SZABADKAI MŰSZAKI SZAKFŐISKOLA
SZABADKA

SZOFTVERTESZTELÉS

Készítette: Balla Renáta
Szak: Internet és elektronkus ügyvitel
Index: 16218209

Szabadka, 2020

Tartalom

Bevezető

Tesztelésre szükség van, mivel emberek készítik a szoftvereket és midenki hibázik. Ezért a szoftverokat termékben üzembe helyezés előtt tesztelni kell, így megtaláljuk a hibákat és növeljük a termék minőségét, megbízhatóságát. 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

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 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 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:

•Feketedobozos (black-box) vagy specifikáció alapú, amikor a specifikáció alapján készülnek a tesztesetek mivel a teszetelő nem látja a forráskódot.

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

A tesztelés szintjei

A tesztelés szintjei a következők:

Komponensteszt: a rendszer egy komponensét teszteli önmagában. Gyakori fajtái az unit-teszt és a 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 *modulteszt* általában a modul nem-funkcionális tulajdonságát teszteli, pl. sebességét, vagy, hogy van-e memóriaszivárgás (memory lake), van-e szűk keresztmetszet (bottleneck).

Integrációs teszt: kettő vagy több komponens együttműködési tesztje.

Rendszerteszt az egész rendszert, tehát minden komponenst együtt, teszteli.

Az átvételi teszt során a felhasználók a kész rendszert tesztelik. Az átvételi teszt legismertebb fajtái a következők: alfa teszt, béta teszt, felhasználói átvételi teszt, 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.

Felhasználó 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.

Üzemeltetői átvételi teszt során 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, hanem tevékenységekből is. A leggyakoribb tesztelési tevékenységek:

- •teszterv elkészítése
- •tesztesetek tervezése
- •felkészülés a végrehajtásra
- •tesztek végrehajtása
- •kilépési feltételek vizsgálata
- •eredmények értékelése
- •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:
- •megtalálni a hibákat
- •növelni a megbízhatóságot,
- •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:

- •A teszt tárgya: A rendszer azon része, amelyet tesztelünk. ez lehet az egész rendszer is.
- •Tesztbázis: Azon dokumentumok összessége, amelyek a teszt tárgyára vonatkozó követelményeket tartalmazzák.
- •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.
- •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 teszttervnek.

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 fejlesztés.

A szoftver életciklusa, módszertanok

A szoftver életciklusa szoftverrel egy idős fogalom. Ha átadunk egy szoftvert a felhasználóknak, akkor a felhasználók előbb vagy utóbb újabb igényekkel állnak elő, ami a szoftver továbbfejlesztését teszi szükségessé. Tehát egy szoftver soha sincs kész, ciklikusan meg-megújul. Ezt nevezzük életciklusnak. Az életciklus lépéseit a módszertanok határozzák meg

V-modell

A nevét onnan kapta, hogy két szára van és így egy V betűhöz hasonlít. Az egyik szára a fejlesztési szár. A másik szára a létrejövő termékek tesztjeit tartalmazza. Ez a tesztelési szár. Az egy szinten lévő fejlesztési és tesztelési lépések összetartoznak, azaz a tesztelési lépés a fejlesztési lépés során létrejött dokumentumokat használja, vagy a létrejött terméket teszteli. A V-modell a vízesés modell kiegészítése teszteléssel. Ez azt jelenti, hogy először végre kell hajtani a fejlesztés lépéseit, ezután jönnek a tesztelés lépései. Ha valamelyik teszt hibát talál, akkor vissza kell menni a megfelelő fejlesztési lépésre. Fő jellemzője a teszt központi szerepe.

Prototípus modell

A prototípus modell válasz a vízesés modell sikertelenségére. A fejlesztő cégek rájöttek, hogy tarthatatlan a vízesés modell megközelítése, hogy a rendszerrel a felhasználó csak a projekt végén találkozik. Gyakran csak ekkor derült ki, hogy az életciklus elején félreértették egymást a felek és nem a valós követelményeknek megfelelő rendszer született. Ezt elkerülendő a prototípus modell azt mondja, hogy a végső átadás előtt több prototípust is szállítsunk le, hogy mihamarabb kiderüljenek a félreértések, illetve a megrendelő lássa, mit várhat a rendszertől.

Iteratív és inkrementális módszertanok

Az iteratív módszertan előírja, hogy a fejlesztést, kezdve az igényfelméréstől az üzemeltetésig, kisebb iterációk sorozatára bontsuk. A folyamatos finomítás lehetővé teszi, hogy mélyen megértsük a feladatot és felderítsük az ellentmondásokat. Minden iteráció kiegészíti a már kifejlesztett prototípust. A kiegészítést *inkrementumnak* is nevezzük. Azok a módszertanok, amik a folyamatra teszik a hangsúlyt, azaz az iterációra, azokat *iteratív* módszertanoknak nevezzük. Azokat, amelyek az iteráció termékére, az inkrementumra teszik a hangsúlyt, azokat inkrementális módszertanoknak hívjuk. A mai módszertanok nagy része, kezdve a prototípus modelltől egészen az agilis modellekig, ebbe a családba tartoznak Az iteratív modell fő ereje abban rejlik, hogy az életciklus lépései nem egymás után jönnek, mint a strukturált módszertanok esetén, hanem időben átfedik egymást. Minden iterációban van elemzés, tervezés, implementáció és tesztelés.

Gyors alkalmazásfejlesztés – RAD

A gyors alkalmazásfejlesztés vagy ismertebb nevén RAD (Rapid Application Development) egy olyan elgondolás, amelynek lényege a szoftver gyorsabb és jobb minőségű elkészítése. Itt a csapat - megrendelő és a csapaton belüli kommunikációban kevésbé formális. Szigorú ütemterv van, így az újítások mindig csak a termék következő verziójában jelennek meg. Komponenseket újrahasznosítsák.

Agilis szoftverfejlesztés

Az agilis fejlesztési módszerek (nevezik adaptívnak is) egyik fontos jellemzője, hogy a résztvevők, amennyire lehetséges megpróbálnak alkalmazkodni a projekthez. Ezért fontos például, hogy a fejlesztők folyamatosan tanuljanak. Az agilis szoftverfejlesztésben a legfontosabb a megrendelő kielégítése használható szoftver gyors és folyamatos átadásával. A scrum is egy agilis szoftverfejlesztés.

Statikus tesztelési technikák

A statikus tesztelési technikák a szoftver forrás kódját vizsgálják fordítási időben. Ide tartozik a dokumentáció felülvizsgálata is. A statikus tesztelés párja a dinamikus tesztelés,

amely a szoftvert futásidőben teszteli. A statikus tesztelést *felülvizsgálat* és *statikus* elemzésre oszthatjuk.

Felülvizsgálat

A felülvizsgálat azt jelenti, hogy manuálisan átnézzük a forráskódot és fejben futtatjuk vagy egyszerűen csak gyanús részeket keresünk benne. Ezzel szemben áll a statikus elemzés, ahol szoftverekkel nézetjük át automatikusan a forráskódot. A felülvizsgálat fehérdobozos teszt, mivel kell hozzá a forráskód. A felülvizsgálat lehet informális, pl. páros programozás, de akár nagyon formális is, amikor a folyamatot jól dokumentáljuk, illetve a két szélsőség közti átmenetek.
Informális felülvizsgálat ha egy tapasztalt programozó átnézi *(review)* a kezdők kódját, ehhez hasonló a páros programozás *(pair programming)* is. Ekkor két programozó ír egy kódot, pontosabban az egyik írja, a másik figyeli. A kódszépítés *(refactoring)* egy másik módja a felülvizsgálatnak. Ilyenkor a már letesztelt, működő kódot lehet szépíteni, ami esetleg lassú, rugalmatlan, vagy egyszerűen csak csúnya. A kódszépítés előfeltétele, hogy legyen sok unit-teszt. A szépítés során nem szabad megváltoztatni a kód funkcionalitását.

Átvizsgálás az már kicsit formálisabb módja a felülvizsgálatnak. Általában a módszertan előírja, hogy az elkészült kisebb-nagyobb modulokat ismertetni kell a csapat többi tagjával, a többi csapattal. Célja, hogy a mások is átlássák az általunk írt kódrészletet kritikai megjegyzéseikkel segítsék a kód minőségének javítását.

Technikai felülvizsgálatra általában akkor kerül sor, ha a szoftver teljesítményével nem vagyunk elégedettek. Azt általában könnyű megtalálni a felhasználói visszajelzések és úgynevezett profiler programok segítségével, hogy mi az a szűk keresztmetszet (angolul: bottleneck), ami a lassúságot okozza.

Inspekció a legformálisabb felülvizsgálat. Ezt is akkor használjuk, ha külső szakértő bevonására van szükségünk. A technikai felülvizsgálattól az különbözteti meg, hogy a szoftver cég és a szakértőt adó cég részletesebb szerződést köt.

Statikus elemzés

A statikus elemzés fehérdobozos teszt, hiszen szükséges hozzá a forráskód. Néhány esetben, pl. holtpont ellenőrzés, elegendő a lefordított köztes kód (byte kód). A statikus elemzés azért hasznos, mert olyan hibákat fedez fel, amiket más tesztelési eljárással nehéz megtalálni.

Statikus elemzés csak a forráskód alapján

Azok az elemzők, amelyek csak a forráskódot használják fel az elemzéshez, azok nagyon hasznosak olyan szempontból, hogy nem igényelnek plusz erőfeszítést a programozóktól a specifikáció megírásához. Ilyen eszköz például a FindBugs. Ezeket az eszközöket csak bele kell illeszteni a fordítás folyamatába. Ezután a statikus elemző felhívja a figyelmünket a tipikus programozói hibákra. Ezek általában programozási nyelv specifikusak, de léteznek nyelv függetlenek.

Statikus elemzés a forráskód és modell alapján

Ebben az esetben a forráskód mellett van egy modellünk is, ami leírja, hogyan kellene működnie a programnak. A program viselkedése ilyen esetben elő- és utófeltételekkel, illetve invariánsokkal van leírva. Ezt úgy érjük el legkönnyebben, hogy kontraktus alapú tervezést (design by contract) használunk. Ez esetben minden metódusnak van egy kontraktusa, amely a metódus elő- és utófeltételében ölt testet. A szerződés kimondja, hogy ha a metódus hívása előtt igaz az előfeltétele, akkor a metódus lefutása után igaznak kell lennie az utófeltételének.

Dinamikus tesztelési technikák

A dinamikus tesztelési technikák elsősorban a komponens teszt, azon belül is főleg a unit-teszt (egységteszt) fázis eszköze. A dinamikus tesztek tervezése alapvetően az alábbi három lépésből áll:

- A tesztelés alanyának, céljának meghatározása (test condition)
- Tesztesetek (test cases) specifikálása
- Teszt folyamat (test procedure) specifikálása A tesztelési folyamathoz kapcsolódnak még a teszt készlet (test suite), hibamodell és lefedettség (test coverage) fogalmak is.

A tesztelés alanya lehet rendszer egy olyan jellemzője, amely ellenőrizhető egy vagy több teszt esettel. Ilyen lehet például: funkció, tranzakció, képesség (feature), minőségi jellemző, strukturális elem.

Egy teszteset célja egy meghatározott vezérlési út végrehajtatása a tesztelendő program egységben, vagy egy meghatározott követelmény teljesülésének ellenőrzése. Egy teszteset végrehajtása esetén a rendszert egy megadott kezdő állapotban kell hozni (prekondíciók), megadott input értékek halmazával végre kell hajtatni a tesztelt elemet, majd a teszt futásának eredményét össze kell hasonlítani az elvárt eredménnyel és ellenőrizni kell, hogy a végrehajtás után a rendszer az elvárt állapotba (posztkondíciók) került-e.

Teszt specifikáció az egy teszteset végrehajtásához szükséges tevékenységek sorozatának a leírása. Szokás teszt forgatókönyvnek (manual test script) is nevezni.

Egy tesztkészlet tesztesetek és hozzájuk tartozó teszt specifikációk halmaza. Csoportosítható egy teszt alanyra, vagy egy vizsgált hibára.

Hibamodell azon (feltételezett) szoftver hibák halmaza, amelyre a teszt tervezés irányul. A tesztesethez kapcsolódó példához a hibamodell azt rögzítheti, hogy az alábbi hibák következhetnek be:

•számítási hibák

•adatbázis lekérdezési hibák (rossz adatokat használunk fel a számításhoz)

•az adatbázisba módosításának hibák (a számítás eredménye rosszul kerül be az adatbázisba).

A hibamodell a tesztesetek tervezéséhez ad támpontokat.

Egy rendszer teljes tesztelésének megtervezéséhez (ami a teszt menedzsment egyik feladata, így a következő fejezetben foglalkozunk vele részletesebben), az alábbiakat foglalja magában:

•a szükséges tesztelési célok meghatározása,

•minden tesztelési célhoz a szükséges tesz készlet definiálása

•az egyes teszt készletekben foglalt tesztek ütemezésének és a végrehajtásuk dokumentálásának megtervezése.

A teszt folyamat terve a rendszer specifikációjának egy fejezete lehet, de összetettebb rendszerek esetén általában külön teszt specifikáció készül.

A teszt lefedettség a számszerű értékelése annak, hogy a tesztelési tevékenység mennyire alapos, illetve hogy egy adott időpontban hol tart.

A teszt tervezési technikák fajtái

- Specifikáció alapú technikák. Ezek a módszerek a teszteseteket közvetlenül a rendszer specifikációjából (modelljéből) vezetik le. Black-box technikáknak is nevezzük ezeket, mert az egyes szoftver modulok belső szerkezetének ismerete nélkül, az egyes modulok által teljesítendő funkcionalitások alapján tervezhetjük meg a teszt eseteket.
- Modell alapú technika (Model-driven testing). Valójában az előző csoportba tartozik, csak formalizáltabb technika. Közvetlenül az UML modellből vezeti le a teszteseteket, és formalizált teszt specifikációt alkalmaz. Erre használható az UML kiterjesztése. (UTP – UML Testing Profile.)
- Struktúra alapú technikák. Ezek a módszerek a kód ismeretében határozzák meg a teszteseteket. White-box technikáknak is nevezzük.
- Gyakorlat alapú technikák. A tesztelőknek a munkájuk során megszerzett tapasztalatira épülő, a szakmai intuíciókat is értékesítő technikák

Integrációs tesztek

Az integrációs tesztek az egységteszteket követik. Az egység teszt szorosan kapcsolódik az implementációs fázishoz, és biztosítja, hogy a részegységek önmagukban már helyesen működnek. Így ha az integrációs tesztek során hibát észlelünk, az feltehetőleg a modulok együttműködéséből adódik. Az integrációs teszteknek következő fázisait különböztetjük meg:

- technikai integrációs teszt (Integration Level Testing, ILT),
- rendszerteszt (System Level Testing , SLT)
- elfogadtatási teszt (User Acceptance Testing, UAT).

Integration Level Testing, ILT

Célja az együttműködő egységek vizsgálata. Ezért a teszteléshez egy részrendszert állítunk össze a már önmagában tesztelt elemekből. Ezek többnyire csak technikai, nem funkcionális részrendszerek, ezért probléma lehet a megfelelő tesztesetek előállítása.

System Level Testing , SLT

A rendszer összes komponensének teljes körű (funkcionális, nem funkcionális) tesztelése. Feladata annak megállapítása, hogy a rendszer kiadható-e a megrendelőnek. Ez tehát egy végső ellenőrzési fázis a fejlesztési folyamatban. Szokás elnevezése még: "release test", a tesztelés tárgyát képező rendszerváltozat pedig gyakran nevezik "alpha version"-nek.

User Acceptance Testing, UAT

Feladata annak megállapítása, hogy a rendszer éles üzembe állítható-e. Célja a felhasználó és minden haszonélvező (stakeholder) elégedettségének vizsgálata. Általában a megrendelő telephelyén, annak közreműködésével, és a végleges üzemeltetési körülmények között kell végrehajtani. A használható tesztelési módszerek hasonlóak, mint a SLT esetén, de azok közül csak a felhasználó számára releváns eseteket kell bemutatni

Specifikáció alapú technikák

Ezek a technikák a tesztelés alapjaként a rendszer specifikációját, esetleg formális modelljét tekintik. Amennyiben a specifikáció jól definiált és megfelelően strukturált, ennek elemzése során könnyen azonosíthatjuk a tesztelés alanyait (test conditions), amelyekből pedig származtathatjuk a teszteseteket.

Ekvivalencia particionálás

Ennek a technikának az alapja az a megfigyelés, hogy vannak olyan különböző input értékek, amelyekre a programnak ugyanúgy kell viselkednie. Ekvivalencia osztálynak nevezzük az input értékek olyan halmazát, amelyre ugyanúgy kell viselkednie a programnak.

Határérték analízis

Ennek a technikának az alapja az a megfigyelés, hogy a határértékek kezelésénél könnyebben követnek el hibát a programozók, mint az „általános” eseteknél. Figyelni kell a kimeneti ekvivalencia osztályok határértékeit is. Ehhez persze sokszor "visszafelé" kell gondolkodni, tehát meg kell határozni azon input értékek halmazát, amelyek határértékként kezelhető kimeneteket produkálnak.

Ok-hatás analízis

Ez a technikai egy döntési táblát épít fel, amelynek az oszlopai adják meg a definiálandó teszteseteket, ezért döntési tábla (decision table) technikának is nevezik. A módszer alapgondolata az, hogy a specifikáció gyakran olyan formában írja le a rendszer által megvalósítandó üzleti folyamatokat, hogy az egyes tevékenységeknek milyen bemeneti feltételei vannak.

Használati eset (use case) tesztelés

A mai fejlesztési projektekben a specifikáció gyakran használt eszköze a használati eset (use case) modell felépítése. Ilyen esetekben a teszt tervezés vezérfonalát a használati eset modell elemei alkotják, sőt, a teszteseteket is leírhatjuk használati esetekkel. Ebben az esetben a használati esetek és a tesztesetek összerendelése hasznos eszköz lehet annak eldöntésére, ellenőrzésére, hogy hol tartunk a tesztelési folyamatban.

Struktúra alapú technikák

Ezeknek a módszereknek az alapja az a tapasztalat, hogy a programozási hibák gyakran a vezérlési szerkezeteket érintik. A tesztelés célja tehát a kód struktúrájának a felderítése és helyességének ellenőrzése, ezért a tesztesetek generálása a forráskód elemzése alapján történik. Ebben a szemléletben a tesztesetek megtervezése során az a cél, hogy a vizsgált kód minden ágát végrehajtva vizsgáljuk annak működését. Mivel egy bonyolult kód végrehajtási útjainak száma nagyon magas lehet, általában nem túl nagy egységek képezik a tesztelés tárgyát. A kódbejárás alapját a kód matematikai modellje, a vezérlési folyamat gráf (control-flow graph, CFG) képezi. A vezérlési folyamat gráf egy olyan irányított gráf, amelyben a csomópontok a program utasításainak felelnek meg, az irányított élek pedig a végrehajtásuk sorrendjét jelölik ki. Egy döntési utasításnak megfelelő csomópontból több él indul ki, a vezérlési ágak összekapcsolódási pontjában elhelyezkedő utasításhoz pedig több él fut be. A ciklust visszafelé irányuló él reprezentálja.

Biztonsági tesztelés

A támadások azok a technológiák, amiket a támadók használnak, hogy kihasználják az alkalmazások sebezhető pontjait. A támadásokat gyakran tévesztik össze a sebezhető pontokkal. Egy támadás leírása azt mondja el, hogy mit tenne a támadó a gyengeség kihasználására, nem pedig az alkalmazás gyenge pontjait ismerteti. A támadás ellenállóság tesztelése általában feketedobozos teszt. Történhet a rendszer kiadása előtt vagy után is. Ha utána történik, akkor általában etikus törési kísérletről beszélünk. Ehhez általában külső szakembereket, fehér kalapos hacker-eket szoktak felkérni. Ha a kiadás előtt történik, akkor általában a legmagasabban képzett belső tesztmérnökök feladata.

Felhasznált irodalom

Ficsor Lajos, Dr. Kovács László, Krizsán Zoltán, Dr. Kusper Gábor: Szoftvertesztelés