1. fejezet - A tesztelés alapfogalmai

Tartalom

Tesztelésre azért van szükség, hogy a szoftver termékben meglévő hibákat még az üzembe helyezés előtt megtaláljuk, ezzel növeljük a termék minőségét, megbízhatóságát. Abban szinte biztosak lehetünk, hogy a szoftverben van hiba, hiszen azt emberek fejlesztik és az emberek hibáznak.

Gondoljunk arra, hogy a legegyszerűbb programban, mondjuk egy szöveges menü kezelésben, mennyi hibát kellett kijavítani, mielőtt működőképes lett. Tehát abban szinte biztosak lehetünk, hogy tesztelés előtt van hiba, abban viszont nem lehetünk biztosak, hogy tesztelés után nem marad hiba. A tesztelés után azt tudjuk elmondani, hogy a letesztelt részekben nincs hiba, így nő a program megbízhatósága. Ez azt is mutatja, hogy a program azon funkcióit kell tesztelni, amiket a felhasználók legtöbbször fognak használni.

A tesztelés alapelvei

A tesztelés alapjait a következő alapelvekben foglalhatjuk össze:

1. A tesztelés hibák jelenlétét jelzi: A tesztelés képes felfedni a hibákat, de azt nem, hogy nincs hiba. Ugyanakkor a szoftver minőségét és megbízhatóságát növeli.

2. Nem lehetséges kimerítő teszt: Minden bemeneti kombinációt nem lehet letesztelni (csak egy 10 hosszú karakterláncnak 256^10 lehetséges értéke van) és nem is érdemes. Általában csak a magas kockázatú és magas prioritású részeket teszteljük.

3. Korai teszt: Érdemes a tesztelést az életciklus minél korábbi szakaszában elkezdeni, mert minél hamar találunk meg egy hibát (mondjuk a specifikációban), annál olcsóbb javítani. Ez azt is jelenti, hogy nemcsak programot, hanem dokumentumokat is lehet tesztelni.

4. Hibák csoportosulása: A tesztelésre csak véges időnk van, ezért a tesztelést azokra a modulokra kell koncentrálni, ahol a hibák a legvalószínűbbek, illetve azokra a bemenetekre kell tesztelnünk, amelyre valószínűleg hibás a szoftver (pl. szélsőértékek).

5. A féregirtó paradoxon: Ha az újratesztelés során (lásd később a regressziós tesztet) mindig ugyanazokat a teszteseteket futtatjuk, akkor egy idő után ezek már nem találnak több hibát (mintha a férgek alkalmazkodnának a teszthez). Ezért a tesztjeinket néha bővíteni kell.

6. A tesztelés függ a körülményektől: Másképp tesztelünk egy atomerőműnek szánt programot és egy beadandót. Másképp tesztelünk, ha a tesztre 10 napunk vagy csak egy éjszakánk van.
7. A hibátlan rendszer téveszméje: Hiába javítjuk ki a hibákat a szoftverben, azzal nem lesz elégedett a megrendelő, ha nem felel meg az igényeinek. Azaz használhatatlan szoftvert nem érdemes tesztelni.

Tesztelési technikák

A tesztelési technikákat csoportosíthatjuk a szerint, hogy a teszteseteket milyen információ alapján állítjuk elő. E szerint létezik:

1. Feketedobozos (black-box) vagy specifikáció alapú, amikor a specifikáció alapján készülnek a tesztesetek.

2. Fehérdobozos (white-box) vagy strukturális teszt, amikor a forráskód alapján készülnek a tesztesetek.

Tehát beszélünk feketedobozos tesztelésről, amikor a tesztelő nem látja a forráskódot, de a specifikációkat igen, fehérdobozos tesztelésről, amikor a forráskód rendelkezésre áll.

A feketedobozos tesztelést specifikáció alapúnak is nevezzük, mert a specifikáció alapján készül. Ugyanakkor a teszt futtatásához szükség van a lefordított szoftverre. Leggyakoribb formája, hogy egy adott bemenetre tudjuk, milyen kimenetet kellene adni a programnak. Lefuttatjuk a programot a bemenetre és összehasonlítjuk a kapott kimenetet az elvárttal. Ezt alkalmazzák pl. az ACM versenyeken is.

A fehérdobozos tesztelést strukturális tesztelésnek is nevezzük, mert mindig egy már kész struktúrát, pl. program kódot, tesztelünk. A strukturális teszt esetén értelmezhető a (struktúra) lefedettség. A lefedettség azt mutatja meg, hogy a struktúra hány százalékát tudjuk tesztelni a meglévő tesztesetekkel. Általában ezeket a struktúrákat teszteljük:

1. kódsorok,
2. elágazások,
3. metódusok,
4. osztályok,
5. funkciók,
6. modulok.

Például a gyakran használt unit-teszt a metódusok struktúra tesztje.

A tesztelés szintjei

A tesztelés szintjei a következők:

1. komponensteszt,
2. integrációs teszt,
3. rendszerteszt,
4. átvételi teszt.

A komponensteszt csak a rendszer egy komponensét teszteli önmagában. Az integrációs teszt kettő vagy több komponens együttműködési tesztje. A rendszerteszt az egész rendszert, tehát minden komponenst együtt, teszteli. Ez első három teszt szintet együttesen fejlesztői tesztnek hívjuk, mert ezeket a fejlesztő cég alkalmazottai vagy megbízottjai végzik. Az átvételi teszt során a felhasználók a kész rendszert tesztelik. Ezek általában időrendben is így követik egymást.

A komponensteszt a rendszer önálló részeit teszteli általában a forráskód ismeretében (fehér dobozos tesztelés). Gyakori fajtái:

1. unit-teszt,
2. modulteszt.

A unit-teszt, vagy más néven egységteszt, a metódusokat teszteli. Adott paraméterekre ismerjük a metódus visszatérési értékét (vagy mellékhatását). A unit-teszt megvizsgálja, hogy a tényleges visszatérési érték megegyezik-e az elvárttal. Ha igen, sikeres a teszt, egyébként sikertelen. Elvárás, hogy magának a unit-tesztnek ne legyen mellékhatása.

A unit-tesztelést minden fejlett programozási környezet (integrated development environment, IDE) támogatja, azaz egyszerű ilyen teszteket írni. A jelentőségüket az adja, hogy a futtatásukat is támogatják, így egy változtatás után csak lefuttatjuk az összes unit-tesztet, ezzel biztosítjuk magunkat, hogy a változás nem okozott hibát. Ezt nevezzük regressziós tesztnek.

A modulteszt általában a modul nem-funkcionális tulajdonságát teszteli, pl. sebességét, vagy, hogy van-e memóriaszivárgás (memory lake), van-e szűk keresztmetszet (bottleneck).

Az integrációs teszt során a komponensek közti interfészeket, az operációs rendszer és a rendszer közti interfészt, illetve más rendszerek felé nyújtott interfészeket tesztelik. Az integrációs teszt legismertebb típusai:

1. Komponens – komponens integrációs teszt: A komponensek közötti kölcsönhatások tesztje a komponensteszt után.
2. Rendszer – rendszer integrációs teszt: A rendszer és más rendszerek közötti kölcsönhatásokat tesztje a rendszerteszt után.

Az integrációs teszt az összeillesztés során keletkező hibákat keresi. Mivel a részeket más-más programozók, csapatok fejlesztették, ezért az elégtelen kommunikációból súlyos hibák keletkezhetnek. Gyakori hiba, hogy az egyik programozó valamit feltételez (pl. a metódus csak pozitív számokat kap a paraméterében), amiről a másik nem tud (és meghívja a metódust

egy negatív értékkel). Ezek a hibák kontraktus alapú tervezéssel (design by contract) elkerülhetőek.

Az integrációs teszteket érdemes minél hamarabb elvégezni, mert minél nagyobb az integráció mértéke, annál nehezebb meghatározni, hogy a fellelt hiba (általában egy futási hiba) honnan származik. Ellenkező esetben, azaz amikor már minden komponens kész és csak akkor tesztelünk, akkor ezt a „nagy bumm tesztnek” (big bang tesztnek) nevezzük, ami rendkívül kockázatos.

A rendszerteszt a már kész szoftverterméket teszteli, hogy megfelel-e:

1. a követelmény specifikációnak,
2. a funkcionális specifikációnak,
3. a rendszertervnek.

A rendszerteszt nagyon gyakran feketedobozos teszt. Gyakran nem is a fejlesztő cég, ahol esetleg elfogultak a tesztelők, hanem egy független cég végzi. Ilyenkor a tesztelők és a fejlesztők közti kapcsolat tartást egy hibabejelentő (bug trucking) rendszer látja el. A rendszerteszt feladata, hogy ellenőrizze, hogy a specifikációknak megfelel-e a termék. Ha pl. a követelmény specifikáció azt írja, hogy lehessen jelentést készíteni az éve forgalomról, akkor ezt a tesztelők kipróbálják, hogy lehet-e, és hogy helyes-e a jelentés. Ha hibát találnak, azt felviszik a hibabejelentő rendszerbe.

Fontos, hogy a rendszertesthez használt környezet a lehető legjobban hasonlítson a megrendelő környezetére, hogy a környezet-specifikus hibákat is sikerüljön felderíteni.

Az átvételi teszt hasonlóan a rendszertesthez az egész rendszert teszteli, de ezt már a végfelhasználók végzik. Az átvételi teszt legismertebb fajtái a következők:

1. alfa teszt,
2. béta teszt,
3. felhasználói átvételi teszt,
4. üzemeltetői átvételi teszt.

Az alfa teszt a kész termék tesztje a fejlesztő cégnél, de nem a fejlesztő csapat által. Ennek része, amikor egy kis segédprogram több millió véletlen egérkattintással ellenőrzi, hogy össze-vissza kattintgatva sem lehet kifektetni a programot.

Ezután következik a béta teszt. A béta tesztet a végfelhasználók egy szűk csoportja végzi. Játékoknál gyakori, hogy a kiadás előtt néhány fanatikus játékosnak elküldik a játékot, akik rövid alatt sokat játszanak vele. Cserébe csak azt kérik, hogy a felfedezett hibákat jelentsék.

Ezután jön egy sokkal szélesebb béta teszt, amit felhasználói átvételi tesztnek nevezünk. Ekkor már az összes, vagy majdnem az összes felhasználó, megkapja a szoftvert és az éles környezetben használatba veszi. Azaz installálják és használják, de még nem a termelésben. Ennek a tesztnek a célja, hogy a felhasználók meggyőződjenek, hogy a termék biztonságosan használható lesz majd éles körülmények között is. Itt már elvárt, hogy a fő funkciók mind működjenek, de előfordulhat, hogy az éles színhelyen előjön olyan környezet függő hiba, ami a teszt környezetben nem jött elő. Lehet ez pl. egy funkció lassúsága.

Ezután már csak az üzemeltetői átvételi teszt van hátra. Ekkor a rendszergazdák ellenőrzik, hogy a biztonsági funkciók, pl. a biztonsági mentés és a helyreállítás, helyesen működnek-e.

A tesztelési tevékenység

Ugyanakkor a tesztelés nem csak tesztek készítéséből és futtatásából áll. A leggyakoribb tesztelési tevékenységek:

1. teszterv elkészítése,
2. tesztesetek tervezése,
3. felkészülés a végrehajtásra,
4. tesztek végrehajtása,
5. kilépési feltételek vizsgálata,
6. eredmények értékelése,
7. jelentéskészítés.

A teszterv fontos dokumentum, amely leírja, hogy mit, milyen céllal, hogyan kell tesztelni. A teszterv általában a rendszerterv része, azon belül is a minőségbiztosítás (quality assurance, QA) fejezethez tartozik. A teszt célja lehet:

1. megtalálni a hibákat,
2. növelni a megbízhatóságot,
3. megelőzni a hibákat.

A fejlesztői tesztek célja általában minél több hiba megtalálása. Az átvételi teszt célja, hogy a felhasználók bizalma nőjön a megbízhatóságban. A regressziós teszt célja, hogy megelőzni, hogy a változások a rendszer többi részében hibákat okozzanak.

A teszterv elkészítéséhez a célon túl tudni kell, hogy mit és hogyan kell tesztelni, mikor tekintjük a tesztet sikeresnek. Ehhez ismernünk kell a következő fogalmakat:

1. A teszt tárgya: A rendszer azon része, amelyet tesztelünk. ez lehet az egész rendszer is.
2. Tesztbázis: Azon dokumentumok összessége, amelyek a teszt tárgyára vonatkozó követelményeket tartalmazzák.
3. Tesztadat: Olyan adat, amivel meghívjuk a teszt tárgyát. Általában ismert, hogy milyen értéket kellene erre adnia a teszt tárgyának vagy milyen viselkedést kellene produkálnia. Ez az elvárt visszatérési érték, illetve viselkedés. A valós visszatérési értéket, illetve viselkedést hasonlítjuk össze az elvárttal.
4. Kilépési feltétel: Minden tesztnél előre meghatározzuk, mikor tekintjük ezt a tesztet lezárhatónak. Ezt nevezzük kilépési feltételnek. A kilépési feltétel általában az, hogy minden tesztest sikeresen lefut, de lehet az is, hogy a kritikus részek tesztlefedettsége 100%.

A teszterv leírja a teszt tárgyát, kigyűjti a tesztbázisból a teszt által lefedett követelményeket, meghatározza a kilépési feltételt. A tesztadatokat általában csak a teszteset határozzák meg, de gyakran a tesztesetek is részei a tesztterveknek.

A tesztesetek leírják, hogy milyen tesztadattal kell meghajtani a teszt tárgyát. Illetve, hogy mi az elvárt visszatérési érték vagy viselkedés. A tesztadatok meghatározásához általában úgynevezett ekvivalencia-osztályokat állítunk fel. Egy ekvivalencia-osztály minden elemére a szoftver ugyanazon része fut le. Természetesen más módszerek is léteznek, amikre később térünk ki.

A tesztesetek végrehajtásához teszt környezetre van szükségünk. A teszt környezet kialakításánál törekedni kell, hogy a lehető legjobban hasonlítson az éles környezetre, amely a végfelhasználónál működik. A felkészülés során írhatunk teszt szkripteket is, amik az automatizálást segítik.

A tesztek végrehajtása során teszt naplót vezetünk. Ebbe írjuk le, hogy milyen lépéseket hajtottunk végre és milyen eredményeket kaptunk. A tesz napló alapján a tesztnek megismételhetőnek kell lennie. Ha hibát találunk, akkor a hibabejelentőt a teszt napló alapján töltjük ki.

A tesztek után meg kell vizsgálni, hogy sikeresen teljesítettük-e a kilépési feltételt. Ehhez a tesztesetben leírt elvárt eredményt hasonlítjuk össze a teszt naplóban lévő valós eredménnyel a kilépési feltétel alapján. Ha kilépési feltételek teljesülnek, akkor mehetünk tovább. Ha nem, akkor vagy a teszt tárgya, vagy a kilépési feltétel hibás. Ha kell, akkor módosítjuk a kilépési feltételt. Ha teszt tárgya hibás, akkor a hibabejelentő rendszeren keresztül értesítjük a fejlesztőket. A teszteket addig ismételjük, míg mindegyik kilépési feltétele igaz nem lesz.

A tesztek eredményei alapján további teszteket készíthetünk. Elhatározhatjuk, hogy a megtalált hibákhoz hasonló hibákat felderítjük. Ezt általában a teszttervek elő is írják. Dönthetünk úgy, hogy egy komponenst nem érdemes tovább tesztelni, de egy másikat tüzetesebben kell tesztelni. Ezek a döntések a teszt irányításához tartoznak.

Végül jelentést kell készítenünk. Itt arra kell figyelnünk, hogy sok programozó kritikaként éli meg, ha a kódjában a tesztelők hibát találnak. Úgy érzi, hogy rossz programozó és veszélyben van az állása. Ezért a tesztelőket nem várt támadások érhetik. Ennek elkerülésére a jelentésben nem szabad személyeskedni, nem muszáj látnia főnöknek, kinek a kódjában volt hiba. A hibákat rá lehet fogni a rövid időre, a nagy nyomásra. Jó, ha kiemeljük a tesztelők és a fejlesztők közös célját, a magas minőségű, hibamentes szoftver fejlesztését.