Útmutató kezdőknek a szoftvertesztelés

- Padmini C

Tartalomjegyzék:

1. Áttekintés 5

A nagy kép 5

Mi az a szoftver? Miért kell tesztelni? 6

Mi a minőség? Mennyire fontos? 6

Mit csinál pontosan egy szoftvertesztelő? 7

Milyen a jó tesztelő? 8

Útmutató az új tesztelők számára..... 9

2. Bevezetés..... ..... .11

Szoftver életciklusa..... 11

Különféle életciklus-modellek 12

A szoftvertesztelés életciklusa 13

Mi az a hiba? Miért fordulnak elő hibák? 15

A hiba életciklusa 16

A hibák kijavításának költsége..... 17

Mikor lehet leállítani/csökkenteni a tesztelést?..... 18

3. Szoftvertesztelési szintek, típusok, kifejezések és definíciók.....19

Tesztelési szintek és típusok..... 19

Tesztelési feltételek..... 22

A leggyakoribb szoftverhibák..... 23

Hibatípusok példákkal..... 23

5. A teszttervezési folyamat..... 25

Mi az a tesztstratégia? Mik az összetevői?.....25

Teszttervezés – Mintastruktúra 25

Főbb teszttervezési feladatok..... 26

6. Teszteset kidolgozása..... 27

Általános irányelvek..... 27

Teszteset – Mintastruktúra..... 27

Tesztesetek tervezési technikái 28

Mi az a használati eset? 30

7. Hibakövetés..... 31

Mi a hiba?..... 31

Melyek a hibakategóriák?..... 31

Hogyan történik a hiba bejelentése?..... 32

Mennyire legyen leíró jellegű a hiba-/hibajelentés?.....32

Mit tesz a tesztelő, ha a hibát kijavították?.....33

8. A tesztjelentések típusai..... 34

9. Szoftverteszt-automatizálás 35

Az automatizálás megközelítései..... 36

A megfelelő szerszám kiválasztása..... 37

A szoftverteszt-automatizálás tíz legnagyobb kihívása.....37

10. Bevezetés a szoftverszabványokba..... 38

Hat Szigma..... 38

ISO 39

11. Szoftvertesztelési tanúsítványok..... 40

12. Tények a szoftverfejlesztésről 41

1. Áttekintés

A nagy kép

Minden szoftveres probléma nevezhető hibának. A szoftverhiba általában akkor fordul elő, ha a szoftver nem azt csinálja, amire szánták, vagy olyasmit tesz, amire nem szánták. A specifikációk, a tervezés, a kód vagy más okok hibái okozhatják ezeket a hibákat. A hibák azonosítása és javítása a szoftver korai szakaszában nagyon fontos, mivel a hibák javításának költsége idővel nő. A szoftvertesztelő célja tehát az, hogy a hibákat a lehető legkorábban megtalálja, és gondoskodjon a javításukról.

A tesztelés kontextus alapú és kockázatvezérelt. Módszeres és fegyelmezett megközelítést igényel a hibák feltárása. Egy jó szoftvertesztelőnek hitelessé kell válnia, és feltáró, hibaelhárító, könyörtelen, kreatív, diplomatikus és meggyőző hozzáállással kell rendelkeznie.

Ellentétben azzal a felfogással, hogy a tesztelés csak a kódolási fázis befejezése után kezdődik, valójában még az első kódsor megírása előtt kezdődik. A hagyományos szoftvertermék életciklusában a tesztelés abban a szakaszban kezdődik, amikor a specifikációkat megírják, azaz a termékspecifikáció vagy termékspecifikáció tesztelésétől.
Ha ebben a szakaszban hibákat talál, rengeteg időt és pénzt takaríthat meg.

Miután a specifikációkat jól megértette, meg kell terveznie és végre kell hajtania a teszteseteket. A megfelelő technika kiválasztása, amely csökkenti az adott szolgáltatást lefedő tesztek számát, az egyik legfontosabb dolog, amelyet figyelembe kell vennie a tesztesetek tervezése során. A teszteseteket úgy kell megtervezni, hogy a szoftver minden aspektusát lefedjék, azaz a biztonságot, az adatbázist, a funkcionalitást (kritikus és általános) és a felhasználói felületet. A hibák a tesztesetek végrehajtása során keletkeznek.

Tesztelőként előfordulhat, hogy különböző körülmények között tesztelést kell végeznie, azaz az alkalmazás kezdeti szakaszban van vagy gyors változásokon megy keresztül, kevesebb ideje van a tesztelésre, előfordulhat, hogy a terméket olyan életciklus-modell segítségével fejlesztik, amely nem támogatja formális tesztelés vagy újratesztelés nagy része. Továbbá gondoskodni kell a különböző operációs rendszerekkel, böngészőkkel és konfigurációkkal történő tesztelésről.

A hibajelentés lehet a legfontosabb és néha a legnehezebb feladat, amelyet szoftvertesztelőként végre kell hajtania. Különféle eszközök használatával és a fejlesztővel való egyértelmű kommunikációval biztosíthatja, hogy a talált hibákat kijavítsák.

A tesztek végrehajtására, szkriptek futtatására és a hibák nyomon követésére szolgáló automatizált eszközök használata javítja a tesztek hatékonyságát és eredményességét.
Ezenkívül, ha lépést tart a terület legújabb fejleményeivel, ez növeli szoftvertesztmérnöki karrierjét.

Mi az a szoftver? Miért kell tesztelni?

A szoftver egy olyan utasítássorozat a számítógép számára, amely egy adott feladatot hajt végre, úgynevezett program; a szoftverek két fő kategóriája a rendszerszoftver és az alkalmazásszoftver. A rendszerszoftver vezérlőprogramokból áll. Alkalmazási szoftver minden olyan program, amely a felhasználó számára adatokat dolgoz fel (táblázat, szövegszerkesztő, bérszámfejtés stb.).

Egy szoftverterméket csak azután szabad kiadni, miután megfelelő fejlesztési, tesztelési és hibajavítási folyamaton ment keresztül. A tesztelés olyan területeket vizsgál, mint a teljesítmény, a stabilitás és a hibakezelés, ellenőrzött körülmények között tesztforgatókönyvek felállításával és az eredmények értékelésével. Éppen ezért minden szoftvert tesztelni kell. Fontos megjegyezni, hogy a szoftvereket elsősorban azért tesztelik, hogy megbizonyosodjanak arról, hogy megfelelnek-e az ügyfelek igényeinek, és megfelelnek-e a szabványoknak. Szokásos norma, hogy a szoftver akkor tekinthető jó minőségűnek, ha megfelel a felhasználói követelményeknek.

Mi a minőség? Mennyire fontos?

A minőség röviden a „kiválóság bizonyos fokaként” definiálható. A jó minőségű szoftverek általában megfelelnek a felhasználói igényeknek. A vevő minőségről alkotott elképzelése sokféle funkciót lefedhet – a specifikációknak való megfelelés, jó teljesítmény a platform(ok)on/konfigurációkban, teljes mértékben megfelel az üzemeltetési követelményeknek (még ha nincs is meghatározva!), kompatibilitás az összes végfelhasználói berendezéssel, nincs negatív hatás a meglévő végfelhasználói bázison a bevezetés időpontjában.

A minőségi szoftver sok időt és pénzt takarít meg. Mivel a szoftvernek kevesebb hibája lesz, ez időt takarít meg a tesztelési és karbantartási szakaszok során. A nagyobb megbízhatóság hozzájárul a vevői elégedettség mérhetetlen növekedéséhez, valamint az alacsonyabb karbantartási költségekhez. Mivel a karbantartás az összes szoftverköltség nagy részét teszi ki, a projekt összköltsége valószínűleg alacsonyabb lesz, mint a hasonló projekteknél.

Az alábbi két eset mutatja be a szoftverminőség fontosságát:

Az Ariane 5 lezuhan 1996. június 4-én

- Az európai Ariane 5 hordozórakéta első repülése körülbelül 40 másodperccel később lezuhant levesz
- A veszteség körülbelül félmilliárd dollár volt
    - A robbanást szoftverhiba okozta
    - Lebegőpontos hiba miatt nem észlelt kivétel: konverzió 64 bites egészről 16 bites előjelű egész számra a vártnál nagyobbra alkalmazva

szám

- A modult újra használták az Ariane 4 megfelelő tesztelése nélkül
- Nem kellett volna hibának megtörténnie az Ariane 4-nél
- Nincs kivételkezelő

Mars Climate Orbiter – 1999. szeptember 23

- Mars Climate Orbiter, eltűnt, amikor elkezdett keringeni a Mars körül.
- Költség körülbelül 125 millió USD

és egy csapat Kaliforniában

- Az egyik csapat angol mértékegységeket (pl. hüvelyk, láb és font), míg a másik metrikus mértékegységeket használt egy kulcsfontosságú űrhajó művelethez.

Mit csinál pontosan egy szoftvertesztelő?

A szoftvertermékek hibáinak ("hibáinak") feltárásán kívül, amelyek megerősítik, hogy a program megfelel a program specifikációinak, tesztmérnökként teszteseteket, eljárásokat, szkripteket kell létrehoznia, és adatokat kell generálnia. Teszteljárásokat és szkripteket hajt végre, szabványokat elemez, és értékeli a rendszer/integrációs/regressziós tesztelés eredményeit. Te is...

- A fejlesztési folyamat felgyorsítása a hibák korai szakaszában történő azonosításával (pl. specifikációs szakaszban) • Csökkentse a szervezet jogi felelősségvállalásának kockázatát • Maximalizálja a szoftver értékét • Biztosítsa a termék sikeres bevezetését, pénzt, időt és a vállalat hírnevét takarítson meg a hibák és tervezési hibák korai szakaszában történő felfedezése, mielőtt a gyártásban vagy a terepen meghibásodások lépnének fel
- Folyamatos fejlődés elősegítése

Milyen a jó tesztelő?

Mivel a szoftvermérnökséget ma már műszaki mérnöki szakmaként tartják számon, fontos, hogy a szoftvertesztelő mérnök rendelkezzen bizonyos tulajdonságokkal, és könyörtelen hozzáállással, hogy kiemelkedjen. Íme néhány.

• Ismerje a technológiát. Az alkalmazás fejlesztésének technológiájának ismerete minden tesztelő számára további előnyt jelent. Segít jobb és hatékonyabb tesztesetek tervezésében a technológia gyengeségei vagy hibái alapján. A jó tesztelők tudják, hogy mit támogat és mit nem, így ha ezekre a vonalakra koncentrál, az segít gyorsan feltörni az alkalmazást. • Perfekcionista és realista. A perfekcionista lét segít a tesztelőknek felismerni a problémát, realistának lenni pedig segít a nap végén tudni, hogy mely problémák igazán fontosak. Tudni fogja, hogy melyiket kell javítani, és melyiket nem. • Tapintatos, diplomatikus és meggyőző. A jó szoftvertesztelők tapintatosak, és tudják, hogyan közöljék a híreket a fejlesztőkkel. Diplomatikusak, miközben meggyőzik a fejlesztőket a hibákról, és szükség esetén meggyőzik őket, és javítják a hibákat. Fontos, hogy kritikusan álljunk a témához, és ne hagyjuk, hogy az alkalmazás kifejlesztője meghökkenjen a megállapításokon.

• Egy felfedező. Egy kis kreativitás és kockázatvállalási hozzáállás segít a tesztelőknek ismeretlen helyzetekbe merészkedni, és olyan hibákat találni, amelyeket egyébként átnéznek. • Hibaelhárítás. A hibaelhárítás és annak kiderítése, hogy valami miért nem működik, segít a tesztelőknek abban, hogy magabiztosan és egyértelműen közöljék a hibákat a fejlesztőkkel. • Emberi készségekkel és kitartással rendelkezik. A tesztelők nagy ellenállással szembesülhetnek a programozók részéről. Szociálisan okosnak és diplomatikusnak lenni nem azt jelenti, hogy határozatlan. A legjobb tesztelők társadalmilag ügyesek és kitartóak ott, ahol ez számít. • Szervezett. A legjobb tesztelők nagyon jól tudják, hogy ők is hibázhatnak, és nem kockáztatnak. Nagyon jól szervezettek, ellenőrző listákkal rendelkeznek, megállapításaik alátámasztására aktákat, tényeket és számadatokat használnak, amelyek bizonyítékként használhatók, és kétszer is ellenőrizhetik megállapításaikat. • Objektív és pontos. Nagyon tárgyilagosak és tudják, mit jelentenek, így pártatlan és értelmes információkat közvetítenek, amelyek távol tartják a politikát és az érzelmeket az üzenetektől. A pontatlan információk bejelentése némileg elveszti hitelességét. A jó tesztelők gondoskodnak arról, hogy eredményeik pontosak és reprodukálhatók legyenek. • A hibák értékesek. A jó tesztelők tanulnak tőlük. Minden hiba egy lehetőség a tanulásra és a fejlődésre. A korán felfedezett hiba lényegesen kevesebbe kerül, mint a későbbi szakaszban észlelt hiba. A hibák komoly problémákat okozhatnak, ha nem kezelik megfelelően. A hibákból való tanulás segít – a jövőbeni problémák megelőzésében, a fejlesztések nyomon követésében, az előrejelzés és a becslés javításában.

Útmutató az új tesztelőknek

• A tesztelés nem tudja kimutatni, hogy nem léteznek hibák. A tesztelés fontos oka a hibák megelőzése. Elvégezheti a teszteket, megtalálhatja és jelentheti a hibákat, de soha nem garantálhatja, hogy nincsenek hibák. • Lehetetlen egy program teljes körű tesztelése. Sajnos ez még a legegyszerűbb programmal sem lehetséges, mert – a bemenetek száma nagyon nagy, a kimenetek száma nagyon nagy, a szoftveren keresztüli utak száma nagyon nagy, és a specifikáció szubjektív a gyakori változtatásokra. • Nem tudja garantálni a minőséget. Szoftvertesztelőként nem tesztelhet mindent, és nem vállal felelősséget a termék minőségéért. A tesztelő kudarcának fő módja az, hogy nem jelenti be pontosan a megfigyelt hibát. Fontos megjegyezni, hogy a minőséget ritkán tudjuk ellenőrizni. • Célkörnyezet és tervezett végfelhasználó. Az alkalmazás előrejelzése és tesztelése abban a környezetben, amelyet a felhasználók várhatóan használnak, az egyik fő szempont, amelyet figyelembe kell venni. Ezenkívül annak mérlegelése, hogy az alkalmazás egyfelhasználós rendszer-e vagy többfelhasználós rendszer, fontos annak bizonyításához, hogy szükség esetén azonnali készenlétben áll rendelkezésre. A Disney Oroszlánkirályának hibaesete ezt szemlélteti. A Disney Company kiadta első multimédiás CD-ROM játékát gyerekeknek, az Oroszlánkirály Animated Storybookot. Nagyon népszerűsítették, és az eladások hatalmasak voltak. Hamarosan hírek érkeztek arról, hogy a vásárlók nem tudták működésre bírni a szoftvert. Néhány rendszeren működött -

valószínűleg azokat, amelyeket a Disney programozói használtak a játék létrehozásához – de nem a nagyközönség által használt leggyakoribb rendszereken.
• Egyetlen alkalmazás sem 100%-ban hibamentes. Ésszerűbb felismerni, hogy vannak olyan prioritások, amelyek miatt néhány kevésbé kritikus probléma megoldatlan vagy azonosítatlan marad. Egyszerű ház az Intel Pentium bug. Írja be a következő egyenletet a számítógépe számológépébe: (4195835 / 3145727) * 3145727 – 4195835. Ha a válasz nulla, akkor a számítógép rendben van. Ha bármi mást kap, akkor egy régi Intel Pentium CPU-ja van, lebegőpontos felosztási hibával. • Legyen az ügyfél. Próbálja meg laikus felhasználóként használni a rendszert. Ahhoz, hogy bepillantást nyerjen ebbe, kérjen meg egy személyt, akinek fogalma sincs az alkalmazásról, hogy egy ideig használja azt, és meg fog lepődni, amikor látja, hogy az illető milyen problémákkal találkozik. Amint látja, nincs eljárás. Ha ezt megteszi, a rendszer váratlan tesztek sorával találkozhat – ismétlés, stressz, terhelés, verseny stb. • Növelje hitelességét. A hitelesség olyan, mint a minőség, amely magában foglalja a megbízhatóságot, a tudást, következetesség, hírnév, bizalom, hozzáállás és a részletekre való odafigyelés. Nem azonnali, hanem idővel kell építeni, és hangot ad a szervezet tesztelőinek. Az Ön kulcsa a hitelesség növeléséhez – azonosítsa erősségeit és gyengeségeit, építsen jó kapcsolatokat, mutasson kompetenciát, és legyen hajlandó beismerni a hibákat, újraértékelni és alkalmazkodni. • Tesztelje, amit megfigyel. Nagyon fontos, hogy tesztelje azt, amit megfigyelhet és amihez hozzáférhet. Kreatív tesztesetek írása csak akkor segíthet, ha megvan a

lehetőség az eredmények megfigyelésére. Szóval ne feltételezz semmit. • Nem minden talált hibát javítunk ki. Annak eldöntése, hogy mely hibákat javítják ki és melyeket nem, kockázatalapú döntés. A hiba kijavításának több oka is lehet, ha nincs elég idő, a hibát elvetik egy új funkcióhoz, a javítás nagyon kockázatos lehet, vagy nem éri meg, mert ritkán fordul elő, vagy van megoldás, ahol a a felhasználó megakadályozhatja vagy elkerülheti a hibát. A rossz döntés katasztrofális lehet. • Tekintse át a versenyképes termékeket. Ha jó betekintést nyer az azonos típusú termékekbe, valamint megismeri azok funkcionalitását és általános viselkedését, akkor könnyebben megtervezheti a különböző teszteseteket, és megértheti az alkalmazás erősségeit és gyengeségeit. Ez azt is lehetővé teszi, hogy hozzáadott értéket és új funkciókat és fejlesztéseket javasoljon termékéhez. • Kövesse a szabványokat és folyamatokat. Tesztelőként Önnek meg kell felelnie a szervezet által meghatározott szabványoknak és irányelveknek. Ezek a szabványok a jelentési hierarchiára, a kódolásra, a dokumentációra, a tesztelésre, a hibák jelentésére, az automatizált eszközök használatára stb. vonatkoznak.

2. Bevezetés

Szoftver életciklusa

A szoftver életciklusa jellemzően a következőket tartalmazza: követelményelemzés, tervezés, kódolás, tesztelés, telepítés és karbantartás. A kettő között előfordulhat, hogy a termékhez műveleteket és támogató tevékenységeket kell biztosítani.

Követelmények elemzése. A szoftverszervezetek megoldásokat kínálnak az ügyfelek igényeire a specifikációiknak leginkább megfelelő szoftver kifejlesztésével. Így a szoftver élete a követelmények eredetével kezdődik. Ezek a követelmények nagyon gyakran homályosak, felmerülőek és mindig változhatnak.

Elemzést végeznek: - A javasolt projekt mélyreható elemzéséhez értékelje a műszaki megvalósíthatóságot, feltárja a rendszer particionálásának módját, azonosítsa, hogy a követelmények mely területeit kell kidolgozni az ügyféltől, azonosítsa a követelmények változásainak hatását, hogy azonosítsa, mely követelményeket melyik összetevőhöz kell hozzárendelni.

Tervezés és specifikációk. A követelményelemzés eredménye a követelményspecifikáció. Ennek felhasználásával kerül kidolgozásra a tervezett szoftver általános kialakítása.

Tevékenységek ebben a fázisban - A szoftver építészeti tervezése, tervezési adatbázis (ha van), felhasználói felületek tervezése, algoritmusok kiválasztása vagy fejlesztése (ha alkalmazható), részletes tervezés végrehajtása.

Kódolás. A fejlesztési folyamat inkább iteratív módon fut végig ezeken a fázisokon, nem pedig lineárisan; Számos modellt (spirál, vízesés stb.) javasoltak ennek a folyamatnak a leírására.

Tevékenységek ebben a fázisban – Tesztadatok létrehozása, Forrás létrehozása, Objektumkód generálása, Működési dokumentáció készítése, integráció tervezése, integráció végrehajtása

Tesztelés. A kifejlesztett rendszer hibakeresési célú használatának folyamata. Az ebben a szakaszban talált hibákat/hibákat visszaküldjük a fejlesztőnek javításra, és újra kell tesztelni. Ez a fázis iteratív mindaddig, amíg a hibákat a követelményeknek megfelelően javítják.

Tevékenységek ebben a fázisban - Tervellenőrzés és érvényesítés, Ellenőrzési és érvényesítési feladatok végrehajtása, Metrikus adatok gyűjtése és elemzése, Terv tesztelése, Teszt fejlesztése Követelmények, tesztek végrehajtása

Telepítés. Az így kifejlesztett és tesztelt szoftvert végre a kliens helyére kell telepíteni. Gondos tervezést kell végezni annak érdekében, hogy elkerülje a felhasználó problémáit a telepítés után.

Tevékenységek ebben a fázisban - Telepítés megtervezése, Szoftver terjesztése, Szoftver telepítése, Szoftver elfogadása működési környezetben.

Működés és támogatás. A támogató tevékenységeket általában a szoftvert kifejlesztő szervezet végzi. Ezekről a tevékenységekről általában mindkét fél dönt a rendszer kidolgozása előtt.

Tevékenységek ebben a fázisban – A rendszer működtetése, technikai segítségnyújtás és tanácsadás, támogatási kérelmek naplójának karbantartása.

Karbantartás. A folyamat nem áll le, amint teljesen implementálják és telepítik a felhasználónál; ebben a fázisban új funkciók, fejlesztések stb.

Tevékenységek ebben a fázisban – Szoftver életciklusának újbóli alkalmazása.

Különféle életciklus modellek

Egy adott alkalmazás tesztelési módja nagyban függ az általa követett életciklus-modelltől. Ennek az az oka, hogy minden életciklus-modell a szoftver különböző aspektusaira helyezi a hangsúlyt, azaz bizonyos modellek megfelelő teret és időt biztosítanak a teszteléshez, míg mások nem. Tehát a kifejlesztett tesztesetek száma, a lefedett funkciók és az egyes kérdésekre fordított idő az alkalmazás által követett életciklus-modelltől függ.

Nem számít, milyen az életciklus-modell, minden alkalmazás ugyanazokon a fázisokon megy keresztül, mint a fent leírtak, mint életciklusa.

Az alábbiakban bemutatunk néhány szoftver életciklus-modellt, azok előnyeit és hátrányait.

Vízesés modell	Prototípuskészítési modell	Spirál modell
Erősségek:	Erősségek: •A	Erősségek:
•Kiemeli az egyik fázis befejezését a költözés előtt tovább • Hangsúlyozza a korai tervezést, az ügyfelek hozzájárulását és a tervezést • Hangsúlyozza a tesztelést, mint az életciklus szerves részét	követelmények beállíthatók korábban és megbízhatóbban • A követelmények egyértelműbben és teljesebben kommunikálhatók a fejlesztők és az ügyfelek között • Követelmények és tervezés a lehetőségek gyorsan és alacsony költséggel vizsgálhatók •További követelmények és	• Elősegíti a meglévő szoftverek újrafelhasználását a fejlesztés korai szakaszában. • Lehetővé teszi a minőségi célok megfogalmazását a fejlesztés során. • Felkészülést biztosít az esetleges fejlődésre szoftver termék. • Kiküszöböli a hibákat és a nem vonzó alternatívákat korai.

	a tervezési hibákat korán észlelik	• Kiegyensúlyozza az erőforrás-kiadásokat. • Nem tartalmaz külön megközelítést a szoftverfejlesztéshez és a szoftver karbantartásához. • Életképes keretet biztosít integrált hardver szoftverrendszer fejlesztéshez.
Gyengeség:	Gyengeség:	Gyengeség:
• Az életciklus korai szakaszában jelentkező rögzítési és fagyasztási igényektől függ • A követelményeknek a tervezéstől való elválasztásától függ • A visszacsatolás csak a tesztelési fázistól az előző szakaszig terjed • Egyes esetekben nem kivitelezhető szervezetek • A termékekre helyezi a hangsúlyt a folyamatok helyett	• Prototípuskészítő eszközt és szakértelmet igényel a használatához – ez költség a fejlesztő szervezet számára •A prototípus gyártási rendszerré válhat	• Ez a folyamat gyors alkalmazásfejlesztést igényel, vagy általában ahhoz kapcsolódik, ami gyakorlatilag nagyon nehéz. • A folyamat nehezebben kezelhető, és a vízesés-modelltől eltérő megközelítést igényel (a vízesés-modell olyan kezelési technikákat tartalmaz, mint a GANTT-diagramok értékelésére)

Szoftvertesztelés életciklusa

A szoftvertesztelés életciklusa hat (általános) szakaszból áll: 1) Tervezés, 2) Elemzés, 3) Tervezés, 4) építés, 5) tesztelési ciklusok, 6) végső tesztelés és megvalósítás és 7) Végrehajtás után. Az életciklus minden fázisa a megfelelő tevékenységekkel van leírva.

Tervezés. Magas szintű teszterv, minőségbiztosítási terv (minőségi célok), azonosítás – jelentési eljárások, problémabesorolás, elfogadási kritériumok, tesztelési adatbázisok, mérési kritériumok (hiba mennyisége/súlyossági szintje és hiba eredete), projektmetrikák és végül a projekt ütemezésének megkezdése. tesztelés. Tervezze meg az összes karbantartását is

tesztesetek (kézi vagy automatizált) egy adatbázisban.

Elemzés. Olyan tevékenységeket foglal magában, amelyek - az üzleti követelményeken alapuló funkcionális érvényesítést fejlesztenek (tesztesetek írása ezen adatok alapján), teszteset formátumot (időbecslések és prioritás-hozzárendelések), tesztciklusokat (mátrixok és idővonalak) fejlesztenek, azonosítják az automatizálandó teszteseteket (ha alkalmazható), meghatározza a stressz- és teljesítményteszt területét, megtervezi a projekthez és a regressziós teszteléshez szükséges tesztciklusokat, meghatározza az adatkarbantartási eljárásokat (mentés, visszaállítás, érvényesítés), áttekinti a dokumentációt.

Tervezés. Tevékenységek a tervezési fázisban - Teszterv módosítása a változtatások alapján, a tesztciklus mátrixok és idővonalak felülvizsgálata, annak ellenőrzése, hogy a teszterv és esetek adatbázisban vagy kellékben vannak-e, folytassa a tesztesetek írását és újak hozzáadása a változások alapján, kockázatértékelési kritériumok kidolgozása , formalizálja a stressz- és teljesítményteszt részleteit, véglegesítse a tesztciklusokat (a tesztesetek száma ciklusonként a tesztesetenkénti időbecslések és a prioritás alapján), véglegesítse a Teszttervet (az egységtesztelés fejlesztését támogató erőforrások becslése).

Építés (egységtesztelési fázis). Végezze el az összes tervet, töltse ki a tesztciklus-mátrixokat és idővonalakat, töltse ki az összes tesztesetet (kézi), kezdje el a stressz- és teljesítménytesztet, tesztelje az automatizált tesztelőrendszert és javítsa ki a hibákat, (támogassa a fejlesztést az egységteszteknél), futtassa a minőségbiztosítási elfogadási tesztcsomagot a szoftverek tanúsításához kész átadni a minőségbiztosításnak.

Tesztciklus(ok) / Hibajavítások (Újratesztelés/Rendszertesztelési fázis). Futtassa a teszteseteket (elülső és hátsó), hibajelentést, ellenőrzést, és szükség szerint módosítsa/adja hozzá a teszteseteket.

Végső tesztelés és megvalósítás (a kód befagyasztásának fázisa). Az összes előtérbeli teszteset végrehajtása - manuális és automatizált, az összes háttérteszt végrehajtása - manuálisan és automatizáltan, az összes stressz- és teljesítményteszt végrehajtása, folyamatos hibakövetési mérőszámok biztosítása, folyamatos összetettségi és tervezési mérőszámok biztosítása, becslések frissítése tesztesetek és teszttervek esetében dokumentálja a tesztciklusokat, a regressziós tesztelést, és ennek megfelelően frissítse.

Végrehajtás után. A megvalósítás utáni értékelő értekezlet lebonyolítható a teljes projekt áttekintésére. Tevékenységek ebben a fázisban - Végső hibajelentés és a kapcsolódó mérőszámok elkészítése, stratégiák azonosítása a hasonló problémák megelőzésére a jövőbeni projektben, automatizálási csapat - 1) Tesztesetek áttekintése a regressziós teszteléshez automatizálandó egyéb esetek értékeléséhez, 2) Az automatizált tesztesetek tisztítása és változók, és 3) Az automatizált tesztelés eredményeinek a kézi tesztelés eredményeivel való integrálásának folyamata.

Mi az a hiba? Miért fordulnak elő hibák?

A szoftverhiba olyan kódolási hibaként definiálható, amely váratlan hibát, hibát, hibát vagy hiányosságot okoz egy számítógépes programban. Más szóval, ha egy program nem a rendeltetésszerűen működik, az valószínűleg hiba .

Vannak szoftverhibák a tisztázatlan vagy folyamatosan változó követelmények, a szoftver bonyolultsága, programozási hibák, idővonalak, hibakövetési hibák, kommunikációs hiányosságok, dokumentációs hibák, szabványoktól való eltérés stb. miatt.
• A tisztázatlan szoftverkövetelmények abból adódnak, hogy a szoftvernek mit kell tennie vagy mit nem kell téves kommunikációból. Sok esetben előfordulhat, hogy az ügyfél nem teljesen világos a termék végső működését illetően. Ez különösen igaz, ha a szoftvert egy teljesen új termékhez fejlesztették ki. Az ilyen esetek általában sok félreértelmezéshez vezetnek bármelyik vagy mindkét oldalról. • A folyamatosan változó szoftverkövetelmények sok zűrzavart és nyomást okoznak mind a fejlesztői, mind a tesztelési csapatban. Gyakran egy új hozzáadott vagy eltávolított szolgáltatás összekapcsolható a szoftver többi moduljával vagy összetevőjével.
Az ilyen problémák figyelmen kívül hagyása hibákat okoz. • Ezenkívül a szoftver egyik részében/összetevőjében lévő hiba javítása egy másik vagy ugyanazon összetevőben előfordulhat. Az ilyen problémák előrejelzésének hiánya komoly problémákat és a hibák számának növekedését okozhatja. Ez az egyik fő probléma, ami miatt hibák fordulnak elő, mivel a fejlesztők nagyon gyakran vannak kitéve nyomásnak az idővonalak miatt; gyakran változó követelmények, a hibák számának növekedése stb.
• Tervezés és újratervezés, UI interfészek, modulok integrációja, adatbázis-kezelés – mindez tovább bonyolítja a szoftvert és a rendszer egészét.

• A szoftvertervezéssel és architektúrával kapcsolatos alapvető problémák problémákat okozhatnak a programozásban. A kifejlesztett szoftverek hajlamosak a hibákra, mivel a programozók is hibázhatnak. Tesztelőként ellenőrizheti az adathivatkozási/deklarációs hibákat, a vezérlési folyamathibákat, a paraméterhibákat, a bemeneti/kimeneti hibákat stb. • Az erőforrások átütemezése, a már elkészült munkák újbóli elvégzése vagy elvetése, a hardver/szoftver követelmények változásai befolyásolhatják a szoftver is. Ha félúton új fejlesztőt rendel a projekthez, az hibákat okozhat. Ez akkor lehetséges, ha nem követik a megfelelő kódolási szabványokat, nem megfelelő kóddokumentáció, nem hatékony tudásátadás stb. A meglévő kód egy részének eldobása nyomot hagyhat a szoftver más részein; az ilyen kód figyelmen kívül hagyása vagy elmulasztása hibákat okozhat. Súlyos hibák különösen nagyobb projekteknél fordulhatnak elő, mivel egyre nehezebb azonosítani a problémás területet. • A programozók általában rohannak, ahogy közeledik a határidő. Ez az az idő, amikor a legtöbb hiba előfordul. Lehetséges, hogy bármilyen típusú és súlyosságú hibát észlel.
• Az összes hiba nyomon követésének bonyolultsága önmagában ismét hibákat okozhat. Ez bejön

nehezebb, ha egy hibának nagyon összetett életciklusa van, azaz amikor a bezárások, újranyitások, elfogadások, figyelmen kívül hagyások stb. száma folyamatosan növekszik.

Bug életciklus

A hiba életciklusa egy nem szándékos szoftverhibával/viselkedéssel kezdődik, és akkor ér véget, amikor a hozzárendelt fejlesztő kijavítja a hibát. Ha hibát talál, közölni kell, és hozzá kell rendelni egy fejlesztőhöz, aki ki tudja javítani. A javítás után a problémás területet újra meg kell vizsgálni. Ezenkívül meg kell erősíteni, hogy a javítás máshol nem okozott-e problémákat. A legtöbb esetben az életciklus nagyon bonyolulttá és nehezen követhetővé válik, ezért elengedhetetlen egy hiba-/hibakövető rendszer.

Lásd a 7. fejezetet – Hibakövetés

A hiba életciklusának különböző fázisai a következők:

Nyitott: A hiba Nyitott állapotban van, amikor a tesztelő problémás területet azonosít

Elfogadva: A hiba ezután hozzá van rendelve egy fejlesztőhöz javítás céljából. A fejlesztő ezután elfogadja, ha érvényes.

Nem fogadja el/Nem javítja: Ha a fejlesztő alacsony szintűnek tartja a hibát, vagy nem fogadja el hibaként, így nem fogadja el /nem javítja állapotba.

Az ilyen hibákat a projektmenedzserhez rendeljük, aki eldönti, hogy a hibát javítani kell-e. Ha kell, akkor visszarendeli a fejlesztőhöz, ha pedig nem, akkor visszaadja a tesztelőnek, akinek be kell zárnia a hibát.

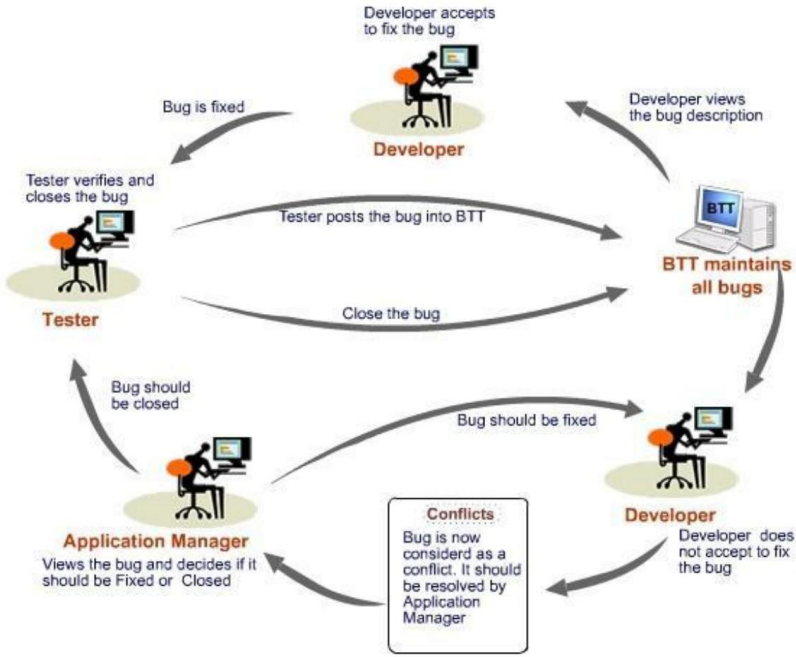
Függőben: Előfordulhat, hogy a fejlesztő által elfogadott hiba nem javítható azonnal. Ilyen esetekben a Függőben állapotba helyezhető .

Javítva: A programozó kijavítja a hibát, és javítottként oldja meg .

Bezárás: A javított hiba a tesztelőhöz lesz hozzárendelve, aki azt a Close állapotba helyezi.

Újranyitás: A javított hibákat a tesztelők újra megnyithatják, ha a javítás máshol problémákat okozna.

Az alábbi képen jól látható a hiba életciklusa és a hiba követésének módja
Hibakövető eszközök (BTT)



A hibák javításának költsége

A költségek logaritmikusak; méretük tízszeresére nő az idő növekedésével. A korai szakaszban talált és kijavított hiba – a követelmények vagy a termékspecifikációs szakasz javítható
egy rövid interakció az érintettel, és szinte semmibe sem kerülhet.

A kódolás során egy gyorsan észlelt hiba kijavítása csak nagyon kevesebb erőfeszítést igényel. Az integrációs tesztelés során a hibajelentés és a hivatalosan dokumentált javítás papírmunkája, valamint az újratesztelés késedelme és költsége.

A rendszertesztelés során ez még több időt vesz igénybe, és késleltetheti a szállítást. Végezetül, a műveletek során bármit okozhat a kellemetlenségtől a rendszerhibáig, esetleg katasztrofális következményekkel járhat egy biztonsági szempontból kritikus rendszerben, például egy repülőgépen vagy egy segélyszolgálatnál.

Mikor lehet leállítani/csökkenteni a tesztelést?

Nehéz meghatározni, hogy pontosan mikor kell leállítani a tesztelést. Íme néhány gyakori tényező, amelyek segítenek eldönteni, hogy mikor állíthatja le vagy csökkentheti a tesztelést:

• Határidők (megjelenési határidők, tesztelési határidők stb.) • Bizonyos százalékos teljesítéssel befejezett tesztesetek • A teszt költségvetése kimerült • A kód/funkcionalitás/ követelmények lefedettsége elér egy meghatározott pontot • A hibák aránya egy bizonyos szint alá esik
• A béta vagy alfa tesztelési időszak véget ér

3. Szoftvertesztelési szintek, -típusok, -kifejezések és -definíciók

Tesztelési szintek és típusok

A tesztelésnek alapvetően három szintje van: egységteszt, integrációs tesztelés és rendszerteszt. Különféle típusú tesztelések tartoznak e szintek alá.

Egység tesztelése

Egyetlen program vagy egy program egy részének ellenőrzése

Integrációs tesztelés

A rendszerelemek közötti interakció ellenőrzése

Előfeltétel: egységtesztelés a rendszert alkotó összes komponensen.

Rendszertesztelés

A teljes rendszer viselkedésének ellenőrzése és érvényesítése az eredeti rendszercélokhoz képest.

A szoftvertesztelés olyan folyamat, amely azonosítja a szoftver helyességét, teljességét és minőségét.

Az alábbiakban felsoroljuk a különféle szoftvertesztelési típusokat és azok definícióit véletlenszerű sorrendben:

• Formális tesztelés: Tesztmérnökök végzik
• Informális tesztelés: A fejlesztők végzik
• Kézi tesztelés: A szoftvertesztelés azon része, amely emberi bevitelt, elemzést vagy értékelést igényel.

• Automatizált tesztelés: Szoftvertesztelés, amely különféle eszközöket használ a tesztelési folyamat automatizálására. Az automatizált teszteléshez továbbra is szakképzett minőségbiztosítási szakemberre van szükség, aki ismeri az automatizálási eszközöket és a tesztelt szoftvert a tesztesetek beállításához. • Fekete doboz
tesztelése: Szoftver tesztelése a rendszer hátterének, a tesztelt modul struktúrájának vagy nyelvének ismerete nélkül. A fekete doboz tesztesetek egy végleges forrásdokumentumból, például specifikációból vagy követelménydokumentumból készülnek.

• Fehér doboz Tesztelés: Olyan tesztelés, amelyben a szoftvertesztelő ismeri a szoftver hátterét, szerkezetét és nyelvét, vagy legalább a célját. • Egységteszt: Az egységteszt egy adott megfelelő program tesztelésének folyamata,

azaz egy ablak, egy jelentés, egy interfész stb. önállóan, mint önálló komponens/
program. Az egységtesztek típusa és mértéke a módosított és az újonnan létrehozott
programokonként változhat. Az egységtesztelést többnyire a programozók végzik, akik a szükséges
egységteszt adatok létrehozásáért is felelősek. • Inkrementális tesztelés: A növekményes
tesztelés egy nem teljes termék részleges tesztelése. A növekményes tesztelés célja, hogy
korai visszajelzést adjon a szoftverfejlesztőknek. • Rendszertesztelés: A rendszerteszt a fekete doboz
tesztelésének
egy formája. A rendszertesztelés célja, hogy ellenőrizze az alkalmazás pontosságát és teljességét
a tervezett funkciók végrehajtása során.

• Integrációs tesztelés: Két vagy több modul vagy funkció tesztelése a modulok/funkciók közötti
interfészhibák feltárása céljából. • Rendszerintegrációs tesztelés: Több platformon
(pl. kliens, webszerver, alkalmazásszerver és adatbázisszerver) elosztott szoftverösszetevők
tesztelése a rendszerintegrációs hibák által okozott hibák (azaz a disztribúcióval és vissza-visszatéréssel
kapcsolatos hibák) érdekében.

• Funkcionális tesztelés: Annak ellenőrzése, hogy egy modul a specifikációban leírtak szerint működik-
e, és meggyőződjünk arról, hogy a program azt csinálja, amit tennie kell. • Végpontok
közötti tesztelés: Hasonló a rendszerteszthez – egy teljes alkalmazás tesztelése olyan helyzetben,
amely a valós használatot utánozza, mint például egy adatbázissal való interakció, hálózati
kommunikáció használata vagy más hardverrel, alkalmazással vagy rendszerrel való interakció. •
Józanságteszt: A józanságtesztet akkor hajtják végre, ha a felületes tesztelés elegendő annak
bizonyítására, hogy az alkalmazás a specifikációknak megfelelően működik. Ez a tesztelési szint a
regressziós tesztelés egy részhalmaza. Általában magában foglalja az alapvető grafikus felhasználói
felület tesztelését az adatbázishoz, alkalmazáskiszolgálókhoz, nyomtatókhoz stb. való
kapcsolódás bizonyítására. • Regressziós tesztelés: Tesztelés annak megállapítására, hogy a
hibajavítások sikeresek voltak-e, és nem okoztak-e új
problémákat. • Átvételi tesztelés: A rendszer tesztelése azzal a céllal, hogy megerősítse a termék
készenlétét és az ügyfél elfogadását. Más néven Felhasználói Elfogadási Teszt. • Adhoc tesztelés:
Tesztelés hivatalos teszterv nélkül vagy teszterven kívül. Egyes projekteknél az ilyen típusú
tesztelést a formális tesztelés kiegészítéseként végzik.
Néha, ha a tesztelésre a fejlesztési ciklus nagyon késői szakaszában kerül sor, ez lesz az egyetlen
olyan tesztelés, amelyet el lehet végezni – általában szakképzett tesztelők végzik. Az ad hoc tesztelést
néha feltáró tesztelésnek nevezik. • Konfiguráció
tesztelése: Teszt annak meghatározására, hogy a termék mennyire működik a hardver-/perifériális
berendezések széles skálájával, valamint a különböző operációs rendszereken és szoftvereken. •
Terhelésvizsgálat: Tesztelés azzal a
céllal, hogy megállapítsa, milyen jól kezelhető a termék
verseny a rendszer erőforrásaiért. A verseny a hálózati forgalom, a CPU kihasználtság vagy a
memóriakiosztás formájában jelentkezhet. • Stressz-
tesztelés: Tesztelés a rendszer lenyomásakor tapasztalható viselkedés értékelésére

túl a törésponton. A cél a gyenge láncszemek feltárása és annak megállapítása, hogy sikerül-e kecsesen helyreállnia a rendszernek. • Teljesítményteszt: Tesztelés annak meghatározására, hogy egy termék mennyire hatékonyan kezeli a különféle eseményeket. Az ilyen típusú teszteléshez leggyakrabban olyan automatizált teszteszközöket használnak, amelyek kifejezetten a teljesítmény tesztelésére és finomhangolására szolgálnak. • Használhatósági tesztelés: A használhatósági tesztelés a „felhasználóbarátság” tesztelése. Módszer annak értékelésére és mérésére, hogy a felhasználók hogyan lépnek kapcsolatba egy szoftvertermékkel vagy -hellyel. Feladatokat adnak a felhasználóknak és megfigyeléseket végeznek. • Telepítési tesztelés: Tesztelés annak meghatározására, hogy a termék kompatibilis-e számos platformmal, és milyen könnyen telepíthető. • Helyreállítási/hibatesztelés: Annak tesztelése, hogy a rendszer mennyire áll helyre összeomlások, hardverhibák vagy egyéb katasztrofális problémák után. • Biztonsági tesztelés: Adatbázisok és hálózati szoftverek tesztelése a vállalati adatok és erőforrások biztonságban tartása érdekében a tévedésből/véletlen felhasználóktól, hackerektől és más rosszindulatú támadóktól.

• Behatolási tesztelés: A behatolási tesztelés azt vizsgálja, hogy a rendszer mennyire védett a jogosulatlan belső vagy külső hozzáféréssel, illetve szándékos károkozással szemben. Az ilyen típusú tesztelés általában kifinomult tesztelési technikákat igényel. • Kompatibilitási tesztelés: Tesztelés annak meghatározására, hogy van-e más rendszerszoftver az olyan összetevők, mint a böngészők, segédprogramok és konkurens szoftverek ütköznek a tesztelt szoftverrel. • Feltáró tesztelés: Minden olyan tesztelés, amelyben a tesztelő dinamikusan változtatja a tesztvégrehajtási tevékenységét a tesztek végrehajtása során tanult információk alapján.

• Összehasonlító tesztelés: Olyan tesztelés, amely összehasonlítja a szoftver gyengeségeit és erősségeit a versenytársak termékeivel. • Alfa tesztelés: A kód utáni tesztelés többnyire teljes, vagy tartalmazza a legtöbb funkciót és az ügyfelek elérése előtt. Néha a felhasználók egy kiválasztott csoportja vesz részt. Ezt a tesztelést gyakrabban házon belül vagy egy külső tesztelő cég végzi, szorosan együttműködve a szoftvermérnöki részleggel. • Bétatesztelés: Tesztelés, miután a termék kódja befejeződött. A bétákat gyakran széles körben terjesztik, vagy akár a nagyközönség számára is terjesztik. • Gamma tesztelés: A gamma tesztelés olyan szoftver tesztelése, amely rendelkezik minden szükséges funkcióval, de nem ment át minden házon belüli minőségellenőrzésen. • Mutációs tesztelés: A tesztelés alaposságának meghatározására szolgáló módszer annak mérésével, hogy a tesztesetek milyen mértékben képesek megkülönböztetni a programot a program csekély változataitól. • Független ellenőrzés és érvényesítés (IV&V): A szoftver használatának folyamata azzal a céllal, hogy a szoftverrendszer megfeleljen a követelményeknek és a felhasználói elvárásoknak, és ne hibásodjon meg elfogadhatatlan módon. Az ezt a munkát végző egyén vagy csoport nem tagja a szoftvert kifejlesztő csoportnak vagy szervezetnek.

• Pilot Testing: Olyan tesztelés, amely közvetlenül a tényleges kiadás előtt vonja be a felhasználókat annak biztosítására, hogy a felhasználók megismerjék a kiadás tartalmát, és végül elfogadják azt. Jellemzően sok felhasználót érint, rövid időn keresztül zajlik, és szigorúan ellenőrzik. (Lásd: béta tesztelés) • Párhuzamos/Audit tesztelés: Tesztelés, ahol a felhasználó egyezteti az új rendszer kimenetét a jelenlegi rendszer kimenetével annak ellenőrzésére, hogy az új rendszer megfelelően hajtja végre a műveleteket. • Üvegdoboz/nyitott doboz tesztelése: Az üvegdoboz tesztelése megegyezik a fehérdobozos teszteléssel. Ez egy tesztelési megközelítés, amely megvizsgálja az alkalmazás programszerkezetét, és teszteseteket származtat az alkalmazás programlogikájából. • Zárt dobozos tesztelés: A zárt dobozos tesztelés megegyezik a fekete dobozos teszteléssel. A tesztelés olyan típusa, amely csak az alkalmazás funkcionalitását veszi figyelembe. • Alulról felfelé irányuló tesztelés: Az alulról felfelé irányuló tesztelés az integrációs tesztelés egyik technikája. A tesztelő mérnök teszt-illesztőprogramokat készít és használ a még nem fejlesztett komponensekhez, mivel az alulról felfelé irányuló tesztelés során először az alacsony szintű összetevőket tesztelik. Az alulról felfelé irányuló tesztelés célja az alacsony szintű komponensek első behívása tesztelés céljából célokra.

• Füstvizsgálat: Szúrópróbaszerű vizsgálat, amelyet a szállítás előtt és a befejezés után végeznek tesztelés.

Tesztelési feltételek

• Bug: A szoftverhiba olyan kódolási hibaként definiálható, amely váratlan hibát, hibát vagy hibát okoz. Más szóval, ha egy program nem a rendeltetésszerűen működik, az valószínűleg hiba. • Hiba: A program és a specifikációja közötti eltérés a program hibája program.

• Hiba: A hiba a kívánt termékattribútumtól való eltérés (lehet hibás, hiányzó vagy extra adat). Kétféle lehet – a termék hibája vagy a vásárló/felhasználó elvárásaitól való eltérés. Ez a szoftverrendszer hibája, és nincs hatása addig, amíg nem érinti a felhasználót/ügyfelet és az operációs rendszert. A hibák 90%-át folyamatproblémák okozhatják.

• Hiba: Olyan hiba, amely működési hibát okoz, vagy negatív hatással van a felhasználóra/vevő.

• Minőségbiztosítás: A hibák megelőzésére irányul. A minőségbiztosítás biztosítja, hogy a projektben érintett valamennyi fél betartsa a folyamatot és az eljárásokat, a szabványokat és a sablonokat, valamint a tesztek készenléti felülvizsgálatát. • Minőség-ellenőrzés: a minőség-ellenőrzés vagy a minőségfejlesztés olyan intézkedések összessége, amelyeket annak biztosítására tesznek, hogy hibás termékeket vagy szolgáltatásokat ne gyártsanak, és hogy a tervezés megfeleljen a teljesítménykövetelményeknek.

• Ellenőrzés: Az ellenőrzés biztosítja, hogy a terméket úgy tervezték, hogy az összes funkciót az ügyfél rendelkezésére bocsátja; jellemzően felülvizsgálatokat és értékezleteket foglal magában a dokumentumok, tervek, kódok, követelmények és specifikációk értékelése céljából; ez megtehető ellenőrző listákkal, problémalistákkal, áttekintésekkel és ellenőrző értekezletekkel. • Érvényesítés: Az érvényesítés biztosítja, hogy a követelményekben meghatározott funkcionalitás a termék tervezett viselkedése legyen; az érvényesítés általában tényleges tesztelést foglal magában, és az ellenőrzések befejezése után történik.

A leggyakoribb szoftverhibák

Az alábbiakban felsoroljuk a leggyakoribb szoftverhibákat, amelyek segítséget nyújtanak a szoftvertesztelésben. Ez segít a hibák szisztematikus azonosításában, és növeli a szoftvertesztelés hatékonyságát és termelékenységét.

Hibatípusok példákkal

• Felhasználói felület hibái: Hiányzó/Rossz funkciók Nem azt csinálja, amit a felhasználó elvár, Hiányzó információ, Félrevezető, Zavaros információk, Rossz tartalom a Súgószövegben, Nem megfelelő hibaüzenetek. Teljesítményproblémák - Gyenge reagálóképesség, Nem lehet átirányítani a kimenetet, A billentyűzet nem megfelelő használata • Hibakezelés: Nem megfelelő - védelem a sérült adatok ellen, felhasználói beviteli tesztek, verziókezelés; Figyelmen kívül hagyja – túlcsordulás, adatösszehasonlítás, hibahelyreállítás – hibák megszakítása, hardverproblémákból való helyreállítás. • Határral kapcsolatos hibák: Határok a hurokban, térben, időben, memóriában, határon kívüli esetek helytelen kezelése. • Számítási hibák: Rossz logika, Rossz aritmetika, Elavult állandók, Számítási hibák, hibás átalakítás egyik adatábrázolásból a másikba, Rossz képlet, hibás közelítés. • Kezdeti és későbbi állapotok: Sikertelen - az adatelem nullára állítása, ciklusvezérlő változó inicializálása vagy mutató újrainicializálása, karakterlánc vagy zászló törlése, Hibás inicializálás. • Vezérlési folyamathibák: hibás visszatérési állapot feltételezett, kivételkezelésen alapuló kilépések, verem alulcsordulás/túlcsordulás, megszakítások blokkolásának vagy feloldásának elmulasztása, Az összehasonlítás néha rossz eredményt ad, Hiányzó/rossz alapértelmezés és Adattípus hibák. • Hibák az adatok kezelésében vagy értelmezésében: le nem zárt null karakterláncok, amelyek felülírják a fájlt hiba kilépés vagy felhasználói megszakítás után.

• Versenyfeltételek: Feltételezés, hogy egy esemény vagy feladat befejeződött, mielőtt a másik elkezdődne, Erőforrás versenyek, A feladatok az előfeltételek teljesülése előtt indulnak, Üzenetek kereszt

vagy nem az elküldött sorrendben érkezik meg.

• Betöltési feltételek: Nem állnak rendelkezésre a szükséges erőforrások, nincs nagy memóriaterület, az alacsony prioritású feladatokat nem halasztják el, nem törli a régi fájlokat a háttértárról, és nem ad vissza a fel nem használt memóriát. • Hardver: rossz eszköz, az eszköz nem elérhető,
az eszköz intelligenciáját nem használják fel,
Félreértett állapot- vagy visszatérési kód, hibás művelet- vagy utasításkód. • Forrás, verzió és azonosító vezérlése:
Nincs cím vagy verzióazonosító, az adatok vagy programfájlok több példányának frissítésének elmulasztása. • Tesztelési hibák: a probléma észlelésének/bejelentésének
elmulasztása, a legígéretesebb teszteset használatának elmulasztása, sérült adatfájlok, félreértelmezett specifikációk vagy dokumentáció, a probléma reprodukálásának elmulasztása, a megoldatlan problémák ellenőrzésének elmulasztása közvetlenül a kiadás előtt , A javítások ellenőrzésének elmulasztása, Összefoglaló jelentés benyújtásának elmulasztása.

5. A teszttervezési folyamat

Mi az a tesztstratégia? Mik a komponensei?

Tesztpolitika – A szervezet szoftvertesztelési filozófiáját jellemző dokumentum.

Tesztstratégia – Magas szintű dokumentum, amely meghatározza a program végrehajtandó tesztfázisait és az ezeken belüli tesztelést. Meghatározza az egyes projektekben követendő folyamatot. Ez meghatározza az egyes projekteknél követendő folyamatok, dokumentumok, tevékenységek stb. szabványait.

Például, ha egy terméket tesztelésre adnak, el kell döntenie, hogy jobb-e a feketedobozos vagy a fehérdobozos tesztelés, és ha úgy dönt, hogy mindkettőt használja, mikor alkalmazza mindegyiket és a szoftver melyik részére? Mindezeket a részleteket meg kell határozni a tesztstratégiában.

Projekt Teszterv – egy dokumentum, amely meghatározza az elvégzendő tesztfázisokat és az ezeken belüli tesztelést egy adott projekt esetében.

A tesztstratégiának egynél több projektet kell lefednie, és a következő problémákkal kell foglalkoznia: A magas kockázatú területek tesztelésének megközelítése először, a tesztelés megtervezése, a folyamat javítása a korábbi tesztelések alapján, a felhasznált környezetek/adatok, Tesztkezelés - Konfigurációkezelés, Problémakezelés, Milyen mérőszámokat követnek, Automatizálódnak-e a tesztek, és ha igen, milyen eszközöket fognak használni, Mik a tesztelési szakaszok és tesztelési módszerek, Tesztelés utáni felülvizsgálati folyamat, Sablonok.

A teszttervezést azonnal el kell kezdeni, amint a projekt követelményei ismertek. Az első elkészítendő dokumentum a Tesztstratégia/Tesztelési Megközelítés, amely meghatározza a tesztelés magas szintű megközelítését, és lefedi a fent említett összes többi elemet.

Teszttervezés – Mintastruktúra

A megközelítés megértése után részletes teszterv írható. Általában ez a teszterv különböző stílusokban írható. A teszttervek projektenként teljesen eltérőek lehetnek ugyanabban a szervezetben.

IEEE SZOFTVERTESZT DOKUMENTÁCIÓ Std 829-1998 – TESZTTERV

Célja
A tesztelési tevékenységek hatókörének, megközelítésének, erőforrásainak és ütemezésének leírása. Nak nek

azonosítsa a tesztelt tételeket, a tesztelendő jellemzőket, az elvégzendő tesztelési feladatokat, az egyes feladatokért felelős személyzetet és a tervhez kapcsolódó kockázatokat.

VÁZLAT

A vizsgálati tervnek a következő szerkezettel kell rendelkeznie:

• Teszterv azonosítója. A teszttervhez hozzárendelt egyedi azonosító. • Bevezetés:
Összefoglalta a tesztelendő szoftverelemeket és jellemzőket, és
beépítésük szükségessége.

• Tesztelemek: Azonosítsa a tesztelemeket, azok átviteli médiáját, amelyek hatással vannak rájuk • Tesztelendő jellemzők • Nem
tesztelendő jellemzők • Megközelítés

• Elfogadási/
nem teljesítési kritériumok •
Felfüggesztési kritériumok és újraindítási követelmények •
Teszteljesítmények

• Tesztelési feladatok
• Környezeti igények •
Felelősségek •
Személyzeti és képzési igények •
Ütemezés

• Kockázatok és előre nem látható
események • Jóváhagyások

Főbb teszttervezési feladatok

A szoftvertesztelés minden más folyamatához hasonlóan a teszttervezés fő feladatai a következők: Tesztstratégia kidolgozása, kritikus sikertényezők, tesztcélok meghatározása, szükséges teszterőforrások meghatározása, tesztkörnyezet megtervezése, tesztelési eljárások meghatározása, tesztelendő funkciók azonosítása, interfészek azonosítása más rendszerekkel vagy komponensekkel, tesztszkriptek írása, tesztesetek meghatározása, tesztadatok tervezése , Tesztmátrix készítése, Vizsgálati ütemezések meghatározása, Információk összeállítása, Terv véglegesítése.

6. Teszteset kidolgozása

A teszteset egy olyan részletes eljárás, amely teljes mértékben tesztel egy szolgáltatást vagy egy szolgáltatás egy aspektusát. Míg a teszterv leírja, hogy mit kell tesztelni, a teszteset leírja, hogyan kell végrehajtani egy adott tesztet. A teszttervben felsorolt minden egyes teszthez teszteseteket kell kidolgoznia.

Általános irányelvek

Tesztelőként a szoftver követelményeknek való megfelelőségének meghatározásának legjobb módja az, ha hatékony teszteseteket tervezünk, amelyek az egység alapos tesztelését biztosítják. A különböző teszteset-tervezési technikák lehetővé teszik a tesztelők számára, hogy hatékony teszteseteket dolgozzanak ki. Emellett a tervezési technikák alkalmazásakor minden tesztelőnek szem előtt kell tartania az általános irányelveket, amelyek segítenek a teszteset tervezésében:

a. Minden teszteset célja a teszt lehető legegyszerűbb lefuttatása.
[Megfelelő technikák - Specifikációból származó tesztek, egyenértékűségi particionálás] b.
Kezdetben a pozitív tesztelésre koncentráljon, azaz a tesztesetnek azt kell mutatnia, hogy a szoftver azt csinálja, amire szánták. [Megfelelő technikák - Specifikációból származó tesztek, ekvivalencia particionálás, állapotátmenet tesztelése] c. A meglévő teszteseteket
tovább kell fejleszteni, és további teszteseteket kell megtervezni annak bizonyítására, hogy a szoftver nem csinál semmit, amit nem írnak elő, pl.
Negatív tesztelés [Alkalmas technikák - Hibakitalálás, Határérték-elemzés, Belső határérték-vizsgálat, Állapotátmeneti tesztelés] d. Adott esetben a teszteseteket úgy kell
megtervezni, hogy olyan kérdéseket kezeljenek, mint a teljesítmény, a biztonsági követelmények és a biztonsági követelmények [Megfelelő technikák -
Specifikációból származó
vizsgálatok] e. Ezután további tesztesetek adhatók hozzá az egységteszt specifikációjához, hogy elérjék a konkrét tesztlefedettségi célkitűzéseket. A lefedettségi tesztek megtervezése után a tesztelési eljárás kidolgozható és a tesztek végrehajthatók [Alkalmas technikák - Elágazás tesztelése, Állapotteszt, Adatdefiníció-használati tesztelés, Állapotátmenet tesztelése]

Teszteset – Mintastruktúra

A teszteset ábrázolásának módja szervezetenként eltérő. Mindenesetre sok teszteset-sablon táblázat formájában van, például egy 5 oszlopos táblázat mezőkkel:

Teszt Ügy ID	Próbaper Leírás Beállítás	Teszt Függőség/	Beviteli adat Követelmények/ Lépések	Várható passz Eredmények	sikertelenség

Tesztesetek tervezési technikái

A tesztesetek tervezési technikái nagyjából két kategóriába sorolhatók: fekete doboz technikák, fehér doboz technikák és egyéb technikák, amelyek egyik kategóriába sem tartoznak.

Fekete doboz (funkcionális) Fehér doboz (szerkezeti)		Egyéb
- Levezetett specifikáció tesztek - Egyenértékű particionálás - Határérték Elemzés - Állapotátmenet tesztelése	- Ágazati tesztelés - Állapotvizsgálat - Adatdefiníció - Használati tesztelés - Belső határérték vizsgálat	- Hiba találgatás

Specifikációból származó tesztek

Ahogy a neve is sugallja, a tesztesetek tervezése a vonatkozó specifikációk alapján történik. Ez egy pozitív teszteset tervezési technika.

Egyenértékű particionálás

Az ekvivalencia-particionálás az a folyamat, amikor az összes lehetséges tesztértéket felvesszük és osztályokba (partíciókba vagy csoportokba) helyezzük. A teszteseteket úgy kell megtervezni, hogy minden osztályból egy-egy értéket teszteljenek. Ezáltal a legkevesebb tesztesetet használ a maximális bemeneti követelmények kielégítésére.

Például, ha egy program csak 1-től 10-ig terjedő egész értékeket fogad el. Egy ilyen program lehetséges tesztesetei az összes egész szám tartománya. Egy ilyen programban minden 0-ig és 10 feletti egész szám hibát okoz. Tehát ésszerű feltételezni, hogy ha a 11 meghibásodik, akkor a felette lévő összes érték meghibásodik, és fordítva.

Ha egy bemeneti feltétel egy értéktartomány, legyen egy érvényes ekvivalenciaosztály a tartomány (ebben a példában 0 vagy 10). Legyen a tartomány alatti és feletti értékek két megfelelő érvénytelen ekvivalenciaérték (azaz -1 és 11). Ezért a fenti három partícióérték tesztesetként használható a fenti példában.

Határérték-elemzés

Ez egy kiválasztási technika, ahol a tesztadatokat úgy választják ki, hogy a bemeneti tartomány vagy a kimeneti tartomány határai mentén helyezkedjenek el. Ezt a technikát gyakran stressztesztnek nevezik, és bizonyos mértékű negatív tesztet tartalmaz a teszttervezésben, előre jelezve, hogy hibák fordulnak elő a partíció határain vagy azok körül.

Például a mező kitöltése kötelező 0 és 10 USD közötti pénzösszegek elfogadásához. Tesztelőként ellenőriznie kell, hogy ez legfeljebb 10 és 9,99 dollárt jelent-e, és hogy a 10 dollár elfogadható-e. Tehát a határértékek: 0, 0,01, 9,99 és 10 dollár.

Most a következő tesztek hajthatók végre. Negatív értéket el kell utasítani, 0-t kell elfogadni (ez a határon van), 0,01 és 9,99 dollárt, nullát és 10 dollárt el kell utasítani. Ily módon ugyanazt a partíciókoncepciót használja, mint az egyenértékű particionálás.

Állapotátmenet tesztelése

Ahogy a neve is sugallja, a tesztesetek az állapotok közötti átmenet tesztelésére szolgálnak az átmenetet okozó események létrehozásával.

Ágazati tesztelés

Az elágazási tesztelés során a tesztesetek úgy vannak kialakítva, hogy az egység áramlási ágait vagy döntési pontjait vezéreljék. Ez általában a döntési lefedettség célszintjének elérését célozza. Ágak lefedettsége, tesztelni kell az IF és az ELSE mindkét ágát. Az ágon belüli összes elágazást és összetett feltételt (pl. hurkok és tömbkezelés) legalább egyszer le kell gyakorolni.

Állapotvizsgálat

A feltételvizsgálat célja tesztesetek tervezése annak bizonyítására, hogy a logikai feltételek egyes összetevői és az egyes összetevők kombinációi helyesek. A tesztesetek a logikai kifejezések egyes elemeinek tesztelésére szolgálnak, mind az elágazás feltételein belül, mind az egység más kifejezésein belül.

Adatdefiníció – Tesztelés használata

Az adatdefiníciós-használati tesztelés teszteseteket tervez az adatdefiníciók és -felhasználások párjának tesztelésére. Az adatdefiníció bárhol található, ahol be van állítva egy adatelem értéke. Az adathasználat bárhol történik, ahol egy adatelemet olvasnak vagy használnak. A cél olyan tesztesetek létrehozása, amelyek meghatározott definíciók és felhasználások közötti útvonalakon keresztül hajtják végre a végrehajtást.

Belső határérték tesztelése

Sok esetben a partíciók és határaik azonosíthatók egy egység funkcionális specifikációjából, amint azt az ekvivalencia-particionálás és a határérték-elemzés részben leírtuk. Egy egységnek azonban lehetnek belső határértékei is, amelyek csak szerkezeti specifikációból azonosíthatók.

Hiba a találgatás során

Ez egy teszteset tervezési technika, ahol a tesztelők tapasztalataik alapján kitalálják az esetlegesen előforduló hibákat, és ennek megfelelően tervezik meg a teszteseteket, hogy feltárják azokat.

A fent leírt teszteset tervezési technikák bármelyikének vagy kombinációjának alkalmazása; hatékony teszteseteket dolgozhat ki.

Mi az a használati eset?

A használati eset leírja a rendszer viselkedését különböző feltételek mellett, amikor az egyik felhasználó kérésére válaszol. A felhasználó interakciót kezdeményez a rendszerrel valamilyen cél elérése érdekében. Különböző viselkedési sorozatok vagy forgatókönyvek bontakozhatnak ki az adott kérésektől és a kérések körülményeitől függően.
A használati eset összegyűjti ezeket a különböző forgatókönyveket.

A használati esetek nagyrészt azért népszerűek, mert koherens történeteket mesélnek el a rendszer használat közbeni viselkedéséről. A rendszer használói láthatják, hogy mi lesz ez az új rendszer, és időben reagálhatnak.

7. Hibakövetés

Mi a hiba?

Amint azt korábban tárgyaltuk, a hiba a kívánt termékattribútumtól való eltérés (lehet hibás, hiányzó vagy extra adat). Kétféle lehet – a termék hibája vagy a vásárló/felhasználó elvárásaitól való eltérés. Ez a szoftverrendszer hibája, és nincs hatása addig, amíg nem érinti a felhasználót/ügyfelet és az operációs rendszert.

Melyek a hibakategóriák?

A tesztelés eddig megszerzett tudásával most már kategorizálhatja a talált hibákat. A hibák különböző típusokba sorolhatók az általuk kezelt alapvető problémák alapján. Egyes hibák biztonsági vagy adatbázis-problémákat érintenek, míg mások működési vagy felhasználói felületi problémákra utalhatnak.

Biztonsági hibák: Az alkalmazás biztonsági hibái általában a felhasználótól az alkalmazásnak küldött adatok nem megfelelő kezelését jelentik. Ezek a hibák a legsúlyosabbak, és a javítás szempontjából a legmagasabb prioritást élvezik.

Példák:

- Hitelesítés: Érvénytelen felhasználónév/jelszó elfogadása
- Engedélyezés: Az oldalak elérése engedély nélkül

Adatminőség/adatbázis hibák: Az adatbázisban lévő adatok nem megfelelő kezelésével foglalkozik.

Példák:

- Az értékeket nem törölték/illesztették be megfelelően az adatbázisba
- Helytelen/rossz/null értékek beszúrva a tényleges értékek helyére

Kritikus működési hibák: Ezeknek a hibáknak az előfordulása akadályozza az alkalmazás alapvető működését.

Példák:

- Kivételek

Működési hibák: Ezek a hibák befolyásolják az alkalmazás működését.

Példák:

- Minden Javascript hiba
- Az olyan gombok, mint a Mentés, Törlés, Mégse, nem látják el a kívánt funkciót
- Hiányzó funkció (vagy) egy funkció, amely nem úgy működik, ahogyan azt tervezték
- A hurkok folyamatos végrehajtása

Felhasználói felület hibái: Ahogy a neve is sugallja, a felhasználói felülettel kapcsolatos problémákkal foglalkozó hibák általában kevésbé súlyosak.

Példák:

- Nem megfelelő hiba/figyelmeztetés/UI üzenetek
- Helyesírási hibák
- Igazítási problémák

Hogyan történik a hiba bejelentése?

Miután a teszteseteket a megfelelő technikák segítségével kifejlesztették, akkor végrehajtásra kerülnek, amikor a hibák előfordulnak. Nagyon fontos, hogy ezeket a hibákat a lehető leghamarabb jelentsék, mert minél korábban jelenti be a hibát, annál több idő marad az ütemezésben a javításra.

Egyszerű példa az, hogy néhány hónappal a termék megjelenése előtt a súgófájlban dokumentált hibás funkciót jelent, nagyon nagy az esélye annak, hogy a hiba javításra kerül.
Ha ugyanazt a hibát néhány órával a megjelenés előtt jelenti, akkor valószínű, hogy nem lesz javítva. A hiba továbbra is ugyanaz, bár néhány hónappal vagy néhány órával a megjelenés előtt jelented, de ami számít, az az idő.

Nem elég csak megtalálni a hibákat; ezeket is világosan és hatékonyan jelenteni/kommunikálni kell, nem beszélve arról, hogy hányan fogják elolvasni a hibát.

A hibakövető eszközök (más néven hibakövető eszközök, hibakövető eszközök vagy problémakövetők) nagyban segítik a tesztelőket a szoftveralkalmazásokban talált hibák jelentésében és nyomon követésében. Eszközöket biztosítanak a projektinformációk egy kulcsfontosságú elemének egy helyen történő konszolidálására. A projektmenedzserek ezután láthatják, hogy mely hibákat javították ki, melyek kiemelkedőek, és mennyi ideig tart a hibák kijavítása. A felső vezetés jelentések segítségével megértheti a fejlesztési folyamat állapotát.

Mennyire legyen leíró jellegű a hiba-/hibajelentés?

A hiba bejelentése során elegendő részletet kell megadnia, szem előtt tartva azokat az embereket, akik használni fogják – tesztvezetőt, fejlesztőt, projektmenedzsert, más tesztelőket, új tesztelőket, stb. Ez azt jelenti, hogy a jelentésnek tömörnek, egyenesnek és világosnak kell lennie. . A jelentésnek tartalmaznia kell a következőket:

- Hiba címe
- Hibaazonosító (szám, azonosító stb.)

- Az alkalmazás neve vagy azonosítója és verziója - A
funkció, modul, szolgáltatás, objektum, képernyő stb., ahol a hiba előfordult - Környezet (OS,
böngésző és verziója)
- Hibatípus vagy kategória/súlyosság/prioritás o
  Hibakategória: Biztonság, adatbázis, funkcionalitás (kritikus/általános), felhasználói felület
  o Hiba súlyossága: A hiba súlyossága az alkalmazást – nagyon magas, magas, közepes, alacsony, nagyon alacsony
  o Hibaprioritás: A hiba javításának javasolt prioritása – P0, P1, P2, P3, P4, P5 (P0 - Legmagasabb, P5 - Legalacsonyabb)
- Hibaállapot (Nyitott, Függőben, Javítva, Lezárva, Újramegnyitás)
- Teszteset neve/száma/azonosítója
- Hibaleírás
- A reprodukálás lépései
- Tényleges eredmény
- Tesztelői megjegyzések

Mit csinál a tesztelő, ha a hibát kijavítják?

Miután a jelentett hibát kijavították, a tesztelőnek újra kell tesztelnie a javítás megerősítéséhez. Ez általában a lehetséges forgatókönyvek végrehajtásával történik, ahol a hiba előfordulhat. Az újratesztelés befejezése után a javítás megerősíthető, és a hiba bezárható. Ez a hiba életciklusának végét jelzi.

8. A tesztjelentések típusai

Az IEEE szoftverteszt-dokumentációs szabványban körvonalazott dokumentumok kiterjednek
teszttervezés, tesztspecifikáció és tesztjelentés.

A tesztjelentés négy dokumentumtípust fed le:

1. A tesztelem átviteli jelentés azonosítja azokat a tesztelemeket, amelyeket tesztelésre továbbítanak
a fejlesztéstől a tesztelési csoporthoz abban az esetben, ha a tesztvégrehajtás formális kezdete
szükséges.

A jelentésbe foglalandó részletek – Cél, Vázlat, Átviteli jelentés azonosítója, Továbbított tételek, Hely, Állapot
és Jóváhagyások.

2. A tesztcsoport tesztnaplót használ a teszt végrehajtása során történt események rögzítésére

A jelentésbe foglalandó részletek – Cél, Vázlat, Tesztnapló azonosító,
Leírás, tevékenység és esemény bejegyzések, végrehajtás leírása, eljárás eredményei,
Környezeti információk, rendellenes események, incidens-jelentési azonosítók

3. A tesztesemény jelentés leír minden olyan eseményt, amely a teszt végrehajtása során következik be,
és amely további vizsgálatot igényel

A jelentésben feltüntetendő részletek – Cél, Vázlat, Teszt-esemény-jelentés azonosítója,
Összegzés, hatás

4. A vizsgálati összefoglaló jelentés összefoglalja az egy vagy több teszttervezési specifikációhoz
kapcsolódó tesztelési tevékenységeket

A jelentésbe foglalandó részletek - Cél, Vázlat, Teszt-összefoglaló-Jelentésazonosító, Összegzés,
eltérések, Átfogóság értékelése, Eredmények összefoglalása, Tevékenységek összefoglalása és Jóváhagyások.

9. Szoftverteszt-automatizálás

A tesztelés automatizálása nem különbözik attól a programozótól, amely kódolási nyelvet használ a programíráshoz bármely kézi folyamat automatizálására. A nagy rendszerek tesztelésének egyik problémája az, hogy túlmutat a kis tesztcsoportokon. Mivel csak kevés tesztelő áll rendelkezésre, a tesztelés lefedettsége és mélysége nem megfelelő az adott feladathoz.

A tesztcsapat egy bizonyos létszámon túli bővítése is problémássá válik a fej feletti munka növekedésével. Ezt a minőségromlás nélkül el lehet kerülni olyan eszközök megfelelő használatával, amelyek hatalmas mértékben növelhetik az egyén kapacitását, miközben a kritikus elemekre összpontosítanak (mélységei).

Vegye figyelembe a következő tényezőket, amelyek segítenek meghatározni az automatizált tesztelőeszközök használatát:

- Vizsgálja meg jelenlegi tesztelési folyamatát, és határozza meg, hol kell azt módosítani az automatizált teszteszközök használatához. • Készüljön fel arra, hogy módosítsa a tesztelés jelenlegi módjait. • Olyan emberek bevonása, akik használni fogják az eszközt az automatizált tesztelés megtervezéséhez folyamat.
- Hozzon létre egy értékelési kritériumkészletet a függvényekhez, amelyeket figyelembe szeretne venni az automatizált teszteszköz használatakor. Ezek a kritériumok a következőket foglalhatják magukban:
    - o Megismételhetőség vizsgálata
    - o Az alkalmazások kritikussága/kockázata
    - o Működési egyszerűség
    - o Könnyű automatizálás
    - o A funkció dokumentáltsági szintje (követelmények stb.)
- Vizsgálja meg meglévő teszteseteit és tesztszkriptjeit, hogy megtudja, melyek a legalkalmasabbak a tesztautomatizáláshoz. • Az emberek képzése az alapvető teszttervezési készségekre.

Az automatizálás megközelítései

A tesztautomatizálásban három tág lehetőség áll rendelkezésre:

Teljes kézikönyv	Részleges automatizálás	Teljes automatizálás
A kézi tesztelésre hagyatkozás	Redundancia lehetséges, de automatizált megkettőzést igényel	tesztelés
Érzékeny és rugalmas Rugalmas		Viszonylag rugalmatlan
Következetlen	Következetes	Nagyon következetes
Alacsony megvalósítási költség -		Magas megvalósítási költség
Magas ismétlődő költség	Automatizálja az ismétlődő feladatokat és a magas megtérülésű feladatokat	Méretgazdaságosság ismétlés, regresszió stb
Automatizáláshoz szükséges -		-
Alacsony képzettségi követelmények -		Magas képzettségi követelmények

A teljesen manuális tesztelés azzal az előnnyel jár, hogy viszonylag olcsó és hatékony. De ahogy a termék minősége javul, a további hibák felkutatásának többletköltsége egyre drágább lesz. A nagy léptékű kézi tesztelés azt is jelenti, hogy nagy léptékű tesztelő csapatok működnek együtt az ehhez kapcsolódó űrköltségek és infrastruktúra. A kézi tesztelés is sokkal érzékenyebb és rugalmasabb, mint az automatizált tesztelés, de hajlamos a tesztelő hibáira a fáradtság miatt.

A teljesen automatizált tesztelés nagyon következetes, és lehetővé teszi a hasonló tesztek megismétlését nagyon alacsony határköltséggel. Az ilyen automatizálás telepítési és beszerzési költségei azonban nagyon magasak, és a karbantartás is ugyanilyen költséges lehet. Az automatizálás is viszonylag rugalmatlan, és átdolgozást igényel a változó követelményekhez való alkalmazkodás érdekében.

A részleges automatizálás csak ott foglalja magában az automatizálást, ahol a legtöbb előny érhető el. Előnye, hogy kifejezetten az automatizálási feladatokat célozza meg, és így a legtöbb hasznot éri el belőlük. Megtartja továbbá a kézi tesztelés nagy részét, amely fenntartja a tesztcsapat rugalmasságát, és redundanciát kínál az automatizálás kézi teszteléssel történő kiegészítésével. Hátránya, hogy nyilvánvalóan nem nyújt olyan kiterjedt előnyöket, mint egyik szélsőséges megoldás sem.

A megfelelő eszköz kiválasztása

• Szánjon időt az eszközigények meghatározására technológia, folyamat, alkalmazások, emberek készségek és szervezés.
• Az eszközök értékelése során rangsorolja, hogy mely teszttípusok a legkritikusabbak a siker szempontjából, és ezek alapján ítélje meg a jelölt eszközöket. •
Ismerje meg az eszközöket és azok kompromisszumait. Előfordulhat, hogy több eszközből álló megoldást kell használnia a magasabb szintű teszttípus lefedettség eléréséhez. Például kombinálnia kell a rögzítési/lejátszási eszközt egy terhelés-teszt eszközzel, hogy lefedje a teljesítménytesztet esetek.
• A potenciális felhasználók bevonása az eszközigények és az értékelési kritériumok meghatározásába. •
Készítsen értékelési pontozókártyát az egyes eszközök teljesítményének összehasonlításához a közös kritériumrendszer. Rangsorolja a kritériumokat a szervezet szempontjából való relatív fontosság szerint.

A szoftverteszt-automatizálás tíz legnagyobb kihívása

1. Rossz eszköz vásárlása 2.
Nem megfelelő tesztcsapat szervezés 3. Vezetői
támogatás hiánya 4. A teszttípusok
hiányos lefedettsége a kiválasztott eszköz által 5. Nem megfelelő
eszközképzés 6. Az eszköz
használatának nehézségei 7. Az
alapvető tesztfolyamat vagy megértés hiánya Mit kell tesztelni 8. A
konfigurációkezelési folyamatok hiánya 9. Az eszközök
kompatibilitása és együttműködési képessége

10. Bevezetés a szoftverszabványokba

Capability Maturity Model – A szoftverközösség fejlesztette ki 1986-ban a SEI vezetésével. A CMM leírja a szoftverfolyamatok érettségének alapelveit és gyakorlatait. Célja, hogy segítse a szoftverszervezeteket szoftverfolyamataik érettségének javításában, az ad hoc, kaotikus folyamatoktól a kiforrott, fegyelmezett szoftverfolyamatok felé vezető úton. A hangsúly a kulcsfontosságú folyamatterületek és a példaértékű gyakorlatok azonosításán van, amelyek fegyelmezett szoftverfolyamatot foglalhatnak magukban.

Miből áll a CMM? A CMM öt érettségi szintre van felosztva:

- Kezdeti •
- Megismételhető •
- Meghatározott •
- Kezelhető • Optimalizálás

Az 1. szintet kivéve minden érettségi szint több kulcsfontosságú folyamatterületre bomlik, amelyek jelzik azokat a területeket, amelyekre a szervezetnek összpontosítania kell szoftverfolyamata javítása érdekében.

1. szint – Kezdeti szint: Fegyelmezett folyamat, Szabványos, Következetes folyamat, Kiszámítható folyamat, Folyamatosan javító folyamat

2. szint – Megismételhető: kulcsfontosságú gyakorlati területek – Követelménykezelés, Szoftverprojekt tervezés, Szoftverprojekt nyomon követése és felügyelete, Szoftver alvállalkozói menedzsment, Szoftver minőségbiztosítás, Szoftverkonfiguráció kezelése

3. szint – Meghatározott: kulcsfontosságú gyakorlati területek – Szervezeti folyamat fókusz, Szervezeti folyamat meghatározása, Képzési program, integrált szoftvermenedzsment, Szoftvertermék tervezés, csoportközi koordináció, Peer Review

4. szint – Kezelhető: Kulcsfontosságú gyakorlati területek – Kvantitatív folyamatmenedzsment, Szoftverminőség-menedzsment

5. szint – Optimalizálás: Kulcsfontosságú gyakorlati területek – Hibamegelőzés, Technológiai változás menedzsment, Folyamatváltozás menedzsment

Hat Szigma

A Six Sigma egy minőségirányítási program a „hat szigma” minőségi szint elérésére. A Motorola volt az úttörője az 1980-as évek közepén, és túl sok más gyártó céget is elterjedt, nevezetesen a General Electric Corporationt (GE).

A Six Sigma egy szigorú és fegyelmezett módszertan, amely adatok és statisztikai elemzések segítségével méri és javítja a vállalat működési teljesítményét azáltal, hogy azonosítja és kiküszöböli a "hibákat" a gyártástól a tranzakcióig, illetve a terméktől a szolgáltatásig. Általában 3,4 hiba/millió lehetőségként definiálják, a Six Sigma három különböző szinten definiálható és értelmezhető: metrika, módszertan és filozófia...

A Sigma képzési folyamatait a Six Sigma Green Belts és Six Sigma Black Belts hajtja végre, és a Six Sigma Master Black Belts felügyeli.

ISO

Az ISO - Nemzetközi Szabványügyi Szervezet egy 150 ország nemzeti szabványügyi intézeteinek hálózata, országonként egy taggal, és Genfben (Svájcban) a központi titkárság koordinálja a rendszert. Az ISO egy nem kormányzati szervezet. Az ISO több mint 13 000 nemzetközi szabványt dolgozott ki különféle témákban.

11. Szoftvertesztelési tanúsítványok

Minősítési információk szoftverminőség-ellenőrző és tesztelő mérnökök számára

CSQE – ASQ (Amerikai Minőségügyi Társaság) programja a CSQE (Certified Software Quality Engineer) számára – információk a követelményekről, a szükséges tudásanyag vázlata, tanulmányi hivatkozások listája és még sok más.

CSQA/CSTE – A QAI (Quality Assurance Institute) programja a CSQA (Certified Software Quality Analyst) és a CSTE (Certified Software Test Engineer) minősítésekhez.

ISEB Szoftvertesztelési Tanúsítványok – A British Computer Society 2 szintű tanúsítási programot tart fenn – ISEB Foundation Certificate, Practitioner Certificate.

ISTQB Certified Tester – A Nemzetközi Szoftvertesztelési Képesítési Testület a németországi székhelyű Európai Minőségügyi Szervezet – Szoftvercsoport része.
A tanúsítványok tapasztalaton, tanfolyamon és teszten alapulnak. Két szint áll rendelkezésre: alapozó és haladó.

12. Tények a szoftverfejlesztésről

Az alábbiakban felsorolunk néhány tényt, amelyek segítségével jobb betekintést nyerhet a szoftverfejlesztés valóságába.

1. A legjobb programozók akár 28-szor jobbak, mint a legrosszabb programozók.
2. Az új eszközök/technikák kezdeti termelékenység/minőség VESZTÉST okoznak.
3. A megvalósíthatósági tanulmányra a válasz szinte mindig „igen”.
4. A 2002. májusi jelentés a National Institute of Standards és
   A Technologies (NIST)(1) megbecsüli a szoftverhibák éves költségét az Egyesült Államokban
   59,5 milliárd dollár.
5. Az újrafelhasználható alkatrészeket háromszor olyan nehéz megépíteni
6. A probléma összetettségének minden 25%-os növekedése esetén 100%-kal nő a megoldás
   bonyolultság.
7. A szoftveres munka 80%-a intellektuális. Jó része kreatív. Kevés az
   irodai.
8. A követelmények hibáit a legdrágább a gyártás során javítani.
9. A hiányzó követelmények a legnehezebben korrigálható követelményhibák.
10. A hibaelhárítás az életciklus legidőigényesebb szakasza.
11. A szoftvereket általában a legjobb esetben 55-60%-os (ági) lefedettségi szinten tesztelik.
12. A 100%-os lefedettség még messze nem elég.
13. A szigorú ellenőrzések a hibák akár 90%-át is eltávolíthatják az első teszteset előtt
    fuss.
15. A karbantartás általában a szoftverköltségek 40-80%-át emészti fel. Valószínűleg ez a szoftver
    legfontosabb életciklus-fázisa.
16. A fejlesztések a karbantartási költségek nagyjából 60%-át teszik ki.
17. Nincs egyetlen legjobb megközelítés a szoftverhibák eltávolítására.

Jó tesztelést!!!