Tesztelési módszertan

Szoftverminség

A tesztelés célja a szoftverfejlesztésben a szoftver minségének biztosítása. Ennek megfelelen a tesztelés módszertanának is elsdleges célja a szoftver minség biztosításának segítése. Olyan tesztelési módszertant kell követni a szoftver élettartama, tehát a fejlesztése, bevezetése és üzemeltetése során, amelyik elsegíti, hogy az egyéb, a szoftverminséget segít eszközökkel együtt garantálni lehessen a szoftver jó minségét, használhatóságát, hogy a szoftver segítsen az alkalmazó cégnek vagy intézménynek a céljai elérésében.

A szoftver minsége akkor jó, ha a szoftver képes a célját betölteni, és ennek megfelelen a szoftvert alkalmazó intézmény, vállalat intézményi, üzleti céljait a szoftvertl elvárt mértékben képes segíteni. Ez a minségdefiníció egyrészrl rendkívül általános, másrészrl viszont pontosan meghatározza azt az értéket, amelyet a szoftvernek kell elállítani. Az általánosság miatt a fenti definíció direkt módon ugyan nem használható a tesztelések tervezése és módszertan kialakítása során, de minden más definíciónak, amely vagy amelyek a szoftver minséget határozzák meg valamilyen aspektus szerint, összhangban kell lennie a fenti definícióval. Minden olyan kérdés, amelyik a szoftverminséggel kapcsolatban felmerül akár konkrét, akár általános módszertani szinten olyan választ kell, hogy kapjon, amelyik a szoftverminség általános definíciója szerinti szoftver használati cél elérését segíti.

Az általános definíció szerinti szoftverminség ugyanazon szoftver esetében más és más lehet különböz felhasználóknál, különböz elvárások és szoftverhasználati célok esetében. Ha alacsony a szoftverrel szembeni elvárás, vagy más célra akarják használni a szoftvert akkor a minség megítélése is más lesz. Ennek megfelelen a projektek során a szoftverhasználat céljait kell elször definiálni, majd pedig azt, hogy milyen módon kell a projekt során ezen célok elérését mérni, tesztelni.

A szoftverminség értékelésére vonatkozó általánosan elfogadott nemzetközi szabvány az ISO 9126. A szabvány célja, hogy a szoftverfejlesztési projektek során általánosan ismert emberi hibákból adódó minségi hibákat segítsen felismerni, és kiküszöbölni. A szabvány a szoftverminségre is egy modellel szolgál, amely modell hat kategóriára, és azon belül számos alkategóriára osztja azokat a tulajdonságokat, amelyeket figyelembe kell vagy lehet venni a szoftverminség értékelése, és így a tesztel tervezése , végrehajtása és kiértékelése során.

Funkcionalitás

A funkcionalitás olyan tulajdonságok halmaza, amelyek a szoftver mködésének funkcióival kapcsolatosak, és amelyek szükségesek a szoftverrel szemben támasztott mködési igények kielégítésére. A funkcionalitás témakörébe tartozó fogalmak.

- A *funkcionális megfelelés* azt határozza meg, hogy a szoftver mennyire felel meg a specifikációnak. Ez aránylag jól mérhet, és definiált fogalom, ez az a minségi elem, amelyet a funkcionális tesztekkel lehet mérni.
- A *megfelelség*, amely azt határozza meg, hogy a szoftver mennyire felel meg arra a célra, amire szánták. Bár ez a fogalom sokkal inkább szubjektív lehet, mint a megfelelés és nehezebben, vagy egyáltalán nem mérhet de sokkal közelebb áll a szoftverminség általános definíciójához. Törekedni kell arra, hogy a megfelelés mérése során, elssorban a mérés, azaz a funkcionális tesztek tervezésekor ne veszítse a tervez szem ell a megfelelséget a megfeleléssel szemben.
- A *precizitás* olyan fogalom, amely arról szól, hogy milyen mértékben, milyen pontosan felel meg a szoftver a követelményeknek.
- Az *együttmködés* fogalmára akkor kell különösen gondot fordítani, amikor a szoftvernek együtt kell mködnie más szoftver alkalmazásokkal. Ebben az esetben az egyes szoftveralkalmazások közötti adatcsere szabványos, vagy speciális (ú.n. proprietary) felületeken keresztül valósul meg. A az együttmködés vizsgálata a speciális felületek használata esetén mindenképpen szükséges, de szabványos felületek használata esetén is fontos lehet, ha mszaki lehetség van a szabvány különböz módokon történ használatára, implementálására.
- A *biztonság* olyan funkcionális fogalom, amely arról szól, hogy a szoftver milyen biztonsággal használható, illetve milyen veszélyeket jelenthez a szoftver használata. A szoftverminség értékelése során a biztonság értékelése külön terület, amelyet biztonsági szakemberek bevonásával lehet végezni.

Megbízhatóság

A megbízhatóság fogalma arról szól, hogy a szoftver mennyire stabilan mködik, és milyen szinten adja azt a teljesítményt (beleértve a funkcionalitást), amelyik megfelel az elvárásoknak. A megbízhatóságot szokás mérni (tesztelni) mérési idszakra (beégetési tesztek), vagy speciális feltételek között (sokk teszt). A megbízhatóság témakörébe tartozó fogalmak a

- *fejlettség*, amely arról szól, hogy a szoftver mennyire lett kifejlesztve, mennyire „fejldött ki” az fejlesztés és a használat során. Várható, hogy egy szoftver az élettartama során az újabb verziók kiadásával egyre érettebbé válik, illetve nagyobb funkcióváltások során az értettség idlegesen csökken, majd ezt követen a hiba javító verziókkal az értettség, és ezzel a megbízhatóság növekszik.
- *visszaállási képesség* arra jellemz, hogy a rendszer mennyire tud extrém, a mködést vagy annak egy részét akadályozó küls körülmények elmúlását követen visszaállni a normál mködésre. Általában az a kívánatos, hogy a szoftverrendszerek az extrém körülmények között se veszítsenek adatot, és az extrém körülmények elmúlását követen automatikusan álljanak vissza a normál mködési módra. Ugyanakkor lehetnek olyan alkalmazások, amelyek esetében megengedhet vagy akár kötelez a manuális, emberi beavatkozást igényl visszaállítás a normál mködési módra. Ennek oka lehet a túl magas, és az elnyöket el nem ért költség, amelyet az automatikus visszaállás kifejlesztése követelne meg, vagy biztonsági megfontolások, amelyek csak azt követen engedik meg a normál üzemre való visszaállást, amikor az üzemeltetés megbizonyosodott arról, hogy az extrém körülmények nem csökkentették a rendszer biztonságát.
- *hibatrés* azt jellemzi, hogy a rendszer mennyire marad mködképes egy vagy több elem meghibásodása esetén. Ez különösen fontos olyan esetben, amikor a mködés kiesése komoly hátrányt jelent a szervezet céljainak folyamatos elérése érdekében. Extrém példák olyan rendszerek, amelyek személyek épségének, egészségének védelmét üzemeltetik, tipikusan beépül gépjárm, veszélyes üzemi és egészségügyi rendszerek. A hibatrés vizsgálata elvi vizsgálattal és hibák elidézésével teszteléssel is történik.

Használhatóság

A használhatóság arra jellemz, hogy a rendszert mennyire könny használni, mennyi energiát kell a használónak belefektetni a rendszer használatába. A használhatóság alacsony szintje nem csak a használati költségeket növeli, hanem egy bizonyos szint alatt növeli a hiba lehetségeket, és így kihathat a szoftveralkalmazás biztonságára, és megfelelségére is. A használhatóság témakörébe olyan fogalmak tartoznak, mint a

- *tanulhatóság*, amely arra jellemz, hogy a rendszer kezelését, használatát mennyire nehéz, vagy éppen könny megtanulni.

- *érthetség* a tanulhatósággal szoros összefüggésben arról szól, hogy mennyire érthet a rendszer mködése. Értelemszeren egy értheten mköd rendszer megtanulása is könnyebb.
- *üzemeltethetség* fogalma azt határozza meg, hogy mennyi energiát igényel a rendszer üzemeltetése, használata. Ez a tanulhatóság és az érthetség mellett olyan tényezket is figyelembe vesz, mint a környezet kialakítása, napi karbantartási igénye.

Hatékonyság

A hatékonyság a rendszer által nyújtott teljesítmény, és az eközben felhasznált erforrások közötti arányt határozza meg. A hatékonyságot olyan erforrások mentén érdemes megvizsgálni, amelyek költsége szignifikáns, és amelyek gazdasági és/vagy mszaki okok miatt korlátozottak. Így tipikusan a háttértárak mérete, memória és processzor használat, de ebbe a témakörbe tartozhatnak olyan erforrások is, mint üzemeltet személyzettl igényelt karbantartási munkák mennyisége. A hatékonyság vizsgálata során két területet kell mindenképpen vizsgálni.

- Az egyik az erforrások használata
- A másik az idszükséglet.

Az idszükséglet vizsgálata azért különül el a többi erforrás felhasználásától, mert az id, mint erforrás speciális. Míg diszk terület, memória és processzorkapacitás bvítése elvileg lehetséges az idnél ez nem képzelhet el. Tipikusan például offline, tehát nem üzem közbeni mentési folyamatoknak üzemszünet alatt, általában hétvégén, vagy estétl reggelig le kell futniuk.

Karbantarthatóság

A karbantarthatóság arra jellemz, hogy a szoftveralkalmazás mennyire módosítható, mennyi energiát kell egy módosításhoz a munkába fektetni. Ezen belül meg szokták említeni a szoftver

- *stabilitását*, amelyik arra jellemz, hogy egy szükséges módosítás a szoftver kódjában mennyire módosít olyan funkciókat, amelyek a módosított résztl távol vannak (nem keverend össze a rendszer üzemeltetési stabilitásával). Amennyiben a rendszer összetett és nem lett a bels struktúrája megfelelen megtervezve, nem modulokból áll amelyek jól definiált bels felületeken keresztül érik el egymás szolgáltatásait, akkor a szoftver stabilitása alacsony lesz, egy kódrészlet megváltoztatása olyan funkciók hibáit is elidézhet, amely funkciók látszólag nem is függnek a módosított kódtól.
- *analizálhatóság* egy olyan fogalom, ami arra jellemz, hogy a szoftvert mennyire könny, vagy éppen nehéz megérteni, vizsgálni. A szoftverkód vizsgálata szükséges minden olyan esetben, amikor valamilyen változtatást kell végrehajtani a programon, legyen az új funkció bevezetése, vagy hiba javítása. A kód elemzése nélkül csak az a személy tudja a kódot módosítani, aki azt készített (és még nem felejtette el). Gyakorlatilag minden kódmódosítás egyik elkészít lépése a kód analízise, még akkor is ha ez nem különül el láthatóan a módosítás folyamatában.
- *változtathatóság* arra jellemz, hogy mennyire könny vagy éppen nehéz a kódot változtatni. Ez a tulajdonság szorosan összefügg a stabilitás és az analizálhatóság kérdésével, hiszen ha azok rosszak, akkor a kód változtathatósága is rossz.
- *tesztelhetség* arra jellemz, hogy a szoftver illetve az egyes kódrészek mennyire könnyen tesztelhetek. A szoftverek mködképességét a használatbavétel eltt csak a teszteléssel lehet bizonyítani, ezért minden kódnak tesztelhetnek kell lennie. Elvileg nincs olyan kód, amelyet ne lehetne tesztelni, hiszen (elvileg) akár az egész éles környezetet is fel lehet építeni még egyszer tesztkörnyezetként, és ha máshol nem, de abban a környezetben a programnak mindenképpen futnia kell. A tesztelhetség kérdése igazából az, hogy a szoftverkód tesztelhet-e elfogadható költségekkel. Ennek megfelelen a kódokat gyakran olyan programozási struktúrák alkalmazásával kell elkészíteni, amelyek nem feltétlenül a használhatóságot, hanem a könnyebb (értsd olcsóbb) tesztelhetséget segítik.

Hordozhatóság

A hordozhatóság az utolsó nagy témakör a szoftverminség fogalmi rendszerében. Ez a fogalom arra jellemz, hogy mennyire könny a rendszert installálni, lecserélni, hozzáigazítani a környezethez. Ebben a témakörben a szokásos felmerül fogalmak a

- *installálhatóság*, ami arra jellemz, hogy mennyi energiát, munkát (költséget) igényel a rendszer üzembe helyezése megfelel hardver és hálózati környezetben.
- *lecserélhetség* arra jellemz, hogy a szoftver mennyire könnyen cserélhet le más rendszerre, vagy a rendszer egy újabb verziójára. Ez azért nehéz fogalom, mert általában a lecserélhetségbe bele szoktuk érteni a másik, bevezetend rendszer tulajdonságait is. Amikor azonban egy szoftver minségérl beszélünk, akkor a lecserélhetségrl a helyette alkalmazandó új szoftvertl függetlenül kell gondolnunk, és szét kell választanunk a cserélési folyamatban azokat a tulajdonságokat, amelyek a kiváltandó rendszer *lecserélhetségére* jellemzek a bevezetend rendszer *installálhatósá gára* való jellemzktl. A lecserélhetségre jellemz tipikus tulajdonság az adatok export képessége, például képes-e a rendszer az adatokat szabványos (pl. XML) formátumban exportálni, hogy utána az egyszer segédprogramokkal betölthet legyen az új rendszerbe.
- *adaptálhatóság* arra jellemz, hogy a rendszert mennyire könny hozzáigazítani megváltozó körülményekhez. Ez jelenthet mszaki körülményt, például másik adatbázis-kezelvel kívánjuk használni a rendszert, de jelenthet mködési igény változást is, például jogszabály követés, amely esetben elfordulhat, hogy a rendszer szoftverkódját is módosítani kell.
- *Hordozhatósági megfelelés*, ami nagyon hasonló a funkcionális megfelelés fogalmának, de itt elssorban azt jelenti, hogy a szoftver mennyire felel meg azoknak a nem feltétlenül, st elssorban nem funkcionális követelményeknek, amelyek a hordozhatóságot segítik el. Ilyen tipikusan a használt adatbázis-kezel, alkalmazás szerver, middleware rendszerek szabványoktól eltér specialitásainak elkerülése.

A tesztelés célja

A tesztelés célja, hogy segítse a szoftverminség kontrolljának egyes részeit, összetevit. A tesztelés önmagában nem elegend a szoftverminség garantálásához. Nem lesz jobb egy szoftver ha csak teszteljük, és nem javítjuk ki a hibákat. Ugyanakkor vannak olyan szoftverminségi mutatók is, amelyek nem javíthatóak teszteléssel.

A tesztelés feladata ketts. Egyik feladata, hogy bizonyítsa a szoftver minségét, a másik, hogy segítse a szoftverminség javítását (tipikusan a hibakeresést). Ezek a feladatok az szoftverfejlesztés során a tervezéstl az üzemszer mködésig tartanak. A fejlesztés korai fázisában a hibakeresésen, mint f feladaton van a hangsúly, és kevésbé a helyes mködés bizonyításán. A szoftver fejldése során a fontosság folyamatosan kerül át a hibakeresésrl a helyes mködés bizonyítására, és az átadási fázisra ez a feladat súlyozás teljesen megfordul, és az ügyfél általi átvételi tesztek során szinte nem is számít, hogy az egyes tesztek mennyire segítik a szoftverben esetleg még maradt hibák megkeresését, azon túlmenen, hogy létüket demonstrálhatóan bizonyítják, ha vannak hibák.

Ezzel párhuzamosan a tesztelés elvégzése, mint feladat is átkerül a fejleszt, szállító cégtl az átvev, üzemeltet céghez, intézményhez. A korai fázisban a tesztelés a szállító feladata, azt saját hatáskörben, saját felelsségére és rizikójára végzi. Ebben a fázisban a megrendel nem is vesz részt a tesztelésben, és nem is érintett ezekben a tesztekben azon túlmenen, hogy bizonyos tesztlési módszertant megkövetelhet a szállítótól, mint általános rezsimet a szállító általános minségének ellenrzésére.

A szállítás során az egyes fázisokban a tesztelés folyamatosan átkerül a megrendel felelsségi körébe. Ez teljesen összhangban van az egyes tesztek céljával: minség bizonyítás és hibakeresés. A szállító számára direkt módon nem érték a szoftver minségnek a bizonyítása, csak annyiban, amennyiben az érdekei a megrendel érdekein keresztül realizálódnak. A szállító számára a tesztelés elssorban hibakeresést segít eszköz. A megrendel ugyanakkor nem érdekelt a hibakeresésben, nem is ért hozzá feltétlenül, és nem a feladata és általában az eszközei sincsenek meg hozzá. A megrendel számára a tesztelésben az a fontos, hogy az bizonyítja számára, hogy a szoftver megfelel minség, és így a tesztelés a megrendel számára csökkenti az szoftver esetleges minségi hiányosságaiból adódó üzleti kockázatot.

Az átvételi teszt — mint az egyik utolsó teszt a szállítási fázisban és amely lezárja az elfogadás, és teljesítési fázist — teljes felelssége az megrendelé, még akkor is ha annak kivitelezésében a szállító segíti szakmai tudásával, vagy éppen a rendszer ismeretével a megrendelt. Azonban ebben az esetben is pontosan tudatában kell lennie a megrendel képviselinek, hogy a megrendel és a szállító ebben a fázisban a tesztelés szempontjából részben ellenérdekelt. A szállító érdeke ebben a fázisban a tesztek helyes lefutása, hiszen ez az ami a gazdasági célját, a szoftverszállítás teljesítését segíti, ekkor már függetlenül a szoftver minségétl. A szállító érdeke azonban a szoftver minségének biztosítása, és a szoftver átvétel megtagadása, ha a szoftver nem éri el azt a minségi szintet, amelyet a megrendel megkövetel a szállítótól a szerzdéses keretek között.

Emiatt a megrendel cég vagy intézmény nem ruházhatja át a tesztelés kési fázisait a szállítóra. A tesztelés felelssége még akkor is a megrendeln marad, ha az átvételi tesztek tervezését és technikai kivitelezését a szállítótól független harmadik, szakmailag kompetens félre bízza.

Ahogyan a tesztelés célja a fázisok során eltolódik a hibakeresés fell a bizonyítás felé, ennek megfelelen a tesztelés technológiája, és kivitelezése is eltolódik a technikai, fejleszti szintrl egyre inkább a felhasználói szint felé.

Természetesen a tesztek minden fázisban technikailag megalapozottak kellenek, hogy legyenek, különben nem képesek betölteni a feladatukat, de a fejlesztés korai szakaszában a tesztek megjelenés és a tesztek eredményének a prezentációja részletes és technikai. A fejlesztés késbbi szakaszában a tesztek eredménye átlátható, olvasható, egyszeren értelmezhet kell, hogy legyen. Amíg a hibakeresés a tesztelés f feladata, addig a teszt eredményének minden olyan adatot tartalmaznia kell, amelyik a hibakeresést segíti. Ezért a tesztelés eredménye ebben a fázisban gyakran szöveges naplófájlok formájában jelenik meg, amelyet a fejlesztk használnak információ forrásként. A tesztelés késbbi fázisában ez a teszteredmény forma használhatatlan lenne. Annak eldöntése, hogy egy teszt hibátlanul lefutott, és így bizonyítja-e a szoftver minségének azt a szeletét, amelyet bizonyítani hivatott nem követelheti meg hosszú, és könnyen „elnézhet” naplófájlok vizsgálatát. A teszteket úgy kell kivitelezni, hogy azok eredményének értelmezése, magyarán annak eldöntése, hogy a teszt sikeres volt-e vagy sem szinte emberi hibalehetség nélkül elvégezhet legyen. Így tipikusan egy átvételi teszt futtatási eredménye tipikusan egy „teszt sikeres” vagy „hibás teszt” kiírás, vagy zöld – piros jel egy eredmény lapon.

A tesztek tulajdonságai

A teszteket úgy kell elkészíteni, hogy megfeleljenek a konkrét teszt céljának. Ennek elérésére a teszteknek

- hatásos
- hatékonynak
- költséghatékonynak
- gyorsnak
- definiáltnak
- értékelhetnek
- reprodukálhatónak

kell lenniük.

Amikor egy tesztet megvizsgálunk, akkor hatásos ha a teszt sikeres futtatása nagy valószínséggel bizonyítja, hogy a tesztelt minségi kritérium a szoftver alkalmazása során is megfelel lesz. Ha például tesztelünk egy funkciót, akkor nem teszteljük végig az összes létez bemeneti adattal a teszt rendszert, mert ez fizikai képtelenség. A tesztünk mégis lehet hatásos, ha olyan tesztadat-sort használunk, amelyik tartalmaz mintát a legfontosabb esetekre, és amelyek alapján joggal következtethetünk arra, hogy a szoftver minden bemen adatra megfelelképpen fog viselkedni.

A teszt akkor hatékony, ha nem csak hatásos, de nem fogyaszt feleslegesen erforrásokat. A fenti funkcionális teszt példánál maradva a teszt nem hatékony ha több olyan bemenettel is teszteli a szoftver mködését, amelyek esetében az egyik lefutása esetén nem marad kétség, hogy a többivel is mködni fog a rendszer.

A teszt költséghatékonysága azért is fontos, mert sokszor a tesztelés igényeit figyelembe nem vev tervezés olyan szoftver eredményez, amelyiknek a tesztelése csak nagyon nagy költséggel végezhet el, és emiatt sokszor el is marad. Ennek a következménye, hogy a szoftver hibái csak a szállítási fázist követen a jótállási fázisban derülnek ki, amikor egyrészrl a szoftver már üzemben van és így minden minségi deficit üzleti hátrányt jelent a megrendelnek; másrészrl pedig ebben a fázisban a megrendel jóval kisebb érdekérvényesít ervel rendelkezik a szállítóval szemben.

A szoftverrendszereket úgy kell megtervezni, és kivitelezni, hogy azok a szoftver élettartama során, tehát nem csak a szállítási fázis lezárásáig hanem azt követen is költséghatékonyan tesztelhetk legyenek.

A költséghatékonyságnál nem csak a tényleges kiadásokat kell figyelembe venni, hanem azokat a költségeket is, amelyek sokszor nem kerülnek pénzbeli kifejezésre. Ilyen tipikusan a tesztelésre fordított id, a szükséges koncentráció, a kiértékelés ideje és hibázási lehetség a kiértékelés során.

A teszt sebessége szorosan összefügg a hatékonysággal és a költséghatékonysággal. Az, hogy egy teszt elég gyors-e beleérthet az elz kettbe, de az id, mint speciális, nem többszörözhet erforrás, szükségessé teszi, hogy a teszt sebességét mindenképpen, mint külön kritériumot vegyük figyelembe.

A tesztnek definiáltnak kell lennie. Ez azt jelenti, hogy pontosan el kell írni a teszt futtatásának kritériumait, az elfeltételeket, hogy hogyan kell a tesztet végrehajtani, milyen módon kell a teszt eredményét begyjteni és az eredményt értékelni. Gyakori hiba, hogy nincsenek definiálva a teszt elfeltételei, és emiatt a teszt reprodukálhatósága csökken. Más esetekben a teszt eredményeinek begyjtése nincs definiálva vagy az eredmények értékelése és e miatt a teszt sikeressége vagy sikertelensége egyes esetekben definiálatlan.

A teszt eredményeknek értékelhetnek kell lenniük. Ezt a definiáltsággal összefügg fogalmat attól függetlenül is ki kell emelni, mert nem elegend, hogy a teszt eredménye definiált, ha az nem értékelhet. Definiált például ha egy teszt eredménye egy megabájt méret bináris fájl, amely nem tartalmazhat egy meghatározott szekvenciát. Ezt az eredményt azonban további segéd programok nélkül nem tudja a teszt végrehajtója értékelni, ez az eredmény ilyen módon nem értékelhet. A konkrét esetben a példa azzá tehet, ha a kiértékel program használatát is a teszt részévé tesszük, és definiáljuk, hogy mely programmal, és milyen módon kell elemezni a keletkezett bináris fájl.

Végül és nem utolsó sorban a teszteknek reprodukálhatóaknak kell lenniük. A tesztelés, mint minség bizonyítási eszköz semmit nem ér ha néha mködik egy teszt eset, néha meg nem. A reprodukálhatóság nagymértékben elsegíthet ha a teszt hatásos, definiált és értékelhet, de nem garantálja a teszt reprodukálhatóságát. Lehetnek a tesztelt mködés során olyan küls körülmények, üzemelési paraméterek, amelyek változnak a tesztfuttatások között és amelyek hatással vannak a teszt eredményre, de nem a teszt bemen paraméterei. Ezeket a hatásokat a tesztek elkészítése során amennyire lehet ki kell küszöbölni. Amennyiben a teszt nem reprodukálható, különböz idben látszólag azonos körülmények között a teszt különböz eredményeket ad, akkor az általában instabil, hibás mködésre utal, és a szoftverminségi szint bizonyítására alkalmatlan.

Teszt típusok

A gyakorlatban a szoftverfejlesztés élettartama során különböz teszteket szoktak használni. Ezek a tesztek sokszor egymásra épülnek, és amennyiben valamelyik teszt hibát tár fel, meg kell vizsgálni a hiba kijavítását követen, vagy annak során, hogy kell-e valamilyen tesztet készíteni a hiba ismételt elfordulásnak megakadályozására az alsóbb szinteken.

Ennek az az oka, hogy egy hiba nem csak egyszer fordulhat el. Annak ellenére, hogy a szoftver kódban egy kijavított hibás kód megszüntet egy hibajelenséget, nem biztos, hogy a hibát is kiküszöbölte. Elvileg is csak annyit lehet állítani, hogy bizonyos teszt adatokra, bizonyos inputra már nem jelenik meg a hiba. Ugyanakkor elfordulhat, hogy más adatokra, vagy a kód egy másik részének megváltoztatása esetén a hiba ismételten jelentkezik. Ez tipikusan abban az esetben fordul el, amikor a hiba kijavítása nem kell körültekintéssel, és kellen általánosan történik meg.

Mivel egy hiba kijavítása annál olcsóbb, minél alacsonyabb szint teszt során derül ki a hiba, ezért érdemes a kijavított hibákra is minden szinten teszteket készíteni, kiegészíteni a meglev teszt eseteket.

Az egymás után következ tesztek a fejlesztési fázisoknak megfelelen egyre nagyobb részeket tesztelnek, míg végül a felhasználói elfogadási teszt már az egész rendszert teszteli. Ennek megfelelen azok a tesztek, amelyek csak egy részt tesztelnek, a többi, a teszt által nem vizsgált részt a tesztelési környezettel helyettesítik, szakszóval mock-olják a környezetet. Ebbl a szempontból is érdemes megkülönböztetni a különböz teszteket, az alapján, hogy a tesztelt részek és a mock környezet milyen felületen kapcsolódik egymáshoz. Amennyiben a megrendel, vagy megbízott technikai szakemberei részt vesznek a felhasználói elfogadási tesztet megelz teszteken is, nagyon fontos tisztázni, hogy a mock környezetek közül mi az amit a szállító biztosít, és mi az amit a megrendeli oldalnak kell nyújtania. Amennyiben a szállító biztosítja a mock környezetet vagy annak egy részét a megrendeli oldalnak különös gondot kell fordítania arra, hogy meggyzdjön a mock környezet megfelelségérl.

A fejlesztési és átadási fázisoknak megfelelen szokásos tesztek:

- fejlesztési unit tesztek,
- modul tesztek,
- integrációs tesztek,
- funkcionális tesztek,
- terheléses tesztek,
- shock tesztek,
- tartóssági teszt,
- felhasználói elfogadási tesztek (UAT),
- üzembehelyezési teszt.

Unit tesztek

A unit tesztek a fejlesztési unitokat tesztelik, és ezeket a teszteket a megfelel iparági gyakorlat szerint a fejlesztk készítik el az egyes unitok fejlesztése során. Egy unit tipikusan lehet egy Java class, amihez általában egy teszt class-t készítenek, amelyik futtatása során teszteli a Java class-t. A unit tesztek futtatása általában minden fordítás során megtörténik, és a fordításvezérl rendszerek többnyire a szintaktikai hibával egyenértéknek tekintik, ha egy unit teszt nem fut le sikeresen.

Ideális esetben az egyes unit tesztek függetlenek egymástól. Az éles futtatás során történ egymástól való függséget mock osztályokkal valósítják meg a unit tesztek, amelyek megfelel, küls fél által fejlesztett mock könyvtárak segítségével hozhatók létre.

A unit teszt értelme, hogy az egyes unitokat egyenként, egymástól függetlenül lehet tesztelni, és ezzel bizonyítani, hogy az egyes részek megfelelen mködnek. Sokszor a unit teszt kódja a fejleszti dokumentáció része, amely természetébl adódóan abszolút precízen írja le a unit mködését.

A unit tesztek biztonságossá teszik a unitok módosítását, például a nagyobb teljesítmény kedvéért. A fordítási idben lefutó unit teszt azonnal megmutatja, ha a módosított kód nem felel meg a korábban megírt unit tesztnek, és gyakorlatilag nem sikerül a fordítás, amíg a kód nem megfelel (regressziós teszt).

A unit tesztek esetében gyakran merül fel a kód lefedettség fogalma, ami azt jelenti, hogy az adott unit minden egyes programsora meghívásra került-e a unit teszt futtatása során. Nem feltétlenül követelmény még egy szigorúan szabályzott fejleszti környezetben sem a teljes lefedettség, mert vannak olyan kódok, amelyeket nem érdemes tesztelni. Ilyenek tipikusan a fejleszti környezet által automatikusan generált setter és getter metódusok. Ugyanakkor a teljes lefedettség sem garantálja a kód hibátlanságát. Ha minden egyes kódsor legalább egyszer futott a unit teszt alatt, az még nem garantálja, hogy minden kódsor a lehetséges lefutások minden kombinációjában, a kód összes lehetséges lefutási útvonalán tesztelve lett, és még kevésbé, hogy az összes lehetséges adattal lett tesztelve, ez triviális programok kivételével ugyanis elvileg sem lehetséges.

Unit tesztet akkor tekintünk megfelelnek, ha a unit minden *dokumentált* funkcióját teszteli minden lehetséges bemeneti adatcsoporttal. (Egy adatcsoportba tartozónak tekintünk két bemeneti adathalmazt, ha az várható, hogy az egyik helyes kezelése esetén a unit a másikkal sem fog hibásan futni.) Ennek megfelelen a unit teszt elkészítéséhez valójában nem is szükséges a unit kódjának ismerete, csak a funkciók ismerete, és néha alkalmazott fejlesztési szabály az is, hogy a unit tesztet a unit kódjának megírása eltt készítik el. Ennek következtében nem is a unit tesztre, hanem a kódra lehet jellemz, ha van olyan lényeges kódsor, amit a unit teszt nem fed le nem használ. Az ilyen kódsor valószínleg szükségtelen.

A unit teszt szállítási szempontból a forráskód része, és a megrendel akkor követelheti meg a szállítótól a unit tesztek kódjainak szállítását, ha a forráskód átadása is a szállítás része. Általában a unit teszteket a megrendeli oldalon nem ellenrzik sem tartalmilag, sem pedig teszt futtatás során. A unit tesztek meglétét és minségét a megrendeli oldalnak legfeljebb szúrópróba szeren kell ellenriznie. Ezzel a megrendel nem a kódot, hanem a szállító a minségét ellenrzi.

Modul tesztek

A modul teszteket a QA szállító a szállított, és tesztelésre készre jelentett modulokon végzi, általában mint fekete doboz teszteket. Modulnak olyan egységeket tekintünk ebben az esetben, amelyek üzemeltetési szempontból külön egységként jelennek meg. A felhasználó számára gyakran nem jelennek meg az egyes modulok, nem feltétlenül kell ismernie az egyes modulokat és azok egyedi funkcionalitását. Lehetnek olyan modulok, amelyek nem is biztosítanak egyáltalán végfelhasználói funkcionalitást, csupán más modulok veszik igénybe az általuk nyújtott szolgáltatásokat.

A modulokra jellemz, hogy jól definiált, általában szabványos felületeken keresztül kommunikálnak egymással. A modultesztelés során az ezeken a felületeken keresztül elért vagy nyújtott szolgáltatásokat mock modulokkal helyettesíti a tesztel rendszer. Szabványos felületek esetében erre kész eszközök állnak általában rendelkezésre, saját felületek esetében a teszt mock modulokat ki kell fejleszteni. Ez a fejlesztés a QA folyamatban külön költséget jelenthet amennyiben a szállító nem szállítja a mock modulokat. Mivel a külön fejlesztett mock modulok maguk is lehetnek a tesztelés során hiba források ezért különösen fontos tesztelési szempontból is a szabványos felületek használata.

Különösen érdekes azoknak a moduloknak a tesztelése, amelyek humán felületen keresztül is kommunikálnak, azaz kezeli felületük is van. Ezek a felületek általában nem annyira jól definiáltak, mint a gépi felületek. Ennek az oka, hogy az ember, mint a felületet használó „modul” intelligens, és egészen más módon értelmez egy felületet, mint egy gép. Egy felhasználói felület lehet úgy is jól használható, hogy géppel nehezen tesztelhet, és lehet úgy is nehezen használható, hogy közben pontos definíció szerint épül fel.

A felhasználói felületek gépi teszteléséhez léteznek különböz módszerek, amelyek a tesztelés végrehajtása során az emberi viselkedést (begépelt karakterek, egérrel történ kattintások) ismétlik el, és így a tesztek regressziós futtatása automatizálható. Sok esetben azonban az ilyen tesztek fals hibát jeleznek, például egy megnyomható gomb arrébb kerül, és az automatizált teszt a gomb mellé kattint mert nem veszi észre automatikusan a változást úgy, ahogy egy ember észrevenné. Emiatt az ilyen felületek tesztelése teljes mértékben nem automatizálható.

A tesztek során nem csak a garantált, „szállított” küls felületeket teszteljük, hanem az egyes modulokat külön, egymástól szeparáltan vizsgáljuk, a másik, éppen nem tesztelt modul által nyújtott interfészeket pedig mock szolgáltatásokkal biztosítjuk. Ezzel nem csak a publikus interfészeket teszteli az integrációs teszt, hanem azokat az interfészeket is, amelyekkel az egyes modulok egymás között kommunikálnak. Ez lehetvé teszi a modulok valódi modularitását és kiválthatóságát a késbbiekben a definiált és tesztelt interfészeken keresztül.

Integrációs tesztek

Az integrációs tesztek során a már összekapcsolt modulok mködését teszteljük. Ilyenkor már nem az egyes modulok mködését vizsgáljuk, hanem azt, hogy az egyes modulok valóban együtt tudnak-e egymással mködni, integrálódnak-e. Elvileg ha a modul tesztek nem találtak hibát, akkor az integrációs teszteknek is sikerülniük kell, feltéve, hogy a tesztel mock implementációk, amelyek a modul tesztek során a nem tesztelt modulok által nyújtott felületeket implementálják tökéletesek és minden szempontból megfelelnek a valós környezetnek. Ez azonban a gyakorlatban nm biztosítható, és elvileg sem cél olyan mock modulokat készíteni, amelyek 100%-ig emulálják a modulok viselkedését.

Az integrációs tesztek során kiderül hibák olyan hibák, amelyek egyben jelzik a modul tesztek valamilyen hibáját is, vagy olyan hibák, amelyek még korábban elkövetett terezési hibára utalnak. Az integrációs tesztek során feltárt hibák esetében mindig meg kell vizsgálni, hogy az adott hiba miért nem derült ki a modul tesztelés során, és vagy javítani kell a modulteszten új teszteset felvételével, vagy valamelyik mock modul kódját, vagy konfigurációját kell módosítani a hiba alapján, és megismételni a modultesztet, ellenrizve, hogy a módosított formában felfedezi-e a hibát.

Az integrációs tesztek során kiderül hibák zöme konfigurációs hiba. Ebben az esetben a modulok összekapcsolásának paraméterei a nem megfelelek. Az ilyen hiba kijavítása nem a kódban történik, hanem a teszt rendszere konfigurációjában, és mint ilyen nem is tekinthet a rendszer hibájának. Amennyiben az derül ki, hogy a hiba a rendszer kódjában van, akkor vissza kell lépni a kód kijavításához, a unit tesztek, modul tesztek kibvítéséhez és ezt követen lehet megismételni az integrációs teszteket.

Ezek a visszalépések adminisztráció és idigényesek. Az adminisztráció célja az, hogy minden tesztelés, és kód módosítás követhet legyen, és így a késbb kiderül hibák esetén tudni lehessen, hogy egy új hibáról van-e szó, ami egyszer már fellépett és javítva lett, de valamilyen oknál fogva ismételten fellépett, vagy egyszeren a hibát még nem javított ki fejleszt. Ez könnyen elfordulhat nagyobb lépték fejlesztés során, amikor a modul tesztek során több száz hiba derül ki. Sok esetben az iteratív tesztelési folyamatban nem vár a projekt addig, amíg a modul tesztben feltárt összes hibát kijavítják, hanem elkezdik az integrációs teszteket tudva, hogy lesznek olyan hibák, amelyek definíció szerint a modul tesztekbl örökldnek, és meg kell, hogy jelenjenek.

Az átlapolódó tesztelési fázisok oka, hogy ebben a projekt struktúrában gyorsabban lehet elre haladni, az ismételt tesztelések extra költségén. A tesztek végrehajtása azonban, fleg ha automatizált tesztekrl van szó, nem olyan magas, mint a tesztek kialakításának költsége, ezért általában a projektek során az egyes teszt fázisok átlapolódnak.

Az integrációs tesztek sikeres lefutása esetén bizonyítják, hogy az egyes modulok képesek együttmködni. Ez különösen abban az esetben fontos az egész rendszer tesztelése eltt, ha a különböz modulokat különböz szállítók szállítják. Az integrációs teszt ebben az esetben ki kell, hogy egészüljön olyan monitoringgal és elemzési lehetséggel, amelyik azt tudja vizsgálni, hogy az egyes modulok közötti felületen milyen kommunikáció zajlik. Ebben az esetben ugyanis nagyon fontos, hogy a két modul közötti integráció hiánya melyik modul nem megfelel mködése miatt történik, és ezt a tesztnek mszakilag ki kell tudnia mutatnia vitathatatlanul.

A modulok kapcsolódási felületeinek ellenrzése hálózati figyel eszközökkel, vagy a kommunikációt továbbító és naplózó egyszer proxy alkalmazásokkal lehetséges. Ilyen jelleg forgalom rögzítés az integrációs teszt során egyébként is ajánlott, akkor is ha a teszt értékelésének nem része a naplófájlok elemzése. Ezek akkor lehetnek hasznosak, ha egy késbbi teszt során valamilyen hiba a felszínre kerül, következtetni lehet a hiba okára és helyére ha kideríthet a naplófájlok alapján, hogy a hiba már korábban is meg volt, csak nem vette észre a teszt, vagy pedig a fejlesztések során keletkezett.

Funkcionális teszt

A funkcionális tesztek a legismertebb tesztek. Azt hivatottak eldönteni, hogy a szállított szoftver rendszer helyesen végzi-e a funkcióit, amelyeket elvárnak tle. Tulajdonképpen a unit tesztek, a modul és az integrációs tesztek is funkcionális tesztek, de céljuk az egyedi unit, modul, illetve a modulok kapcsolódásának a tesztelése az integrációs teszt, és nem azon funkcióknak a tesztelése amelyek a megrendel számára fontosak. Úgy is megkülönböztethet a unit, modul és integrációs teszt a funkcionális teszttl, hogy bár az els három is funkciókat tesztel ezek a funkciók elssorban bels funkciók és nem azok, amelyek a felhasználó számára is direkt módon érdekesek. A felhasználó számára fontos, úgynevezett felhasználói funkciók a bels funkciókra épülnek, így ezek mködése is alapvet, de nem a funkcionális teszt vizsgálatának direkt tárgyai.

A funkcionális tesztek lehetnek manuális és automatizált tesztek. Mint minden tesztnek tervezettnek kell lennie, és nem ad-hoc jellegnek. A funkcionális tesztek elvégzése eltt pontosan meg kell határozni, hogy milyen teszt esetek lesznek, és definiálni kell az egyes teszt eseteket. A teszt esetek definiálása során nem csak mszaki, hanem projekt menedzsment paramétereket is figyelembe kell venni. Így tipikusan a funkcionális teszt elkészítése során a tervben a következket kell rögzíteni minden egyes teszt esetre:

- funkció
- súly
- elfeltételek
- elkészítés
- végrehajtás
- dokumentálás
- értékelés

A *funkció* mondja meg, hogy a szoftver rendszer melyik funkciója az, amit a teszt vizsgál. Egy teszt leltár során világosan látni kell, hogy nem maradt-e ki olyan funkció amelyet a megrendel a szállítótól követel, és amelyet teszt nem vizsgál. Amennyiben teszt nem bizonyítja egy funkció mködképességét a funkciót nem lehet biztonsággal mködnek, meglevnek tekinteni.

A teszt *súlya* adja meg, hogy mennyire fontos a teszt sikeres lefutása. Lehetnek olyan teszt esetek, amelyek sikertelen lefutása valamely funkció hiányát, vagy hiányosságát jelzik, de a funkció hiánya, vagy hiányossága nem olyan súlyos, hogy emiatt a projekt ne léphessen a következ fázisba. Tipikusan ilyenek a felhasználói felület ú.n. kozmetikai hibái, amelyek a használhatóságot nem csökkentik, de ergonómiai hatásuk hosszabb távon lehet. Ilyen esetben a rendszer átvehet, mert a használatba vétel, még a csökkent funkcionalitással is nagyobb gazdasági haszonnal kecsegtet, mint egy késbb használatba vett javított rendszer. Természetesen az a szempont sem elhanyagolható, bár nem a tesztelés szigorúan vett hatáskörébe tartozik, hogy a szállító a szállítást követen kijavítja-e ezeket a hibákat szavatossági körben.

Tipikus teszt eset súlyok lehetnek a

- blokkoló
- komoly
- fontos
- apró
- kozmetikai

hibák.

A blokkoló hiba olyan amelyik megakadályozza a rendszer használatát, amíg a rendszerben ilyen súlyos hiba akárcsak egy is van a rendszer nem tudja ellátni a feladatát.

Komoly hiba az, amely mellett még elképzelhet a rendszer üzembe helyezése, de a rendszer gazdasági haszna lényegesen kisebb, mint ha a hiba nem lenne. Ilyen lehet egy fontos, de nem életbe vágó funkció hiánya, vagy egy olyan funkció hiánya, amely a program használata során csak egy késbbi idpontban válik szükségessé, például évváltáskor. Ilyen esetben a szoftver rendszer használatba vehet, de általában a szállítótól a megrendel csak részteljesítést fogad el, és a hiba kijavítása is szigorú határid mellett történik.

Fontos, hogy amennyiben a rendszer átvételre kerül komoly hibával, akkor a szoftverszállítási projektmenedzsmentben a rizikó menedzsment foglalkozzon azzal a kérdéssel, hogy milyen következményei vannak annak, ha a szállító nem szállítja a javítást az elvárt, vagy a projektben kalkulált legvégs határidig.

Fontos hiba az, amelyik nem akadályozza meg a rendszer használatát, és nem csökken a rendszer gazdasági haszna sem, de a használat költségesebb, vagy (ezzel egyenértéken) nehézkesebb, mint ha a hiba nem lenne jelen. A fontos hibák kijavítását joggal várja el a megrendel a szállítótól, de ezek javítása nem feltétlenül az átadás, és a teljesítés során történik.

Apró hibáról beszélünk olyan esetben, amikor a hiba jelenlétével együtt is jól használható a rendszer. Egy hiba kozmetikai akkor ha gyakorlatilag funkció nem sérül csak a felhasználói felületen van olyan eltérés a tervezettl, amely észlelhet.

A teszt esetek súlyának elssorban átvételi tesztek során van szerepe, ahogyan a fentiekben részleteztük is, hogy az átvételre milyen hatással kell lennie a hibáknak a súlyozástól függen. A teszt tervnek, amennyiben az átvételi teszt meg kell határoznia azt is, hogy az egész teszt mikor tekinthet sikeresnek. Természetesen blokkoló hiba egy sem fordulhat el, de komoly, fontos, apró és kozmetikai hibákra is meg lehet és érdemes adni maximális számot, ami tipikusan az összes teszt eseten 5-10%-a.

A teszt elfeltételei olyan feltételek, amelyek nem tartoznak ugyan a teszt elkészítés teendi közé, de szükségesek ahhoz, hogy a tesztet végre lehessen hajtani. Ilyenek a tesztelend rendszer és a mock részek installálása, ami az egész tesztelés, és nem a teszt eset elkészítéséhez tartozik, vagy a megfelel sávszélesség, amennyiben az nem triviális. A teszt tervben azokat az elfeltételeket kell felsorolni, amelyekkel szemben várható, hogy elfordulhat, hogy nem állnak rendelkezésre, és ez különösen akkor fontos ha valamelyik elfeltétel hiánya olyan hibás tesztfutást eredményez ami miatt fals negatív értékelést kaphat a teszt, holott csak az elfeltételek nem voltak adottak.

Az elkészítésben olyan teendket kell felsorolni, amelyek nem a teszt részei, de amelyeket a teszt eset elindítása eltt végre kell hajtani, hogy a tesztelt rendszer a teszt eset végrehajtására megfelel állapotba kerüljön. Ezek a lépések általában minden alkalommal végrehajtandók a teszt eset ismételt futtatása esetén is, és hasonlóan az elfeltételekhez figyelni kell arra, hogy a teszt hibája esetén nem a hibás teszt elkészítés-e a valódi ok.

A végrehajtás részben azt kell leírni, hogy hogyan kell magát a tesztet végrehajtani. A végrehajtás leírásában nagyon precízen, egyértelmen és konkrétan kell fogalmazni, hogy ki legyen zárva a teszt végrehajtásának különböz lehetsége. Amennyiben például várni kell egy eredményre, akkor nem megfelel az olyan megfogalmazás, hogy „kivárható idn belül megjelenik az eredmény az ablakban”. Az, hogy mi a kivárható id a tesztel személyétl, hangulatától is függhet, így a teszt eredménye nem determinált. Helyes megfogalmazás viszont, hogy „10mp-en belül megjelenik az eredmény az ablakban”.

A dokumentálás részben azt kell leírni, hogy a teszt végrehajtása során a teszt jegyzkönyvben milyen adatokat kell rögzíteni, illetve a számítógépes rendszerben vagy a tesztelkörnyezetben keletkezett állományok közül melyeket kell begyjteni, és csatolni a teszthez, mint dokumentumokat.

Az értékelés rész pontosan és precízen kell, hogy definiálja, hogy mikor tekinthet a teszt sikeresnek. Nem létezhet olyan fogalom, hogy egy teszt majdnem sikeres, vagy tulajdonképpen sikeres. Ha olyan értékeléssel találkozunk, hogy a teszt „egyes részei" sikeresen lefutottak, az valószínleg azt jelenti, hogy valójában a teszt esetet szét kellene vágni több tesztre. Egy funkcionális teszt lehet sikeres, vagy sikertelen.

Az átvételi tesztek során a funkcionális teszteket a teszt tervekbl derivált teszt jegyzkönyvekkel kell dokumentálni, ahol minden egyes teszt esetre rögzíteni kell, az elfeltételek ellenrzését és az ellenrzés eredményét, az elkészítés lépéseit, a végrehajtást, a dokumentáció begyjtését, illetve a dokumentációt és végül az értékelést (sikeres, vagy sikertelen).

Az átvétel során ha egy teszt azért nem sikerült, mert voltak megfelelek az elfeltételek, akkor tesztelési szempontból a tesztet sikertelennek kell tekinteni. Így teszt csak akkor kerülhet elfogadásra, ha minden blokkoló súllyal rendelkez teszt eset sikeresen lefutott, és a megadott maximális hibaarányon kívül sikeresen futott minden más súlyú teszt eset.

Terheléses tesztek

A terheléses tesztek elsdleges célja nem a rendszer funkcionális mködésének ellenrzése, hanem a rendszer terhelés alatti viselkedésének vizsgálata. A terheléses tesztek között van a terheléses teszt, a sokk teszt és a tartóssági teszt. Ezek mellett lehetséges más terhelés típusú teszteket is definiálni, de ez az a három alaptípus, amelyik a legfontosabb.

Ezeknél a teszteknél a rendszer teljesítményét mérjük, és ehhez definiálni kell a konkrét rendszerre a teljesítmény fogalmát. Pontosan meg kell határozni, hogy mik azok a mérszámok, amelyeket a rendszer tesztelése során mérni akarunk, és milyen értékeket tudunk elfogadhatónak tekinteni.

Ezek az értékek a terminológia szerint kulcsfontosságú teljesítmény mérk (Key Performance Indicators, KPI), amelyekre a teszt terv minségi értékeket ír el, valamint definíció szeren meghatározza ezek mérésének pontos módszertanát, és azt, hogy az egyes PKI-k egyenként, és összességükben hogyan garantálják a rendszer szolgáltatási szintjeit, amelyet a Service Level Agreement (SLA) megkíván.

Terheléses teszt

A terheléses teszt a rendszer terhelésre adott válaszát vizsgálja. A terheléssel kapcsolatos tesztek körében nem egységes a terminológia, így ide szokták érteni a túlterhelési vagy sokk teszteket, a beégetési (burn) vagy más néven tartóssági teszteket.

A terheléses teszt feladata, hogy ellenrizze, a rendszer képes az elvárt válaszidkön belül nyújtani a funkcionalitását olyan terhelés alatt, amilyen terhelésnek az éles rendszerben lesz maximálisan kitéve.

A terheléses tesztek természetszerleg csak olyan rendszeren végezhetk, amelyek hardver és kapacitás szempontjából megegyeznek az éles rendszerrel, ellenkez esetben a kapott eredmények nem fognak megfelelni az éles rendszernek. Ha a terheléses teszt alatt a rendszer sokkal lassabb, mint azt elvárjuk, de a hardver is kisebb teljesítmény, az még nem jelenti azt, hogy az éles rendszeren sem lesz képes az elvárt teljesítményt hozni. Ugyanakkor fordítva is igaz, ha a terheléses tesztet nagyobb rendszeren végezzük (ezt nem tipikus), mint ami az éles rendszeren áll majd rendelkezésre a sikeres terhelési eredmények nem garantálják, hogy az éles rendszeren is megfelel lesz a teljesítmény.

Amíg a funkcionális jelleg tesztek esetében nem fontos, hogy a rendszer teljesítménye azonos legyen az éles rendszerével, addig a terheléses tesztek esetében ez alapvet. Ez a terheléses teszteket költségessé teszi, és ezért ezeket csak a sikeres funkcionális tesztek lezárását követen, és késbb nem ismételten végzik olyan hardveren, amelyet csak ideiglenesen rendelnek a projekt céljaira.

A terheléses teszt eredménye gyakran nem olyan egyértelm, mint a funkcionális teszteké. A kapott válaszidket statisztikai módszerekkel lehet érdemes elemezni, vizsgálni a válaszidk minimum, maximum értékét, eloszlását. A gyakorlatban sokszor elegend az olyan statisztikai kritérium, mint megadni két három értéket a válaszidkre és megkövetelni, hogy a teszt alatt a válaszok 90%-s, a legkisebb, 97% a második legkisebb stb. kritérium idértéknél kisebb legyen. Gyakran a rendszerrel szemben megkövetelnek maximális válaszidt is, amely túllépése esetén a kérést meg nem válaszoltnak tekintik, és a terhelési kérdések megválaszolási arányára is egy 100% közeli, de annál természetesen kisebb értéket adnak meg.

A terheléses tesztek során a terhelések megfelelnek a rendszer funkcióinak, és általában a tipikus funkciókkal terhelik a rendszert a funkciók közötti olyan eloszlással, amelyik tipikusan várható az éles környezetben is. Ezek a terhelések azonban nem vizsgálnak minden funkciót. És a funkciók helyes eredményeit sem ellenrzi általában a terheléses teszt sem valós idben, sem pedig azt követen. Ennek oka, hogy az ellenrzés valós idben a tesztelt rendszernél lényegesen nagyobb erforrást igényelne, ami rendkívüli, a legtöbb projekt számára financiálisan elviselhetetlen mértékben megnövelné a tesztelés költségét. A teszt lefutását követ idszakban, a naplófájlok vizsgálatából megállapítani a funkcionális megfelelséget pedig gyakran nem lehetséges. Ezért a terheléses tesztek során a funkcionális megfelelséget csak szúrópróba szeren ellenrzik a tesztel rendszerek, illetve azt vizsgálják minden egyes válasz során hogy a visszaadott válasz nem a terhelés következtében általában fellép rendszerhiba.

Sokk teszt

A sokk teszt során a rendszert olyan terheléssel teszteljük, amely rövid ideig sokkal magasabb, mint a maximálisan megkívánt terhelés. A teszt célja annak vizsgálata, hogy a rendszer képes meghibásodás nélkül elviselni az extrém terhelést, és nem történik ennek hatására adatvesztés, vagy más meg nem engedhet hiba, illetve a rendszer képes-e a terhelés lecsökkenését követen automatikusan visszaállni normál üzemre.

Nem kívánalom a rendszerekkel szemben, hogy a sokk idtartama alatt funkcionálisan helyesen mködjenek, csak az, hogy ne történjen olyan változás a rendszerben, amely a normális mködést a késbbiekben gátolja.

A sikeres sokk teszt bizonyítja, hogy a rendszer nem támadható hatékonyan (Denial of Service) DoS támadással, és hasonló terhelés esetén is a lehetségekhez képest automatikusan helyreáll a mködés, és nem történik adatvesztés.

Vannak olyan módon felépített rendszerek, amelyek költséghatékonysági okok miatt eleve úgy vannak tervezve, hogy a nominális terhelésnél nagyobb terhelést nem viselnek el, és összeomlanak. Az ilyen rendszereket általában terhelés elosztó rendszerek, tzfalak védik a túlzott terheléstl. Természetszerleg a sokk tesztek során a tesztelésbe ezeket az elemeket is be kell vonni, és valójában ilyen esetekben a tesztek a terhelés limitáló tzfalak funkcionalitását tesztelik, nem a szállított szoftver rendszerét.

A sokk terheléseknél a következket vizsgáljuk:

- A rendszer a terhelést követen visszaáll a normál mködésre.
- A rendszer a terhelés alatt és azt követen sem enged jogosulatlan hozzáférést.
- A rendszer a terhelést követen nem használ indokolatlan erforrást (memória vagy más erforrás szivárgás jele).

A sokk tesztek kivitelezésénél gondosan oda kell figyelni arra, hogy a rendkívül nagy terhelést milyen módon állítja el a tesztel rendszer, gyakran sok géprl, hálózati elemeken keresztül kell létrehozni a terhelést. Meg kell határozni a teszt tervezése során a sokk mértékét, és azt is, hogy a terhelés milyen sebességgel és emelkedéssel fut fel, mennyi ideig tart és milyen a lefutása. Ezekre az értékekre nincs általános szabály, ezeket a várható, vagy elképzelhet túlterheléseknek, vagy éppen a rendelkezésre álló terhel erforrásoknak, és a tesztelésre fordítható idnek megfelelen szokás beállítani.

A rendes mködésre való visszaállás idejét azonban a megkívánt rendelkezésre állási paraméterek alapján kell megkövetelni, és elre rögzíteni a teszt tervben. Ebben figyelembe kell venni, hogy a rendszer milyen görbe szerint áll vissza a rendes üzemre, milyen szolgáltatási szint érhet el a visszaállás alatt, illetve, hogy mennyi id a rendszert leállítani és újraindítani. Csak olyan visszaállási id fogadható el, amelyik rövidebb, mint az egész rendszer újraindítása (ellenkez esetben nagyobb szolgáltatási szint érhet el a rendszer újraindításával), illetve annál nem sokkal nagyobb abban az esetben ha a visszaállási id alatt is elérhet a szolgáltatás megnövekedett, de használható válaszidkkel. Tipikusan a visszaállási id pár perctl néhány óráig tarthat a rendszer összetettségétl függen.

Tartóssági teszt

A tartóssági teszt hasonló a sokk teszthez, de nem egyszeri kirívóan nagy terhelésnek teszi ki a rendszert, hanem hosszan terheli azt, és azt vizsgálja, hogy a rendszer képes-e hosszú idn keresztül folyamatosan a specifikációs paramétereknek megfelelen mködni.

A tartóssági teszt sikeressége biztosítja, hogy a rendszer hosszú idn keresztül képes ellátni a feladatát, és biztosítani azokat a gazdasági elnyöket, amelyeket a rendszertl a megrendel elvár. A tartóssági teszt két esetben tekinthet sikeresnek. Az egyik, amikor a rendszer semmilyen mködési degradációt nem mutat a teszt alatt. Ebben az esetben a teszt azt mutatja, hogy a tesztelési idszak alatt a rendszer ugyanolyan jól és ugyanolyan jó paraméterekkel mködik, mint közvetlenül indulás után.

A mködési degradáció lehet olyan paraméter változása, amelyet a felhasználók éreznek, tipikusan a rendszer válaszidejének megnövekedése, de lehetnek olyan paraméterek változása is, amelyeket a felhasználók direktben nem tapasztalnak, de mszakilag biztosnak lehet tekinteni, hogy hosszabb távon a felhasználók által érzékelt paraméterek is degradálódnának. Ilyen tipikusan a memória elfolyás, vagy diszk terület betelése, vagy a rendszerben felgyüleml halott szálak, adatbázis kapcsolatok elfogyasztása, és fel nem szabadítása. Ezek a paraméterek általában valamilyen rendszer erforrás elfogyasztását és fel nem szabadítását jelentik.

A másik eset, amikor a tartóssági tesztet sikeresnek kell tekinteni, ha a rendszer ugyan mutat a teszt ideje alatt valamilyen degradációt, ez azonban nem olyan mérték, hogy egy megkövetelt folyamatos üzemelési periódus alatt ne lenne elfogadható. A megkövetelt folyamatos üzemelési periódus az az idtartam, ameddig a rendszernek folyamatosan üzemelnie kell újraindítás, karbantartás nélkül. Ha a rendszer üzemeltetése során az üzletileg megkívánt rendelkezésre állás megengedi, hogy a rendszert naponta újraindítsák, akkor a tartóssági teszt elegend ha egy napig tart. Ha a rendszerben ez alatt van némi degradáció, akkor a rendszert lehetséges napi újraindítással üzemeltetni.

A rendszer természetesen ilyen esetben nem tökéletes. A rendszerben valamilyen hiba van, amely a degradációt okozza, de az üzemelés ezzel a hibával együtt is elfogadható. Ennek ellenére az ilyen hibák kijavítására is törekedni kell akár az átadás, vagy a határidknek és a gazdasági elnyöknek megfelelen a szavatossági idtartam alatt. Ennek oka nem csak az, hogy a rendszerkarbantartás, újraindítás költséget jelent, hanem az is, hogy minden felfedezett és ismert, de be nem határolt hiba rizikót jelent, és nem garantálható, hogy változó üzemelési körülmények között a degradáció nem növekszik, vagy akár nem akadályozza meg teljesen a rendszer funkcionalitását (teljes leállás, akár adatvesztés).

A be nem határolt hibák veszélye nem csak a tartóssági teszteknél létezik, de ez az a tesztelés, amelyiknél a legnagyobb súllyal kell figyelembe venni az ilyen jelenségeket. A funkcionális, terhelési és sokk teszteknél fellép hibák oka általában könnyebben behatárolható, és akár javítható a hiba vagy, becsülhet a hiba mködésre való kihatása. A tartóssági teszteknél sokkal gyakrabban fordul el, hogy a hiba jelenség nem drasztikus, nem egy adott funkcióra és ennek egy adott kódrészletre specifikus és emiatt nehezen behatárolható a hiba helye.

A tartóssági tesztek azok, amelyek a legtovább tartanak, hiszen ha egy rendszernek képesnek kell lennie folyamatosan üzemelni egy hónapon keresztül, akkor a tartóssági teszt sem lehet ennél rövidebb. Ez azért fontos kérdés, mert ez a kritérium nem feltétlenül az a követelmény, amelyet a megrendel a szállítóval szemben megfogalmaz. A megrendeli követelmények megfogalmazása során gyakran olyan kitételek szerepelnek, mint „a rendszernek képesnek kell lennie folyamatosan üzemelnie”, amibl elvileg végtelen hosszú tartóssági teszt következne. Valójában a tartóssági teszt hosszát az üzleti és üzemeltetési követelmények határozzák meg, amelyek nem feltétlenül jelennek meg direktben a szállító felé követelményként. Ezért a megrendeli oldalon a tartóssági teszt hosszát szokták sokszor követelményként feltüntetni.

Felhasználói elfogadási teszt

A felhasználói elfogadási teszt a hivatalos átadási, átvételi teszt. Erre a tesztre akkor kerül sor, amikor a szállító a többi tesztet már lefuttatta, és úgy ítéli meg, hogy a rendszer alkalmas az átadásra, nincsen benne olyan hiba, amelyik megakadályozza a teljesítést.

A felhasználói elfogadási teszt (User Acceptance Test, vagy UAT) elfogadója a megrendel, és nem a felhasználó. Sok esetben ez a két személy ugyanaz, de nagyobb rendszereknél nem a felhasználó az aki a rendszer gazdasági elnyeiért felel, csak kezeli, és ezért az elfogadás felelssége gazdasági értelemben a szállítás igazolása sem az feladata. Mégis ez a terminológia terjedt el, mert ugyan nem a felhasználó a felels, de mégis az a személy aki ezeken a teszteken, mint a megrendel a tesztek végrehajtására delegált képviselje részt vesz.

Sokszor az UAT-ot csak mint funkcionális tesztet tekintik,és valóban a leggyakrabban az UAT tényleges végrehajtása során csak a funkcionális tesztet hajtják lépésrl lépésre végig. Ennek az oka, hogy a terheléses tesztek ismételt végrehajtása nagyon költséges lenne, és nem is járna mszaki elnyökkel. A terheléses teszteket általában a megrendel képviseljének, vagy képviselinek jelenlétében és felügyeletével hajtják végre, és ezek elfogadott jegyzkönyve az UAT része.

Az UAT funkcionális teszt végrehajtása során a megrendel és a szállító a teszteket együtt hajtják végre olyan módon, hogy a teszt végrehajtás ellenrzéséért és a teszteredmények kiértékeléséért a megrendel a felels. Az UAT minden esetben formális jegyzkönyvvel rögzített és dokumentált.

Üzembehelyezési teszt

Az üzembehelyezési teszt az éles rendszeren történik és célja annak ellenrzése, hogy a rendszer az éles környezetben is megfelelen mködik, nem történtek installációs, konfigurációs hibák.

Ezt a tesztet sokszor nem tekintik tesztnek, hanem az üzembehelyezés részének. Mégis érdemes a tesztek közé sorolni és úgy kezelni, mint a korábbi teszteket, mert ugyanolyan precízen és hasonló elvek mentén kell végrehajtani, és ellenrizni, mint a funkcionális teszteket.

Az üzembehelyezést követen a rendszer elkezd mködni, és éles inputokkal mködik. Üzletileg kritikus rendszereknél az üzemelés során is folyamatosan figyelni kell, hogy a rendszer mködik-e, és ez nem csak az üzletileg kritikus rendszereknél hanem minden rendszernél különösen fontos az éles indulást követen.

Arról meggyzdni, hogy a rendszer helyesen mködik csak úgy lehet, ha definiáljuk a helyes mködés kritériumait, és azt, hogy ezeket milyen módon fogjuk mérni, a mérési eredményeket kiértékelni.

Az üzembehelyezési teszthez ugyanazokat az SLA-nak megfelel PKI-kat kell mérni, mint amiket az üzemelés során, de ezeket ki lehet egészíteni további indikátorokkal, amelyek mérése nem üzemeltetési cél a folyamatos üzem során, de a beüzemelés során fontos információt adhat a rendszer állapotairól.

Az üzembehelyezési teszt tervezése és végrehajtása során, mivel ekkor már éles rendszer üzemel, különösen gondosan kell ügyelni arra, hogy ne legyenek olyan teszt lépések, amelyek a rendszer környezetét, azokat a rendszereket, amelyeket üzemszeren használnak a felhasználók, feleslegesen terhelik, esetleg mködésüket gátolják.

Teszt tervek formái

A teszt tervek elkészítése a rendszer követelmény specifikációja alapján történik. Ennek megjelenési formája az iparági gyakorlatban általában CRC kártyákat, követelmény diagrammokat vagy más UML leírást jelentenek.

A tesztesetek kidolgozása során minden egyes teszt eset definiálja, hogy mely követelményt milyen mértékben fed le a teszt futtatása. Ennek nyilvántartása UML szerkeszt eszközben történhet, amely biztosítja, hogy tervezés során bármikor lekérdezhet a követelmények tesztek által történ lefedettsége, valamint minden teszthez lekérdezhet, hogy mely követelményeket tesztel, és fordítva minden követelményhez meghatározható, hogy mely tesztek azok, amelyek biztosítják a követelmény teljesülését.

Ezzel a követelmények változása során is folyamatosan nyilvántartható, hogy mely tesztek azok, amelyeket módosítani kell, mely tesztek váltak esetleg feleslegessé, és milyen új teszteket kell tervezni a követelmények változott halmazának lefedettségéhez.

Sokszor a teszt tervek elkészítéséhez szövegszerkesztt, Wiki rendszert használnak. Ez utóbbinak az lehet az elnye, hogy vannak olyan Wiki alapú rendszerek, amelyek a formális tesztleírásokat automatikusan végre is tudják hajtani, így a tesztek automatizálása a teszt definíciókból indul ki minimalizálva az emberi hiba lehetségét.