Tesztelés

Az utolsó részhez érkeztünk, ahol az elkészült alkalmazásunkat tesztelni fogjuk. A pontos és megbízható működés érdekében meg kell vizsgálni a létrejött applikációnkat. Az összetettség és a magas felhasználói/vevői elvárások miatt a vizsgálat kihívást jelenthet. A tesztelésre sok időt nem tudunk fordítani, ezért ott szeretnénk tesztelni, ahol a legvalószínűbb, hogy hiba van.

**A tesztelés előtt/után elvégzendő feladatok listája:**

* Elkészítjük a teszttervet
* Teszteseteket tervezünk
* Felkészülünk a teszt végrehajtására
* Tesztek végrehajtása
* Kilépési feltételek vizsgálata
* Eredmények értékelése
* Jelentés elkészítése

Lássuk milyen lehetőségeink vannak a tesztelésre…

Két módszer szerint lehetséges tesztelni: amikor a forráskódot vizsgáljuk és amikor a működést és a viselkedést.

**Ellenőrzések, amiket végrehajthatunk:**

* *Kódellenőrzés*: a program tervet összehasonlítjuk a forrás programmal.
* *Szintaktikai hibák ellenőrzése*: fordító, interpreter használatával.
* *Formai ellenőrzés*: forráskód tagoltság ellenőrzése.
* *Tartalmi ellenőrzés*: forrásprogram vizsgálata.

Dinamikus módon is vizsgálódhatunk. A programot tesztadatokkal lefuttathatjuk, a lefutás végén kapott outputokat elemezzük, és következtetünk a program működésének helyességre.

**A program tesztelésnek két módját használhatjuk:**

* Fekete doboz módszer (black-box) / specifikáció alapú teszt
* Fehér doboz módszer (white-box) / struktúra alapú teszt

Ezeknek a következő az alkalmazása…

*Fekete doboz módszer (forráskód nem áll rendelkezésre):* Tegyük fel, hogy a program egy fekete doboz, melynek adunk adatokat és arra reagál. Az adatokat osztályokra osztjuk (ekvivalencia osztály létrehozása). A kipróbálásnál az adatokat úgy választjuk, hogy minden osztályt lefedjenek. Ha a program rosszul válaszol, akkor az osztály többi adatára is rosszul válaszolna. Ugyanígy következtethetünk a jó válaszra is. Ha minden osztályra jól reagál, akkor minden inputra megfelelő lesz a program. Úgy gondoltuk, hogy a leggyakoribb formáját fogjuk alkalmazni. Ami annyit jelent, hogy az adott bemenetre már tudjuk, hogy milyen kimenetet szeretnénk kapni válaszul. A futtatás után a kimenetet összehasonlítjuk az elvárttal.

*Fehér doboz módszer (forráskód rendelkezésre áll):* A módszerrel már mindig egy kész struktúrát, programkódot tesztelünk. Amiket tesztelhetünk: kódsor, elágazás, metódus, osztály, funkció, modul. A program logikai menetéhez igazodunk. Az összes lehetséges utasítás végrehajtás sorrendjét ki kéne próbálnunk, ami nem lehetséges. Ezért az utasítások egyszeri lefedését használjuk: a tesztelni kívánt adatokat úgy választjuk ki, hogy minden utasítás legalább egyszer végrehajtódjon.

**Hibakeresési technikák, amiket találtunk a program vizsgálatához:**

* *Indukciós módszer*: rendezzük a tesztelési adatok, és azok alapján próbálunk következtetni a hibára.
* *Dedukciós módszer*: összes hiba felsorolása és csökkentése.
* *Visszalépési módszer*: attól a ponttól keresünk visszafelé, ahol a hiba jelentkezett, addig a pontig, ahol még nem volt hiba.

**Hibakeresési eszközök**, amikkel könnyen és gyorsan megtaláljuk a hibát: részegységekre bontás (kisebb egységenkénti vizsgálat), kiíratás és ellenőrző feltételek.

**Komponens tesztelése:**

* Külön álló egységek esetén: itt kiderül, hogy a komponensek működése helyes.
* Modultest esetén a magasabb szintű egységeket teszteljük.

**Rendszert és alrendszer is tesztelhetünk.** A rendszer vizsgálat során az alrendszerek közötti kommunikációt figyeljük, működés és illeszkedés szempontjából. Az alrendszerek vizsgálatakor a modulok együttesét teszteljük. A kommunikációs hibák itt kiderülnek.

**Tesztelési formák:**

* *Top-down*: A legfelső hierarchiai szintről indulva végig vizsgáljuk az összes útvonalat miközben ellenőrizzük a program részek megfelelő működését.
* *Bottom-up*: Alulról felfelé. A legalapvetőbb programrészektől haladunk felfele s vizsgáljuk a részek közti átadott érték helyességét.

**Végső teszt folyamatok:**

Mindenképpen ezt a tesztet szeretnénk végrehajtani, ami a következő: **Béta teszt**

Amikor a belső teszteléskor a program a teszt adatokkal megfelelően működik, de visszacsatolásra van szükség. Így lehetőséget adunk néhány kiválasztott felhasználónak, hogy a rendszert tesztelhesse valós adatokkal. Ezzel tudjuk a legjobban kiszűrni a hibákat. Jöhetünk rá összefüggésekre, problémákra a további fejlesztés elindításához. Az alkalmazásunk kapcsán, úgy gondoljuk, hogy ez a tesztelés a legfontosabb az üzembe állítás előtt.

**RC (Release Candidate):** A kiadásra kész alkalmazást még egy körös tesztelésen átfuttatjuk, ezzel közelebb kerülünk a végleges állapothoz. Itt a tesztelő felhasználok köre bővül. Hibák száma kevesebb.

**Bug Report**: Mivel tesztelés folyamán nem lehetséges mindent letesztelni a gyártó terén. Ezért a kiadás után a felhasználok találhatnak hibákat még. Így a felhasználók is valamilyen szinten tesztelőknek számítanak. Egy tökéletes program megvalósítása és egy jól használható hibamentes rendszer érdekében.

***Szótár***

**Ekvivalencia osztály:** Ekvivalencia-osztályokat úgy határozunk meg, hogy megfelelő és nem megfelelő input adatokat is tartalmazzon