibák megtalálásához szükséges, a felszínt vizsgáló tesztelői megközelítésre egyre inkább nincs szükség.

A modern tesztelőnek nem kell tudnia full-stack alkalmazást írni, de valószínűleg elegendő tudással rendelkezik a hozzáférhetőség, a többnyelvűség, a különböző platformok, a Wi-Fi sebesség szimulálás, a hálózatok, a HTML, a CSS, a TCP/IP és a JavaScript területén, hogy képes legyen webes alkalmazásokban a hibakeresésre. Sokan közülük valószínűleg képesek naplóelemzéshez alkalmazást írni, megértik a szerver naplókat, és ismerik annyira a virtualizációs technikákat, hogy képesek legyenek virtuális munkaállomások és szerverek használatára, nem is említve egy CI/CD rendszer összerakását és működtetését.

Ilyen a modern web jellege. A modern tesztelők szakterületekre szakosodnak, mint például adatbázisok, nagy szöveges adatkészletek, natív mobilalkalmazások, API vagy örökölt rendszerek, de a következő munkához a tesztelőknek képesnek kell lenniük új technológiák megtanulására.

És ezek csak a technikai képességek. Természetesen meg kell értenünk a gyakori meghibásodási módokat is, megfelelő emberismerttel kell rendelkezniük, hogy ügyfeleinket minél jobban megismerjük, és folyamatosan tanulni kell a gondolkodást.

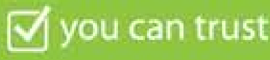
Kezdésnek mára, teszteljétek a palindrom webes alkalmazást, ha tetszik, adjátok tovább. Írjátok meg a véleményeteket, hogyan lehetne jobbá tenni, és arról, hogyan tudnánk mindannyian fejleszteni tesztelési képességeinket. ■

Szerző: ***Matthew Heusser***

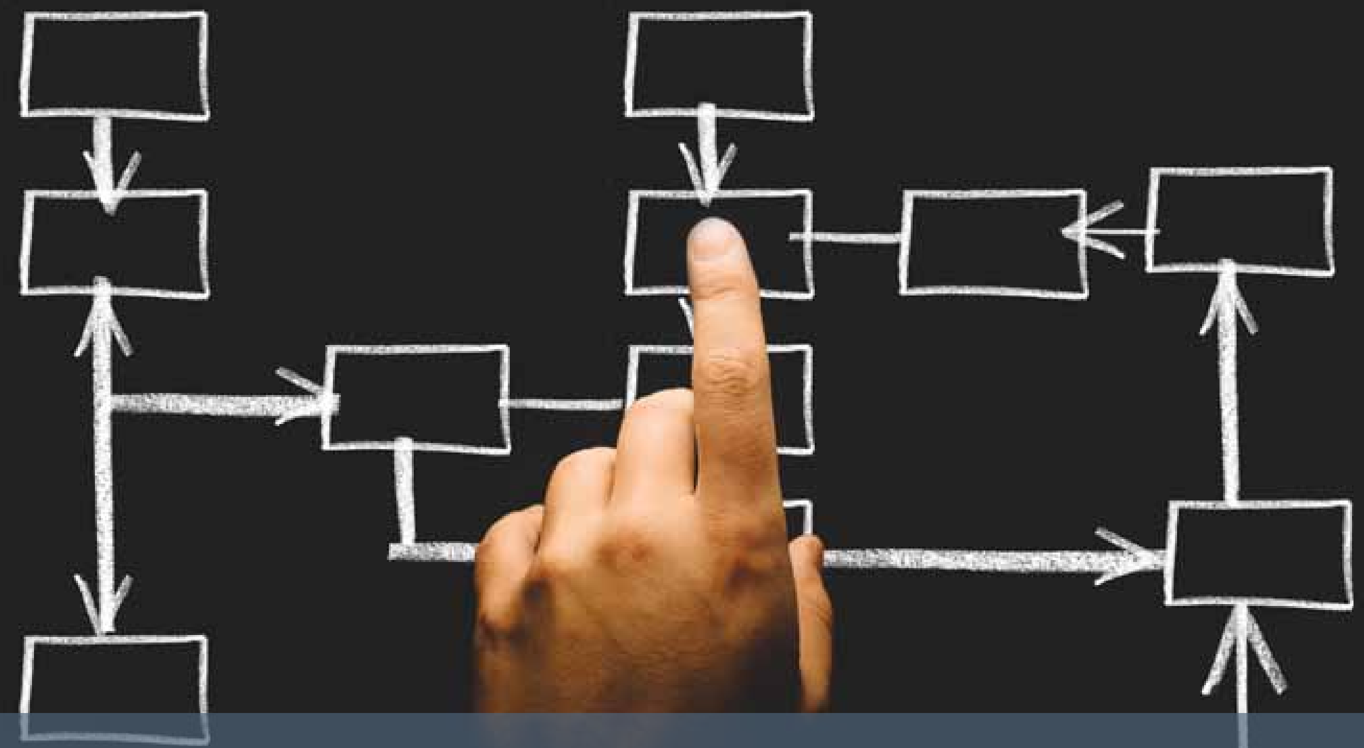
Forrás: https://www.stickyminds.com/article/next-generation-exercises-software-testers

Matthaew Heusser

A "Managing Consultant at Excelon Development" Matt Heusser talán legismertebb írása. Matt a Stickyminds.com korábbi technikai szerkesztője, és ő volt a vezető szerkesztője a „Hogyan csökkenthető a szoftvertesztelés költsége" című könyvnek is ("How To Reduce The Cost Of Software Testing" Taylor and Francis, 2011). Matt mind az Association for Software Testing igazgatótanácsának, mind a Calvin College informatikai rendszerének részmunkaidős előadójaként szolgál.



The testing company



A manuális tesztelésnek se múltja, se jövője. Soha nem is létezett!

Ha esetleg a blogolvasók számára nem lenne világos, mert erről általában konferenciákon beszélek – abszolút nem értek egyet a 'manuális tesztelés' kifejezés létjogosultságával és szükségességével.

Őszintén azt gondolom, hogy a tesztelés sosem volt manuális, és soha nem is lesz az. Sőt messzebb megyek – szeretném, ha bárki, aki úgy gondolja, hogy a 'manuális tesztelés' fogalma szükséges és hasznos, megmagyarázná, miért is segít ez a kifejezés jobban leírni és megérteni a tesztelést. Íme a gondolataim.

Mi a gond a 'manuális' teszteléssel?

Az első probléma ezzel a kifejezéssel, hogy a tesztelés soha nem csak abból állt, hogy végletekig leegyszerűsített módon, a gondolkodást mellőzve hajtunk végre egy cselekménysorozatot, akár egy gyári munkás a futószalag mellett, ahol újra és újra ugyanazok a műveltek ismétlődnek.

A tesztelést általában a teszteszköz-forgalmazók szokták a manuális teszteléssel azonosítani, mintegy lebutítva azt, hogy eladhassák eszközeiket a tesztelés megkönnyítése céljából, azoknak az embereknek, akik nincsenek otthon a tesztelés világában.

A másik probléma, amit ezzel kapcsolatban látok, az az, hogy az emberek a tesztelést egy egységes, és szükségtelenül formális dologként kezelik, így be tudják magukat skatulyázni 'manuális tesztelőként'. Ezekről az alábbi cikkemben írtam nemrég: The non-manual, unautomated tester[1] .

Miért helytelen a 'manuális tesztelés' kifejezés?

A tesztelés soha nem lehet csak és kizárólag manuális. Ennek oka, hogy túl sok gondolkodással és szellemi munkával jár. Hogy egy híres mondást idézzek – hiába püföli a billentyűzetet, és hiába kattintgatja végig 100 majom a szoftverterméket 100 órán keresztül, attól még nem fognak magas szintű tesztelési szolgáltatást nyújtani.

A sok gondolkodás és szellemi munka alatt a következőket értem, a teljesség igénye nélkül:

- **modellezés** – képesek vagyunk a terméket, és annak belső működését elképzelni,
- **kritikus gondolkodás** – képesnek lenni elméleteket gyártani a termékkel kapcsolatban, majd megkérdőjelezni azokat, és megpróbálni bizonyítani az ellenkezőjét,
- **társadalomismeret** – a szoftver nemcsak egy kód, mivel emberek életét hivatott segíteni, meghatározott társadalmi kontextusban kell működnie,
- **kockázatcsökkentés** – azon agyalni mindig mi romolhat el, mielőtt túl késő lenne,
- **kognitív pszichológiai ismeretek** – ismerni a saját gondolkodásmódunkat, és tudni hogyan ejthet az csapdába, a tudás átka,
- **metakogníció** – avagy a saját tudásunkról rendelkezésre álló tudás, kívülállóként figyelni a saját megközelítésmódunkat vagy gondolkodásunkat, így lehetőségünk nyílik módszereink hatékonyságának felmérésére,

- **logika** – következtetések levonása a jelekből, deduktív-, induktív-, és abduktív érvelés segítségével,
- **episztemológia** – ismeretelmélet – honnan tudjuk azt, amit tudunk, hogy lehetünk biztosak abban, hogy egy szoftverrel kapcsolatos állítás igaz, honnan tudjuk, hogy a szoftvernek egy adott módon kell működnie,
- **kreatív gondolkodás** – problémák felfedése új utak és stratégiák segítségével.

Ne aggódj, ezeket általában anélkül is használod, hogy tisztában lennél az elnevezésükkel, vagy akár tudnál a létezésükről. Ezek a kifejezések csak magyarázatul szolgálnak arra, amit csinálunk.
Megjegyzés: Ha szeretnél jobban elmerülni ezekben a témákban, akkor ajánlom a “What software testing is…”[2] blogsorozatomat. Jelen cikk írásakor ez még nincs kész, de hamarosan készen lesznek.

Amit ezzel kapcsolatban szeretnék kihangsúlyozni, az leginkább a fekete hattyú jelenséggel írható le – az, hogy te nem láttál még fekete hattyút, nem jelenti azt, hogy nem is létezik. **Tehát attól még, hogy a tesztelés kapcsán általában nincs szó magáról a gondolkodásról, az nem szükségszerűen azt jelenti, hogy nem használjuk az agyunkat tesztelés közben,** vagy egyből ’manuális tesztelésként’ lehetne címkézni.

A tesztelés egy olyan cselekvés, mely szorosan kötődik a gondolkodáshoz és észleléshez, ha mondjuk egy testrésszel kéne jellemeznem, akkor az **’agyi tesztelés’** kifejezés lenne a legmegfelelőbb. Nevetségesen hangzik igaz? Ezek után mondja nekem bárki is, hogy a manuális tesztelés fogalmának van bármi értelme…

Miért olyan népszerű mégis a ’manuális tesztelés’ szóhasználat?

Ha kíméletlenül őszintén kéne megmondanom, akkor Scott Berkun-t idézném[3] : *„Az emberek szeretik az egyszerű, kétalkotós modelleket, és büszkélkednek azzal, hogy primitív elméleteik igazolására, a tudományt használják ürügyként.”*
Pontosan a fentiek miatt létezhet oly sok ideje a manuális/automatizált kettős marhaság. A tesztelők **inkább részesítik előnyben az egyszerű, de helytelen kettős felosztást, mintsem kicsit megerőltessék magukat** és előálljanak egy jobb válasszal.

Mit értenek az emberek pontosan ’manuális tesztelés’ alatt?

Megfigyelésem szerint, mikor tesztelők egymás közt a manuális tesztelésről beszélnek, általában **’nem programozott módon végrehajtott’** tesztelésre gondolnak, vagy másképpen fogalmazva - **olyan tesztelés, melynek végrehajtásához nincs szükség kód írására, hanem magunk futtatjuk.** Mindenesetre ettől a tesztelés még nem lesz manuális, úgyhogy javasolnám e hiba kiküszöbölésére az ’emberi tesztelés’ fogalmának bevezetését. Szerintem ez a kifejezés nagyobb rálátást biztosít arra, mi is áll mögötte pontosan.

A ’felfedező teszt’ elnevezés sem segít

A fenti kifejezés, leginkább a programkóddal támogatott tesztelést végzők körében népszerű, de szélesebb körökben is terjedni látszik.

Akármikor szóba kerül az automatizálás piramisa (amivel a Hindsight automatizálási blog-leckékben[4] fogok foglalkozni), a legtöbben a piramis csúcsán látható ’felhőre’ felfedező tesztelésként hivatkoznak. Sokszor konferenciákon is azt látom, hogy a prezentáló cégek úgy adják elő a tesztelési folyamatot, hogy egy része automatizálva van, a többit pedig felfedező tesztként hajtják végre.

Nos, úgy érzem a társadalom lassan kezd ráérezni, miért vagyunk allergiásak a ’manuális tesztelés’ szóhasználatra, és elkezdték ’felfedező tesztelésnek’ hívni, mert az menőbben hangzik. Ez azonban **nem orvosolja az eredeti problémát,** ha továbbra is ugyanazokat a régi elavult dolgokat értik manuális tesztelés alatt. Másfelől ez eltorzítja a felfedező tesztelés eredeti jelentését, felruházva azt olyan tulajdonságokkal, melyek nem igazak rá.

Befejezésként…

A legtöbbször azt látom, hogy a ’manuális tesztelés’ fogalmát sokan túl leegyszerűsítik és nyilvánvalóként kezelik, majd ebből hibás következtetéseket vonnak le, mint például: a tesztelés manuális, tehát egyszerű, vagy: a tesztelés manuális, tehát sokkal hatékonyabb lenne, ha automatizálnánk. **Ez mind igen csekély ismeretre vall a tesztelésről és magáról a szoftverek tárgyköréről úgy általában.** A ’manuális tesztelés’ kifejezése valójában egy szinonima lett az ’alacsony minőségű, kevés emberi szaktudást igénylő ellenőrzés’-re. Le merném fogadni, hogy egyetlen épeszű tesztelő sem szeretné, ha így kategorizálnák őket.

Az én javaslatom az, hogy töröljük el ezt a ’manuális’ kifejezést – **a tesztelés sose volt manuális, és nem is lesz az.**

Azok számára, akik nem értenek egyet velem - a kihívás nyitva áll – írd le a megjegyzésekhez a legjobb érveket, hogy miért hasznos a ’manuális teszt’ kifejezés a tesztelési szerep leírásában? ■

Szerző: ***Viktor Slavchev***

Forrás: https://mrslavchev.com/2017/12/11/manual-testing-never-existed/
https://mrslavchev.com/2017/01/09/the-non-manual-unautomated-tester/
https://mrslavchev.com/2016/11/04/software-testing-is-part-1/
http://scottberkun.com/2017/creativity-advice-explained/
https://mrslavchev.com/2017/11/29/hindsight-lessons-about-automation-start-automating/

Viktor Slavchev

Tesztelő, társadalomtudományi háttérrel, gyakran beszél és ír tesztelésről és tudományról vagy a tesztelésről, mint tudományról. Számára a tesztelés több tudományágat érint, több rétegű, több dimenziós szakma, mely egy biztos alapon nyugszik – az emberi intelligencián. Minden más az ennek a meghosszabbítása. Jelenleg szenior minőségbiztosítási tesztelőként dolgozik a Siteground (https://www.siteground.com) nevű cégnél és részmunkaidőben oktat a Pragmatic-ban (https://pragmatic.bg). Érdeklődési köre főleg a nehézsúlyú kritikus gondolkodás témakörében fellelhető, melyet a tesztelés jobb átgondolásához és a felfedező tesztelés és automatizáció keresztezésére használ. Nézeteiről és a gondolatairól a tesztelés területén, a blogján olvashatsz többet: http://mrslavchev.com

Tesztelői döntéshelyzet: duplikált hibajegyek

Mikor lehet eldönteni, hogy egy hibát fölösleges rögzíteni mert már szerepel a hibajegykezelő rendszerben? Ha megkérdezed a fejlesztőktől és két hiba ugyanazért a hibás kód miatt eredményez helytelen működést, ezeket a hibákat elég csak egyszer rögzíteni. Michael Stahl azonban jó érvekkel támasztja alá, hogy tesztelő szemszögéből miért jobb inkább több hibát rögzíteni.

Egyszer megkérdeztem egy ismerősömet, egy matematika professzort, mit tett azért, hogy megkapja a PhD-jét.

„*Adtak nekem egy megoldandó problémát*" mondta.
„*És*?" Kérdeztem
Válasz: „*Már nem probléma*"

A szoftvertesztelés során számos olyan témát találunk, melyek megoldhatónak tekinthetők, ha a jól ismert és általánosan elfogadott módon kezeljük őket. Az egyik ilyen példa a hibakezelés.

Elméletileg, ez egy világos folyamat: Vegyünk egy hiba-státusz megbeszélést, amin részt vesznek a fejlesztők a projektmenedzser és még azok, akik esetleg érdekeltek. A hibajelentések ellenőrzésre kerülnek, hogy tartalmaznak-e világos leírást, naplót, képernyőképeket és ésszerű leírást a hiba reprodukálására. Ezután megvitatásra kerül a hiba súlyossága. Egyes esetekben ez könnyű feladat, néha pedig bonyolult és alapos hozzáértést igényel az alaptechnológiákban. Gyakran alakul ki vita – ami néha heves – egy konkrét hiba súlyosságának megítélésével kapcsolatban. A release határidejéhez közeledve a politika is szerepet játszik. Miután a súlyosságot elfogadták és a prioritás is be van állítva: Megoldjuk a hibát azonnal, vagy várhatunk vele? Itt is van lehetőség a vitákra. De általánosságban elmondható, hogy a folyamat egyértelmű és többnyire működik.

Ennélfogva, ismételten meglepődők, hogy a folyamat aspektusai soha nem tűnnek teljes mértékben elfogadhatónak, sőt kétségbe vonják még akkor is, ha már régen átkerültek a „Már nem probléma" kategóriába.

Az egyik ilyen szempont az ismétlődő hibajegyek kérdése. Pontosabban, mikor lehet egy hibajelentést fölöslegesnek tekinteni, mert az már szerepel a hibakezelő rendszerben? Ilyen esetben az új jelentés felesleges, nem érdemes vitázni róla és bezárható.

A duplikáltként kezelt hibákat csapásként tekinthetjük a validálást tekintve (és így is kell látnunk). Ez azt jelenti, hogy a validációs csapat nem szakszerű: A tesztelőnek meg kell vizsgálnia a hibakezelő rendszerben, hogy a hiba jelentve van-e már és tartózkodnia, hogy ne adjon hozzá újabb zavaró tényezőt az adathoz ugyanazon hiba jelentésének megismétlésével.

Ezért a validációnak nem érdeke, hogy egy hiba duplikáltként legyen megjelölve, és megpróbálja igazolni az új hiba jelentését. A fejlesztés viszont szeretné minimalizálni a nyitott hibajegyek számát és szeretnék elérni, hogy egy hibajegyet úgy lehessen lezárni, hogy nem dolgoznak rajta. Éppen ezért a nézeteltérések gyarapodni fognak és a hiba-státusz megbeszélés fölösleges vitába torkollik.

Hogyan döntsük el, hogy egy hibajegy duplikátum-e?

Attól függ kit kérdezel. Ha a fejlesztőt kérdezzük, akkor, ha két hiba ugyanarra a hibára mutat rá a kódban elég egyszer jelenteni azt. A második hiba automatikusan

megjavul miután az elsőt kijavítják. A fejlesztők úgy látják (és sokszor a projekt menedzser is ezt gondolja), hogy ebben az esetben, ha mindkét hibajegy külön van kezelve az rontja a kód minőségét.

Itt egy meggyőző példa (ha fejlesztő vagy): Tegyük fel, hogy van egy alkalmazásod a hallgatók évfolyamainak tárolására és jelentésére. A hallgatók társadalombiztosítási számát (SSN) kell használni, hogy meghatározzuk az évfolyamot az adatbázisból, egy hallgató azonosítása végett. Az alkalmazás megengedi, hogy a diákok más adatok használatával azonosítsák magukat az SSN mellett: teljes név, biztonsági kód (PIN), telefonszám, és bankkártya szám. Az első körben az alkalmazás az azonosító adatokat használja az SSN kitöltéséhez a hallgató személyi adataiból. Ekkor az SSN segítségével határozzák meg az évfolyamot. (Minden adatbázis-szakértőnek, aki a haját tépi sírva, hogy „Így nem szabad csinálni": Nyugalom! Azt mondtam, hogy „Tegyük fel „.)

Öt teszteset definiálnak rendszer szinten. Mindegyik az évfolyamkiválasztás minőségét ellenőrzi különböző azonosítási metódusokkal. Mi történik ezután, ha van olyan regresszió a kódban, ami kivonja az évfolyamot az adatbázisban szereplő második táblából? Mind az öt teszteset hibás lesz!

Milyen a kódminőség ebben az esetben? Ha jelentünk öt hibát, átadunk egy üzenetet, hogy az évfolyam meghatározás modulban súlyos károkat szenvedett, mely erősen kihathat a projektre. De ez magától értetődően helytelen. Egyetlen hiba van a kódban és az egész fél nap alatt meg fog oldódni, beleértve az előzetes tesztelést is.

Ez így egy kimondottan jó érv arra, hogy az öt hibából négy miért duplikált. Viszont van egy kis ellentmondás is. Mennyire lehetünk biztosak abban, hogy mind az öt hibajelenségnek ugyanaz a kiváltó oka? Valójában a példának több oldala van. „Bizalmasan", tesztelők között: Ha elrontották azt a kódot, ami az évfolyam azonosítását végzi, még az SSN azonosítását is elronthatták. Teljes mértékben bízom a fejlesztőkben. Biztos vagyok benne, hogy tudják, hogyan lehet két hibát létrehozni egy adott verzióban.

Ha négy hibát bezárunk, mint duplikátum, mert mind az öt egy hibára mutat, előfordulhat, hogy csak egy hiba fog megoldódni. A többi hiba pedig várni kényszerül amíg újra futtatjuk a tesztjeinket. Mivel nem volt nyitott hibajegy ezekkel a tesztesetekkel szemben, nem fogjuk újra tesztelni sem ezeket. Ráadásul, ha az évfolyam meghatározási tesztekre alacsony prioritást állítunk be, elég sokáig tarthat, amíg újra nem futtatjuk őket. Miután a termék eléri a release állapotát egyértelműen nem fogjuk ezeket újra futtatni (ez az a pont, ahol Murphy törvénye szerint a hiba azonnal megjelenik).

Az álláspontom az, hogy a tesztelőnek kötelessége jelenteni azt, ami történik. Azt, hogy miért történt — a kiváltó ok — nem releváns, ha megbeszéljük a hibajelentést. A fejlesztő követelése a duplikátumokkal szemben azon a feltételezésen alapul, hogy rendszer szinten történt hibák kódszinten ugyanazhhoz a hibához köthetők. A tesztelők ellenvetése pedig az, hogy rendszer szinten a hibajelenségek eltérők voltak.

Azt állítom, hogy egy hibát csak akkor kell duplikáltnak tekinteni, ha mind a végrehajtás lépései, mind a hibás eredmény azonos.

Létezik egy másik figyelmeztető jel, amikor egy hiba nem duplikátum. Általában egy hiba akkor fog kétszer szerepelni, ha a két tesztelő ugyanazt a hibát találta meg és jelentette anélkül, hogy megnézte volna, szerepel-e már a rendszerben. Érdemes tehát azt gondolni, hogy ha mindkét hibát ugyanaz a tesztelő nyitotta meg, akkor az nem egy duplikált hiba. Miért akarná ugyanaz a tesztelő ugyanazt a hibát kétszer felvinni a rendszerben? Ez plusz munkát jelent a tesztelő számára, aki biztosan emlékszik az első hibajelentésre. Nyilvánvaló, hogy akármi is történt, a hiba nem teljesen azonos — még ha fejlesztők ezt is gondolnák.

Fontos beszélni ezzel a témával kapcsolatban, elszakadva attól, amivel éppen foglalkozunk és megállapodnunk az érintett felekkel arról, hogy egy hibát duplikáltként kell-e tekinteni. Először is ez elkerüli a megismételt beszélgetéseket a hibajavítás során és megakadályozza a negatív hatójelenségek számát, melyek akkor fordulnak elő, ha erős a tendencia a hibajegyek megismétlésére. Eddig két ilyen jelenséget tapasztaltam:

- A tesztelők úgy döntenek, hogy nem jelentenek be néhányat ezek közül az események közül mondván „úgyis duplikátumként végzi"
- A tesztelők ezt a magatartást tovább folytatják: Ha hibát tapasztalnak és úgy gondolják, hogy megértik a kód viselkedésére gyakorolt hatását, akkor nem nyitnak hibákat olyannal kapcsolatban, amelyről azt a következtetést vonják le, hogy az első hiba eredménye miközben az új hiba teljesen máshogy nyilvánul meg.

Jobb, szkeptikusnak lenni és megválogatni, mely hibajegyeket jelölünk duplikáltként. Néhány duplikált hiba által okozott kár alacsonyabb, mint a nem bejelentett hiba okozta kár. Jobb, ha a jelentés oldalon tévedsz.

Ha egy hibajelenség nem pont ugyanaz, mint egy korábban bejelentett hiba, hozzunk létre egy új hibaejegyet. Ha mindkettőt ugyanaz a hiba okozza, akkor a fejlesztők egy árért kettőt kapnak cserébe. Ilyen eseteben fontos, hogy összekapcsolják a hibákat a hibajelentő rendszerben (a legtöbb rendszer támogatja az ilyen összekapcsolásokat). Ez megmondja a fejlesztőknek, hogy úgy gondoljuk, hogy ez a két hiba kapcsolatban állhat egymással, vagy ugyanazon hiba más-más megnyilvánulása lehet. A linkek még nem elegendők — a fejlesztők általában túlugranak rajtuk. Be kell írni egyértelműen mindkét hiba szöveges leírásánál: „Lehetséges, hogy ezt a hibát ugyanaz a kód-hiba okozza, mint az X, linkelt hibát"

Abban reménykedem, hogy a fentiek hatására ez a probléma is átkerül a „nem probléma többé" kategóriába.

Szerző: ***Michael Stahl***

Forrás: https://www.stickyminds.com/article/when-testers-should-consider-bug-duplicate

Michael Stahl

Michael Stahl egy szoftvervalidációs architekt az Intelnél. Ebben a szerepkörben tesztelési stratégiákat és munkamódszereket határoz meg a tesztcsapatok számára, és néha tesztelni is próbál - amit a legjobban élvez. Michael a SIGiST Izraelben, a STARWestben, az EuroStar-ban és más nemzetközi konferenciákon is tanít és szoftvertesztelést is a Jeruzsálemi Héber Egyetemen. Mielõtt 2000-ben elkezdte pályafutását a tesztelésben, Michael dolgozott az Intel gyárában, Jeruzsálemben, mint chipszintû tesztmérnök.
Michael az Izraeli Teszt Tanúsító Testület (ITCB) igazgatósági tagja, teljes körû ISTQB tanúsítvánnyal rendelkezik, és az ITCB tanácsadó testületét vezeti. Michael néhány bemutatója és írása megtalálhatók a www.testprincipia.com címen.

AutoHotKey

A napi munka legkevésbé kedvelt részei, mikor ugyanazon műveleteket kell végrehajtani nap, mint nap: bejelentkezés alkalmazásokba, azonos karaktersorozatok gépelése, ugyanazon űrlapok kitöltése. A monotonitás az alkotómunka elől veszi el a levegőt. A jó hír az, hogy létezik segítség.

Az AutoHotKey szkriptnyelv segítségével a napi munka hatékonyabbá tehető. Az AutoHotKey ingyenes, nyílt forráskódú program, amely lehetővé teszi felhasználói számára, hogy könnyedén készítsenek kis és összetett parancsfájlokat mindenféle feladatra, mint például: űrlapkitöltések, automatikusan végrehajtott műveletek, billentyűzet makrók, egérvezérlő makrók. A segítségével akár Windows alapú alkalmazást is készíthetünk.

Telepítés – A legfrissebb verzió közvetlenül a https://autohotkey.com címről szerezhető be. A telepítő megkérdezi, hogy az UNICODE vagy az ANSI változatot szeretnéd-e telepíteni. Az ANSI csak az angol karaktereket támogatja, ezért számunkra előnyösebb az UNICODE változat használata.
A telepítés után azonnal használhatjuk a programot, hiszen egyszerű szöveg-szkripteket kell készíteni, amit a Windows jegyzettömb (notepad) programjával is megtehetünk.

Indítás és használat – Az AutoHotKey önmagában nem indul el. Az AutoHoKey csak szkript futtatására jó, ezért el kell készíteni az első szkriptünket. Nyissuk meg a jegyzettömböt és írjuk be:

```
::tgy::tesztelesagyakorlatban.hu
```

A fenti sort mentsük el proba.ahk néven. Ha duplán klikkelünk a proba.ahk-ra, akkor az AutoHotKey betölti a szkriptet és a háttérben folyamatosan futtatja egészen addig, amig mást nem kérünk tőle. Ha most nyitunk például egy böngészőt és az URL sávban begépeljük, hogy „tgy „ (idézőjel nélkül, szóközzel a végén), akkor a tgy-t a szkriptünk hatására lecseréli „tesztelesagyakorlatban.hu”-ra. Már csak az entert kell lenyomni. Ha ezt is szeretnénk megspórolni, arra is van lehetőség. Módosítsuk a fentieket:

```
::tgy::tesztelesagyakorlatban.hu{enter}
```

Mentés után újra dupla klikk a proba.ahk-ra. Meg fogja kérdezni, hogy az előző változatot biztosan cserélni szeretnénk-e. Válasszuk az „Igen” lehetőséget. A böngésző címsorába beírva a „tgy „-t meg is nyitja számunkra az oldalt.
Ha e cikk online megkeresését is szeretnénk megoldani a szkriptünkkel, akkor ezt is könnyen megtehetjük:

```
::tgy::tesztelesagyakorlatban.hu{enter}{tab}Eszköz-bemutatás: AutoHotKey{enter}
```

Az előzőekben megismert módon ki is próbálhatjuk szkriptünket.

AutoGUI – Sokat könnyíthetünk a saját dolgunkon, ha letöltjük az AutoHotKey-hez készített AutoGUI[1] programot, ami egy integrált fejlesztési környezet (IDE) az AutoHotKey számára. Az AutoGUI egy szkriptszerkesztő, egy GUI tervező és debugger eszközöket tartalmaz. A legérdekesebb tulajdonsága, hogy maga egy AutoHotKey szkripthalmaz. A forráskódját megnézhetjük és akár át is írhatjuk, de most csak elindítjuk. (Indítása: dupla klikk a kicsomagolt AutoGUI könyvtárban az AutoGUI.ahk állományra.)
Az indítás után érdemes a megnyíló üres fület rögtön elmenteni, hogy az IDE felismerje, hogy AHK programot írunk. Innentől színkiemelésekkel (is) segíti munkánkat.

A szkriptek mentés után azonnal kipróbálhatók, az IDE gyorsmenüjében található „Execute” gomb segítségével.

Kedvcsináló példák – Az alábbi példák a teljesség igénye nélkül igyekeznek megmutatni, hogy hogyan lehet használni az AutoHotKey szkriptnyelvet. Egy állományban írhatunk több makrót is, futáskor mindegyiket figyelembe veszi az AHK, ezért következő példák írhatók egyetlen egy ahk állományba.

Egyszerű billentyűmakrók

Ha nem akarunk lezáró billentyűt (szóköz) használni, akkor az első „::” közé tehetünk csillagot:

```
:*:tgx::tesztelesagyakorlatban.hu{enter}{tab}
Eszközbemutatás: AutoHotKey {enter}
```

Futtatás után a böngészőben az url mezőbe gépelt „tgx” hatására (szóköz nélkül) működésbe lép a szkriptünk.

Ugyanezt megtehetjük gyorsbillentyűmakróként is:

```
#t::SendInput tesztelesagyakorlatban.hu{enter}{tab}
Eszközbemutatás: AutoHotKey{enter}
```

Windows + t hatására ugyanazt teszi, mint az előző szkript. Mivel nem „szócsere” történik, hanem billentyűkombinációhoz rendelt parancsot kell futtatni, ezért szükség van egy utasításra is, ami esetünkben a SendInput.

Ha html kódot kell írnunk és kevesebbet szeretnénk gépelni:

```
:*:ahr::<a href="">{left 2}
```

Az „ahr” hatására a csere szövege <a href=""> lesz és a kurzor az idézőjelek közé áll.

Többsorosok

Van mód különféle Windows információkhoz hozzáférni, például a pontos időhöz:

```
:*:pido::
 SendInput %A_YYYY%-%A_MM%-%A_DD% %A_Hour%:%A_Min%
return
```

A fenti sorok hatására a „pido” karaktersor a gépelés után a pontos dátumra és pontos (24 órás) időre módosul. Egy makró-utasítás több soros is lehet, ilyenkor „return” sorral tudjuk lezárni a végét.

Az alábbi esetben megvizsgáljuk, hogy van e már megnyitott jegyzettömb és ha van csak aktív ablakká állítjuk, különben megnyitjuk.

```
#n::
IfWinExist Untitled - Notepad
    WinActivate
else
    Run Notepad
return
```

Program vagy weboldal megnyitása:

```
#w::
 run http://tesztelesagyakorlatban.hu/
 Sleep 1500
 IfWinExist Tesztelés a Gyakorlatban - A szakértő
tesztelők lapja
   {
    WinActivate
    SendInput {tab}Eszközbemutatás:
AutoHotKey{enter}
    return
   }
 return
Windows + w
```

hatására az alapértelmezett böngésző új fülén nyílik meg a Tesztelés a gyakorlatban oldala. A Slepp 1500 pontosan 1.5 másodpercet várakozik. A várakozás megfelelő beállítása gyakori kísérletezést igényel.

Kereshetünk kijelölt szövegrészt a Google segítségével gyorsan, a Win+q kombinációval:

```
#q::
 Send, ^c
 Sleep 50
 Run, http://www.google.com/search?q=%clipboard%
 Return
```

Teszt „fél-automatizálás” – Az AHK fejlett szkriptnyelvvel rendelkezik, ami képessé teszi komolyabb feladatok automatizálására is Windows felületen. Ahhoz, hogy tesztelési célból igénybe vegyük, ismernünk kell a határait:

- Az AHK-t arra találták ki, hogy a windows-os felületen különböző műveleteket hajtson végre és nem arra, hogy ellenőrzéseket végezen.
- Legkönnyebben olyan műveleteket tudunk automatizálni, amik gyorsbillentyűkhöz kötöttek.
- A programokban a navigáció legegyszerűbb módja a tabulátor egymás utáni alkalmazása. Ezért azok a programok, ahol nem működik a tabbal való navigálás, vagy gyakran változik a tabsorrend, nehezebben automatizálható AHK-val.
- Van mód egérparancsok pozícionált végrehajtására, de ez nem megbízható módszer, ha többfajta eszközön is szeretnénk futtatni teszteket, hiszen ezek képernyő méretei eltérhetnek, így az alkalmazás belső pozíciói is eltérhetnek.
- Nehezen kezelhetők a várakozási idők.

A fentiek miatt úgy gondolom, hogy egy komplex tesztrendszer nehezen megvalósítható AHK-val. Inkább a manuális tesztelők egyes tevékenységeinek az automatizálást látom kivitelezhető célnak.

„Fél-automata keretrendszer” koncepciója

A manuális tesztelők smoke és regressziós tesztjeihez készíthetünk könnyen karbantartható, kis időráfordítással megvalósítható szkripteket, amelyek segítik és meggyorsítják az olyan részfeladatokat, mint például: bejelentkezések különböző rendszerekbe, űrlapok kitöltése, adat összehasonlítások képernyők között, számított érték ellenőrzése, adatfeltöltés CSV állományból, adatleolvasás CSV állományba.

Az egyes részfeladatokat gyorsbillentyűkhöz köthetjük vagy létrehozhatunk egy vezérlő felületet, ahonnan a megírt gyorsfunkciók gombnyomással előhívhatok. A vezérlő felületen lehetőséget tudunk biztosítani egyes funkciókhoz paraméterek megadására is.

Függvények, objektumok, API-k kezelése

Az AHK-ban van lehetőség objektumorientált programot írni és van lehetőség különböző API könyvtárak egyszerű használatára. Ez lehetővé teszi komplexebb elképzelések megvalósítását is. Például könnyen vezérelhetünk Excel-t, vagy Internet Explorert is. (lásd: ComObjCreate() függvény) ■

Szerző: ***Szőke Ármin***

Szőke Ármin

2000-ben szerezte meg a tanári diplomáját matematika-számítástechnika szakon. 2006-ig tanárként dolgozott. 2006 óta szoftver-teszteléssel foglalkozik. Főként banki és telekommunikációs területen szerzett tesztelési, tesztautomatizálási és tesztvezetői tapasztalatokat. Jelenleg több projekten szakmai vezető és ennek a magazinnak a főszerkesztője.

7 általános szoftvertesztelési tevékenység, amit ismerned kellene kezdés előtt

A szoftvertesztelési ipar olyan ütemben nő, mint azelőtt soha. Nem meglepő, hogy emiatt egyre több és több ember szeretne tesztelővé válni. Milyen is maga a szoftvertesztelés? Milyen tesztelési feladatok várnak rád?

A tesztelők emellett sokszor küldenek nekem e-maileket is arról, hogy mennyire élvezik az utazásukat a tesztelés világában. Örülök, hogy ilyesmiről hallok és örömmel fogadom sikereiket.

Sajnos emellett látok ellenpéldákat is különböző emberektől, hogy ők mennyire nem szeretik a szoftvertesztelést. Úgy érzik, hogy rosszul választottak munkát és elvesztegették az idejüket a teszteléssel, miután pár hónapja már teszteléssel foglalkoztak.

Miért történt ez? Nem ezek voltak azok az emberek, akik korábban annyira érdekeltek voltak a tesztelésben és rendkívül izgalmasnak látták a szoftvertesztelést?

Van néhány oka annak, hogy megmagyarázzuk mi is történt a színfalak mögött, de számomra akad egy mindentől nagyobb ok erre: kiábrándulás. Azt hitték ezek az emberek, hogy a szoftvertesztelés könnyű munka, de most már látják, hogy ez nem így van.

Azt hitték, hogy a szoftvertesztelés a hibák megtalálásáról szól, most pedig már felismerték, hogy több tevékenység is kapcsolódik ehhez a munkához, amiket viszont már egyszerűen nem élveznek napi szinten megismételve.

Összezavarodnak és bizonytalanná válnak, hogy biztos a tesztelés-e a megfelelő karrierválasztás, vagy nem. Ez rossz, amikor megtörténik. Viszont el tudod kerülni az ehhez hasonló helyzeteket azzal, hogy tudod, milyen is maga a szoftvertesztelés a nagy képben, milyen tesztelési feladatok várnak rád, így pedig előre el tudod dönteni, hogy ezt akarod-e választani… egyáltalán. Pontosan ezt szeretném megosztani veled ebben a cikkben.

Megjegyzés: Mielőtt elkezdenél olvasni ezekről a tesztelési feladatokról, kérlek, vedd figyelembe, hogy a szoftvertesztelési feladatok nincsenek azokra a feladatkörökre limitálva, amiket ebben a cikkben megemlítek. A tapasztalataim alapján próbálom listázni ezeket, de biztos vagyok benne, hogy ez nem a teljes lista. Valamint emellett vedd azt is figyelembe, hogy néhány cég más fogalmakat használ, mint amiket én használtam a lista megírásánál. Most már kezdhetünk.

1) A dokumentumok elolvasása

Végre nyeregben vagy. Felvettek néhány kör interjú után, nehéz és jó kérdésekkel (néhány buta kérdéssel is), de ez már nem is számít. Most már kvalifikált vagy a munkára és most már tesztelő vagy.

Képzeld el, hogy ez az első munkanapod. A menedzsered/főnököd bemutat téged a csapat tagjainak. Ha szerencséd van, a csapattagokkal

közel dolgoztok egymáshoz és mindenki bemutatkozik, minden archoz tudsz nevet társítani. Miután bemutattak nagyjából mindenkinek, izgatottan várod, hogy mi lesz az első feladatod.

„Itt vannak a projekt dokumentumai, kezd el őket olvasni.” – mondja a menedzsered.

Oké, egy kicsit túlzok talán, de alapvetően az lesz az első feladatod, hogy elolvass néhány dokumentumot, pl. a specifikációt, az útmutatót, segítséget, stb., hogy többet megtudj arról a rendszerről, amit tesztelni fogsz.

Míg te arra számítasz, hogy egy rock sztár leszel azzal, hogy találsz egy csomó hibát, amik majd a főnökeidet is le fogják nyűgözni, azt kell, hogy mondjam, a dokumentációk elolvasása annak érdekében, hogy ismerd a rendszert, ami tesztelés alatt van, az egyik legfontosabb első lépés, amit meg fogsz tenni a projektben.

Szóval használd ki ezt a lehetőséget az olvasásra és kérdezz annyit, amennyit csak tudsz a rendszerről, amit tesztelni fogsz. Ha a projektednek nincsenek dokumentumai, kérdezd meg a főnöködet, hogy hogyan tudhatnál meg többet a rendszerről.

Miután befejezted a dokumentáció átolvasását, lehet, hogy a tudásodról és a megértésed szintjéről be kell számolnod a csapattársaidnak/főnöködnek. Emellett ezt a tudást el is kezdheted felhasználni arra, hogy elkezdj más teszteléssel kapcsolatos tevékenységeket, pl. teszteseteket kitalálni, amiről később fogok írni.

Milyen hosszú ez a tevékenység?
Attól függ, hogy hogy áll a projekt. Néhány projekt kicsi, szóval csak 1-2 nap szükséges az olvasáshoz, megértéshez. Néhány projekt nagy, lehet, hogy több napra is szükséged lesz.

Mennyire érdekes ez a tevékenység?
Unalmas. Sosem láttam még egy tesztelőt sem, aki azt mondta volna, hogy szereti a tesztspecifikációk olvasását. Még inkább unalmas olyan esetekben, ahol nincs hozzáférésed a tesztelni kívánt programhoz, miközben olvasod a dokumentumokat. Ez azt jelenti, hogy a dokumentum olvasása közben el kell képzelned, hogy a rendszer hogyan fog működni és az egyes elemei hogyan illenek össze.

Mennyire fontos ez a tevékenység?
Nagyon fontos. Még ha a dokumentum olvasás unalmas is természeténél fogva, ez a tevékenység nagyon fontos. Minél többet tudsz a tesztelés alatt álló rendszerről, annál jobb teszteseteket tudsz írni és jobb eséllyel találhatod meg a hibákat.

2) Tesztesetek kidolgozása vagy teszteset írás

Ha a projekted egy tipikus projekt, ami a megszokott szoftverfejlesztési életciklust követi és a projekt elején vagytok, akkor biztosan részed lesz ebben a tevékenységben. Ez a tevékenység a teszteset kidolgozás vagy teszteset írás.

Azt hiszem már sejted, hogy mire gondolok, amikor teszteset kidolgozásra gondolok. Ha nem, akkor lehet, hogy érdemes utánanézned a neten fellelhető (egyébként ingyenes) kurzusomnak, amely arról szól, hogy hogyan dolgozzunk ki teszteseteket.

Itt jegyezném meg, hogy több módja is van a teszteseteid megírásának, bemutatásának. Alapulhatnak egy pipálós lista elvén, mindmap felépítésű elven vagy egy lépésről-lépésre kidolgozott teszteset elvén is, mint amilyet a kurzusomon mutatok be.

Ez most túl soknak tűnhet, de ne aggódj emiatt. A felettesed sok esetben el fog látni valamiféle instrukcióval, azzal kapcsolatban, hogy hogyan szokás abban a környezetben tesztesetet írni, így pedig te is könnyebben fogod tudni követni.

Egy másik, a teszteset készítéssel kéz a kézben járó tevékenység a következő:

Tesztesetek felülvizsgálata:
Mivel új vagy, sok esetben sok figyelemmel vesznek körbe. Ez azt jelenti, hogy a dolgok, amiket csinálsz, sok esetben lesznek ellenőrizve a teszt vezetőddel vagy feletteseddel.

Ha végeztél a teszteseteid megalkotásával, a főnököd/felettesed le fog ülni veled beszélgetni, hogy átnézzétek és jóváhagyja a teszteseteket, hogy biztos legyen, hogy a megfelelő dolgokat és jól végzed. Ha már elsőre is megfelelően végzed a munkád, az nagyszerű, de a legtöbb esetben vissza kell majd menned az alkotóasztalhoz és megváltoztatnod a teszt eseted néhány helyen.

Ne fogd ezt fel rossz tapasztalatként. Az mindig jó, ha van valaki, aki segít az általad alkotottak felülvizsgálatában, véleményt alkot és jobbá tesz téged. Ez szintén egy nagyszerű lehetőség, hogy feltedd a kérdéseidet és tisztázzátok a dolgokat, hogy még többet megtudj a rendszerről.

Még ha a tesztesetek kidolgozása nem is a legizgalmasabb tevékenység, annyira azért nem is unalmas. Néhány tesztelő, akit ismerek, egészen szereti ezt a tevékenységet, mert van esélyük kiélni a kreativitásukat olyan tekintetben, hogy hogyan akarják tesztelni a rendszert.

ÁLLÁSHIRDETÉS

Szoftvertesztelésre szakosodott személyzeti tanácsadó, a TesterJob megbízói számára munkatársat keres

TESZTELŐ
pozícióba, budapesti munkavégzéssel.

Feladat:
– Tesztesetek tervezése, létrehozása, önálló kivitelezése és adminisztrációja
– Bejövő hibajelentések esetén hibaelhárítás (hibaelemzés, hibajavítás, stb.)
– Folyamatos kommunikáció kollégákkal, csoportvezetőkkel és az ügyfelekkel
– Részvétel olyan karbantartó tevékenységekben, mint a hibajegyek kezelése, problémamegoldás

Elvárások:
– Felsőfokú szakirányú végzettség
– Legalább 2-3 év tesztelési vagy egyéb informatikai tapasztalat
– Aktív, középfokú német ÉS angol nyelvismeret szóban és írásban
– Terhelhetőség, magas problémamegoldó, csapatjátékos attitűd

Előny:
– SAP BW, BI, ABAP ismeretek
–SQL ismeret
– Nemzetközi projektben, külföldön szerzett tapasztalat

Amit kínálunk:
–Versenyképes jövedelem és juttatási csomag
–Ergonómikus, modern munkakörnyezet az Infoparkban
– Folyamatos fejlődés lehetősége az informatika és a nyelvek területén.
– Karrier lehetőség
– Multinacionális környezet: részvétel a T-Systems nemzetközi projektjeiben

Jelentkezés:
Állásajánlatainkat megtalálja és közvetlenül jelentkezhet hirdetéseinkre www.testerjob.hu oldalon.

ÁLLÁSHIRDETÉS

Szoftvertesztelésre szakosodott személyzeti tanácsadó, a TesterJob megbízói számára munkatársat keres

TESZTELŐ

pozícióba, budapesti munkavégzéssel.

Feladat:
– Tesztesetek tervezése, létrehozása, önálló kivitelezése és adminisztrációja
– Bejövő hibajelentések esetén hibaelhárítás (hibaelemzés, hibajavítás, stb.)
– Folyamatos kommunikáció kollégákkal, csoportvezetőkkel és az ügyfelekkel
– Részvétel olyan karbantartó tevékenységekben, mint a hibajegyek kezelése, problémamegoldás

Elvárások:
– Felsőfokú szakirányú végzettség
– Legalább 2-3 év tesztelési vagy egyéb informatikai tapasztalat
– Aktív, középfokú német ÉS angol nyelvismeret szóban és írásban
– Terhelhetőség, magas problémamegoldó, csapatjátékos attitűd

Előny:
– SAP BW, BI, ABAP ismeretek
–SQL ismeret
– Nemzetközi projektben, külföldön szerzett tapasztalat

Amit kínálunk:
–Versenyképes jövedelem és juttatási csomag
–Ergonómikus, modern munkakörnyezet az Infoparkban
– Folyamatos fejlődés lehetősége az informatika és a nyelvek területén.
– Karrier lehetőség
– Multinacionális környezet: részvétel a T-Systems nemzetközi projektjeiben

Jelentkezés:
Állásajánlatainkat megtalálja és közvetlenül jelentkezhet hirdetéseinkre www.testerjob.hu oldalon.

3) Tesztek végrehajtása/ regresszió

Ha van egy új verziód tesztelésre, akkor újra kell futtatnod azokat a teszteseteket, amelyeket megírtál és változás érhette őket. Ezt a folyamatot nevezzük regressziós tesztelési tevékenységeknek.

A regressziós tesztelés alapvetően a következő: Ha van 10 teszteseted, amely azt a felületet érintette, ahol változás állt be, mind a 10 tesztet újra kell futtatni. Ha van 100 teszteseted, akkor 100 esetet kell újra futtatnod.

Remélem érted a lényeget.
A regressziós tesztelés nagyon kifizetődő tud lenni a tesztelés szempontjából, de lehet, hogy egy új problémával szembesülsz: a teszteset frissítéssel.

Amikor újrafuttatod a teszteseteidet az új felületet vizsgálva, feltűnhet, hogy néhány funkció a rendszerben megváltozott és a megírt teszteseteid elavulttá váltak. Ilyenkor vissza kell menned a teszteseteidhez, naprakéssé tenni őket és futtatni őket. Ez nem nagy dolog, ha a változtatás kicsi és csak néhány tesztesetet kell frissíteni. Viszont igen idegesítő problémává nőheti ki magát, ha egy nagyobb változás megy végbe, mely több különböző funkciót is érint és neked több száz tesztesetet kell frissítened emiatt… ebben már semmi vicces nincs.

De ne aggódj, a rendszered nem változik meg folyamatosan. Néhány jobb napon a teszteseteid megteszik, amit kell és még néhány fontos hibát is felfedezhetsz az esetek futtatásával.

Mennyire érdekes ez a regressziós tesztelési tevékenység?
Is-is. Ha csak pár környi regressziós tesztelést kell futtatnod, akkor nincs gond, de ha a regressziós tesztelési ciklusod rövid és újra és újra le kell futtatnod a tesztsorozatodat, unalmassá válhat a folyamat. Viszont, ahogy már mondtam, a regressziós tesztelés nagyon fontos dolog, úgyhogy tetszik vagy sem, előbb vagy utóbb, muszáj leszel megcsinálni.

4) Felderítő tesztelés

Az eddigi tesztelési tevékenységek eléggé unalmasak voltak, de tesztelőként biztos vagyok benne, hogy ezt a tevékenységet imádni fogod: felderítő tesztelés (vagy ad- hoc tesztelés, ha az jobban tetszik).

Ez most a te időd, hogy csillogtasd a tehetségedet. Ez egy olyan lehetőség, ahol felhasználhatod minden tudásodat a rendszerről ahhoz, hogy hibákat találj a benne.

Attól függetlenül, hogy ezt a tevékenységet munkamenet-alapú technikával, vagy szabad tesztelési metódussal végzed, el kell ismernem, ez a folyamat jó szórakozás. Találni fogsz hibákat, amiket a teszteseteid nem fedtek le. Szabadjára engedheted a kreativitásodat és képzelőerődet, hogy olyan esetekre gondolj, ahol a végfelhasználók a rendszert használják, és hogyan ronthatnák el azt.

Mivel a felderítő tesztelés egy fontos tevékenység és szórakoztató is emellett, mindig azt javaslom, hogy szánjunk erre egy nagyobb időintervallumot, mikor ilyen típusú tesztelést végzünk a projektben. Ez tényleg sokat tud segíteni.

5) Hibák jelentése

Ha tesztelő vagy, előbb vagy utóbb hibát is fogsz találni és azokat le kell jelentened. Alapvetően ez csak egy olyan tevékenység, ahol te elmondod az embereknek, hogy milyen problémát találtál, felhívod rá a figyelmüket és ők kijavítják.

Ez nehéz?
A legtöbb esetben a csapatodnak a rendelkezésére áll néhány irányelv vagy formátum, amit követhettek. Csak kövesd az instrukciókat és nem lesz semmi gond.

Ez unalmas?
Nem igazán. Nem ismerlek téged, de a hibák lejelentése számomra egy egész érdekes tevékenység. Nem csak azért, mert így „megmutathatom a világnak” a tesztelési eredményességemet, de azért is, mert segít az íráshoz kapcsolódó képességeim gyakorlásában. Ha egy jó hibajelentést tudok írni, akkor az emberek gyorsabban megértik a problémát és azt, hogy ennek javítása mennyire fontos.

Ez fontos?
Nagyon fontos. Vedd figyelembe, hogy amikor tesztelsz, akkor nem csak teszt eredményeket gyártasz, hanem emellett szolgáltatsz is és az a munkád, hogy annyi információt adj a tesztelés alatt álló rendszerről, amennyit csak lehetséges és a hibalejelentés ezen információk egyike. Ha megfelelően csinálod, az emberek meg fogják becsülni a munkádat.

Néhány csapda, amibe könnyű belesétálni:

- Kerüld a hibák duplikált lejelentését.
- Kerüld a hatályon kívüli hibák lejelentését.
- Kerüld a megismételhetetlen hibák lejelentését.
- Ne hibáztass másokat a hibák előfordulásáért.
- Figyelj arra, hogy a fontos információk a hibalejelentésből, mint a cím, leírás, lépésről-lépésre reprodukálás, stb. ne hiányozzanak.

6) Meetingek

„Nos, a meetingek”… kedveljük vagy sem, a meetingek elkerülhetetlenek a projektekben. A meetingeknek különböző fajtái kerülnek elő egy projekt életciklusa során. Itt van néhány az általánosabbak körül:

(A képforrása: http://dilbert.com/strip/2016-01-25)

Napi meeting:
Ha a csapatod Scrum-ban dolgozik, a napi meetingek kötelező jellegűek. Ezen a meetingen minden csapattag számot ad a munkásságáról a következő 3 kérdés megválaszolásával:

- Mit csináltál tegnap?
- Mit fogsz ma csinálni?
- Van olyan tényező, ami gátolja a haladásodat?

Ha nem Scrumban dolgozol, a napi meetingek akkor is nagyon fontosak olyan esetekben, amikor nagy a csapatod és a csapattársak nem tudnak gyakran kommunikálni.

Hiba átbeszélő meeting / Hiba prioritás meeting
Ezen a meetingen a csapattagok, csapatvezető és/vagy a menedzser fogja átnézni a talált hibákat a tesztelő csapattal (jelentése: veled) és meg fogjátok vitatni a találtakat annak érdekében, hogy:

- Tisztázzatok néhány felmerült kérdést a hibákkal kapcsolatban vagy több információra van szükség a tesztelőktől.
- Döntsétek el, hogy melyik hibát kell minél hamarabb kijavítani és melyik ér rá a későbbiekben.

Az emberek általában nem szeretik a meetingeket. Viszont, ha a projekted nagy és bonyolult, vagy a csapatod nem egy légtérben dolgozik, a folyamatos meetingek erősíthetnek a csapaton belüli kommunikáción. Szóval használd ki ezt a lehetőséget, hogy megfelelő kérdésekkel többet tudj meg a projektről, a tesztelés alatt álló rendszerről és a projekt tervezetéről. Minél többet tudsz, annál jobban tudod a rendszert tesztelni.

7) Jelentsd a tesztelésben való haladásodat

Ha a csapatodban nem túl gyakori a napi meeting vagy egy külsős csapat része vagy, lehet, hogy szükséges lesz a tesztelésben való haladásod, teszt eredményeid jelentésére. Jelentheted a haladásodat e-mailben is, meetingek esetén is, vagy… mindkét módon. Ezt meg lehet csinálni napi vagy heti bontásban a csapatodtól és a csapatodban betöltött szerepedtől függően.

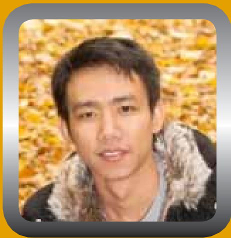
Ne becsüld alá a teszteredmények jelentésének fontosságát. A tesztelésben való haladásod, eredményeid és észlelt blokkoló tényezők megosztása segíteni fog a vezetőidnek az információk gyűjtésében és ennek megfelelően a döntések meghozásában.

Nagyjából ennyi lenne. Megosztottam veled néhány alapvető tesztelési tevékenységet, amikről nem árt előre tudnod tesztelőként. Már írtam korábban és szeretném ismét kiemelni a fontosságát annak, hogy ezek a tevékenységek azok, amiket én megtapasztaltam és nem az összes tevékenység, ami fellelhető ebben a szakmában. Továbbá, amit itt megosztottam, azok alapja az én nézőpontom, érzéseim és a projektjeim tapasztalata. Viszont, ezek tudatában már kaphatsz egy képet arról, hogy mi várhat rád a jövőben.

Ezek tudatában megválaszolhatod a kérdést, hogy te szeretnéd-e ezeket a tevékenységeket űzni… néha minden egyes nap. Valójában ezeket a tevékenységeket fogod vagy nem fogod csinálni minden nap, amiket itt megosztottam. Használd ezt referenciaként. Néha kísérletezned kell és meglátni a dolgokat a saját oldaladról is. ■

Szerző: ***Thanh Huynh***

Forrás: http://www.asktester.com/7-common-software-testing-activities/



Thanh Huynh

Thanh egy vietnámi tesztelő, valamint az AskTester.com oldal tulajdonosa amely segít az új tesztelőknek jobban megérteni a tesztelést. Több éves tapasztalata van a tesztelésben, menedzselésben és tesztelési projektek vezetésében is. Számára az ügyfelek elégedettsége, a fenntarthatóság és a költséghatékonyság a legfontosabb egy tesztelési munka folyamán. Más tesztelés-orientált tevékenységek is érdeklik, mint a blogolás, konferenciákon való előadások és a tesztelői közösség építése. Egy introvertált, csendes és családorientált ember, aki ha nem teszteléssel foglalkozik, akkor általában blogokat, könyveket olvas, a kisfiával játszik, kávét iszik és melodikus metál zenét hallgat… nos, néha mindezt egyszerre.

Publikálj nálunk!

Tesztelési tapasztalataim

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

TESZTELÉS
A GYAKORLATBAN
2013. DECEMBER
A SZAKÉRTŐ TESZTELŐK LAPJA
2014/I. szám
Szakmai ismeretek
Gyakorlati tapasztalatok
Új trendek, módszerek
Kizárólag szoftvertesztelésről
szóló cikkek
Negyedévente olvashatod
2013. JÚNIUS
2013. SZEPTEMBER