Tesztelési stratégiák 2

Állapotgép tesztelés

Eredeti név state machine testing. Black box tesztelési stratégia. Az inputok változtatása output vagy állapot változtatást eredményez a tesztelt entitásban (pl SUT – software under test). Lényegében megvizsgáljuk a tesztelt dolog működését különböző inputok esetén. Fel lehet vele deríteni a működő és nem működő felhasználási eseteket.

Az állapotgép egy olyan rendszer, ahol megtörténhet az, hogy ugyanarra az inputra különböző output érkezik, annak függvényében, hogy korábban mi történt a rendszerrel – milyen állapotban volt a rendszer.

Láttunk már erre példát a döntési tábláknál, amikor bizonyos mennyiségű elrontott jelszó után zárolva lett a felhasználói fiók (az ID).

Felhasználás

Mikor használjuk:

* A bemenetek száma véges (kezelhető mennyiségű)
* Amikor események sorozatát akarjuk tesztelni (pl pénzfelvétel egy ATM-ből)
* Amikor állapotgépünk van (az outputok függnek a korábbi inputoktól/eseményektől)

Mikor ne használjuk:

* Amikor csak annyi a rendszerünk, hogy beadunk valamit és várunk valamit egyszer (triviális ábra)
* Amikor túl nagy a rendszer és túl sok dolgot kell ellenőriznünk (böszme nagy ábra)

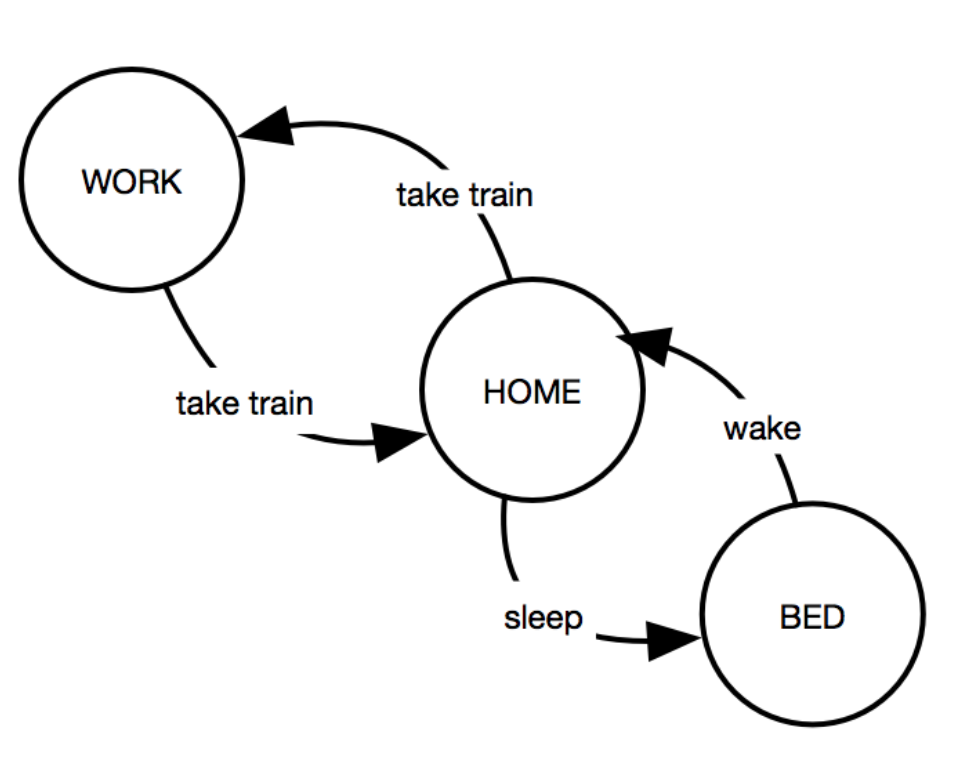
Az állapotátmenet ábra

Elemei:

1. Állapotok – a szoftvernek az állapota (pl alaphelyzetben van egy ATM)
   1. lényeges a kezdő és végállapot
2. Átmenetek – változás a szoftver állapotában
   1. nyilakkal jelöljük
   2. van lehetséges és nem lehetséges átmenet
3. Események – Amik az állapotváltozásokat okozzák
4. Eredmények – Az eseményeknek a következményei

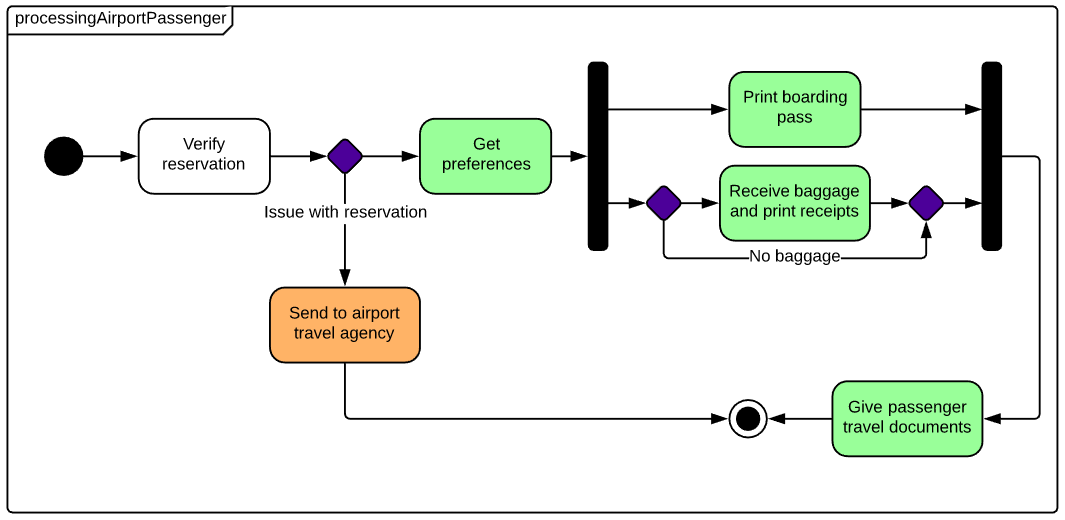
Példa egyszerű állapotátmenetre

* Állapotok:
  + work
  + home
  + bed
* Átmenetek:
  + valid:
    - work -> home
    - home -> work
    - home -> bed
    - bed -> home
  + invalid:
    - work -> bed
    - bed -> work

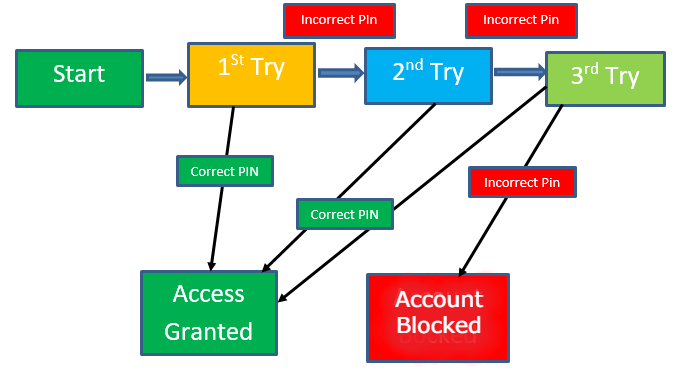


Példa komplexebb állapotátmenetre:

* Állapotok:
  + Kezdő
  + Végső
  + Preferenciák bekérve
  + Dokumentumok készenállnak
* Átmenetek
  + Kezdő -> Végső
  + Kezdő -> Preferenciák bekérve
  + Preferenciák bekérve -> Dokumentumok készenállnak
  + Dokumentumok készenállnak -> Végső
* Események
  + Foglalás ellenőrzése
* Interakciók
  + Preferenciák bekérése
  + Boarding pass nyomtatása
  + Csomagok érkeztetése és bizonylatok nyomtatása
  + Papírok átadása az utasnak



Táblázattá alakítás (példa)



* Ha nincs blokkolva és helyes PIN-t ír be, beengedjük
* Ha helytelen PIN-t ír be, még egyszer próbálkozhat (de összesen 3x az elsővel együtt)
* 3. helyetelen PIN esetén zároljuk a fiókot

Állapotok:

* S1 – start
* S2 – első PIN próbálkozás
* S3 – második PIN próbálkozás
* S4 – harmadik PIN próbálkozás
* S5 – beengedtük
* S6 – zároljuk a fiókot

Táblázat

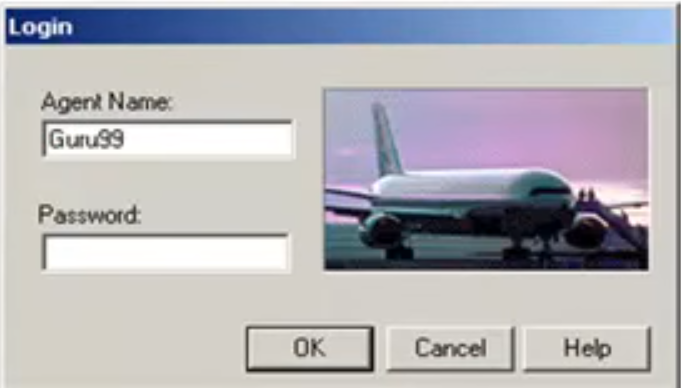
| Állapot | Kártya behelyezése | Jó a PIN (okozó esemény) | Nem jó a PIN (okozó esemény) |
| --- | --- | --- | --- |
| S1 | S2 | - | - |
| S2 | - | S5 | S3 |
| S3 | - | S5 | S4 |
| S4 | - | S5 | S6 |
| S5 | Visszaadja a kártyát | - | - |
| S6 | Elnyeli a kártyát pl | - | - |

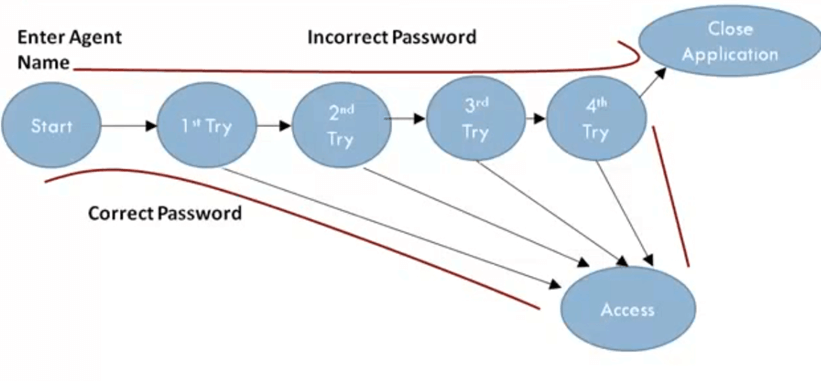
Lehetséges útvonalak:

* S1 -> S2 -> S5
* S1 -> S2 -> S3 -> S5
* S1 -> S2 -> S3 -> S4 -> S5
* S1 -> S2 -> S3 -> S4 -> S6

Invalid útvonalak

Black box testing, van egy alkalmazás amit tesztelünk.





Állapotok

* S1 – Kezdő állapot
* S2 – Első próbálkozás
* S3 – Második próbálkozás
* S4 – Harmadik próbálkozás
* S5 – Negyedik próbálkozás
* S6 – Beengedjük az ügyfelet
* S7 – Bezárjuk az alkalmazást

Táblázat

|  | Megnyitjuk az alkalmazást | Helyes jelszó | Helytelen jelszó |
| --- | --- | --- | --- |
| S1 - Kezdő állapot |  |  |  |
| S2 - Első próbálkozás |  |  |  |
| S3 - Második próbálkozás |  |  |  |
| S4 - Harmadik próbálkozás |  |  |  |
| S5 - Negyedik próbálkozás |  |  |  |
| S6 - Beengedi az ügyfelet |  |  |  |
| S7 - Bezárja az alkalmazást |  |  |  |

Tesztesetek:



Reportolás

* konkrét eset, ebben az esetben a teszt eset
  + pl nem jutottunk el a végéig
  + lépések arra, hogy hogy tudja ezt reprodukálni
  + a környezetet leírod hogy hogy néz ki
* mit tesztelünk
  + egyértelműen azonosítani, hogy melyik verzióról van szó
* kiváltó okok elemzése (root cause analysis)
  + implementáció
  + specifikáció
  + követelmény
  + ügyfél igény

Miért hasznos

* Előnyök
  + Vizuális és/vagy táblázatos reprezentációt nyújt a rendszer viselkedésének felderítésére, ami hozzásegíti a tesztelőt a rendszer megértéséhez és tesztesetekkel lefedéséhez
  + Biztosak lehetünk benne, hogy miden lehetőséget lefedtünk
* Hátrányok
  + Olyan rendszer kell hozzá, ami véges állapottérrel rendelkezik és le van írva, hogy mi után mi következik
  + Nem skálázódik jól (eddigiekben kis rendszerek voltak)

Felhasználási eset tesztelés

Use case testing.

Tesztelési technika, ahol abból indulunk ki, hogy van egy leírásunk egy actor vagy egy felhasználó szempontjából a tesztelt dolgunkról. Rendszer és elfogadási teszt szinten nagyon gyakran használják.

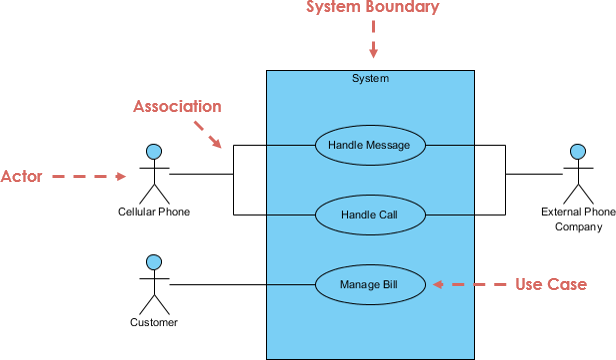
Felhasználási eset

Leírja azt, hogy a felhasználó hogy használja a rendszert, hogy elérjen vele egy meghatározott célt. A felírásához 3 dolgot használ

* Mit írunk le vele (ez a rendszer)
* Ki használja a rendszert (ez az actor)
* Mit akar az actor elérni a rendszerrel (ennek a kifejtése a felhasználási eset)

Ebben az esetben nem feltétlenül írunk le minden összefüggést felhasználási eset, rendszer és actor között (ez az íróra van bízva, hogy mennyire bocsátkozik részletekbe).

Általános példa



Mutatja azt, hogy

* Meddig tart a rendszer
* Milyen actorok vannak vele kapcsolatban
* Milyen felhasználást biztosít a rendszerünk az actoroknak
* Kinek milyen kapcsolata van a rendszerrel

Actor

* Interakcióban van a rendszerrel (valamelyik hat a másikra)
* Valamilyen üzleti szerepe van
* Nem csak felhasználó lehet
* Ők kezdeményezik a felhasználási eseteket a rendszerben
* Biztosít inputokat és outputokat vár
* Egy személy lehet több actor is

Felhasználási eset

* Ő egy rendszer funkcionalitás (feature)
* „Valamit csinál a rendszer valamivel” vagy „valamit csinál a rendszer”
* Minden actornak csatlakoznia kell legalább egy felhasználási esethez

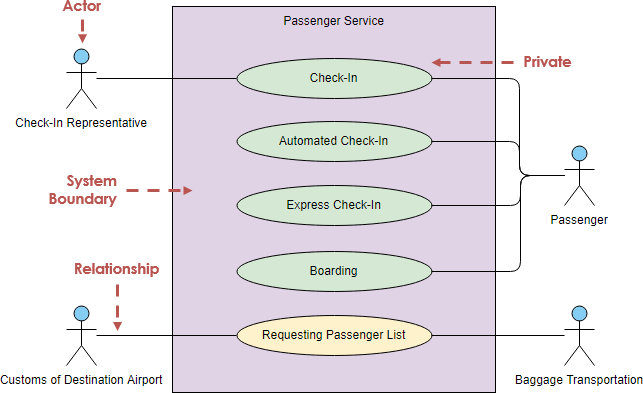
Kommunikációs kapcsolatok

* Folytonos vonal – az actor részt vesz a felhasználási esetben
* Egyéb vonal – az actor és a felhasználási eset csak kommunikál

Rendszer határai

* Elvileg ahogy a rendszer le van írva a követelményekben (system requirement specification)
* Egy termék lehet egy komplex rendszer, ahol több modulra lehet bontani az rendszert (vagy alrendszerre)
  + Pl a google dolgai, hogy más egy fiók, meg más a levelezés és a drive
* Attól függhet, hogy milyen szolgáltatást akarunk éppen tesztelni

Példa



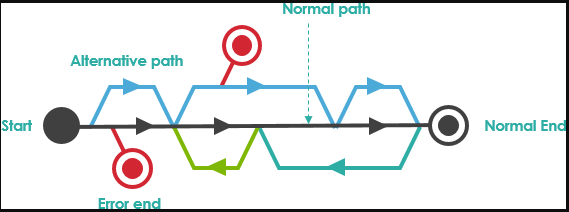
Hogy lesz ezekből teszteset

Leírjuk az actor és a rendszer közötti interakciókat. Mit csinál pl a felhasználó és hogy reagál erre a rendszer.  
Ezeknek a pontjai a következők:

* Leírás
* Előfeltétel (precondition)
* Mi fog előállni a végén (postcondition)
* Fő út
* Alternatív utak
* Kivételek (ezek nem jutnak el a célig)

Amire figyelni kell:

* Egy célja legyen
* Egy meghatározott pontból induljon (vö teszt írás, tesztelhetőség)
* Egy meghatározott pontban végződjön
* Lehet több útvonal kezdő és végpont között



Klasszikus példa: ATM

| Felhasználási eset neve | Készpénz felvétel |
| --- | --- |
| Actor(ok) | Ügyfél (elsődleges), Banking System (másodlagos) |
| Leírás | Lehetővé teszi a bank bármelyik ügyfelének, hogy készpénzt vegyen fel a bankszámlájáról |
| Prioritás | alacsony, közepes, magas |
| Állapot |  |
| Előfeltétel | * Az ügyfélnek van kártyája amit be tud adni az ATM-nek * Az ATM online állapotban van |
| Mi fog előállni a végén | * Az ügyfél megkapja a készpénzét (és opcionálisan a nyugtát) * A bank levonta az összeget a folyószámláról és rögzítette a tranzakció adatait |
| Fő út | 1. Ügyfél behelyezi a kártyáját az ATM-be 2. Az ATM ellenőrzi hogy valid e a kártya 3. Az ATM kéri a PIN-t 4. Az ügyfél beírja a PIN-jét 5. Az ATM ellenőrzi, hogy a kártyához megfelelő e a PIN 6. Az ATM megjeleníti a lehetséges szolgáltatásokat, amiben benne van a készpénz felvétel 7. Az ügyfél kiválasztja a készpénz felvételt 8. Az ATM megjelenít lehetőségeket a pénz mennyiségére 9. Az ügyfél kiválaszt vagy bevisz egy pénzmennyiséget 10. Az ATM ellenőrzi, hogy van e elég pénz benne 11. Az ATM ellenőrzi, hogy az ügyfél nem lépte e túl a készpénzfelvételi keretet 12. Az ATM ellenőrzi, hogy az ügyfél bankszámláján van e elég pénz 13. Az ATM levonja a pénzt a pénzt az ügyfél bankszámlájáról 14. Az ATM kiadja a kártyát 15. Az ügyfél kiveszi a kártyáját 16. Az ATM kiadja a készpénzt 17. Az ügyfél elveszi a készpénzt |
| Alternatív útvonalak | 2a. Érvénytelen kártya  2b. Rosszul behelyezett kártya  5a. Lopott kártya  5b. Helytelen PIN  10a. Nincs elég pénz a gépben  10b. Nincs minden szükséges címlet  11a. Pénz mennyisége a limiten túl van  12a. Nincs elég pénz az ügyfél számláján  14a. A kártya beragadt a gépbe  15a. Ügyfél nem veszi el a kártyát  16a. A pénz beragadt a gépbe  17a. Ügyfél nem veszi el a pénzt   * ATM nem tud kommunikálni a Banking System-el * Ügyfél nem reagál az ATM kérdéseire |
| Üzleti szabályok (business rules) | B1: Megengedett PIN formátumok  B2: PIN próbálkozások száma  B3: A lehetséges szolgáltatások  B4: Felvett összeg lehetőségei  B5: Felvett összeg felső korlátja  B6: A kártyát el kell venni mielőtt a pénzt kiadjuk |
| Nem funkcionális követelmények | NF1: A tranzakció végbemenetelének ideje  NF2: A PIN beütésénél a biztonság  NF3: Az idő a kártya és a pénz elvételére  NF4: Nyelvek  NF5: Látásban korlátozottak támogatása |

Innen, mivel megvan adva a kiinduló állapot és a lépések, minden útvonalat végig tudunk nézni és meg is vannak a tesztjeink. Meg tudunk különböztetni:

* Fő útvonal
* Alternatív útvonalak
* Kivételek (amikor nem érjük el a kívánt eredményt)

Hogy lesznek nekünk felhasználási esetek

A felhasználási esetek a rendszer szempontjából írják le a működését. Az ügyfél igényeire vagy a felhasználás módjára (egy probléma megoldása) támaszkodva tudjuk ezeket felderíteni. Az ügyfél szempontjából íródott felhasználás leírók a felhasználói történetek.

Példák felhasználási esetekre

Például:

* Hogy akarom használni a légkondícionálómat
* Hogy akarom használni a ruhatárat

Error guessing

Ebben az esetben próbáljuk megtippelni, hogy hol lehetnek hibák a rendszerben. Erősen támaszkodik a tesztelő tapasztalataira. Van egy listánk vagy fizikai vagy mentális arról, hogy hol lehet hiba, vagy hibára hajlamos rész. Ezekre írunk célzott teszteket és futtatjuk őket.

* Kell hozzá korábbi tapasztalat hasonló szoftver teszteléséről
* Érteni kell hozzá a tesztelt dolgot
* Kell hozzá ismeret a szokásos implementációs hibákról
* Kell hozzá ismeret a korábban sok hibával rendelkező részekről

Exploratory testing

Ez inkább egy megközelítés, mint technika.

Ebben az esetben a teszteket nem írjuk meg előre, hanem nagyjából egy időben a futtatással. A szolgáltatásokat ellenőrizzük ad-hoc módon. Ha olyan a fejlesztői módszertan, hogy nincs kifejezetten előre megtervezett teszt fázis, és szeretnénk ellenőrizni a fejlesztést még a vége előtt.

Akkor működik jól, ha van egy struktúrája és minden fontosra kiterjed.

Nem követünk script-eket, amik alapján ellenőrizzük a funkcionalitást, inkább együtt gondolkodunk és dolgozunk a fejlesztéssel.

Hogy csináljuk

1. Összeírjuk, hogy milyen hibákra vadászunk (bug taxonomy)
   1. Korábbi projektekben vagy hasonló termékekben talált hibákat vizsgálunk
   2. Megvizsgáljuk ezeknek a hibáknak a gyökerét (root cause analysis)
   3. Összegyűjtjük a hibák kockázatait és végiggondoljuk, hogy mit hogy teszteljünk
2. Összeírjuk, hogy mit csinálunk
   1. Mit tesztelünk (beleértve azt is, hogy mikor tudjuk, hogy megvagyunk (definition of done))
      1. Felhasználói történetek segítenek benne
      2. Vagy követelmények
   2. Hogy lehet tesztelni
   3. Mit keresünk
      1. Teszt ötletek lesznek majd amivel tovább tudunk menni
3. Time box
   1. Meddig csináljuk
   2. Mekkora etapokban tesztelünk (pl 90 perc)
   3. Hányan dolgoznak együtt
4. A tesztek feldolgozása
   1. Defektek feldolgozása
   2. Visszajelzés magunknak és a fejlesztésnek
   3. Fedés ellenőrzése
5. Zárás
   1. Eredmény dokumentálása
   2. Összehasonlítás a célokkal (ez inkább agilis dolog)
   3. Felmérni, hogy van e szükség további tesztelésre (összevetni a definition of done-al)

* Érdemes a felmerült kérdéseket rögzíteni, követni és tisztázni.
* Azt szokták ajánlani, hogy párban érdemes csinálni
* Fontos dokumentumok:
  + Teszt fedettség és összevetése a céllal
  + Kockázatok és hogy kezelve lettek e
  + Teszt végrehajtási log
  + Kérdések és követésük

Használata

Mikor:

* Hiányos vagy hiányzó dokumentáció
* Kreatívnak kell lennünk, vagy szeretnénk azok lenni, vagy a szokványos stratégiák korlátoznak minket
* Kevés az idő/korán kell tesztelni
* Megvan hozzá a tapasztalat
* Kritikus a szolgáltatás és van tapasztalt tesztelő
* Betanítunk új kollégákat a termékre/domain-re

Előnyök:

* Új ötleteket tudunk gyűjteni tesztelés közben
* Meg tudjuk határozni, hogy milyen mélységben akarunk tesztelni
* Kevesebb formai kötöttség

Hátrányok:

* Tapasztalat kell hozzá
* Domain tudás kell hozzá
* Hosszú futási időknél nyűgös
* A hibák reprodukálása gondot tud okozni
* Problémás lehet mindent rögzíteni, ami futtatás közben történik
* A jelentésekhez gyakran saját formátumot kell készíteni