**1. modul**

Introduction to Basics

ISTQB

ISTQB - CTFL

# Bevezetés

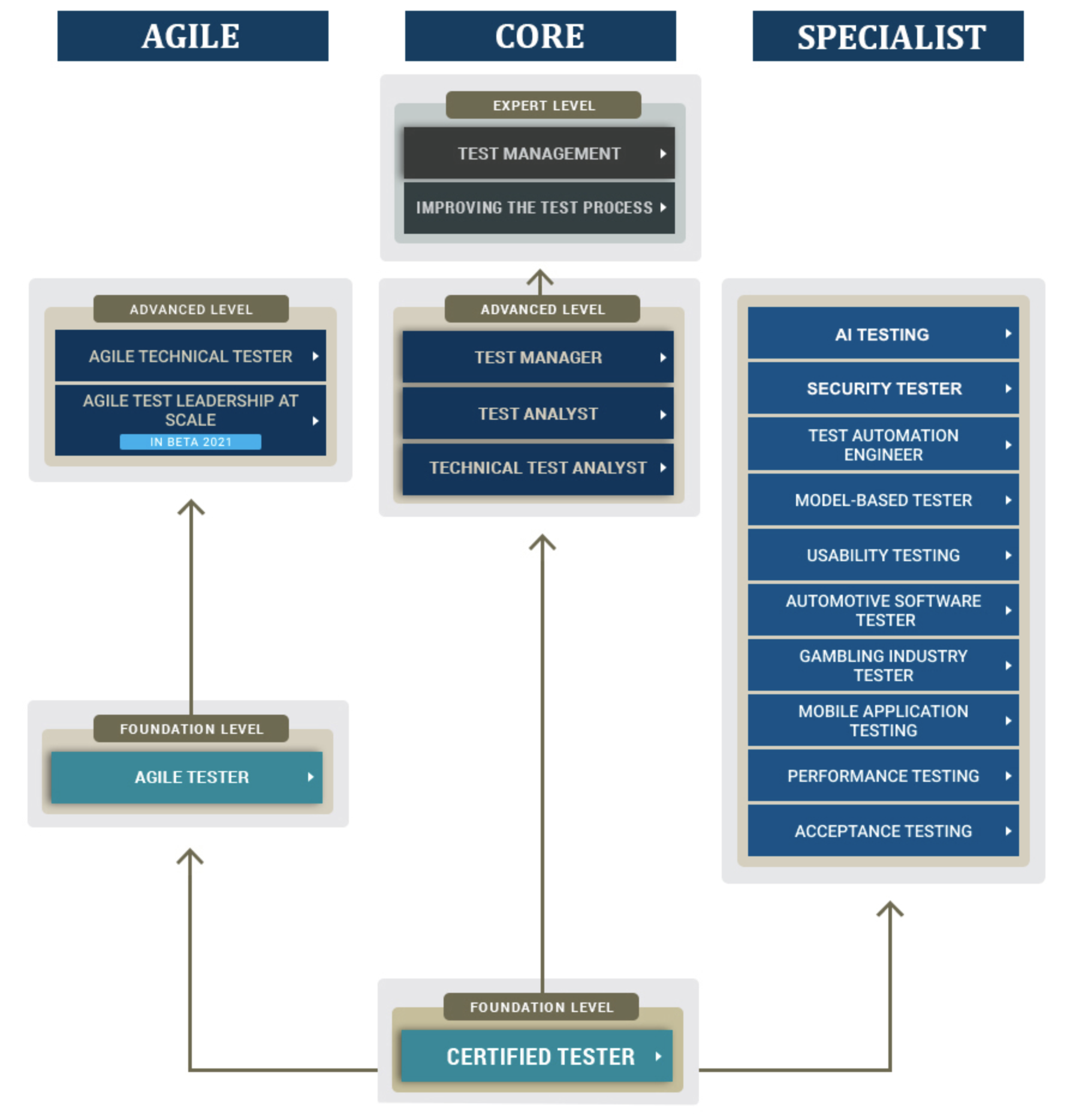
Mint a legtöbb szakmának, úgy a tesztelői szakmának is egyre jobban kidolgozott tudásbázisa, vizsgarendszere és hozzá tartozó tanúsítványok vannak. Ami tízen-x éve még nem is létezett mint különálló szakma, az ma hatalmas közösségeket épít. Ezt pedig leginkább az [ISTQB](https://en.wikipedia.org/wiki/International_Software_Testing_Qualifications_Board), International Software Testing Qualifications Board igyekszik összefogni. Itthon, magyar oldalon pedig a [HTB](https://www.hstqb.org), Hungarian Testing Board.

(Egyébként hasznos tesztelői rendezvények is vannak, úgy mint a Teszt&Tea ami egy meetup sorozat vagy HUSTEF, ami magyar szervezésű, de nemzetközi tesztelői konferencia.)

# Miért jó egy ilyen tanúsítvány?

* magas minőség (maga az anyag illetve maguk az emberek akiknek van ilyen)
* nemzetközi tanúsítvány elismertség
* nemzetközi, nyílt nyelvezet
* nyilvános hozzáférés az anyagokhoz
* függetlenség és folyamatos megújulás

# Tanúsítványok



# ISTQB-CTFL

A képzésen az alapokat, a Certified Tester Foundation Level-hez kapcsolódó tananyag egy részét fogjuk átvenni. Főleg amik a napi munkához és a sikeres interjúhoz kellhetnek.

# Szoftvertesztelés

Egy rendszer vagy program kontrollált körülmények melletti futtatása, és az eredmények kiértékelése. Felelős a szoftver termékekkel kapcsolatos hibák megtalálásáért.

A szoftverminőség-biztosítás a szoftverfejlesztés szerves részét képezi. A szoftverfejlesztési életciklus (SDLC) minden részéhez kapcsolódik. Megállapítja, hogy a szoftver, mint termék teljesíti-e a meghatározott követelményeket, megfelel-e a célnak. Statikus és dinamikus folyamat is lehet, a szabályoknak, előírásoknak megfelelőségét biztosítja. *(A tesztelők azonosítják a pontokat, ahol a rendszer nem az elvárt módon működik).*

**Tesztelésre szükség van, mert:**

* a szoftverben mindig van hiba, hisz emberek fejlesztik
* a hibáknak mindig ára van (hírnév, költség és idő)
* a termékben lévő hibákat még az üzembe helyezés előtt (mielőtt a végfelhasználók használni kezdenék) meg kell találni, hogy a termék minőségét, megbízhatóságát növeljük

**A tesztelés segít:**

* meghatározni, hogy a termék teljesíti-e a meghatározott követelményeket és célokat
* a szoftver fejlesztése, karbantartása és működése során fellépő programhibák és ezekkel járó meghibásodások megtalálásában
* azért használjuk, hogy segítsen csökkenteni a meghibásodások kockázatát egy működő környezetben és hozzájáruljon a szoftver jobb minőségéhez (a tesztelés a fejlesztési kockázatok csökkentésének egyik eszköze)
  + **kockázat**: ami még nem történt meg és lehet, hogy nem is fog soha megtörténni, de a jövőben negatív következményeket okozhat ezért lehetséges problémaforrás, amit kezelni kell! A kockázat kiszámítható: „hatás” és „valószínűség” formájában jelenik meg, ezért vizsgálni kell a lehetséges hatásokat és a bekövetkezés valószínűségét meg kell minden esetben becsülni és menedzselni az adott kockázatot.

## Általános tesztelési tevékenységek:

* Tesztelés tervezése
* Tesztelés irányítása
* Teszt feltételek megválasztása
* Feltételek alapján tesztesetek megtervezése
* Tesztesetek végrehajtása
* Eredmény ellenőrzése
* Teljesítési és kilépési feltételek vizsgálata
* Tesztelési folyamatról és rendszerről jelentés
* Teljesítési (összefoglaló) jelentés készítése

# Tesztelési alapelvek

## Korai tesztelés

A tesztelési ciklust a lehető legkorábban el kell kezdeni.

A szoftver- vagy rendszerfejlesztési életciklus legkorábbi stádiumában kezdjük el, és az előre meghatározott célokra összpontosítsunk – pl. dokumentációk tesztje, komponensek, integráció és rendszer tesztelése) Fő célja: a lehető legtöbb meghibásodást idézze elő

Gazdaságosság: könnyen és olcsón javítható

## A tesztelés függ a körülményektől

A különböző alkalmazásokat különbözőképpen kell tesztelni.

Ha a biztonság kritikus szempont, másképp kell tesztelni, mint egy e-kereskedelmi oldalt, vagy egy telefonos alkalmazást – pl.: háttérbe le lehet rakni telefonon, több típusú telefonon is ki kell próbálni; másképp tesztelünk ha csak 1 napunk lenne letesztelni valamint, mintha 5 napunk)

## Hibák csoportosulása

Ahol már találtunk hibát/hibákat nagy valószínűséggel többet is találhatunk.

A véges idő miatt a tesztelést a legfontosabb modulokra *(amely nélkül használhatatlan a program)* kell végrehajtani. A hibák eloszlása nem egyenletes a programban, a megtalált hibák többsége a legösszetettebb modulban van illetve a legújabb modulokban időszakosan az élesítést követően mindig több van mint később, amikor már használatban van egy ideje.

## Féregirtó paradoxon

Ha mindig ugyanazokat a teszteket futtatjuk, egy idő után az azonos tesztkészlettel nem találunk új hibát.

Az újratesztelés során (pl. regressziós tesztnél) a teszteket, tesztadatokat idővel felülvizsgálni, változtatni, bővíteni kell.

## 

## Hibák látszólagos hiánya

A tesztelés csak a programhibák jelenlétét jelzi. A tesztelés képes felfedni a hibákat, de azt nem, hogy nincs hiba.

A teszteléssel csökken az esélye, hogy a szoftverben felfedezetlen programhibák maradnak, de ha nem találunk hibát, az nem a rendszer hibátlanságának bizonyítéka. Bármennyi fehér hattyút látunk, nem állíthatjuk, hogy minden hattyú fehér.

## Nem lehetséges kimerítő tesztelés

Mindennek ​​minden kombinációját nem lehet kipróbálni.

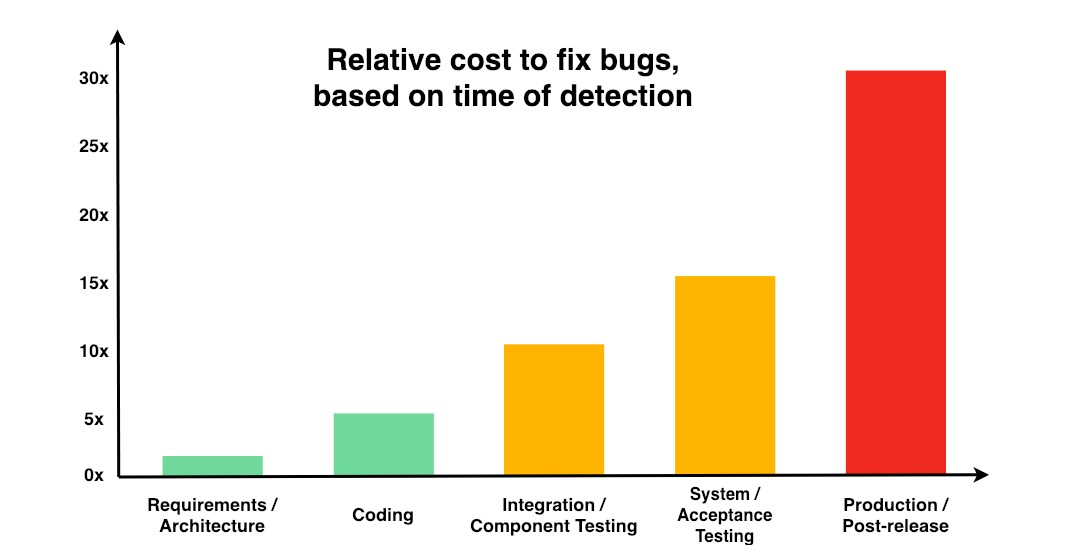
Azaz a bementek és előfeltételek minden kombinációjára – kiterjedő tesztelés nem lehetséges *(helyette elemezzük a kockázatokat és állítsunk fel fontossági sorrendet az összpontosítás hatékonysága érdekében)*

## Hibátlan rendszer téveszméje

A hibák megtalálása és kijavítása hasztalan, ha a fejlesztett rendszer használhatatlan, vagy nem felel meg a felhasználói igényeknek, elvárásoknak.

# Megtérül a tesztelés?

**ROI: Return on Investment** - annak a megállapítása, hogy a befektetés mikor térül meg a többihez képest



Tehát azt lehet mondani, hogy igen megtérül, a hiba kijavításának költsége annál nagyobb, minél később találjuk meg. Akárcsak ha arra gondolunk, hogy ha újra kell végezni több folyamatot + összes addigi tesztet meg kell ismételni.

Célszerű már a tervezési fázisban bevonni a tesztelőket, egyrészt az elkészült dokumentumok ellenőrzésére, másrészta tesztelhető kritériumok megfogalmazására. Ez a **shift-left testing**.

Ha látszólag a rendszerterv követelményeinek megfelelően alakul a fejlesztés, de hibás követelmények vagy hibás rendszerterv alapján kezdődik meg, akkor a fejlesztés során születendő termék bukásra van ítélve.

# 

# Mi az a hiba?

Két fő szempont szerint tudjuk csoportosítani a hibákat.

### Emberi eredetű hiba

Angolul gyakran hívják mistake-nek, bug-nak, defect-nek. Ebbe a kategóriába tartoznak azok a hibák, amely emberi tevékenység által következnek be. Például a fejlesztő egy relációs jel felcserélésével véletlenül engedélyezi, hogy a szerencsejátékkal kapcsolatos oldalát csak 18 éven aluliak használják. Másik ebbe a kategóriába tartozó eset lehet, amikor a felhasználó nem rendeltetésszerűen használja a terméket, ami miatt az meghibásodik. Például a felhasználótól országkód nélkül kérjük a telefonszám rögzítését, de ő azzal teszi ezért helytelen telefonszám kerül elmentésre a rendszerbe. E miatt pedig nem kap meg egy fontos azonosítót a telefonjára.

### Program futásában keletkező hiba

A program futásakor keletkező belső hiba, nem tudja teljesíteni az elvárt működést ezért részlegesen vagy teljesen leállhat, azaz eltér a működése az elvárttól. Ilyen esetek lehetnek, amikor például elfogy a hely a szerveren vagy meghibásodik a szerveren egy hardver. Ez a típusú hiba program failure-höz, program meghibásodáshoz vezethet.

## Lehetséges hibaforrások

* a szoftver és a rendszer specifikációjának, tervezésének és implementációnak hibája
  + a követelmények meghatározása és tervezése során keletkezett programhibák az összes programhiba felét teszik ki (rejtett hiba)
* korábbi hibák, programhibák és meghibásodások lehetséges következményei
* szoftver kódolási hiba (emberi hiba): tesztelés során kiderül, javítható
* helytelen használat (pl. a felhasználók nem az útmutató szerint használják)
* környezeti tényezők *(pl. programhiba: 50 fok van és nincs klíma, akkor túlmelegszik a szerver és zárlatos lesz)*
* szándékos rongálás

## Hibák típusai

* **algoritmikus hibák**: emberi hiba, programozó figyelmetlensége miatt (többsége működés közben derül ki) pl. relációjeles, általában feltételek (értékeket rosszul ír be-házhozszállítás 15E felett ingyenes, ő pedig 150E Ft-ot ír)
* **számítási és pontosságbeli hiányosságok:** az algoritmus, amiből kiindul a programozó helyes, de a kódolás során hiányosan (numerikus, adatstruktúrabeli hibákkal) kerül megvalósítással (pl. banknál kamatszámításnál a programozó csak egész számokat vett figyelembe, pedig a tört számokkal is kell dolgozni)
* **dokumentációs hibák:** legnehezebben felfedezhető, a valós és a dokumentációban leírt működés közötti eltérés (követelményekben megfogalmazott – ált. a tesztelő veszi észre)
* **túlcsordulás hatására bekövetkező hibák:** adatszerkezetek, eljárások túllépik az előre meghatározott memória mennyiséget és a program futása megszakad (irreálisan szélsőséges értékek megadása ami nincs leprogramozva, ezért hiba lesz, nem csinálja amit kértünk)
* **kapacitásbeli problémák:** működés során eléri a teljesítménye határait és bizonytalan állapotba kerül (pl. memóriahiány)
* **hardverrel és rendszerszoftverrel kapcs. probl.:** a hardver és a szoftverkörnyezet nem elégíti ki a dokumentációban foglalt működési feltételeket
* **helyreállítási problémák:** amikor egy más fajta hiba már előzőleg bekövetkezett és a visszaállítás után a rendszer nem úgy működik, ahogy elvárnánk. Pl.:; üzemzavar, aminek adatvesztés lehet következménye

## Hibák csoportosítása

Kategorizálni kell a hibákat, súlyosság és fontosság szerint, hogy a fejlesztőcsapatok tudják rangsorolni a javításukat.

**Fokozatai**: minor (kevésbé), medium, blocker (magas-meg kell oldani) vagy low, medium, high

**Típusai**:

Priority (fontosság): alkalmazás szempontjából – akár a termék használhatatlan is lehet

Severity (súlyosság): ügyfél/felhasználó szempontjából (tesztelő v. ügyfél dönti el)

**Példák**

* Telekom piros logó: használhatóságot nem befolyásolja, az ügyfélnek nagyon sürgős a javítása
* Megszűnik az egyik programnyelv támogatása, amivel működik a program (fontosság: blocker, severity: minor)
* Bankkártyás fizetés (bizonyos megyékből ha fizetnek a program lefagy): lehet ügyfélnek nem fontos, mert az adott megye nem célközönség (severity: minor). Nem lehet fizetni bankkártyával (fontosság: blocker)
* Vessző nincs a szövegben (fontosság: minor, severity: minor)

Sokszor mérlegelni kell a hibák kijavítását, mert lehet hogy a rendszerben a javítás egy vagy több újabb hibát idézhet elő. De olyan is előfordulhat, hogy a hibát csak részben sikerült javítani.

# 

# Minőség

Az a szint, amikor a komponens, rendszer vagy folyamat megfelel a meghatározott követelményeknek és/vagy a felhasználó/ügyfél igényeinek, elvárásainak (fontos megérteni, hogy az ügyfél mit ért minőségen és mik az elvárásai).

Hogy megfelelő-e azt két szempontból kell megvizsgálni

* **Validáció**: helyes-e a specifikáció, jól készítjük-e el a terméket
* **Verifikáció**: a rendszer megfelel-e a specifikációnak

## Mérési szempontok

* megbízhatóság:
  + a meghibásodások között eltelt idő mérése. (pl.: csak akkor lesz kiadva a szoftver ha eltelt 12 óra)
* használhatóság:
  + a felhasználók megkérdezése, elégedettek-e (pl.: NPS)
* ár-érték arány, elérhetőség, ráfordított idő:
  + korlátozni a tesztelés idejét (pl.: a tesztelésnek 7 hét alatt be kell fejeződnie, hogy az adott keretet ne lépje túl)
* szubjektív vélemény, egyén érzései egy termékkel vagy beszerzővel kapcsolatban:
  + ha szeretnek a bizonyos fejlesztői csapattal dolgozni nem gond néhány hiba
* tesztelés elemzése
  + Az eljárások ismerete és a megfelelő alkalmazása. (pl.: csak akkor adják ki, ha kevesebb, mint 5 jelentős programhibát találnak a tervezett tesztekben)

# 

# Tesztelés folyamata

Ami nem folyamat specifikus és minden folyamatrészre igaz, az az hogy a dokumentumok mindenki számára bármikor elérhetőek kell legyenek. **Minden fontos információnak írásos nyoma kell legyen!!!**

## Teszttervezés

**Eredménye**: Projekt célok és követelmények megértése, melynek eredménye a tesztelési küldetés, megbízás.

**Kitűzzük**: a tesztelés célját és feladatait

**Meghatározzuk**: a tesztelési megközelítést és tesztelési stratégiát

### Teszt stratégia (Test strategy)

Magas szintű dokumentum, amely a végrehajtandó tesztelési szinteket írja le, valamint azok részleteit tartalmazza, a szervezetre vagy programra (egy vagy több projektre vonatkozóan). Itt fogalmazhatunk meg tesztelési irányelveket: szervezeti vagy szoftver rendszer szintű alapelveket, amik általános szabályokat adnak a teszteléshez (pl. mindig felülvizsgáljuk a tervdokumentumokat)

### Teszt megközelítés (Test approach)

A tesztelési stratégia megvalósítása egy konkrét projektre. Jellemzően a projekt céljain a kockázatelemzésen alapuló döntéseket, a tesztelési folyamatok kiindulópontját, az alkalmazandó műszaki **tesztterv**ezési technikákat, belépési és kilépési kritériumait, tesztelés fajtáit tartalmazza.

### 

### Tesztterv (Test plan)

* meghatározzuk, hogy a teszteléssel mi az elsődleges célunk *(pl. a követelményeket lefedni tesztesetekkel és azok végrehajtása, majd a felmerülő hibák kijavítása)*
* tesztelés hatókörének (scope) megállapítása. Mire terjed ki a tesztelés és mire nem, milyen más termékek tartoznak a tesztelés hatókörébe, milyen funkciók kerülnek letesztelésre. *(pl.: éles környezetben az adatok más adatbázisból jönnek, ezért példa tesztadatokkal történik)*
* kezelendő üzleti-, termék-, projekt-, technikai kockázatok (ami a tesztelésre jelent kockázatot)
* tesztelési megközelítések meghatározása
  + technikák *(pl. automatizált v. manuális, UI tesztelés- felhasználói felület tesztelés)*
  + tesztelést végrehajtó személyek
  + tesztelési csapatok összetételének kijelölése
  + lefedettség (mit és milyen alaposan kell tesztelni)
  + Kapcsolattartás módja a csapatok és egyének között
* teszteléshez szükséges jelszavak meghatározása *(pl.: előre meghatározott, vagy kitől lehet kérni, esetleg magam is létrehozhatok)*
* beszerzésre kerül az összes szükséges hardver és szoftver
* tesztelési környezetek *(pl. éles környezeten folyik a tesztelés, vagy bizonyos teszt környezeten)*
* ütemterv: teszt elemzésének, műszaki tervezés feladatainak, teszt megvalósításának, végrehajtásának ütemezése, minden teszteléssel kapcsolatos feladat *(pl. 1 hét a követelmények értelmezése, a köv. 2 hétben ezt csinálom, azt csinálom stb.)*
* belépési és kilépési feltételek (entry/exit criteria) meghatározása
  + Belépési feltételek - mik azok a feltételek amiknek teljesülnie kell, hogy elkezdődhessen a munka
  + Kilépési feltételek - mik azok a feltételek amiknek teljesülnie kell, hogy befejezettnek lehessen tekinteni egy munkát

Röviden

|  |  |
| --- | --- |
| Mindenképp tartalmaznia kell | Opcionálisan kell tartalmaznia (cégenként változó) |
| * bevezető * tesztterv azonosító * teszt megközelítés * magas szintű tesztesetek * csapattagok * ütemterv * teszt forgatókönyv * teszt riport eredménye, helye * teszt szintek * kockázatok * feltételezések * belépési feltétel * kilépési feltétel * változások jegyzéke | * tesztegységek * tesztelendő funkciók * nem tesztelendő funkciók * elem sikerének feltétele * felfüggesztési és újrakezdési feltételek * átadandó tesztkimenetek * tesztfeladatok környezeti igények * személyzeti és képzési igények * jóváhagyások * aláírók * dokumentumot megkapják * szükséges erőforrások |

#### 

## Tesztelemzés, műszaki teszttervezés

Az a tevékenység, melynek során a tesztelés általános céljait lefordítjuk kézzelfogható **tesztelési feltételek**ké és **műszaki teszttervek**ké (kidolgozzuk a műszaki teszttervet és tesztelési eljárásokat). Tesztelési feltétel a rendszer egy olyan eleme vagy eseménye, amelyet egy tesztesettel ellenőrizni lehet, pl. funkció, tranzakció, jellemző, minőségi attribútum.

A **tesztbázis** felülvizsgálata azaz az összes dokumentum/követelmény, ami alapjául szolgál a fejlesztésnek. A tesztelő a teszttervet csinálja, a fejlesztő a fejlesztési tervet; Ezek a fő dokumentumok, amelyeken a tesztesetek alapulnak.

A megvalósítandó szoftver **értékelése tesztelhetőség szempontjából** (pl. nem lehet mindent tesztelni – adótorony és telefonhívás kapcsolata vagy a biztosításnak 64 bites titkosítást kell alkalmaznia)

Meghatározásra kerülnek a teszt feltételek és azok fontossága. Olyan feltételek amik a tesztelőnek kellenek ahhoz, hogy elkezdjek dolgozni. *(pl. az alkalmazásból minden nap elejére kerüljön kiadásra a teszt környezetre egy friss verzió)*

**Tesztesetek** megtervezése és priorizálása (kb. milyen tesztesetekre lesz szükség)

**Tesztadatok** meghatározása (adatok, értékek, amik kellenek a tesztesethez, tesztelési eljárások végrehajtásához) – megj.: ha valódi adatbázist használunk, szükséges, hogy ügyeljünk arra, hogy legyen másolat

**Tesztkörnyezet** megtervezése – eszközök (teszteszközök, egyéb felhasznált eszközök: pl.: szövegszerkesztő, projekttervező), infrastruktúra meghatározása *(a tesztelőnek egy elszeparált környezet kell, hogy a tesztek reális eredményt adjanak vissza, ne tudjon más is azon dolgozni egy időben)*

**Nyomonkövethetőség** kialakítása (dokumentumot készítek – pl. tesztesetek tárolom, futási eredményeket riportálom, mindennek legyen nyoma)

## Teszt megvalósítás, végrehajtás

Miután kész a műszaki tesztterv, elkezdődhet a tesztek létrehozása (a tesztelési feltételeket átalakítjuk tesztesetekké és tesztelési eljárásokká, olyan tesztelemekké amik végrehajtható lépések sorozata, mint pl. a teszt szkript, vagy másnéven **teszteset**.

A tesztesetek halmazát **tesztkészlet**nek hívjuk, melybe olyan logikailag összetartozó tesztesetek tartoznak, amik ugyanazon rendszerkomponensre készültek és/vagy egyik teszt utófeltétele egy másik teszt előfeltétele. *(Pl.: van 5 darab tesztesetünk ami egy webáruház kosár funkcióját teszteli.)*. Szokták még hívni **teszthalmaz**nak (test set)

**Tesztkörnyezet ellenőrzése** (meggyőződünk róla, hogy megfelelően be lett-e állítva, működik-e ott az alkalmazás, bizonyos ellenőrző teszteket futtatunk rajta)

Ha minden előfeltétel adott, akkor megkezdődhet a **teszt végrehajtás**, a tesztesetek futtatása, mely során megkapjuk a rendszer aktuális állapotát, viselkedését az adott teszt feltételek szerint.

Naplózás, dokumentálás, a teszt végrehajtáshoz kapcsolódó részletek időrendi rögzítése, azaz a **teszt napló** ( „test log, test record”)

**Kapott és elvárt eredmények összehasonlítása**. A futtatást követően a lényeg, az eltérések issue-ként (incidensként) való jelentése. Amennyiben pedig bebizonyosodik, hogy hiba, akkor hibariport (bug report) felvétele.

Tesztelési tevékenység meg**ismétlése** (előzetesen sikertelen tesztet) – a javítások ellenőrzése miatt újratesztelés, ellenőrző teszt futtatása.

## Kilépési feltételek ellenőrzése és riportálás

Kilépési feltétel: általános és speciális feltételek halmaza, amelyet minden érintettel egyeztetve egy folyamat hivatalos befejezései feltételének tekintjük.

Tesztnaplók és teszttervben foglalt kilépési feltételeinek **összehasonlítása** célja, hogy megakadályozzuk az olyan feladatok befejezettnek tekintését, amelynek még van függőben lévő, be nem fejezett részei. Tehát el kell döntenünk, hogy szükséges-e további teszt?

Ebben a kiértékelésben segíthetnek bizonyos kritériumok:

* Teljesítési feltételek teljesültek-e, például az összes válasz időre vonatkozó tesztesetet lefuttattuk!?
* Lefedettségi feltételek teljesültek-e, például az összes megrendeléshez kapcsolódó tesztesetek lefuttatásra kerültek-e
* Átvételi kritériumok teljesültek-e, például a production, éles környezetben nem találtunk kritikus hibát

**Szoftver értékelése**: a teszt végrehajtását a meghatározott célok fényében kiértékeljük.

**Teszt összefoglaló jelentés** elkészítése. Nem elég ha csak a tesztelők ismerik az eredményeket. Csak így hozhatnak megfontolt döntéseket a szoftverről *(hány teszteset, mennyi sikeres-sikertelen, hány hibát rögzítettem, mely milyen fontosságú, mennyi lett kijavítva, mennyi, ami nincs kijavítva, összefoglalás a tesztelési tapasztalatokról, kinek jelentettem)*

## Teszt lezárása

Több esemény kiválthatja a teszt lezárását, nem csak a **projekt elkészülte és átadása**. Ugyanilyen esemény lehet a projekt leállítása, vagy egy nagy mérföldkőhöz való érkezés.

Ilyenkor a tervezett átadandó elemek **ellenőrzése** egy nagyon fontos feladat, például nyitott kérdések tisztázása.

**Tapasztalatok feldolgozása**, ami segítség lehet a jövőbeni termékekhez vagy projektekhez és a tesztelési folyamat érettségének fejlesztéséhez.

Amit csak lehet **dokumentáljunk** le, mentsünk le a későbbi **újrafelhasználhatóságot** figyelembe véve. (pl.: dokumentáció, programkód, inputok, elvárt eredmények, eljárások, fájlok, adatbázisok, környezetek, ill. bármilyen egyéb szoftver)

A rendszer átvételéről szóló dokumentum előkészítése (**dokumentáljuk** a szoftverrendszer **elfogadását vagy elutasítását**)

# Tesztelési technikák

Célja, hogy segítsen a teszt feltételek, tesztesetek és teszt adatok meghatározásában. Annak eldöntésére, hogy melyik teszt technikát alkalmazzuk az sok tényezőtől függ:

* Komponens vagy rendszer komplexitása
* Szabályozó rendelkezések
* Ügyféligény
* Rendelkezésre álló dokumentáció
* Tesztelő tudása és képességei
* Idő és költségvetés
* Rendszerben várt hibák típusai

Azt, hogy tesztjeinek milyen rendelkezésre álló információ alapján kerül előállításra, aszerint három fő csoportra oszthatjuk.

## Black-box testing (fekete dobozos tesztelés) (nem ismerjük a kódot, nem tudunk semmit sem a fejlesztés hátteréről)

A rendelkezésre álló **specifikáció**k, ügyféligények alapján megírt **tesztek**, anélkül, hogy bármit tudnánk a rendszer belső működéséről vagy a forráskódjáról. *(kizárólag a dokumentum alapján megírt teszteset futtatom le)*

Adott bemenetre tudjuk, milyen elvárt kimenetet kellene adjon az alkalmazás. *(milyen eredménye kell, hogy legyen)*

A teszt feltételek, tesztesetek és tesztadatok a tesztbázisból származnak, ami lehet egy szoftverkövetelmény, bármilyen **specifikáció**, használati eset vagy felhasználói történetek.

Cél, azonosítani a követelmények és megvalósítás közötti eltéréseket.

### Ekvivalenciaparticionálás

Az adatokat partíciókra osztja úgy, hogy az egy partíciókba rakott elemek viselkedése ugyanolyan. Minden érték csak egy partícióba tartozhat. A teszteseteknek minden partícióból legalább egy esetet le kell fedniük.

### Határérték elemzés

Az ekvivalencia partíciók kiterjesztése, mely csak numerikus adatokon alkalmazható. Rendezett partíciók esetén a partíció minimum (alsó) és maximum (felső) értékei maguk a határértékek.

Például: osztályzatokat lehet kiosztani egy digitális naplóban, amibe tegyük fel hogy csak egész számok vihetők be egy helyiértéken.

Így az érvényes tartomány 1-5 lesz. Ekkor három ekvivalencia partíció van

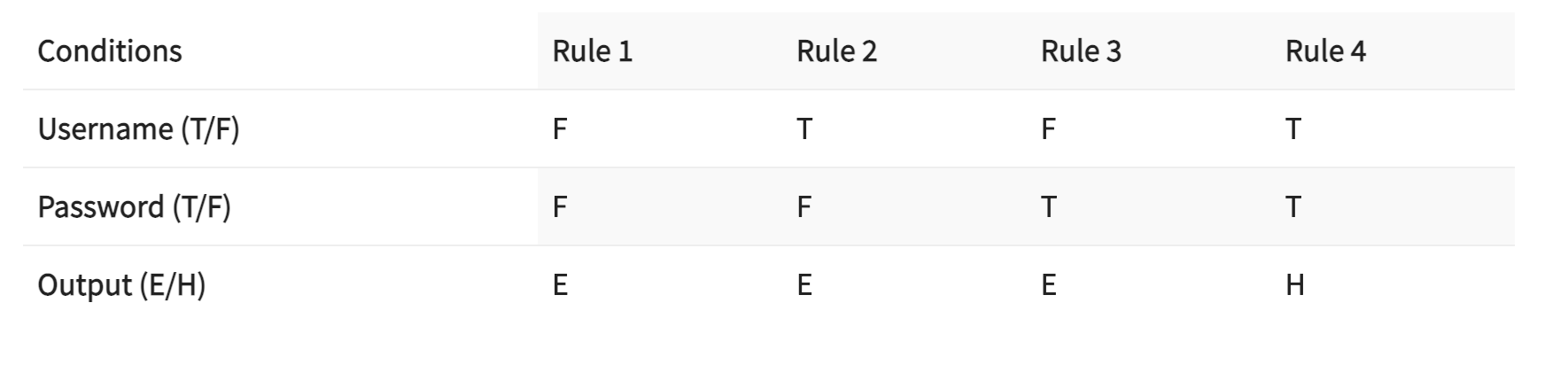
* Érvénytelen túl kicsi érték (**0**)
* Érvényes értékek (**1**,2,3,4,**5**)
* Érvénytelen túl nagy értékek (**6**,7,8,9)

Határértékek a félkövér számok. A legbiztosabb technikák hárompontos határértékek használatát javasolja. (**határérték előtt, határérték és határérték utáni**).

Tehát azokat a teszteseteket kell minimum elkészíteni, melyben a beírt jegy: 0,1,2,4,5,6.

Az ekvivalenciapartíciók határain nagyobb valószínűséggel helytelen a viselkedés, mint a partíciók belsejében.

### Döntési tábla tesztelés

Jól alkalmazhatók olyan komplex üzleti szabályok rögzítésére, melyeket a rendszernek meg kell valósítania. Bemenetek és kimenetek határozzák meg a pontos követelményt, melyek a táblázat sorait jelentik. Minden oszlop egy döntési szabálynak felel meg. 

**Magyarázat:**

T – Helyes username/password

F – Hibás felhasználónév/jelszó

E – Hibaüzenet jelenik meg

H –Sikeresen belépett, megjelenik a kezdőképernyő

**Tesztesetbe ültetés:**

Teszteset 1 – Mind a felhasználónév mind a jelszó hibás. Hibaüzenet jelenik meg a felhasználónak.

Teszteset 2 – Helyes felhasználónév, de helytelen jelszó. Hibaüzenet jelenik meg a felhasználónak.

Teszteset 3 – Helytelen felhasználónév, de helyes jelszó. Hibaüzenet jelenik meg a felhasználónak.

Teszteset 4 – Mind a felhasználónév, mind a jelszó helyes. A felhasználó sikeresen bejelentkezik és betölt a kezdőképernyő.

**Előnyei:**

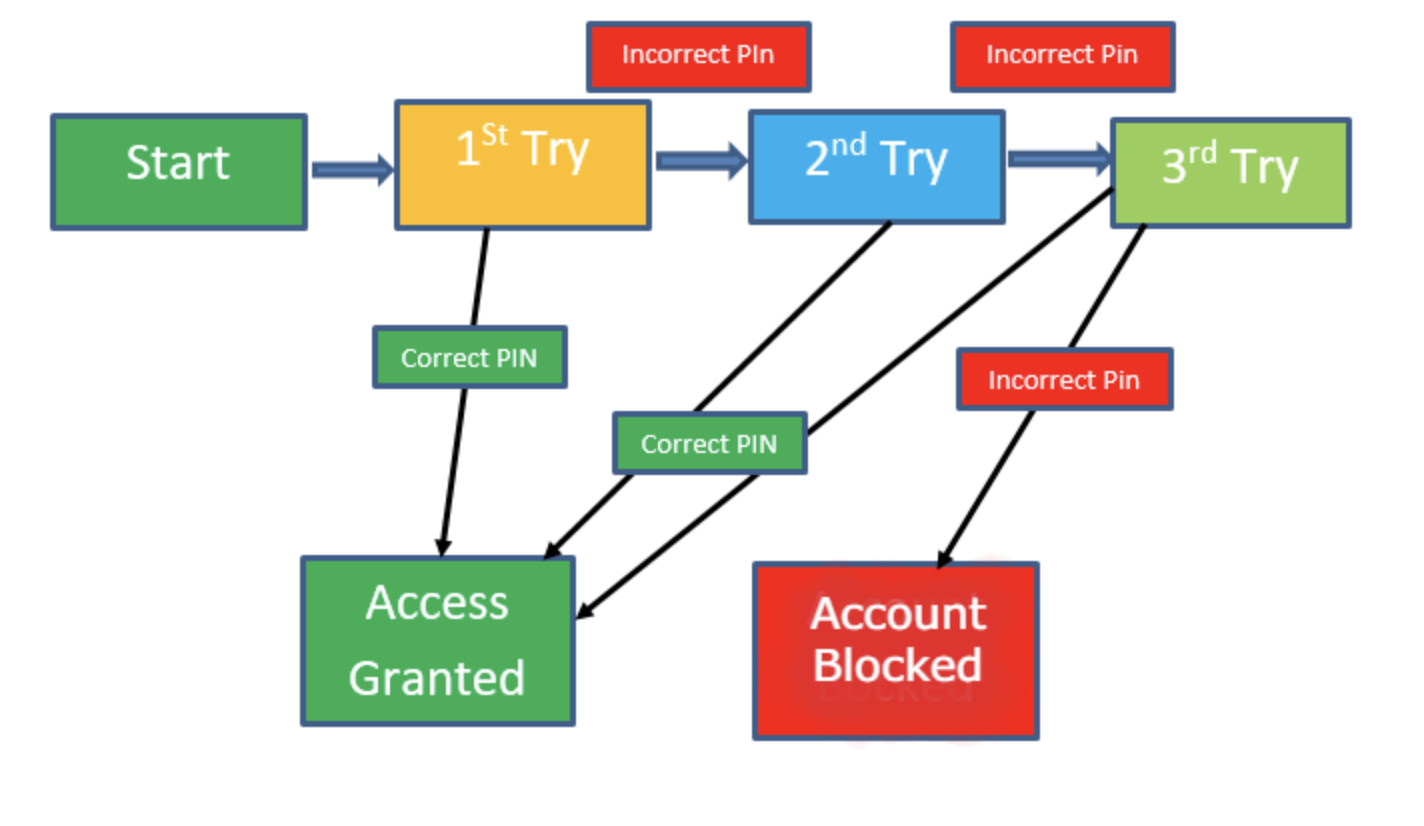
* Amikor a követelmények által leírt különböző viselkedések és bemeneti feltételek nagyon változatosak akkor az ekvivalencia partíció és határérték elemzés technika nem tud segíteni.
* Könnyen kiszúrhatóak a követelmények hiányosságai bizonyos esetekben
* A táblázatból könnyű teszteseteket készíteni és 100%os követelmény lefedettséget biztosítani

**Hátrányai:**

* Nagyszámú feltételek esetén a táblázat tud nagyon összetett és sok oszlopból/sorból álló lenni

### Állapotátmenet tesztelés

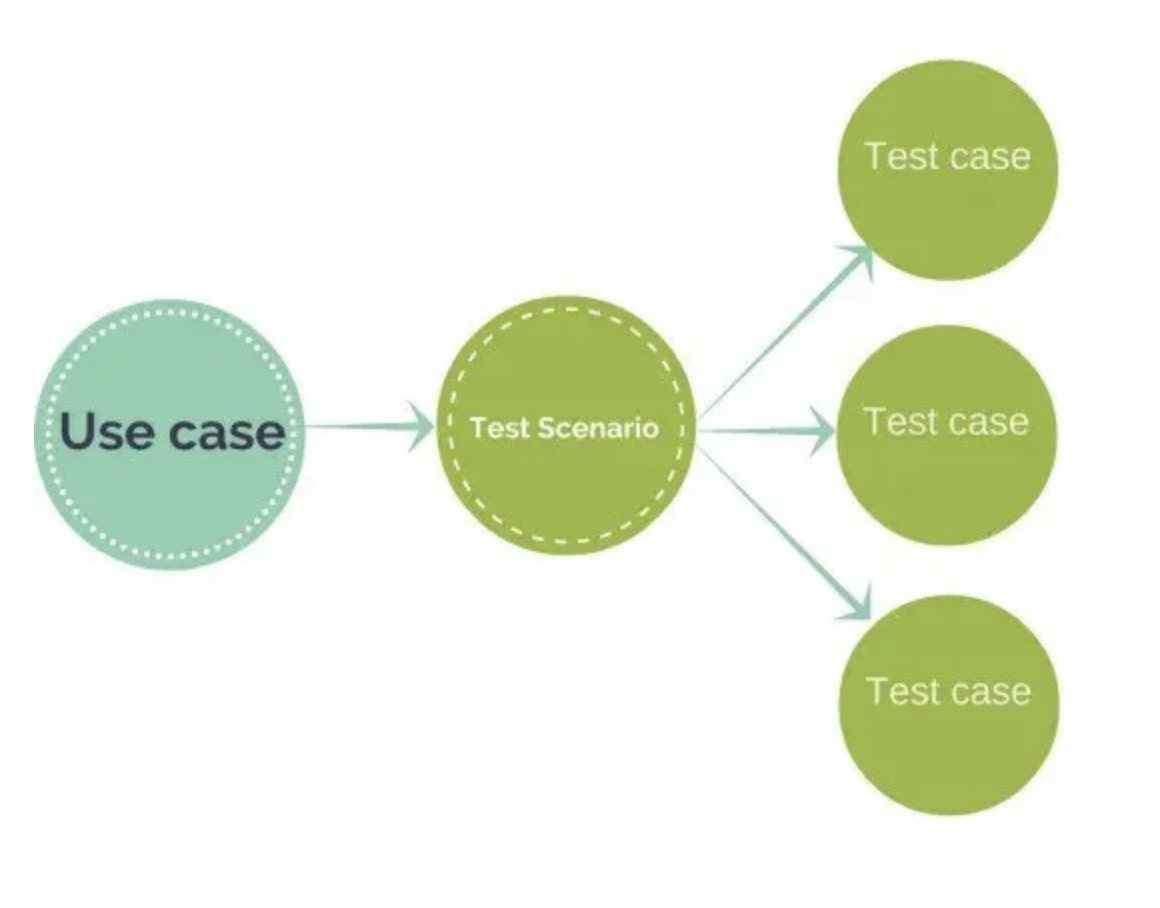
Rendszerek különbözőképpen reagálhatnak az eseményekre a jelenlegi állapotok vagy előzmények függvényében.



### Használati eset tesztelés

A tesztek származtathatók a használati esetekből (use case-ekből). Minden használati eset egy viselkedést határoz meg, amelyet a tárgy hajt végre együttműködésben egy vagy több aktorral.

A használati eset az alábbiakkal írható le: kölcsönhatások és tevékenységek, elő és utófeltételek, illetve maga a leíró nyelv.



**Használati eset - Use case** - “Felhasználó nem tud belépni a helytelen adatokkal”

**Teszt forgatókönyv - Test scenario** - “A felhasználó beírja a felhasználónevet és beírja a jelszót, majd a belépés gombra kattint”

**Teszt esetek - Test cases**

* Felhasználó helyes felhasználónévvel de helytelen jelszóval nem tud belépni
* Felhasználó helytelen felhasználónévvel de helyes jelszóval nem tud belépni
* Felhasználó helytelen felhasználónévvel és jelszóval nem tud belépni

## White-box testing (fehér dobozos tesztelés)

Forráskód alapú, azaz a megírt **forráskód** struktúráján, szoftver architektúra ábrán vagy részletes terven alapuló **teszt**ek. (jó-e a kód felépítése, van-e értelme)

Leggyakoribb formája a **code review**, amikor egy másik személy átnézi és véleményezi az adott kódot. (Az alkalmazás, vagy az automatizált tesztelő által megírt kód) pl.: specifikáció tesztelése - többen elolvassák.

De léteznek alkalmazások, amik jól kidolgozott feltételrendszer, szabályok alapján ellenőrzik a kód minőségét. Ilyen például a Sonarqube.

A kódminőség mellett ebbe a területbe tartozik a kódbázis architektúrája és a fejlesztési munkafolyamat is.

### Utasítás és tesztlefedettség

A tesztek által végrehajtható utasítások és az összes végrehajtható utasítás hányadosa.

### Döntési tesztelés és lefedettség

A döntési tesztelés a kódban lévő döntéseket hajtja végre és a döntések kimeneteként végrehajtott kódot teszteli. Ehhez a tesztesetek az egy döntési pontból kiinduló vezérlési folyamokat követik

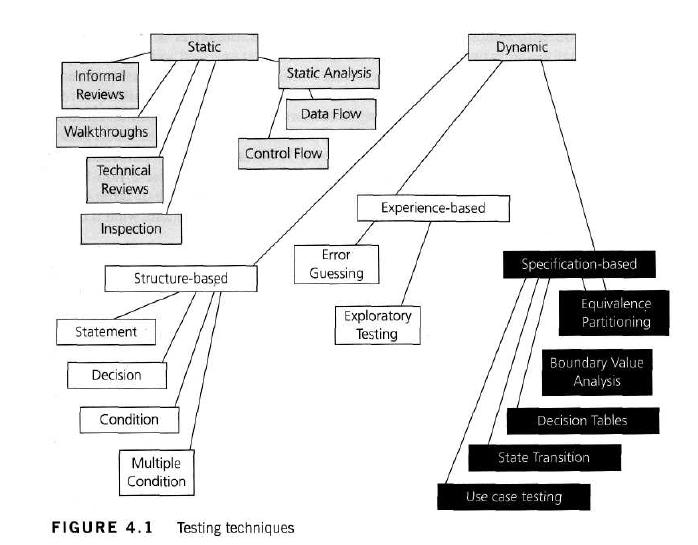
100%-os döntési lefedettség 100%-os utasítás lefedettséget garantál de fordítva ez nem igaz.

## Gray-box testing (szürke dobozos tesztelés)

A fekete és fehér dobozos tesztelés kombinációja.

**Statikus tesztelés**, amikor a kód nem kerül futtatásra (fordítási időben történik) a vizsgálat manuálisan vagy adott eszközökkel (valamilyen ellenőrzéssel) történik. a pl.: kód vagy dokumentumok felülvizsgálatát végezzük manuálisan). Fajtái: felülvizsgálat és statikus elemzés.

**Dinamikus tesztelés**: a forráskód futási időben kerül tesztelésre, a teszt eredményeinek bemutatása céljából futtatjuk. Pl.: a 0-val való osztást vagy olyan elemre való hivatkozást, ami nem létezik. Ezt maga a fejlesztő környezet automatikusan jelzi ha valami nem helyesen van megírva (pl.: kulcsszót kihagy); helyesírás ellenőrző program futtatása a szövegre



## Experience-based testing (tapasztalat alapú tesztelési technikák)

A teszt feltételek, tesztesetek és tesztadatok a tesztbázisból származnak, ami lehet a tesztelők, fejlesztők és felhasználók, érintettek tudása és **tapasztalata.**

### Hibasejtés

Tapasztalat alapú technika, mely során megjósolják a hibák előfordulását a tesztelő tudása, tapasztalata alapján. (Hogyan működött a múltban, milyen tipikus hibák szoktak bekövetkezni…)  
Például: elfogy a hely előbb utóbb az alkalmazás alatt vagy lejár a ceritificate

### Felderítő tesztelés

A felderítő tesztelés során informális (előre nem definiált) teszteket terveznek, hajtanak végre, naplóznak és értékelnek ki a teszt végrehajtása közben dinamikusan. A teszteredményeket arra használják, hogy további ismereteket szerezzenek a komponensről vagy a rendszerről, illetve, hogy teszteket hozzanak létre a további tesztelést igénylő területekhez.

A felderítő tesztelés akkor a leghasznosabb, amikor kevés vagy nem megfelelő specifikáció áll csak rendelkezésre vagy szoros a határidő.

### Ellenőrző lista alapú tesztelés

Ellenörzőlista alapú tesztelés során a tesztelők egy ellenőrzőlistában található tesztfeltételek lefedésére tervezik, valósítják meg és hajtják végre a teszteket.   
Pl.: repülőgép pilótáknak felszállás előtt vagy bevásárlólista

# Tesztelési szintek

A tesztelés során jól elkülöníthető szinteket kell meghatározni. A tesztek megvizsgálják, hogy a visszatérési érték(ek)/viselkedés(ek) megegyeznek-e az elvárttal. Ha igen, akkor a teszt sikeres, amennyiben nem, úgy sikertelen.

Általános elvárás, hogy maguknak a teszteknek ne legyenek mellékhatásaik. (Ne befolyásolják a program működését)

## Egységteszt - Unit test

Az alkalmazás legkisebb önállóan működőképes egységei, moduljai, objektumosztály, alrendszereinek tesztelése. A metódusok végrehajtják az utasításokat, ahogyan meg lett tervezve vagy sem.

(fejlesztők, automatizált tesztelők végzik)

## Komponens teszt - Component test

Az alkalmazás egy jól elkülöníthető komponense, *(például a felhasználói regisztráció során a felhasználói adatokkal foglalkozó önálló egység)* helyesen működik-e.

(fejlesztők, automatizált tesztelők végzik)

### API tesztelés

Annak ellenőrzése, hogy a komponens vagy modul amilyen elérési pontokat tudat a külvilággal, azok a megfelelő bemenő adatok hatására ténylegesen jó adatokat adnak-e vissza illetve megtörténik-e az adott művelet. (Többnyire Api teszteléssel ellenőrzik a frontend és backend végpontok közötti adatok továbbításának pontosságát. Vagyis, jól működnek e a backend által kiajánlott végpontok, a frontend általi lehívás hatására.)

## Komponens kapcsolati teszt vagy más néven Integrációs teszt - Integration test

A modulok és komponensek közötti kapcsolat/együttműködés tesztelésére szolgál. (fejlesztők, automatizált tesztelők végzik)

### Integrációs tesztelés

Például az adatbázisból érkező adatokat feldolgozó modul helyesen kommunikál-e és jól elküldi a megjelenítő modul, a felület számára az adatokat. (hotel-weboldalról megnyitott adatok példa, adatok nem lényegesek, csak, hogy jön-e adat, vagy hibaüzenetet ír ki). Illetve ha a bejelentkező modul egyszer bejelentkeztette a felhasználót és be tudott lépni a felületre, akkor amikor pár perc múlva frissít az oldalon az még mindig emlékszik rá.

Összegyűjt egyes modulokat és megnézi, hogy tudnak-e együtt dolgozni, működni funkcionálisan, mint egy nagyobb egész.

### Pact Contract tesztelés

Egyfajta API teszt kibővítési teszt típus. Nem csak azt ellenőrzi, hogy az adott komponens, adott végpontjai helyesen működnek-e, hanem két komponens végpontjai tudnak-e helyesen, hiba nélkül kommunikálni. Kevésbé van a hangsúly a funkcionalitáson, mint a kommunikáció sikerességén.

## Rendszerteszt - System test

A teljes (integrált rendszer: minden be van kapcsolva, pl. az adatbázisból lekért adatok ténylegesen onnan érkeznek és nem csak kézzel van begépelve; másik alkalmazás segítségével jelenítünk meg adatokat – legyen bekapcsolva – pl.időjárás programja) egészének tesztelése, hogy az alkalmazás megfelel-e a követelményeknek.

Tulajdonképpen a teszttervben szereplő összes teszteset futtatása.

(tesztelők végzik)

### Regressziós tesztelés

Az összes meglévő teszteset lefuttatása annak biztosítása érdekében, hogy az új verzió kiadásával nem-e vezettünk-e be valahol hibát a rendszerbe

### Release tesztelés

Azon tesztek halmaza, amely az adott új verzió kiadásának biztosítására szolgálnak. Ha csak egy bejelentkezés modult érintő funkcióról van szó, akkor az annak a működésének biztosítását lefedő tesztesetekről van szó.

### Smoke tesztelés

Csak azon tesztesetek futtatása amik elégségesek annak eldöntésére, hogy a teljes rendszer működőképes és a **kritikus** funkciók használhatóak.

### Sanity tesztelés

Azon tesztesetek halmaza ami azt hivatott eldönteni, hogy elindult-e az alkalmazás.

## Elfogadási/átvételi teszt - Acceptance test

A szállított termék elfogadása az ügyfél által, vagyis megfelel-e az ügyfél követelményeknek. Kialakul-e a bizalom a rendszer minőségével kapcsolatban illetve megfelel-e a jogszabályi és szabványi követelményeknek.

Cél felhasználók tudják-e és szeretik-e használni, az üzemeltetők tudják-e üzemeltetni és karbantartani.

(Rendszerint az ügyfél, vagy az általa kinevezett felhasználók egy csoportja végzi.)

### Felhasználói elfogadási tesztelés (UAT)

A felhasználók valós vagy szimulált éles működési környezetben validálják azt, hogy a rendszer a leendő felhasználók számára alkalmas lesz-e a munkára.

### Üzemeltetési elfogadási tesztelés (OAT)

A rendszeren az üzemeltetés által végrehajtott elfogadási tesztelés a valós vagy szimulált éles környezeten. Megvizsgálják a többek között a biztonsági mentés és visszaállítás tesztelését, telepítést és eltávolítást, felhasználókezelést, karbantartási feladatokat, adatbetöltési és migrációs feladatokat, sebezhetőség ellenőrzését és teljesítménytesztelést.

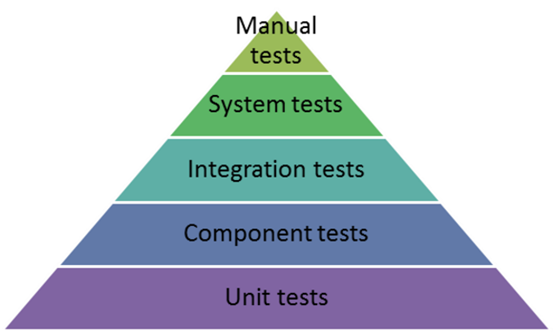
### Szerződéses és szabály alapú elfogadási tesztelés

Szabály alapú elfogadási tesztelést végzünk minden olyan előírás ellenében, amelyet be kell tartani, mint például a kormányzati, jogi vagy biztonsági előírások. A szabály alapú elfogadási tesztelést gyakran a felhasználók vagy a független tesztelők végzik, néha a szabályalkotó végzi az eredmények tanúsítását vagy auditját.

### Alfa és béta tesztelés

Egyfajta visszajelzés megszerzése a végfelhasználóktól az adott funkció vagy a termék végleges bevezetése előtt. Az **alfatesztelés** a fejlesztést végző szervezetnél történik, de a tesztelést nem a fejlesztők, hanem a potenciális vagy meglévő ügyfelek, és/vagy üzemeltetők vagy egy független tesztcsapat végzi. A **béta tesztelés**t a potenciális vagy a meglévő ügyfelek és/vagy az üzemeltetők a saját telephelyükön végzik el. A bétatesztelés az alfatesztelés után vagy előzetes alfatesztelés nélkül is történhet.

# Teszt piramis



# Teszt típusok

## Funkcionális tesztelés

A rendszer funkcionális tesztelése olyan teszteket foglal magában, melyek kiértékelik a rendszer által végzendő funkciókat, viselkedését, így a **fekete doboz technikák** használhatók rá leginkább. Az adott tesztelési szinttől határozza meg pontosan a hozzá tartozó teszt típust.

A funkcionális követelmények leírhatók a munkatermékekben, például az üzleti követelmények specifikációiban, az epic-ekben, a felhasználói történetekben, a használati esetekben vagy a funkcionális specifikációkban, vagy akár dokumentálatlanok is lehetnek. A funkció az, amit a rendszernek tennie kell.

Alapossága a funkcionális lefedettséggel mérhető, legtöbbször százalékos formában. Például a tesztek számának a funkcionalitásokhoz viszonyított aránya.

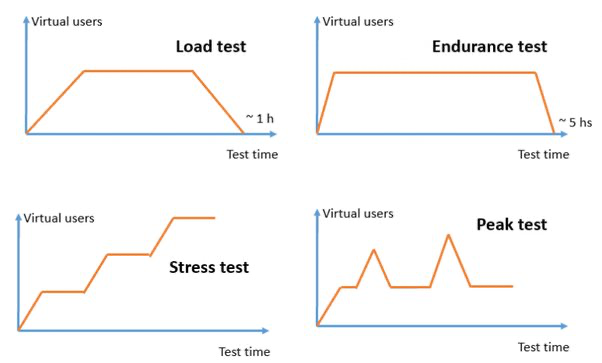
**Az előzőekben említett teszt szintekhez tartozó teszt típusok mind funkcionális teszt típusok.**

## Nem funkcionális tesztelés

A rendszerek és szoftverek jellemzőit vizsgáljuk, például használhatóságot, a teljesítmény-hatékonyságot vagy a biztonságot.

### Teljesítményteszt (Performance test)

Az alkalmazás teljesítményét mérjük különböző szempontok szerint kialakított terhelés alatt.



**Stressztest (Stress test)**

Az alkalmazás hardveres erőforrásait *(elérhető számítási kapacitás, sávszélesség vagy memória)* csökkentjük a normál értéktől, és azt vizsgáljuk hogy az erőforráshiány milyen hatással van a teljesítményre. Vagy annak vizsgálata, hogy a rendszer vagy komponens képes kezelni kiugró terhelést ami meghatározott várható terhelés vagy a feletti mértékű.

**Terheléses teszt (Load test)**

Az alkalmazást vizsgálja olyan szempontból, hogy hogyan viselkedik jól megtervezett, növekvő darabszámú egyidejű kérésekre. Például 1000 felhasználót szimulálva használjuk az alkalmazás pár másodpercen, percen keresztül.

**Kitartási teszt (Endurance testing)**

Olyan terheléses teszt, mely kissé kisebb terheléssel, de huzamosabb ideig, órákig, több napig tarthat. Ez a típusú teszt megerősíti, hogy nem lesz erőforrás probléma (pl.: memóriaszivárgás, adatbázis kapcsolati vagy szálkiosztási hiba) ami azt eredményezheti, hogy végül ronthatja a teljesítményt és/vagy meghibásodást okozhat.

**Csúcsteszt (Spike test)**

Normál terhelést követően, hirtelen egy nagy terheléssel árasztják el a rendszert egy kis időre, majd visszaállítják normál terhelésre a használatot.

**Skálázhatósági test (Scalability test)**

A rendszert bővíthetőségének ellenőrzésére szolgáló teszt, mely során azt vizsgálják, hogy ha új erőforrásokkal bővítik a rendszert, akkor a folyamatok ugyanúgy, stabilan működnek tovább. Illetve ha a több felhasználó nagymértékű adattárolást igényel. Biztosítva azt, hogy a jövőbeli követelményeknek is meg tud felelni a rendszer olyan módon, hogy a teljesítménykövetelmények sérülnek.

**Konkurencia teszt (Concurrency Testing)**

A konkurencia teszt azokra a szituációkra fókuszál, ahol bizonyos események egyidejűleg történnek. Például sok felhasználó egyszerre kattint a bejelentkezés gombra.

**Kapacitás tesztelés (Capacity Testing)**

Meghatározza hogy hány felhasználót és vagy tranzakciót tud a rendszer kiszolgálni olyan módon, hogy még mindig találkozik a teljesítmény célokkal. Ezeket a célokat a tranzakciókból származó adatmennyiségek tekintetében is meg lehet fogalmazni, lehet változtatni.

### Használhatósági teszt (Usability test)

Megmutatja a rendszerről azt, hogy mennyire felhasználóbarát és mennyire könnyen tanulható a használata.

### Nemzetköziesség teszt (i18n)

A rendszer képes-e több nyelven kommunikálni. A különböző országok törvényeinek és szokásainak megfelelően működik-e.

Olyan tevékenységek tartoznak bele, mint a képernyőn látható feliratok fordítása, mértékegységváltás támogatása, lokáció specifikus betűk vagy olvasás irányváltásának támogatása.

### Biztonsági teszt (Security test)

Cégen belüli vagy cél által megbízott külső csapat ellenőrzi, hogy a rendszer mennyire sebezhető rosszindulatú támadások ellen.

### Kompatibilitás teszt (Compatibility testing)

Maga a rendszer képes-e több platformon helyesen működni.

### Accessibility test (a11y)

Hátrányos helyzetű, sérült felhasználóknak mennyire válik lehetővé az alkalmazás használata.

# Teszttípusok és tesztszintek

Lehetőség van bármely teszttípus bármely tesztszinten történő elvégzésére. A szemléltetés érdekében a funkcionális, nemfunkcionális, fehérdoboz és a változásokhoz kapcsolódó tesztelésre mutatunk példákat minden tesztszintre, egy banki alkalmazás esetében, a **funkcionális** tesztekkel kezdve:

* A komponens teszteléséhez a tesztek tervezése azon alapszik, hogy egy komponensnek hogyan kell számítania a kamatos kamatot. (akár egy egyszerű javascript függvény)
* A komponens integrációs tesztelés esetén a teszteket úgy tervezik meg, hogy a felhasználói felületen rögzített számlaadatok átadásra kerüljenek az üzleti logikának. (már összetettebb, mert nem csak a kamatszámítást teszteli, hanem azt is, hogy a rendszerben is jól működik e)
* A rendszerteszteléshez a teszteket úgy tervezik meg, hogy a számlatulajdonosok kérhessenek hitelkeretet a folyószámlájukon. (több funkciót (akár mindet) foglal magában)
* A rendszer integrációs teszteléséhez a teszteket úgy tervezik meg, hogy a rendszer egy külső mikroszolgáltatást használjon a számlatulajdonos hitelminősítésének ellenőrzésére. (külső alkalmazással való együttműködés tesztelése)
* Az elfogadási teszteléshez a tesztek tervezése azon alapszik, hogy a bankár hogyan kezeli egy hitelkérelem jóváhagyását vagy elutasítását. (ez már a felhasználó, vagyis a banki dolgozó)

Példák a **nemfunkcionális** tesztekre:

* Komponenstesztelés esetén a teljesítményteszteket egy komplex kamatos kamat számításhoz szükséges utasításciklusok számának kiértékelésére tervezik. (komplexebb számítás hatása a teljesítményre)
* Komponens integrációs tesztelés esetén a biztonsági tesztek a puffertúlcsordulási sebezhetőséget célozzák meg, mivel itt a felhasználói felület az üzleti logikának továbbít adatokat.
* A rendszertesztelésnél a hordozhatósági vizsgálatok célja annak ellenőrzése, hogy a prezentációs réteg minden támogatott böngészőn és mobil eszközön működik-e.
* A rendszer integrációs tesztek során a megbízhatósági teszteket úgy tervezzük, hogy értékeljék a rendszer robusztusságát, ha a hitelbírálati mikroszolgáltatás nem válaszol.
* Az elfogadási tesztelés esetén a használhatósági tesztelés célja, hogy értékelje a bankár hitel kezelési felületének hozzáférhetőségét a fogyatékkal élők számára.

Példák **fehérdoboz** tesztekre:

* Komponenstesztelés esetén a teszteket úgy tervezik, hogy teljes utasítás- és döntési lefedettséget érjenek el minden olyan komponens esetében, amely pénzügyi számításokat végez.
* Komponens integrációs tesztelésnél azt teszteljük, hogy a böngésző interfésze minden képernyőn átadja-e az adatokat a böngésző következő képernyőjének és az üzleti logikának.
* A rendszerteszteléshez a teszteket úgy tervezzük, hogy azok lefedjék azon weblapokat, amelyek a hitelkeret alkalmazás során megjelenhetnek.
* A rendszer integrációs tesztek során a teszteket úgy tervezzük meg, hogy a hitelbírálati mikroszolgáltatásra küldött összes lehetséges lekérdezési módot végrehajtsa.
* Az elfogadási teszteléshez a teszteket úgy tervezzük, hogy lefedjék az összes támogatott pénzügyi adatfájl struktúrát és a banki átutalások értéktartományait.

Végül példák a **változásokhoz** kapcsolódó tesztekre:

* A komponensteszteléshez minden egyes komponensre automatizált regressziós tesztet készítünk és a folyamatos integráció keretrendszerébe illesztjük.
* Komponens integrációs tesztek esetén a teszteket az interfészhez kapcsolódó hibák javításának megerősítéséhez tervezzük, mivel a hibajavításokat a kódrepositoryban ellenőrzik.
* A rendszerteszteléshez az adott munkafolyamat minden tesztjét újra végrehajtják, ha a munkafolyamat bármely képernyője megváltozik.
* A rendszer integrációs tesztelés során a hitelbírálati mikroszolgáltatással kölcsönhatásban álló alkalmazás tesztjeit naponta újra végrehajtják a mikroszolgáltatás folyamatos telepítésének részeként.
* Elfogadási tesztelés esetén az összes korábban sikertelen tesztet újra végrehajtják, miután az elfogadási tesztben talált hiba javítása megtörtént.

Nem szükséges az alkalmazásoknál minden teszt szintre vonatkozóan minden teszt típust alkalmaznunk.

# Dokumentumok

Át fogjuk venni, hogy milyen dokumentumokkal találkozhatunk munkánk során és mely dokumentum mit kell, hogy tartalmazzon.

De mik is azok a követelmények: **követelménynek, specifikációnak** nevezünk minden olyan dokumentációt, ami a termék sikeres kiadását biztosítja (abban az esetben, ha a termék tükrözi a követelményeket, a követelmények pedig tükrözik az ügyfél igényeket. ) Olyan feltétel vagy képesség, amely a felhasználó számára egy probléma megoldásához vagy egy adott cél eléréséhez szükséges.

A tesztelések egy részénél arra összpontosítunk, hogy összevetjük a terméket annak specifikációjával, majd tesztelőként továbbítjuk a projekt megfelelő résztvevőinek, hogy megítéljék a minőséget. Megvizsgáljuk, mennyire felel meg a rendeltetésnek, a felhasználóknak.

## BRD - Business Requirements Document

Követelmények az ügyféltől, az ügyfél szemszögéből mely leírja a rendszer jellemzőit. Az egyes üzleti folyamatoknak, logikának hogyan kell működnie, azaz válaszol arra a kérdésre, hogy ügyfél mit akar elérni. Mi az a probléma, amit az ügyfél meg akar oldali a különböző megszorításokat figyelembe véve, a megadott idő alatt.

Egy könnyebben érthető gyakorlati eset, amikor egy meglévő sokszor papíron végzendő kézi folyamatot öntenek digitális formába. Például a sárga csekkes postai befizetést kiváltó csekkbefizető weboldalak, applikációk működésének üzleti leírása - a csekk mely mezőinek mi a feladata, melyek kötelezőek és milyen megszorításaik vannak.

Tartalmazza:

* termék okát, a kiváltásra szánt munkafolyamat leírását, hátterét
* üzleti szereplőket (stakeholders)
* célt, amit hogyha meg tud valósítani a termék, azt mondhatjuk hogy az sikeres
* kapcsolódó külső rendszereket (ha van ilyen)
* modelleket, ábrákat az üzleti folyamatokról
* szótárat, az üzleti kifejezések megértéséhez

A fő oka, hogy közös megegyezés legyen minden érintett között a terméktől minden oldalról elvárt igényeit illetően.

## SRS - Software Requirements Specification

A BRD alapján szokta elkészíteni a fejlesztő csapat egyik, általában magasabb senioritású tagja. Részletes leírása annak, hogy hogyan is kellene a terméknek működnie. Célja az, hogy csökkentse a későbbi termék újraírás esélyét. Elkerülje azt, hogy nem az ügyfél igényeinek megfelelő termék készüljön el. Tartalmazni szokott becslést (estimation) az egyes funkciók leszállítására és azok határait, feltételezéseket, kockázatokat és ütemtervet is. A fejlesztőknek, tesztelőknek szükséges hogy közös és egyértelmű megértésük legyen a szállítandó terméket illetően. Funkcionális és nem funkcionális követelmények is fellelhetőek benne.

**Funkcionális követelmény:** adott terméknek vagy a termék egy adott moduljának mit és hogyan kell tudnia végrehajtania. Például: helytelen adatokat megadva a belépés gombra kattintva *Hibás adatok* hibaüzenet jelenik meg.

**Nem funkcionális követelmény:** leírja, hogy a termék elkészültekor a funkcióknak hogyan kell működniük és milyen körülmények között. Például: a helyes adatokat megadva belépés gombra kattintva 2mp-en belül jelenjen meg a főképernyő.

A jó SRS tartalmaz használati eseteket (**use-case**-eket). Például:

**Adott**, egy regisztrált felhasználó aki,

**amikor** a Belépés gombra kattint,

**akkor** az alkalmazásnak be kell lépnie.

(Given-When-Then formula) Amennyiben ez előre van meghatározva és a fejlesztők ezen use-case-ek alapján dolgoznak, azt nevezzük Behavior Driven Developmentnek (**BDD**) azaz Viselkedés vezette fejlesztés.

## 

## Tesztelési alapelv (Test policy)

Magas szintű, általánosan megfogalmazott dokumentum, amely a szervezet elveit, megközelítési módját, céljait mutatja be a tesztelésre vonatkozóan.

## Teszt stratégia (Test strategy)

Leírást lásd fentebb. Példa [teszt stratégia](https://docs.google.com/document/d/1J-NuP_bkG00ULSEtsYzRX3BNgvWBTovYsQMA2Zf8yDo/edit?usp=share_link).

## 

## Tesztterv (Test plan)

Leírást lásd fentebb. Példa [teszt terv](https://docs.google.com/document/d/1jrXo3nK5GG2wkPfL2uu77OHj8QaXbVysTLo53ZF5RDA/edit?usp=share_link).

## Teszteset (Test case)

Végrehajtási előfeltételek, adatok (bemeneti értékek), lépések halmaza, amik a várt eredmény, viselkedés eléréséhez vezetnek, amelyet egy konkrét célért, vagy tesztért fejlesztettek.

* High-level test case: teszteset melyben nem konkretizáljuk a bemenő teszt adatokat, illetve nem részletezzük az elvárt eredményt
* Low-level test case: melyben már minden részletesen ki van fejtve, minden feltétel adott, hogy bárki bármikor le tudja futtatni a teszt esetet az alkalmazás ismerete nélkül is

**Tartalmaznia kell:**

* **Cím**: rövid jellemzés
* **Teszteset azonosító**: pl.: teszteset05
* **Leírás**: rövid jellemzése a tesztelendő funkciónak (pl. az összes a-betűvel kezdődő termék listázása)
* **(Előfeltétel)**: bármi, ami szükséges ahhoz, hogy el tudjam kezdeni a teszteset futtatását (pl. böngésző megnyitása és alkalmazásba való belépés)
* **Teszt lépések**: amit csinálok és ami kell, hogy történjen arra az eseményre (pl. a regisztrál gombra kattintok – ekkor regisztrálás sikeres üzenet jelenik meg)
* **Teszthalmaza vagy tesztforgatókönyv azonosítója**: minden tesztesetet hozzárendelünk 1 v. több teszthalmazba v. tesztforgatókönyvbe (pl. belépéses teszt, vagy kapcsolatform)

Példa: [teszt eset](https://docs.google.com/document/d/1VlPBT_cNsY2NbGRQfB0QqWQq1BQXyUxLJ0c-RGr3ufI/edit?usp=share_link)

## Hibariport

Minőségbiztosítási szempontból a legfontosabb az, hogy a termék kiadását követően ne találjon az ügyfél, felhasználó hibát (**defect**-et). A cél ezek mihamarabbi megtalálása, kiszűrése. Amikor az alkalmazás használata vagy tesztesetek futtatása során valami nem az általunk elvártnak megfelelően viselkedik, történik akkor azt problémának (**issue**nak) hívunk. Ennek a fő oka az, hogy attól hogy szerintünk az "hiba", még lehet az ügyfélnek az a természetes és helyes viselkedés. Illetve lehet, még ami ezt a "hibát" okozza, még nincs is a fejlesztők által lefejlesztve. Ezért szokás egyre jobban azt mondani, hogy issue-t rögzítünk (ami később persze kiderülthet hogy defect).

Példa: [hibariport](https://docs.google.com/document/d/1MJV3HAnNNEn0fji68-kWEcr2-TUbMXk4Ilaos1M-iRg/edit?usp=share_link)

**Tartalmaznia kell:**

* Hiba neve és azonosítója
* Terület ahol jelentkezik a hiba
* **Alkalmazás verziószáma**
* **Súlyosság/fontosság**
* **Környezet**
* Személy aki találta
* Előfordulásának ideje
* **Hiba reprodukálásának lépései**
* **Aktuális eredmény**
* **Elvárt eredmény**
* **Logok/screenshotok**
* Teszt adat (test user)
* Platform/Böngésző és böngésző verzió
* Egyéb (linkelt teszt eset, linkelt fejlesztés, megjegyzés…stb)

## Teszt jegyzőkönyv (Test completion report)

Más néven teszt riport.A tesztelők a tesztforgatókönyvnek megfelelően elvégzik a tesztelést és az **eredményt teszt jegyzőkönyv**ekben dokumentálják. A teszt kimenetelét minden esetben jelenteni kell a teszt koordinátornak. Amennyiben a hiba fennáll, és harmadik félen múlik a megoldása eszkalálni kell a problémát a projekt menedzsernek.

Gyakran tartalmazza magát a **teszt jelentés**t. Ami egy dokumentum, ami összegzi a tesztelési tevékenységeket és futási eredményeket mindenki számára érthető formában.

Példa: [tesztjegyzőkönyv](https://docs.google.com/document/d/1lOB34ddaIXe_1Uwgwd9DWUAhruPbV2SUmcsgEE6tuzQ/edit?usp=share_link)

# 

# 

# Feladatok

## ------Ekvivalencia partíció és határérték elemzés feladatok------

1. Egy formon lévő beviteli mező 18 és 25 közötti számokat fogad el.
   1. Mik a valid/invalid ekvivalencia partíticók?
   2. Melyek azok a tesztesetek, amikkel minimum tesztelnéd?
2. Egy Jr. Tesztautomatizáló vizsgán a jelöltnek 24 pontot kell elérjen, hogy átmenjen. A vizsga maximuma 40 pont.Mik a valid/invalid ekvivalencia partíticók?
   1. Mik a valid/invalid ekvivalencia partíticók?
   2. Melyek azok a tesztesetek, amikkel minimum tesztelnéd?
3. Egy formon lévő beviteli mező alfanumerikus értékeket fogad el.
   1. Mik a valid/invalid ekvivalencia partíciók?
   2. Melyek azok a tesztesetek, amikkel minimum tesztelnéd?
4. A klíma automatikusan kikapcsol, amikor a hőmérséklet 18 fok alá esik és bekapcsol, ha a hőmérséklet több mint 21 fok.
   1. Mik a valid/invalid ekvivalencia partíciók?
   2. Melyek azok a tesztesetek, amikkel minimum tesztelnéd?
5. Egy tetszőleges program numerikus input mezőjének validálása a következő: ha a bevitt érték kisebb, mint 10, akkor el van utasítva, ha 10 és 21 közötti, akkor elfogadásra kerül, de amennyiben az nagyobb vagy egyenlő, mint 22 akkor szintén el van utasítva.
   1. Mik a valid/invalid ekvivalencia partíciók?
   2. Melyek azok a tesztesetek, amikkel minimum tesztelnéd?
6. Egy tetszőleges rendszer arra lett tervezve, hogy az alábbi módon működjon az adó befizetése kapcsán:
   1. A munkavállaló 4000 £-ig adómentes.
   2. Ha a munkavállaló ezen felül 1500 £-vel többet keres, akkor az adó 10%
   3. Ha a munkavállaló az előbbin felül 28000 £-vel többet keres, akkor az adó 22%
   4. Minden további az előbbieken felüli összeg 40%-os adó

Az alábbi hármasok közül melyik az, amelyik legalább három ekvivalencia partícióba esik?

a) £4000; £5000; £5500

b) £32001; £34000; £36500

c) £28000; £28001; £32001

d) £4000; £4200; £5600

1. Az alábbi specifikáció alapján az alábbi korok közül melyik esik bele ugyanabba az ekvivalencia partícióba?
   1. Követelmény 1: Ha fiatalabb, mint 18 akkor túl fiatal biztosítottnak lenni.
   2. Követelmény 2: 18 és 30, valamint ezeken közötti korúak 20% kedvezményt kaphatnak
   3. Bárki aki 30 felett van, semmilyen kedvezményre nem jogosult, de biztosítást köthet.

a) 17, 18, 19

b) 29, 30, 31

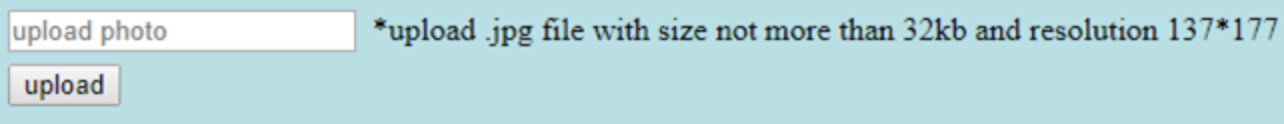
c) 18, 29, 30

d) 17, 29, 31

## 

## 

## ------Döntési tábla feladatok------

1. Adott az alábbi fotó feltöltési felület. 
   1. Vedd fel a döntési táblát

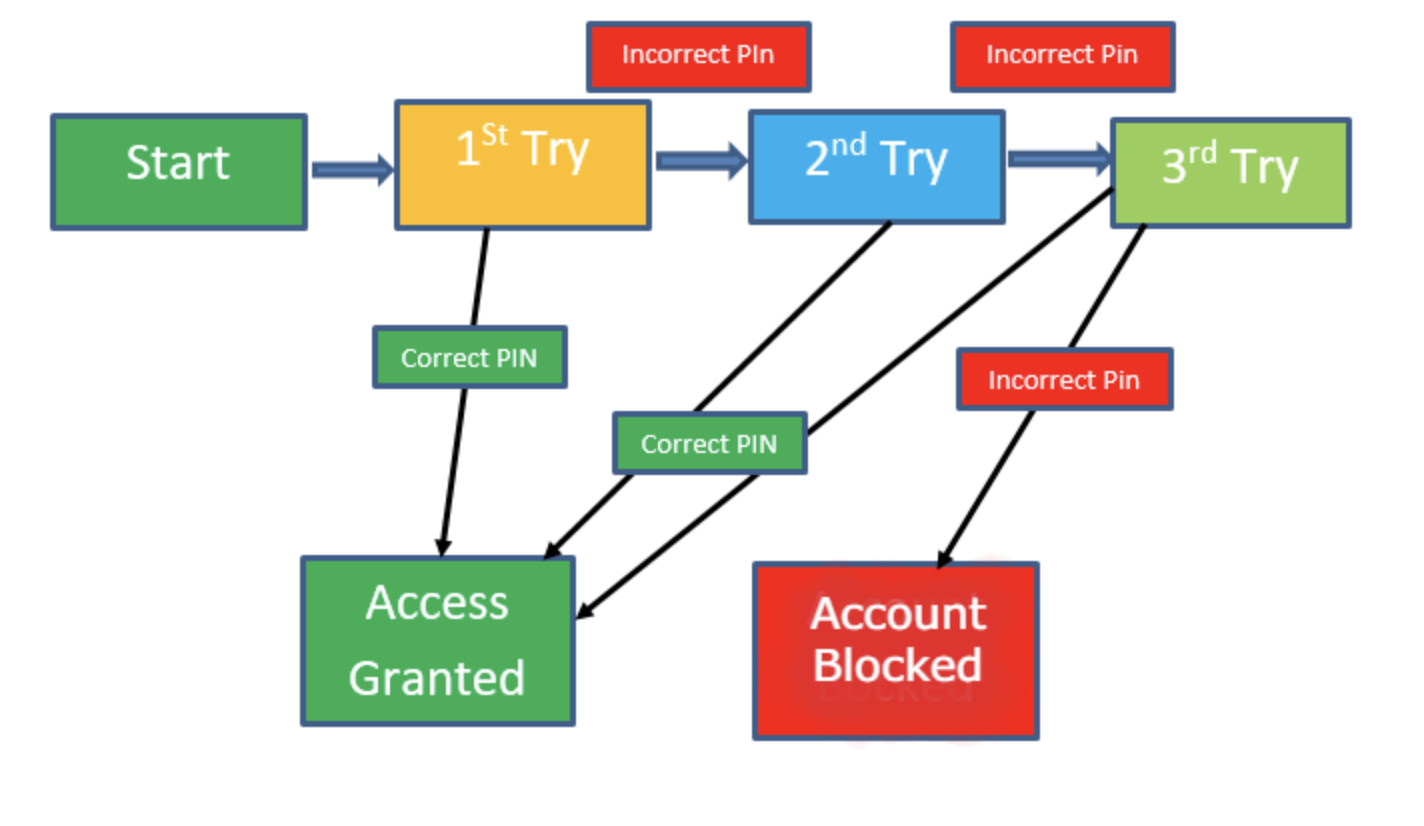
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feltételek / Hatás** | **Szabály 1** | **Szabály 2** | **Szabály 3** | **Szabály 4** | **Szabály 5** | **Szabály 6** | **Szabály 7** | **Szabály 8** |
| **.jpg (Igen/Nem)** |  |  |  |  |  |  |  |  |
| **32kb (Igen/Nem)** |  |  |  |  |  |  |  |  |
| **137\*177 (Igen/Nem)** |  |  |  |  |  |  |  |  |
| **Eredmény - Hatás (Sikerült/Hiba)** |  |  |  |  |  |  |  |  |

* 1. Milyen tesztesetek lehetne megfogalmazni a döntési tábla alapján?

1. Az online rendelések házhozszállítás díját a szállítási mód alapján határozzák meg. Budapestre szállítás esetén express 1.500 Ft, normál 1.000 Ft. Országos szállítás express esetén 2.000 Ft, normál 1.500 Ft.
   1. Vedd fel a döntési táblát
   2. Milyen tesztesetek lehetne megfogalmazni a döntési tábla alapján?
2. Egy áruház pontgyűjtő kártyát bocsát ki. Minden vásárló, akinek van ilyen kártyája, minden vásárlása során dönthet, hogy 5% kedvezményt kér a számla összegéből, vagy a kártyán lévő pontjait növeli meg. Az a vásárló, akinek nincs ilyen kártyája, szintén megkaphatja az 5% kedvezményt, ha 50.000 Ft felett vásárol.
   1. Vedd fel a döntési táblát
   2. Milyen tesztesetek lehetne megfogalmazni a döntési tábla alapján?
3. Adott az alábbi követelmény: Adott 10 lakás egy társasházban, amikor azon házban szól a riasztó, ahol minden héten megszólal, akkor hagyjuk figyelmen kívül. Egyébként ellenőrizzük a lakást és ha az ajtó/ablak nem tűnik gyanúsnak, akkor szintén hagyjuk figyelmen kívül. Egyébként hívjuk a rendőrséget.
   1. Vedd fel a döntési táblát
   2. Milyen tesztesetek lehetne megfogalmazni a döntési tábla alapján?
4. Adott egy hotel foglalási oldal, amely ingyen üveg bort ad az első foglalóknak valamint a törzsvendégeknek (3 vagy a feletti foglalás~~/év~~). Ha a vendégnek születésnapja van az ottlét alatt, akkor ajándék csokiválogatást kapnak, de csak akkor ha nem kaptak még ajándék bort. A törzsvendégek születésnap alkalmával viszont 2 üveg bor jár. Viszont minden vendégnek akik kevesebbszer voltak a hotelben, mint 2 alkalom~~/év~~, azoknak ki kell tölteni egy elégedettségi kérdőívet távozáskor.
   1. Vedd fel a döntési táblát
   2. Milyen tesztesetek lehetne megfogalmazni a döntési tábla alapján?
5. Adott egy banki rendszer bejelentkezési felülete, aminek 4 különböző követelményt kell teljesítenie, figyelembe venni. ID érvényes vagy nem, password érvényes vagy nem, kétfaktoros azonosítás sikeres vagy nem illetve van-e hozzáférés engedélyezve a rendszerhez vagy sem. Érvénytelen ID esetén “Hiba: Érvénytelen ID” hibaüzenet jön, helytelen password esetén “Hiba: Érvénytelen jelszó” hibaüzenet, illetve amennyiben a hozzáférés meg van tagadva “Hiba: Fiók zárolva” hibaüzenet érkezik.
   1. Vedd fel a döntési táblát
   2. Milyen tesztesetek lehetne megfogalmazni a döntési tábla alapján?

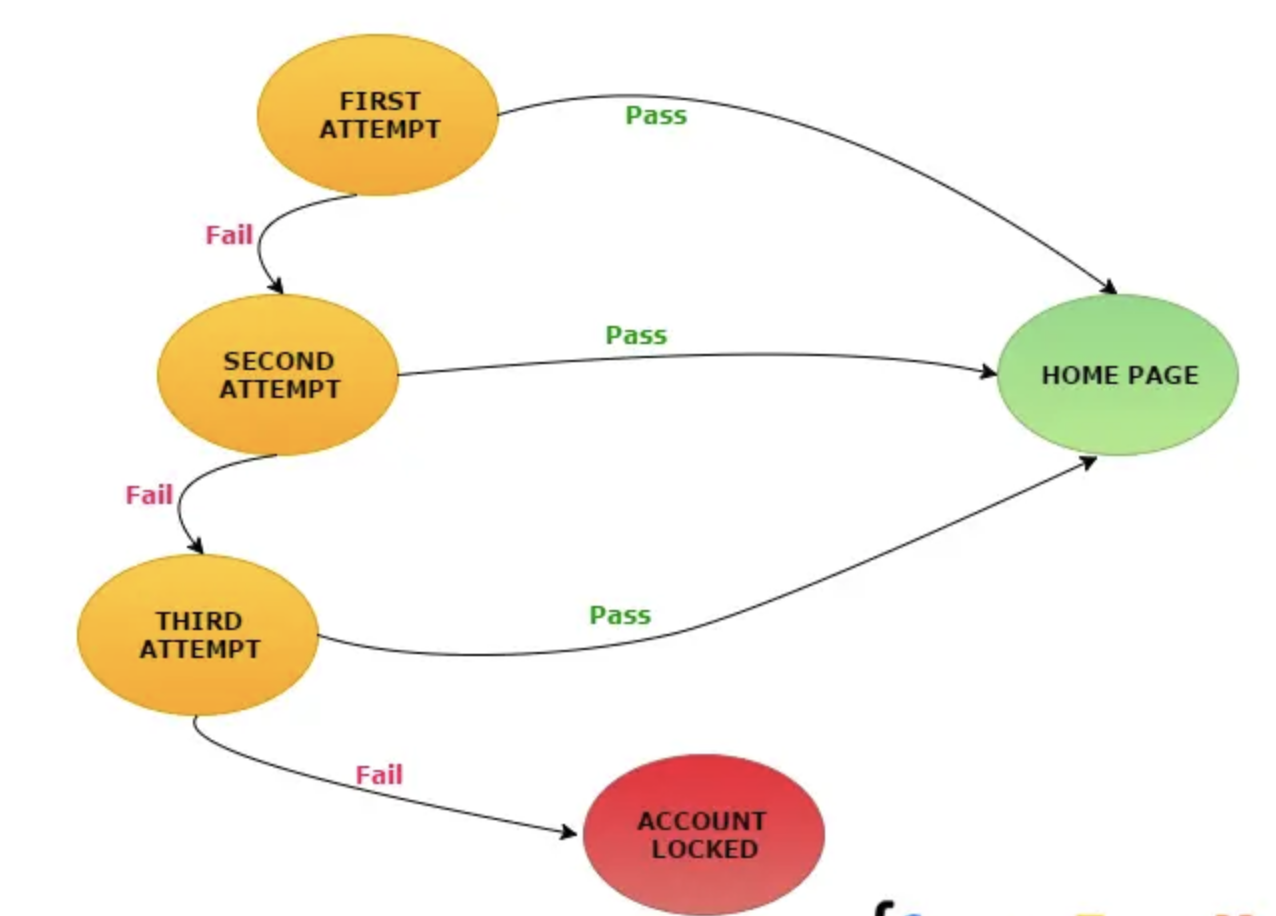
## ------Állapotátmenet feladatok------

1. Adott az alábbi állapotátmenet:



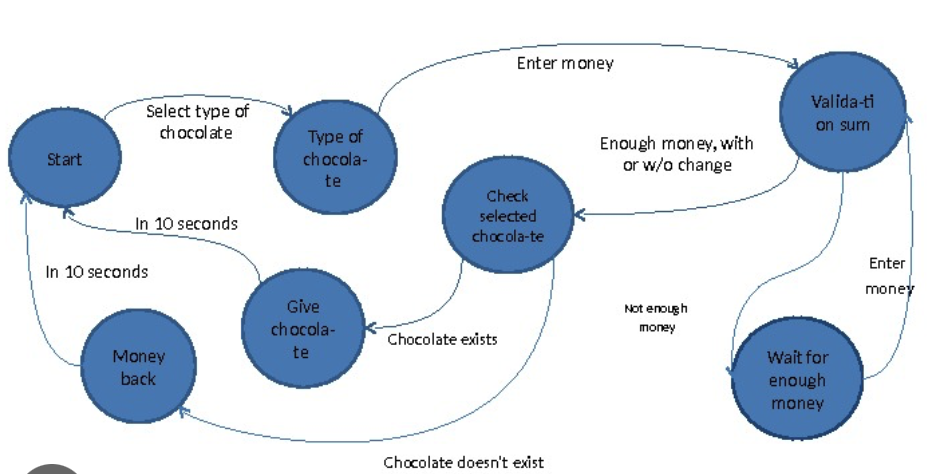
* 1. Milyen állapotátmenet táblázatot vennél fel?
  2. Milyen teszteseteket írnál a táblázat alapján?

1. Adott az alábbi állapotátmenet:



* 1. Milyen állapotátmenet táblázatot vennél fel?
  2. Milyen teszteseteket írnál a táblázat alapján?

1. Adott az alábbi állapotátmenet:



* 1. Milyen állapotátmenet táblázatot vennél fel?
  2. Milyen teszteseteket írnál a táblázat alapján?

1. <https://www.saucedemo.com/> - Ekvivalenciapartíció-határértékelemzés - Kosárban lévő elem mennyiségének változtatására. Legalább 3 különböző teszteset létrehozása.
2. <https://www.saucedemo.com/> - Állapotátmenet ábra - felületek közötti navigáció. Legalább 3 különböző teszteset létrehozása.
3. <https://www.saucedemo.com/> - Döntési tábla - bejelentkezés felületi validációjára. Legalább 3 különböző teszteset létrehozása.

# 

# Linkek

<https://www.hstqb.org/>

<https://www.istqb.org/>