Távközlési szoftverek tesztelése

DIBUZ SAROLTA

Ericsson Magyarország Kft., K+F Igazgatóság
sarolta.dibuz@ericsson.com

Kulcsszavak: szoftverfejlesztés életciklusa, funkcionális tesztelés, teljesítmény-tesztelés, tesztelés-automatizálás

Ebben a cikkben röviden összefoglaljuk a szoftverek, különösképpen a távközlési szoftverek tesztelésének, minőségbiztosításának folyamatát. Foglalkozunk a tesztelés szervezési részével, valamint a tesztelés automatizálásával.

1. Bevezetés

A szoftverek (nemcsak a távközlési szoftverek) komoly tesztelési folyamaton kell átessenek, mielőtt a felhasználóhoz jutnak. Tapasztalati tény, hogy a távközlési szoftverek fejlesztési költségeinek több mint 50%-át a tesztelés költségei teszik ki.

Egy hiba kijavítása annál olcsóbb, minél hamarabb találják meg a hibát a tesztelési folyamatban. Általános tapasztalat az, hogy ha egy tesztelő szakember talál meg egy hibát, akkor tízszer annyi idő illetve költség azt kijavítani, mintha maga a fejlesztő találná meg. Ilyenkor már többen vannak a folyamatban: a tesztelést végző szakember és a fejlesztő, esetleg nem is az, aki a kódot írta, hanem valaki más. A kód írása már régebben történt, és már nem emlékszik pontosan a fejlesztő, hogy hogyan is működik az adott kódrészlet, mindez drágítja a hibajavítást. Ehhez képest is tízszeres a költsége azon hibák kijavításának, amelyeket már egy eladott szoftver termékben a felhasználó talál meg. Ekkorra még több idő telik el a fejlesztés óta, és a teljes szoftverrendszerben, ami több elemből állhat, nagyon nehéz megtalálni azt, hogy melyik elem melyik modulja okozta a hibát. Ekkor már nem azok foglalkoznak a szoftverrel, akik írták, hanem az üzembe helyezők és a szoftverkarbantartást végzők. Kódolási hibák javítását persze kódolók végzik, de a legtöbb esetben nem az, aki eredetileg is írta a kódot, hiszen lehet, hogy már más feladatot kapott azóta.

A következőkben azt mutatjuk be, hogy a távközlési szoftverek tesztelésekor milyen feladatokkal állunk szemben. A második fejezet arról szól, hogy hogyan kell előkészíteni a tesztelést a szoftver fejlesztő projektekben. A harmadik fejezet a tesztelés helyét és szerepét mutatja be a szoftverfejlesztés folyamatában. Végül a teszt automatizálásról szólunk, amely nagy mértékben javítja a tesztelés hatékonyságát a projektekben.

2. Amit a tesztelésről tudni kell

Teszteléssel csak a hibákat tudjuk megtalálni, jobb nem lesz tőle a szoftverünk. Ezért nagyon fontos, hogy gondos rendszertervezés előzze meg a fejlesztést, aminek része a tesztelés gondos megtervezése is. A szoftverek fejlesztése során a tesztelés gyakorlatilag a rendszer tervezésével párhuzamosan elkezdődik, hiszen a tesztelést is meg kell tervezni, s még ha automatikus teszteket is készítenek, még több idő szükséges a tesztek előállításához. Ezért aztán időben hozzá kell látni hogy a tesztek rendelkezésre álljanak amikor végre kell hajtani azokat.

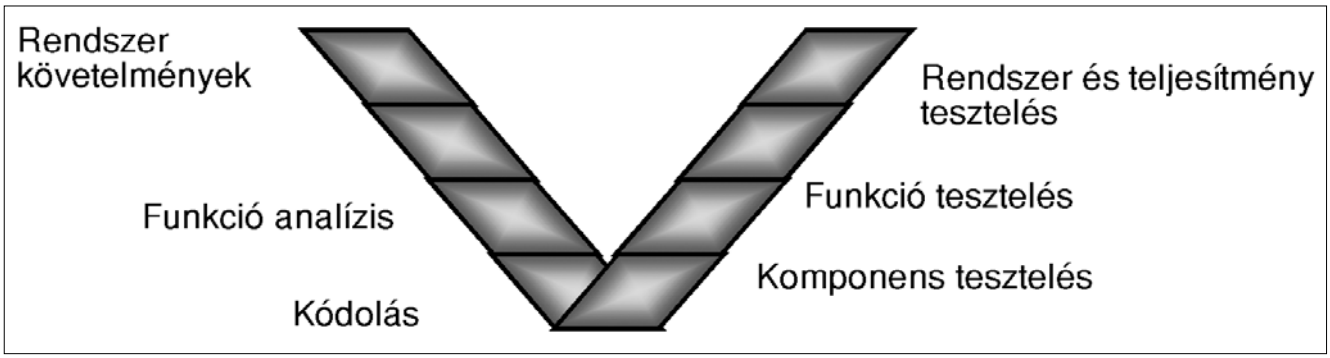
A tesztek tervezése során meghatározzák, hogy milyen eszközökkel, és milyen módon fognak tesztelni. Ezenkívül összegyűjtik a teszteseteket, tehát meghatározzák, hogy milyen tesztfeladatokat fognak elvégezni. A tesztek végrehajtásának első lépése a tesztkörnyezet felállítása. Ezután következik a tesztfeladatok végrehajtása. Ezek eredményeit kiértékelik, ehhez fontos a tesztek lezajlását mutató logok analizálása.

Megkülönböztetünk statikus és dinamikus tesztelést. Statikus tesztelés valójában a kód vagy kódmódosítás alapos átnézését, ellenőrzését jelenti. Ezt mindig más kódoló végzi, mint aki a kódot írta. Dinamikus tesztelés során már működésbe hozzák a szoftvert, amit tesztelnek. Gerjesztik a bemenetén és figyelik a válaszokat a kimeneten.

Beszélhetünk fehér doboz és fekete doboz tesztelésről is. Fehér doboz tesztelés során kihasználjuk azt, hogy látjuk a kódot, ismerjük annak felépítését, struktúráját. Fekete doboz tesztelés esetén csak a tesztelt szoftver külső viselkedése ismert. Ekkor szükséges a dinamikus tesztelés, mely során az interfészein keresztül gerjesztik a szoftvert és ellenőrzik, hogy a gerjesztés alapján a várt, helyes választ kapják-e.

3. A tesztelés helye a szoftverfejlesztésben

A távközlésben használt szoftverek nagyon nagy méretűek. Fejlesztésük modulonként történik, később a modulokat összeépítik, és így áll elő az egyre nagyobb méretű kód. A tesztelést is érdemes fázisokra bontani, a szoftverfejlesztési folyamatának megfelelően. A különböző teszt fázisokat mutatja az *1.ábra*.



1. ábra A szoftverfejlesztés és tesztelés kapcsolata

Már annak is ellenőriznie kell a kód helyességét, aki a kódot írta. Ha fejlesztő is talál hibákat, akkor nyilván saját maga javítja is ki a legkönnyebben.

Az első tesztfázis a fejlesztés során az elkészült modulok tesztelése, ezt nevezik modul- vagy komponenstesztnek. Ennek során a kódoló olyan teszteket végez el, amellyel ellenőrzi az általa írt kódot függetlenül a rendszer többi részétől. Ezek a tesztek elsősorban fehér doboz tesztek.

Amikor a modulokat összerakják, integrálják, további tesztek következnek. Ilyenkor már a kódolóktól független teszterek ellenőrizik, hogy a kód, illetve a több modulból összeállított rendszer megfelelően működike. Eközben valamennyi új funkcionalitást ellenőriznek tipikusan fekete doboz megközelítéssel. Általában nemcsak helyes adatokkal gerjesztik a rendszert, hanem azt is megnézik, hogy hibás bemenetekre vagy helyzetekre hogyan reagál, hogyan tűri ezeket.

Ha már leellenőriztük, hogy a rendszer tudja azokat a funkciókat, amiket a fejlesztés során meg kívántunk valósítani, még azt is meg kell nézni, hogy a rendszer hogyan fog viselkedni várhatóan a valóságos környezetében: bírja-e majd azokat a forgalmi terhelési viszonyokat, ami elvárható, milyen teljesítménnyel fogja végrehajtani a funkciókat.

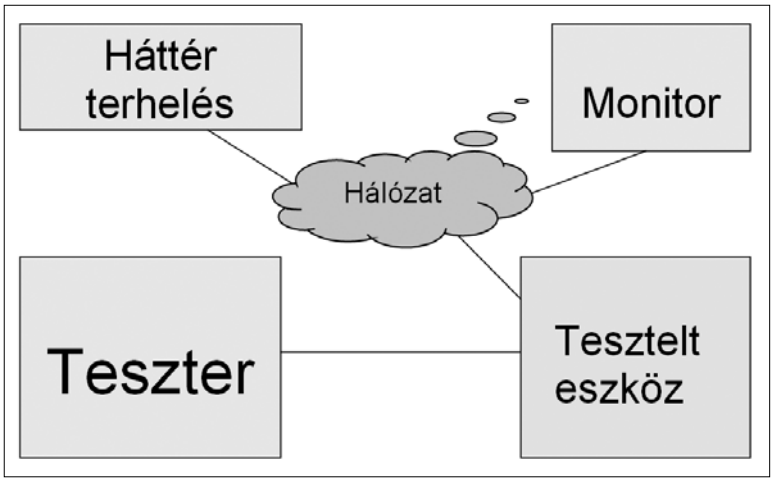
Ez a teljesítmény tesztelés feladata *(2. ábra)*, ahol a tesztelést végző eszközön kívül a tesztkonfiguráció gyakran tartalmaz háttérforgalmat generáló eszközt is, ezzel biztosítva azt, hogy különböző forgalmi helyzetekre el tudjuk végezni a mérést.

Távközlési szoftverek esetében mindig fontos a szabványos interfészek ellenőrzése és az, hogy együtt tude működni a rendszerünk más gyártók hasonló célra fejlesztett szoftverével. Ezt a konformancia és az együttműködési tesztek során érik el. Erről e szám egy másik cikkében olvashatunk részletesen.

A termékek új verzióit úgy alakítják ki, hogy módosítják, kiegészítik újabb modulokkal, elemekkel, amik az új funkcionalitást valósítják meg a szoftverben. Ilyenkor nem elég az új működést tesztelni, hogy az jól került-e megvalósításra, hanem azt is ellenőrizni kell, hogy nem rontottunk-e el valamit a rendszer régi működéséből. Ezt nevezik regresszió-tesztelésnek.

A regresszió-tesztelés tulajdonképpen minden olyan esetben nagyon hasznos, ha már letesztelt, ellenőrzött szoftverhez újabb szoftver részeket adunk, akár a termék új verziójának készítése, akár inkrementális módon történő szoftverfejlesztés során. Az inkrementális fejlesztés azt jelenti, hogy arra törekszünk a szoftverfejlesztés során, hogy a sok új és módosított modulból nem egyszerre hozzuk létre az új szoftvert, hanem egyszerre csak kisebb új részeket adunk a szoftverhez, és ezután ellenőrizzük, hogy az így keletkezett szoftver jól működik-e. Csak ezután következhet a következő modul integrálása. Ezzel érjük el, hogy nem egyszerre kell a nagy mennyiségű új szoftverben megtalálnunk, hogy hol a hiba a tesztelés során, hanem mindig abban a részben kell csak a hibát keresni, amit újonnan tettünk a kódba, hiszen előtte már jól működött. Ennek a logikus megközelítésnek az a hátránya, hogy gyakran kell lefuttatnunk szinte ugyanazokat a teszteket, azaz regresszió-tesztelni. A regresszió-teszt hatékonysága érdekében érdemes ezeket a teszteket automatizálni.

2. ábra Tesztkonfiguráció teljesítmény tesztelésre



4. Teszt-automatizálás

A tesztelés automatizálása azt jelenti, hogy automatikusan hajtjuk végre a tesztelést, és automatikusan értékelődik ki a tesztelés helyessége. Ehhez egy tesztelést végrehajtó eszközre, szoftverre van szükség, valamint arra, hogy meghatározzuk, előre definiáljuk, hogy hogyan tesztelünk, azaz elkészítsük a

tesztsorozatot. Így pontosan átgondolt, alapos teszteket lehet létrehozni, amiket emberi beavatkozás nélkül sokszor, gyorsan végre lehet hajtani. Ezek a tesztek akár éjjel is futhatnak, és a kiértékelésük is igen hatékony, általában szintén automatikus. Ezzel nemcsak hogy sok emberi munkát spórolunk meg, hanem ismétlődő, unalmas emberi munkát, hiszen ugyanazokat a teszteket többször kellene végrehajtani. A tesztek automatizálásához alapvetően új gondolkodási módra van szükség. Hiszen ehhez nemcsak a tesztelt rendszer működését kell megérteni, hanem a tesztelő rendszer működését is. A tesztsorozatok írásához pedig programozói ismeretekre is szükség van, ami nem feltétlenül szükséges a hagyományos teszteléshez.

A teszt-automatizálás teljesen új szemléletet hoz be a tesztelésébe. A tesztelés előkészítése során tovább tart az automatikus tesztek elkészítése, hiszen az automatikusan végrehajtható tesztprogramokat el kell készíteni.Viszont a teszt végrehajtása nagyon gyors. Ennek megfelelően kell tervezni a teszt-projektet. Azt is figyelembe kell venni, hogy bár a tesztek kiértékelése is automatikusan történik, mégis nagyon fontos, hogy a tesztelést felügyelő teszter értse a tesztrendszer és a tesztsorozat működését is, nem csak a tesztelt szoftverét. A tesztprogramok megírásához a teszteréknek is inkább programozói feladatokat kell ellátniuk. Azonban az egyre bonyolultabbá váló távközlési szoftverek tesztelésében már nem képzelhető el a megfelelő tesztek végrehajtása automatizálás nélkül.

5. Összefoglalás

A cikk röviden ismerteti a szoftvertesztelés fontosságát, szerepét és helyét a szoftver fejlesztésben, és hogy miért fontos a tesztelés automatizálását bevezetni a szoftverfejlesztő projektekben. Ez a tesztelésben alapvető szemléletváltással járó lépés, a tesztelés automatizálása elkerülhetetlen.

Ezt ma már nem kérdőjelezik meg egyetlen szoftverfejlesztő projektben sem. Azt azonban, hogy mely teszteket érdemes illetve lehet automatizálni, és milyen módon, mindig az adott fejlesztési környezet és a szoftverrel szemben támasztott követelmények döntik el. A projektek tervezésében figyelembe veszik a teszt automatizálásának igényeit, és azt is, amilyen lehetőségeket biztosít az automatikus tesztelés a projektben. A szoftvertesztelés módszere folyamatos átalakuláson megy át az automatizálás bevezetése miatt. További kérdéseket is felvet a módszer, például, hogy hogyan ellenőrizhető az automatikus teszteléshez fejlesztett tesztszoftver helyes működése.

Ha a távközlésben használt szoftvereink egyre bonyolultabbá válnak, akkor az is elmondható, hogy az ezeket automatikusan tesztelő szoftverek még bonyolultabbak lesznek. Így egy további kérdés, hogy az automatizált tesztelésben használt szoftvereinket hogyan tudjuk minél jobban felhasználni újabb projektekben.

Hírek

Magyar fejlesztésű szoftvert alkalmaz a Cisco a hálózati beléptetési rendszerében

A Cisco Systems az EagleEyeOS™ magyar fejlesztésű, adatlopás elleni EagleEyeOS Professional biztonsági szoftverét alkalmazza Network Admission Control (NAC) hálózatbiztonsági programjában.

A Cisco NAC programja a kliensoldali biztonsági megoldások hálózati szintű ellenőrzését biztosítja, és elsősorban a hálózati férgek és vírusfenyegetések által okozott károk eshetőségét korlátozza. A NAC automatikusan érvényesíti a vállalati biztonsági házirendet az összes NAC kompatíbilis eszközön, és csak az informatikai biztonságért felelős rendszergazdák által beállított biztonsági házirendeknek megfelelő – például a legfrissebb telepített vírusvédelmi szoftvert futtató – kliensgépek, illetve végberendezések számára engedélyezi a hozzáférést.

Az EagleEyeOS Professional szoftverének alkalmazása révén a védelem újabb dimenziója, a belső adatlopás elhárítása valósul meg a NAC program használói számára. A szoftverrel többek között nyomon követhető és szabályozható a dokumentumok vándorlása, módosítása, sőt maguknak a külső eszközöknek a mozgása is.

A két szoftver együttes hatékonyságának legszemléletesebb példája a NAC rendszer védelmi, és a EagleEyeOS Zone funkciójának együttműködése. Amennyiben a NAC-ot és EagleEyeOS-t használó hálózathoz olyan eszköz csatlakozik, amelyen egyik program sem fut, akkor alapesetben a NAC kizárja azt. Ezzel szemben ha a NAC-ot és EagleEyeOS-t egyaránt használó gépet más hálózatra csatlakoztatják, a gép védelmét a továbbiakban az EagleEyeOS látja el, azaz lezárja a gép védett zónáját az idegen hálózat egyéb elemei elől. A NAC alapszabályainak megfelelően az integrált EagleEyeOS esetében is jelentős szerepet kap majd a céges biztonsági házirend: ha olyan gép csatlakozik a hálózathoz, amelyre telepítettek EagleEyeOS-t, de eltérő szabályrendszer fut rajta, akkor azt a NAC felismeri és kizárja a hálózatból.