2.

Egy desktop ügyviteli szoftver tesztelését kell megvalósítani. Készítse el a tesztelési tervet, tegyen javaslatot a tesztelési környezet kialakítására!

– Definiálja a tesztelés fogalmát és jellemzőit!
– Határozza meg a szükséges teszttípusokat!
– Határozza meg a tesztelés szintjeit!
– Jellemezze a tesztelési módszereket!
– Definiáljon egy teljes tesztelési környezetet!
Kulcsszavak, fogalmak (a vizsgakérdést tartalmazó lapon ezek nem szerepelnek, de a vizsgáztatók lapján igen, tehát kérdezhetik őket**):**
– Validáció és verifikáció.
– Tesztelés szintjei: komponens teszt, modul teszt, integrációs teszt (alrendszer, rendszer teszt), elfogadási teszt.
– Tesztelési módszerek: statikus és dinamikus tesztelés, fekete doboz, fehér doboz.
– Szoftvertesztelés módszertana és folyamata.
– Tesztelési vezérlevek.
– Rendszertesztelés, integrációs tesztelés, végtesztelés.
– Teljesítménytesztek (volumen, stressz teszt).
– Automatikus tesztelési eszközök (pl. JUnit, NUnit, xUnit).
– Tesztvezérelt fejlesztés (napi build, release)**.**
A tételhez segédeszköz nem használható.

A szoftver tesztelés

Egy szoftver készítése során több különböző fázison megy keresztül. Az egyes fázisok után, de kitüntetetten bizonyos fázisokban elengedhetetlen a tesztelés folyamata. Ennek oka nagyon egyszerű. A fejlesztés folyamata megköveteli az emberi intelligenciát, és mivel az ember könnyen hibázik, szükségszerű, hogy az egyes fázisok elkészülte után valamilyen ellenőrzést hajtsunk végre a terven, kódrészleten, stb. Ennek hiányában a készülő szoftver valószínűleg tele lesz hibákkal, így a megrendelő nem fogja elfogadni.

Számos technika alakult ki a tesztelés folyamatának segítésére. A következőkben általánosan áttekintjük ezt a folyamatot.

Verifikáció és validáció

A verifikáció és a validációt (V&V) általánosan szoftvervalidációnak nevezik. Legfőbb célja, hogy megmutassa, a rendszer konform a saját specifikációjával, és hogy a rendszer megfelel a rendszert megvásárló ügyfél elvárásainak. Ez olyan ellenőrzési folyamatokat foglal magában, mint például szemléket, felülvizsgálatokat a szoftverfolyamat minden egyes szakaszában a felhasználói követelmények meghatározásától kezdve egészen a program kifejlesztéséig. Röviden tehát:

1. Verifikáció: a terméket jól készítetjük el?
2. Validáció: a megfelelő terméket készítetjük el?

A verifikáció magába foglalja annak ellenőrzését, hogy a szoftver megfelel-e a specifikációnak, azaz eleget tesz-e a funkcionális és nem funkcionális követelményeknek. Általánosabban: a szoftver megfelel-e a vásárló elvárásainak.

A validáció ennél kicsit általánosabb fogalom. Végcélja az, hogy megbizonyosodjunk arról, hogy a szoftverrendszer „megfelel-e a célnak”. Azaz teljesül-e a vásárló elvárása, amibe beleértendőek olyan nem funkcionális tulajdonságok is, mint a hatékonyság, hibatűrés, erőforrásigény.

A V&V két, egymást kiegészítő különböző perspektíva segítségével végzi az ellenőrzési folyamatot:

1. **Statikus**: szoftverátvizsgálások. Olyan technikák, melyek kimondottan csak a rendszer reprezentációját elemzik. Ilyen a követelmény dokumentum, a tervek, és forráskódok.

2. **Dinamikus**: a klasszikus értelemben vett szoftvertesztelés. Csakis az implementáció fázisában végezhető el. Valamely tesztadatok segítségével ellenőrzi, hogy a rendszer adott bemenetre a megfelelő kimenetet nyújt-e.

Az átvizsgálási technikák közé tartoznak a programátvizsgálások, az automatizált forráskód elemzés és formális verifikáció. A rendszert csak akkor tesztelhetjük, ha elkészült egy végrehajtható változatának prototípusa. Az inkrementális fejlesztés előnye, hogy a különböző inkremensek külön tesztelhetők, és a korábban elkészülő egyes funkciók már hamarabb tesztelhetők.

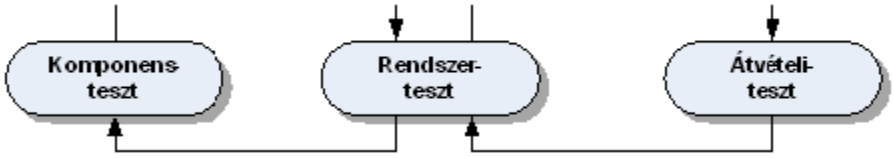
Ne felejtsük el, hogy a tesztelési folyamat önmagában véve nagyon általános. A klasszikus értelemben vett „futtatom a programot és megvizsgálom jó lett-e az eredmény" csak egy részét képezi a teljes folyamatnak. A tapasztalatok szerint az átvizsgálási folyamat a szoftverfejlesztés minden olyan lépésében használható és alkalmazni is kell, ahol már rendelkezésre áll valamilyen kézzel fogható, olvasható reprezentáció a rendszerről. Ez általában a követelmények feltárása fázisban a követelmények validálásával kezdődik és egészen a végleges rendszer leszállításáig terjed. Kijelenthető, hogy gyakorlatilag minden részfolyamat validációja szükséges.

A tesztelés folyamata általánosan

A kis programok kivételével a rendszerek nem tesztelhetők magukban, mint monolitikus egységek. Egy elkészült rendszer már túl komplex ehhez, ezért fontos, hogy a tesztelési folyamatot a fejlesztési modelltől függően lehetőleg minél kisebb komponensek vagy egységek szintjére szorítsuk le. Ez azt jelenti, hogy minden elkészült komponensnek, inkrementális fejlesztés esetén pedig minden inkremensnek rendelkeznie kell a működésüket igazoló tesztekkel. A fejlesztés során az elkészült részegységek így önmagukban tesztelhetők lesznek, ez pedig a hibák jobb felderíthetőségéhez vezet. Az elkészült komponensek integrálása során új funkciókkal bővül a rendszer. Minden integrációs lépés azonban további teszteket von maga után, nem szabad megelégedni a komponensek külön-külön működő tesztjeivel. Az integráció nem várt hatásokat eredményezhet, melyeket csak szisztematikus teszteléssel deríthetünk fel.

A következő ábra egy háromlépéses tesztelési folyamatot mutat be, ahol teszteljük a rendszer komponenseit, majd az integrált rendszert, és végezetül a teljes rendszert a megrendelő adataival.

A tesztelési folyamat



A programban felderített hibákat természetesen ki kell javítani. Ez olyan következménnyel járhat, hogy a tesztelési folyamat egyéb szakaszait is meg kell ismételni. Ha a programkomponensekben található hibák az integrációs tesztelés alatt látnak napvilágot, akkor a folyamat iteratív, a későbbi szakaszokban nyert információk visszacsatolandók a folyamat korábbi szakaszaiba.

A tesztelési folyamat szakaszai:

1. **Komponens (vagy egység) tesztelése**: az egyedi komponenseket tesztelni kell, és biztosítani kell tökéletes működésüket. Minden egyes komponenst az egyéb rendszerkomponensektől függetlenül kell tesztelni.
2. **Rendszer tesztelése:** a komponensek integrált egysége alkotja a teljes rendszert. Ez a folyamat az alrendszerek és interfészeik közötti előre nem várt kölcsönhatásokból adódó hibák megtalálásával foglalkozik. Ezen túl érinti a validációt is, vagyis hogy a rendszer eleget tesz-e a funkcionális és nem funkcionális követelményeknek és az eredendő rendszertulajdonságoknak.

3. **Átvételi tesztelés:** ez a tesztelési folyamat legutolsó fázisa a rendszer használata előtt. A rendszert ilyenkor a megrendelő adataival kell tesztelni. Ez olyan hiányosságokat vethet fel amelyek más esetben nem derülnének ki.

Az átvételi tesztelést **alfa-tesztelés**nek is szokták nevezni. Az alfa-tesztelési folyamatot addig kell folytatni, amíg a rendszerfejlesztő és a kliens egyet nem ért abban, hogy a leszállított rendszer a rendszerkövetelményeknek megfelelő.

Amikor egy rendszer, mint szoftvertermék piacra kerül, gyakran egy másik tesztelés is végbemegy, amelyet **béta-tesztelés**nek nevezünk. A béta-tesztelés magában foglalja a rendszer számos potenciális felhasználójához történő leszállítását, akikkel megegyezés történt a rendszer használatára, és ők jelentik a rendszerrel kapcsolatos problémáikat a rendszerfejlesztőknek. Ezáltal a rendszer valódi használatba kerül, és számos olyan hiba válik felfedezhetővé, amelyeket a rendszer építői esetleg nem láthattak előre. Ez után a visszacsatolás után a rendszer módosítható és kiadható további béta-tesztelőknek, vagy akár általános használatra is.

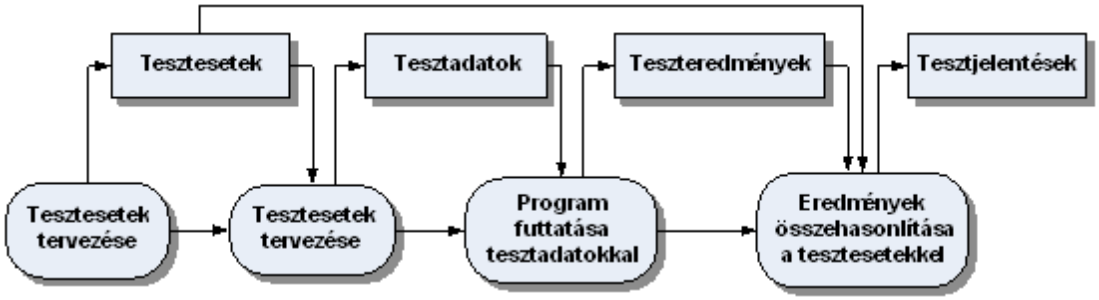
A dinamikus tesztelés

A szoftvertesztelés, mint dinamikus V&V a gyakorlatban inkább alkalmazott technika. Ekkor a programot a valós adatokhoz hasonló adatokkal teszik próbára. A kimenetek eredményei lehetőséget adnak anomáliák, problémák feltárására. Ezen tesztelés két fajtája ismert:

1. **Hiányosságtesztelés:** célja a program és a specifikációja között meglévő ellentmondások felderítése. Az ilyen teszteket a rendszer hiányosságainak feltárására tervezik, nem a valós működés szimulálására.
2. **Statisztikai (vagy validációs) tesztelés:** a program teljesítményének és megbízhatóságának tesztelése valós körülményeket szimulálva. Annak megmutatása, hogy a szoftver megfelel-e a vásárlói igényeknek.

A dinamikus tesztelés folyamatának általános modellje látható következő ábrán.

A szoftvertesztelési folyamat modellje



A teszteset nem más, mint a teszthez szükséges inputok és a rendszertől várt outputok specifikációja. A tesztadatok kifejezetten a rendszer tesztelésére létrehozott bemenő paraméterek. A tesztadatok néha automatikusan generálhatók, viszont az automatikus teszteset-generálás általában nem lehetséges. A tesztek outputjait csak azok tudják előre megjósolni, akik értik, hogy a rendszernek mit kellene csinálnia.

Beszélhetünk kimerítő tesztelésről is, ahol az összes lehetséges program-végrehajtási szekvenciát teszteljük. Ez a gyakorlatban nem praktikus, ezért a tesztelésnek a lehetséges tesztesetek egy részhalmazán kell alapulnia. Erre irányelveket kell kidolgozni a szervezetnek, nem pedig a fejlesztőcsoportra hagyni.

Szoftverteszt-típusok

1. Funkcionális teszt

A rendszer funkcionális specifikációján lapuló teszt, melynek során a rendszer működését vizsgáljuk, vagyis hogy „mit csinál” a rendszer (vagy a komponens). A funkcionális tesztek alapjait a dokumentált funkciók

képezik, amelyek minden tesztelési szinten végrehajthatók. A funkcionális teszt az ISO9126-os szabvány alapján összpontosíthat az alkalmasságra, együttműködő képességre, biztonságra (pl. vírusokkal szembeni védelem), pontosságra és a megfelelőségre. A tesztelő arra összpontosít, hogy mit csinál a szoftver, nem pedig arra hogy hogyan teszi ezt. **Fekete doboz teszt**ként is utalnak rá, bár az elnevezést nem teljesen helytálló, mivel az funkcionális teszteken túl nem funkcionálisakat is tartalmaz.

2. Nem funkcionális tesztelés

A rendszer minőségi jellemzőinek, vagyis nem funkcionális tulajdonságainak tesztelése. A tesztelés minden szintjén vizsgálni kell ezeket a jellemzőket is.

A nem funkcionális teszt magában foglalja a:

- teljesítménytesztet,
- a terheléses tesztet (egy bizonyos terhelés mellett keressük a rendszer szűk keresztmetszeteit),
- a stressztesztet (a normál működés során fellépőnél jóval nagyobb terheléssel dolgozunk)
- a használhatósági tesztet,
- a karbantarthatósági tesztet,
- a megbízhatósági tesztet és a
- hordozhatósági tesztet.

A nem funkcionális tesztelés azt vizsgálja, hogy a rendszer „milyen jól” működik. Az ISO9126 szabvány a következő minőségi jellemzőket javasolja vizsgálni a nem funkcionális tesztelés keretében: megbízhatóság, használhatóság, hatékonyság, karbantarthatóság, hordozhatóság.

3. Strukturális teszt

A rendszer vagy komponens szerkezetét, felépítését vizsgáljuk. Gyakran használják a **„fehérdoboz teszt”** kifejezést is erre a szakaszra, hiszen az érdekel minket, hogy mi történik „a doboz belsejében”. Alapjául szolgál a rendszer felépítése, például a **híváshierarchia**. A tesztesetek előkészítéséhez a szoftver belső struktúráját használják fel.

Minden tesztelési szinten végrehajtható, de jellemzően a komponens és az integráció szintjén alkalmazzák.

4. Ellenőrző teszt és regressziós teszt

Ezen tesztelési eljárások során a szoftvert érintő változásokat és hatásukat vizsgáljuk. Ehhez tudni kell, hogy ha valamit megváltoztatunk az alkalmazásban, akkor annak működése, teljesítménye és struktúrája is megváltozik.

Ellenőrző teszt (újratesztelés)

Ha egy szoftver a tesztelésen megbukik és a meghibásodást egy a szoftverben levő hiba okozza, akkor a hiba jelentése után elkészítik a javított változatot, amit a tesztelők megkapnak és elvégzik az ellenőrző tesztelést (újratesztelést). Fontos, hogy a szoftver újratesztelését pontosan ugyanúgy végezzük, ahogyan a hiba megjelenését kiváltó első alkalommal is tettük. Ha a szoftverteszt sikeres az még nem jelenti, hogy a szoftver egésze hibátlan, csupán azt hogy a korábban hibát tartalmazó része most már hibátlan. A javításnak ugyanis lehetnek „mellékhatásai”, vagyis felfedhet egy addig rejtett hibát vagy akár egy újabb hiba is keletkezhet. Ezek felfedésére szolgál a regressziós teszt.

Regressziós teszt

Ez a tesztelési mód is a korábban már elvégzett tesztek újbóli végrehajtását jelenti. Lényeges különbség viszont, hogy a regressziós teszt azokat a teszteseteket vizsgálja újra, amelyek korábban nem okoztak hibát. Tehát lényegében a korábbi helyes működésű modulok újraellenőrzésével azt vizsgálja meg, hogy a szoftveren vagy környezetén végzett változtatások nem okoztak-e kedvezőtlen mellékhatásokat, illetve hogy a rendszer továbbra is megfelel-e a követelményeknek. Az a legjobb, ha mindegyik tesztelési szinthez rendelkezünk regressziós tesztkészlettel.

Regressziós tesztet minden olyan esetben futtatnunk szükséges, amikor megváltozik a szoftver, legyen a változás akár hibajavítások, akár új illetve megváltozott funkcionalitás eredménye. Ha a szoftverkörnyezetben változik meg valami (pl. új változatú adatbázis-kezelő), akkor is célszerű lefuttatni a regressziós tesztet.

Fontos a regressziós tesztkészlet folyamatos karbantartása, hogy az fokozatosan, a szoftverrel párhuzamosan fejlődjön. A tesztkészlet így óriásira duzzadhat, ezért a tesztelés belátható időkorlátok közé szorítása érdekében bizonyos eseteket el szoktak hagyni, pl. a gyakran ismételteket, illetve a hosszú ideje programhibát nem okozóakat.

5. Karbantartási teszt

Bevezetésüket követően a szoftverrendszereket gyakran évekig vagy évtizedekig használják. Ez idő alatt a rendszert vagy környezetét gyakran javítják, megváltoztatják vagy bővítik. A szoftver életciklusának ezen szakaszában végzett tesztelést karbantartási tesztnek nevezzük.

A karbantartási teszt különbözik a **karbantarthatósági teszt**től, amellyel azt határozzuk meg, hogy mennyire könnyű a rendszert karbantartani.

Karbantartás során a tesztelési folyamat ugyanazon lépéseit alkalmazzuk, mint a fejlesztési tesztek során. A végrehajtott változtatásoktól függően a tesztelés több szintjét is használhatjuk: komponens teszt, integrációs teszt, rendszerteszt, átvételi teszt.

A karbantartási teszt általában két részből áll:

- a változások tesztelése
- regressziós tesztelés – annak ellenőrzése, hogy a karbantartási munkák nem voltak-e kihatással a rendszer többi részére

Operációs rendszerek esetén a karbantartási tesztet a szoftver módosítása, más platformra való migrációja vagy lecserélése okán végzünk. Módosítás lehet tervezett vagy javító- és sürgősségi is (időközben felbukkant kritikus hibák gyors javítása).

Tesztelési szintek

Komponensteszt

A komponensteszt - más néven egység-, modul- vagy programteszt – programhibákat keres a szoftver önállóan tesztelhető részeiben (pl. modulok, osztályok), és ellenőrzi azok működését.

Magában foglalhatja a funkcionalitás, valamint meghatározott nem funkcionális jellemzők tesztelését is. A teszteseteket olyan termékekből készítjük, mint például az adatmodell-specifikáció.

A komponensteszt során általában hozzáférünk a tesztelt forráskódhoz és segítségként felhasználjuk a fejlesztési környezet egyes részeit (pl. egységteszt-keretrendszer, hibakereső eszközök). Néha a komponenstesztet másik fejlesztő végzi, ezzel függetlenné téve a tesztelést. A programhibákat általában már megtaláláskor javítják.

Integrációs teszt

Az integrációs teszt a tesztek közötti, a komponensek közötti, a rendszer különböző részei közötti vagy éppen a rendszerek közötti kölcsönhatásokat vizsgálják.

Több szintje lehet, például:

- Komponensintegrációs teszt: a komponensteszt után végezzük és a szoftverkomponensek közötti kölcsönhatásokat teszteljünk vele.
- rendszerintegrációs teszt: a rendszerteszt után végezzük és a különböző rendszerek közötti kölcsönhatásokat teszteljük.

Változatai:

- „Nagy bumm” integrációs teszt: az összes komponenst vagy rendszert egységes egészként teszteljük. Előnye, hogy minden rész készen van a tesztelés megkezdésekor, nem szükség a befejezetlen részek szimulálására, helyettesítésére. Hátránya, hogy időigényes, illetve a késői integráció miatt nehéz feltárni a meghibásodások okát.
- Inkrementális integrációs teszt: minden programot egyesével integrálunk és minden lépés után végrehajtunk egy tesztet. Fajtái: felülről lefelé haladó, alulról felfelé haladó, funkcionális.

Célszerű az integrációs tesztet azokkal az interfészekkel kezdeni, melyekkel kapcsolatban a legtöbb probléma merülhet fel. Így csökkenthető a programhibák késői felhasználásának kockázata.

Az integrációs tesztet maguk a fejlesztők is végezhetik.

Rendszerteszt

A termék egészének viselkedését vizsgálja, amelyet a fejlesztési projekt vagy termék alkalmazási területe határoz meg. Általában ez a fejlesztés során alkalmazott végső teszt, mely ellenőrzi, hogy az átadandó rendszer megfelel-e a specifikációnak, célja pedig a lehető legtöbb programhiba megtalálása. Leggyakrabban erre szakosodott tesztelők vagy tesztcsoport hajtja végre.

A rendszerteszt során a rendszer funkcionális és nem funkcionális követelményeit is vizsgálni kell. A funkcionális követelmények rendszertesztje feketedoboz-alapú technikák alkalmazásával kezdődik, majd fehérdoboz-tesztek is alkalmazhatók. A tipikus nem funkcionális tesztek a teljesítményt és a megbízhatóságot vizsgálják.

Rendszertesztek esetén elengedhetetlen az ellenőrzött tesztkörnyezet. Ennek a lehető legjobban kell hasonlítania a végfelhasználási vagy termelési környezetre, hogy minél kisebb kockázata legyen annak, hogy a környezetspecifikus meghibásodásokat esetleg ne találja meg a teszt.

Átvételi teszt

Miután a fejlesztő szervezet elvégezte a rendszertesztet és kijavította az összes, vagy legalábbis a legtöbb programhibát, a rendszer a felhasználóhoz vagy az ügyfélhez kerül átvételi tesztre. Legtöbbször az ő feladatuk a tesztelés ezen szakaszának lefolytatása. Ehhez szükség van egy tesztkörnyezetre, amely a lehető legpontosabban reprezentálja a valós termelési környezetet.

A teszt célja a rendszerbe vagy annak nem funkcionális jellemzőibe vetett bizalom megteremtése. Leggyakrabban nem a programhibák megtalálására, hanem validációs tesztelésre összpontosít, amely során igyekszünk meghatározni, hogy a rendszer megfelel-e a céloknak. Bár célja a rendszer kibocsáthatóságának és használhatóságának az elemzése, nem minden esetben jelenti a tesztelés utolsó szintjét. Követheti például egy nagyobb méretű rendszerintegrációs teszt.

Típusai:

- **Felhasználói átvételi teszt:** a funkcionalitásra összpontosít és a rendszer alkalmasságát vállalati felhasználók bevonásával ellenőrzi. A rendszerteszt bizonyos szakaszaival együtt is végezhető.
- **Működési átvételi teszt:** ellenőrzi, hogy a rendszer megfelel-e a működéshez szükséges követelményeknek. Hozzátartozhat a mentés/helyreállítás, az összeomlás utáni visszaállítás és a karbantartási feladatok tesztelése, valamint a biztonsági rések periodikus ellenőrzése.
- **A szerződésre vonatkozó átvételi teszt:** a szerződésben meghatározott átvételi kritériumok teljesülésének vizsgálata.
- **Megfelelőségi, vagy más néven az előírásokra vonatkozó átvételi teszt** (pl. jogi, biztonsági előírásoknak való megfelelés tesztelése).

Tömegpiacra fejlesztett rendszerek, például **kereskedelmi dobozos szoftverek esetében** nem praktikus, vagy egyenesen lehetetlen a tesztelést egyéni tesztelőkkel vagy ügyfelekkel végeztetni. Az ilyen rendszer átvételi tesztjét gyakran két szakaszban hajtjuk végre.

Az első szakasz az **alfa tesztelés**, amit a fejlesztő szervezetnél végeznek. Ennek során egy a potenciális felhasználókból és a fejlesztői szervezet tagjaiból álló csoport használja a rendszert, közben pedig a fejlesztők megfigyelik a felhasználókat és dokumentálják a problémákat. Az alfa tesztelést egy független, erre **szakosodott tesztelési csapat** is végezheti.

A **béta tesztelés**, vagy más néven valós környezetben történő tesztelés során a rendszert külső tesztelésre küldik a felhasználók bizonyos csoportjához, akik telepítik és valós feltételek mellett használják a rendszert. A felhasználók a rendszerrel kapcsolatos incidensekről, működési rendellenességekről készített feljegyzéseiket megküldik a fejlesztő szervezetnek, ahol a programhibákat kijavítják.

Az átvételi tesztet egyes szervezetek eltérően is nevezhetik, például működési átvételi tesznek vagy helyszíni átvételi tesztnek.

Smoke tesztelés

A smoke testing egy előkészítő tesztelési mód, melyet a szoftver tesztelési időszakának megkezdése előtt végzünk el. Az ilyen tesztelés során csak a funkcionális követelmények teljesülését vizsgáljuk. Olyan egyszerű, kis hibák felderítése történik, melyek bár jelentéktelennek tűnnek, mégis eléggé súlyosak ahhoz, hogy kockára tegyék a készülő szoftver időben történő kiadását. Ha tehát egy szoftver modul, inkrementum vagy prototípus megbukik a smoke teszten, akkor annak részletes tesztelése is kedvezőtlen eredményt hozna. Mivel viszont a smoke egy gyors és előzetes teszt, gyors és kis költségű teszteléssel is jelentős erőforrásokat és időt takarít meg a fejlesztő számára.
Az, hogy egy adott szoftver esetén melyek azok a funkciók, amiket vizsgálnunk kell smoke teszteléssel, mindig az adott alkalmazástól függenek, tehát minden fejlesztésnél egyediek.
Például egy webalkalmazás esetén smoke teszt lehet például az applikáció néhány, vagy összes képernyőjének megjelenítése, alapvető regisztrációs és bejelentkezési űrlapok működése, stb. Vagyis azok az elemek kerülnek a smoke tesztelés látókörébe, amelyek hibás működése a többi rész működését is ellehetetleníti.
Az adott projekthez készített smoke tesztekből érdemes egy gyűjteményt összeállítani. Ennek előnye, hogy az adott bulid (nem release!) elkészülte utáni ellenőrzésével meggyőződhetünk arról, hogy a jelenlegi változtatások nem okoztak semmilyen súlyos hibát az applikációban. Akkor is hasznos lehet a tesztgyűjtemény, ha egy már kiadott szoftverhez egy sürgős patchet kell kiadni, és az időbeli korlátok miatt nincs lehetőség teljes regressziós tesztelésre. Ez ritka eset, de sajnos előfordulhat.

A smoke tesztelés előnyei

- Az egész tesztelési folyamatot hatékonyabbá teszi. Ez magától értetődő, hiszen ha egy alapvetően hibás szoftver tesztelésébe kezdünk, csupán értelmetlenül pazaroljuk az erőforrásainkat.
- A szoftverhibák korábban felszínre kerülnek. Bár a smoke testing csak a tesztesetek kb. 20%-át fedi le, a hibák 80%-át mégis már ebben a stádiumban megtaláljuk.
- Regressziós tesztkor talált és egyéb hibák gyorsabb javítása. A smoke teszt utáni későbbi tesztelés során természetesen találhatunk további hibákat, azonban ezek javítása és az okaik megtalálása sokkal egyszerűbb lesz a smoke tesztek eredményének ismeretében.

Mely teszteseteket vegyünk be a smoke tesztelésbe?

A smoke testing lényege, hogy az alapvető hibák korai felfedezése segítségével információt ad a fejlesztőknek arról, hogy egyáltalán érdemes-e részletesebben tesztelni a szoftvert, vagy sem. Emiatt az itt alkalmazott tesztesetek körébe azokat érdemes bevenni, melyek a felhasználói interfészhez (*user interface - UI*) kapcsolódnak, így például a felhasználói interakciók teszteseteit (scriptjeit).
Azt is érdemes megvizsgálni, hogy hány tesztet célszerű bevenni a smoke teszt gyűjteménybe. Ez az alkalmazástól függ, de a gyakorlat azt mutatja, hogy a legtöbb esetben ez a szám 20 és 50 között van. Ha a

tesztjeink száma ezen a sávon kívül esik, akkor valószínűleg vagy túl kevés esetet fedtünk le, vagy a felvett tesztesetek meghaladják a smoke testing kereteit.
A smoke testing a szoftverhibák megtalálásának költséghatékony módja, szakértők szerint a legjobb módja, így érdemes megfontolnunk a használatát.

Tesztelési környezet
A tesztelési környezet megtervezése és létrehozása a szoftvertesztelési stratégia kialakításakor az első feladatok között szerepel. E nélkül a környezet nélkül nem tudunk hozzáfogni a teszteléshez, hiszen az ebben a közegben történik.
Azért van szükség tesztelői környezetre, mert a szoftverfejlesztés során bármikor megváltozhatnak a környezet jellemzői és a tesztelni kívánt szoftver paraméterei is, hiszen a fejlesztői csapat folyamatosan változtathatja a forráskódot. Az éles környezetben pedig azért nem célszerű tesztelni, mert akkor a cégen kívüli szereplők (végfelhasználók, versenytársak, stb.) is láthatják, hogy mire is készülünk valójában, milyen újítást kívánunk bevezetni, valamint az éles környezetben a még kiforratlan szoftverrel akár maradandó károsodást is okozhatunk.

Mi a tesztkörnyezet?
- Tesztkörnyezet: Az a környezet, ami hardvert, szimulátorokat (a valós rendszernek a tesztkörnyezetbe nem vihető részeinek kiváltására), szoftvereszközöket és egyéb elemeket tartalmaz annak érdekében, hogy a tesztelést le lehessen folytatni.
- A tesztkörnyezetek különböző szolgáltatásokat nyújtanak a tesztek módszeres elvégzéséhez.
- Jellemzője, hogy minden, az eredeti rendszeren végzett módosítás ide is átvezetésre kerül, az éles rendszerben használt érvényes és aktuális adatokat tárolja, ugyanazokat a frissítéseket kapja, adatbázisa naprakész stb.

A tesztkörnyezet komponensei
- teszteket végrehajtó komponens (test driver, test harness, test bench)
- tesztkonfiguráció menedzser komponens (test configuration manager)
- a tesztelendő egység „beműszerezésére" szolgáló komponens (instrumentation)
- a tesztelt egység külső szoftver környezetét szimuláló elemek (stubs)

A **teszteket végrehajtó komponens** egy adott hardver és szoftver környezetben (operációs rendszer, megfelelően telepített szoftver elemekkel) működik. Több szolgáltatást is biztosíthat (pl. automatizált tesztelés, debugger, profilok kezelése, dokumentálás stb.), melyek közül választva a tesztkörnyezet tervezője határozza meg, hogy pontosan mire van szükség. Ezt előre meg kell tervezni.

A **tesztkonfiguráció menedzsment**nek kell biztosítania a megfelelő szoftveregységek tesztelését, a teszteseteket és a dokumentálást. Ez a szolgáltatás teremti meg a teszteket végrehajtó komponens működésének feltételeit, illetve felelős magáért a működtetésért is.
Feladatai:
- a tesztelni kívánt egység "beműszerezése"
- a tesztelő által megadott tesztesetek és teszt paraméterek alapján a teszt beállítása
- teszt elvégzése és az eredmények értékelése, kezelése

A **műszerezés** azt jelenti, hogy a teszteléshez a szoftveregységet felszereljük a működéséről tájékoztató kapcsolódási pontokkal. Ezek kapcsolják össze a tesztvégrehajtó komponenst és a tesztelt egységet.
Kétféle beműszerezésről beszélhetünk:
- a forráskód beműszerezése (a szoftver forráskódjába a hibakeresést segítő, adatokat naplózó programkódot írunk)
- a végrehajtható kód beműszerezése (fordításkor történik, a forráskód megváltoztatása nélkül)

A forráskód műszerezés széles körben alkalmazható és nagy szabadságot biztosít a teszteléskor, de nehéz karbantartani, megnöveli a fordítás/tesztelés ciklus idejét és nem utolsó sorban módosítja az eredeti kódot, aminek mellékhatásai lehetnek.

A végrehajtható kód műszerezése ennél korlátozottabban használható, de működése kevésbé zavaró a tesztelendő szoftver szempontjából. Léteznek úgynevezett *patch szintű eszközök*, melyek a forráskód módosítása nélkül működnek, a végrehajtható kód futásakor lépnek működésbe.

A tesztelt egység **külső szoftver környezetét szimuláló elemek** jellemzői:

- A hierarchia legfelső szintjén álló elem teszteléséhez az eggyel lejjebb álló elemek viselkedését és interfészét szimuláló ideiglenes elemek (csonkok) szükségesek.
- A csonkok bár ugyanolyan interfésszel rendelkeznek, mint a komponens, de funkcionalitásuk erősen korlátozott.
- Megadható, hogy teszteléskor az egyes meghívásokkor milyen visszatérési értékei legyenek a szimulált függvénynek.

A csonkok validálásnál jól használhatóak, segítségükkel a felhasználó jól átlátja a szoftverkomponens tevékenységét. Ha nem megfelelő számára, akkor ezt jelzi, és a fejlesztőknek egyből lehetőségük van a gyors változtatásokra (a csonkok egyszerű szimulátorok, módosításuk a szimulált komponenséhez képest igen hamar megoldható).

Egy jól használható tesztkörnyezet jellemzői:

- a fejlesztők / tesztelők munkavégzésének támogatása: A munka esetenként külsős fejlesztők bevonásával történik. A rendszernek hozzáférést kell biztosítania a külsős (jellemzően interneten keresztül kapcsolódó) és a belsős fejlesztők számára egyaránt.
- az éles rendszerekkel megegyező konfigurációjú: A fejlesztéshez elengedhetetlen a kialakítandó környezetek egyezősége az éles rendszerekkel. Az ügyfél rendszerében bekövetkező változások követésére is lehetőséget kell biztosítani.
- fejlesztői / tesztelői segédalkalmazások elérhetősége: pl. verziókövető és hibakezelő rendszerek
- mentés / visszaállítás lehetőségének biztosítása: A teljes szoftverteszt-környezet menthető és visszaállítható az előző stabil verzióra.
- az éles rendszertől teljesen független környezet: Mivel a fejlesztői környezetben az éles rendszerrel adott esetben megegyező konfigurációjú szerverek találhatóak, szükséges a teljes leválasztás. A teszt szerverek semmilyen kommunikációt nem folytathatnak az éles környezetben levő gépekkel.

Mi szükséges egy jól használható tesztkörnyezet kialakításához?

- Koncepció kialakítása:
  - Mit szeretnénk tesztelni?
  - Kik használják a környezetet?
  - Szoftver belépési (entry) és kilépési (exit) kritériumok meghatározása.
  - Teszt és tesztmenedzsment eszközök kiválasztása stb.
- Szükséges erőforrások felmérése: hardver, szoftver, üzemeltetési.
- Költségelemzés készítése .

Milyen hardver környezet szükséges a teszteléshez?

- tesztelés / fejlesztés függő
- Hardver specifikus:
  - speciális hardver szükséges (pl.: IBM Power Solution Edition szerverek)
  - a szoftver tesztelhetősége függ a hardvertől (pl. firmware változás miatt)
- Szoftver specifikus:
  - speciális szoftver környezet szükséges hozzá (pl. egyedi front end)

Napjainkban a tesztelés gyakran **virtualizációs megoldások** segítségével történik.

Milyen operációs rendszeren történjen a tesztelés, milyen hardverrel rendelkezzen a számítógép?

- Egy jól használható tesztkörnyezet kialakításakor célszerű virtuális gépekre építve biztosítani a fejlesztéshez szükséges erőforrások megfelelő kihasználását, mely a tesztkörnyezetek újrafelhasználásának lehetőségét is magában foglalja.

- Abban az esetben, ha az adott komponens nem virtualizálható (pl. az operációs rendszere nem támogatja), lehetőség van valós számítógépek bevonására a tesztelés érdekében.
- Néhány lehetséges virtualizációs megoldás pl.: VMWare, Microsoft Virtual Server és Virtual PC, Oracle VirtualBox

Szoftver követelmény példa:
- VMWare ESX Server
- VMWare Virtual Center
- VMWare Lab Manager
- VMWare Virtual Desktop Infrastructure VMWare Backup Agent Server
- Windows 10
- Windows Server 2016
- teszt és tesztmenedzsment eszközök

Hardver követelmény példa:
- IBM Power Solution Edition szerver
- IBM Blade szerver

Tesztkörnyezet kialakításának lépései:
Három fő lépést kell megkülönböztetni:
- Tesztkörnyezet tervezés.
- Kockázatokra való felkészülés.
- Költségelemzés.

A tesztelés automatizálása

A szoftverfejlesztés elengedhetetlen része a tesztelés, amely során automatizált eszközök is a tesztelő szakemberek rendelkezésre állnak. Az automatizálás főként akkor hasznos, ha nagyszámú tesztet kell végrehajtani, illetve ha nagy mennyiségű tesztadattal kell dolgozni.
A fejlesztendő szoftverrendszerek egyre komplexebbé válnak, ugyanakkor elvárás, hogy az adott termék a lehető leggyorsabban kerüljön piacra. Ez a két egymással szembenálló tény a tesztautomatizálás irányába hat és a folyamat velejárója a manuális tesztelés iránti igény csökkentése, vagy szélsőséges esetben a tesztelés teljes automatizálása (ha lehetséges).
A tesztautomatizálás célja a kézi teszteléshez szükséges erőforrások minimálisra csökkentése. Az automatizálás azonban nem problémamentes, mert alkalmazása nem jelent minden esetben előnyt, sőt, van hogy buktatót rejt magában, illetve káros következménnyel jár.

A tesztautomatizálás előnyei:
• időt és pénzt takarít meg;
• sebesség, hatékonyság és minőség optimalizálása;
• javítja a pontosságot;
• segíti a fejlesztőket és a tesztelőket;
• javítja a termék minőségét;
• könnyebben dokumentálható;
• hiba esetén nagy valószínűséggel könnyebben reprodukálható.

Gyakorlati tapasztalatok az automata tesztelő eszközökkel kapcsolatban

„Van egy nagyon nagy hátránya az automata teszteknek. Ha hatékonyan akarod őket használni, direkt automata teszteléshez kell megírni mindent. A normális dokumentáció is hozzá tartozik. Durván háromszor annyi munkád lesz a fejlesztéssel, mint nélküle. Mindazt csak azért, hogy hibatűrő jelzéseket kapj arról, ha valamelyik fejlesztés a 3 közül gyengélkedik. Talán nem kell jellemeznem a mai világot, mennyire cheapskate cégekkel van tele. Szóval ne vedd biztosra, hogy foglalkoznod kell az automata tesztelés hasznosságával. Csak

nagyon kevés cég engedheti meg magának a gyakorlatban. A többieknek az semmi több, mint savanyú szőlő.”
Forrás: prog.hu

Lényege:

- Már a specifikáció készítésekor be kell tervezni azt, hogy majd automata tesztelő eszközöket kívánunk használni (és melyiket).
- A tervezés során ez jelentős többletmunkát jelent, mivel a tesztelő eszköz működéséhez kell igazítanunk a fejlesztett alkalmazás architektúráját.
- A fejlesztési idő is számottevően megnő, mert ahhoz hogy az automata tesztelők megfelelő minőségű kimenetet adjanak, előbb precízen illesztett bemeneteket kell számukra produkálni.
- A nagyszámú cheapskate (fösvény), vagyis a jó minőségű munkaerőt megfizetni nem akaró cégek inkább foglalkoztatnak nagyobb létszámú olcsó munkaerőt, ami feleslegessé teszi az automata tesztelést (amihez jól képzett és fizetett szakemberek kellenének).

Tesztvezérelt fejlesztés

Tesztvezérelt fejlesztés (test-driven development, TDD) alatt egy olyan szoftverfejlesztési megközelítést értünk, amely a tesztelés és a kódfejlesztés folyamatát együttesen, egymástól szét nem választható módon, párhuzamosan végzi. A kód kifejlesztése inkrementális módon történik, és mindig magával vonja az adott inkremens tesztjeinek a fejlesztését is. Mindaddig nem léphetünk a következő inkremens fejlesztésére, amíg a korábbi kódok át nem mennek a teszteken.

A tesztvezérelt fejlesztés eredetileg az extrém programozásnak (extreme programming, XP) nevezett agilis szoftverfejlesztési módszertan részeként jelent meg, de sikerrel alkalmazhatjuk nemcsak agilis, de hagyományos (terv központú) szoftverek kifejlesztése során is.

A tesztvezérelt fejlesztés három törvénye:

- Tilos bármilyen éles kódot írni mindaddig, amíg nincs hozzá olyan egységtesztünk, amely elbukik.
- Tilos egyszerre annál több egységtesztet írni, mint ami ahhoz szükséges, hogy a kód elbukjon a teszt végrehajtásán. A fordítási hiba is bukásnak számít!
- Nem szabad annál több éles kódot fejleszteni, mint amennyire ahhoz van szükség, hogy egy elbukó egységteszt átmenjen.

Ez a három törvény azt jelenti számunkra, hogy először mindig egységtesztet kell készítenünk ahhoz a funkcionalitáshoz, amelyet le szeretnénk programozni. Azonban a 2. szabály miatt nem írhatunk túl sok ilyen tesztet: mihelyst az egységteszt kódja nem fordul le, vagy nem teljesül valamely állítása, az egységteszt fejlesztését be kell bejezni, és az éles kód írásával kell foglalkoznunk. A 3. szabály miatt azonban ilyen kódból csak annyit (azt a minimális mértékűt) szabad fejlesztenünk, amely a tesztet lefordíthatóvá és sikeresen lefuttathatóvá teszi.

Kiegészítés: a szoftvertesztelő munkakör

Amikor felbukkan egy megrendelői igény egy szoftverre, abban a pillanatban kezdődik egy tesztelő munkája. A szoftverhez meghatározott követelmények kellenek, egy gyakorlott tesztelő a fejlesztőket támogatva már ennek kidolgozásában is részt vehet, de mindenképp fontos, hogy ismerje ezeket. Már ekkor körvonalazódnak a végrehajtandó tesztek forgatókönyvei. Vállalattól és biztonsági szinttől függően már ekkor el is lehet kezdeni írni a teszteseteket. Pontosan megfogalmazva, dokumentálva (a dokumentumkezelés lényeges, a munkaidő jelentős része erre megy el).
Ez után kezdődhetnek a tesztek. A teszteseteken végig kell haladni, minden tesztadattal ki kell próbálni a prototípust. Ez a feladat egy idő után monotonná válik.

Ha viszont hibát találunk, akkor indul az érdekesebb munka. Először is le kell szűkíteni, hogy mi okozhatta a hibát, riportot írni a hibáról, értesíteni a fejlesztőket, és a vállalattól függően részt lehet venni a hiba okának kiderítésében. A javított verziót különböző módokon újra kell tesztelni, a teszteseteket ismételve.
Teszt automatizáció is használható, de ez igényel programozói tudást is.
Egy jó tesztelőnek nagyon pontosan ismerni kell a program működését, de nemcsak az új funkciókat, hanem a régieket is. Pontosnak, precíznek lenni. Mivel mindig lesznek új funkciók, új hibák, így ez mindennapos tanulást igényel, váratlan helyzeteket teremt. Ugyanakkor a munka másik fázisa pedig inkább monoton. Ez a kettősség a szakma jellemzője.

Előnyei:

- A megfelelő hozzáállással, és tudással gyakorlat nélkül is könnyen el lehet helyezkedni benne.
- Nagyon keresett, piacképes szakma, jó fizetéssel.
- A szoftvereké a jövő, azokat pedig mindig kell tesztelni, tehát jövője is van.
- Van előrelépési lehetőség (manuális tesztelés, automatizált tesztelés, programozás, tesztvezetés, teszt menedzseri pozíció stb.).
- Folyamatos tanulást igényel

Hátrányai:

- Folyamatos tanulást igényel.
- Nem mindenki alkalmas rá. Nem minden ember alkalmas a programozásra, vagy éppen a tesztelésre. Olyan tulajdonságokat igényel, amik nehezen tanulhatóak.
- A munkatempó változó. Vannak nagyon sűrű időszakok, amik túlórát, feszített munkatempót igényelnek.

Mi kell a jó tesztelőnek?

- analitikus gondolkozás, logika
- tanulási hajlandóság
- angol nyelvtudás
- ISTQB (International Software Testing Qualifications Board) tananyag ismerete
- Linux használata
- jó kommunikációs képesség, emberismeret, kis pszichológia (pl. azt közölni a fejlesztővel, hogy hibát találtunk a szoftverében, néha kényes kérdés)
- a széleskörű programozási ismeretek hatalmas előnyt jelentenek