Szoftvertesztelés ZV tételek 2021. június

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tétel** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** |
| **Állapot** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** |



1. ábra Tipikus tesztelési szintek és technikák (nem kimerítő ábrázolás, nincs fent minden, de sokat segít a megértésben)

# Bevezetés. Ismertesse a szoftvertesztelés célját, fontosságát. Miért szükséges tesztelni? Fogalmazzon meg 2 lehetséges teszt célt. Miben különbözik a szoftvertesztelés a hibakereséstől és a minőségbiztosítástól?

**Bevezetés. Ismertesse a szoftvertesztelés célját, fontosságát.**

A szoftvertesztelés egy fejlesztéssel létrehozott, integrált, módosított, karbantartott szoftver rendszerrel kapcsolatos, tudatosan tervezett, definiált, végrehajtott, követett, mért, elemzett tevékenység-sorozat, amely arra irányul, hogy a hibákat minél előbb megtaláljuk, kijavítsuk és a hibák okát felderítsük, újabb előfordulásukat megakadályozzuk.

A szoftvertesztelés olyan folyamat, amely segít meghatározni, hogy a számítógépes szoftver mennyire helyes, teljes, milyen minőségű. Teszteléssel a szoftver helyességét nem lehet teljeskörűen igazolni. A hibamentességet csakis formális módszerek alkalmazásával lehet bizonyítani.

**Miért szükséges tesztelni?**

Azért tesztelünk, hogy a szoftver bizonyos jellemzőit megismerjük, kipróbáljuk, helyességükről és az elvárt működésükről meggyőződjünk. Illetve, hogy erősödjék a rendszerbe vetett bizalmunk, és hogy feltárjuk a rendszernek és működtetésének kockázatait.A szoftverben hibák lehetnek, és a tapasztalat azt mutatja, hogy hibák vannak. Ezek között súlyosak, életet veszélyeztetők is lehetnek. Megfelelő teszteléssel esélyünk van megtalálni a hibákat.

**Szoftver**: „vezeti” a repülőgépeket, gyorsítja, fékezi a gépkocsikat, „mozgatja” a pénzünket, tehát életünk minden területébe „beleszól”

**Hibás szoftve**r: emberi életeket veszélyeztethet, pénzügyi csődöt eredményezhet, világkatasztrófához vezető folyamatokat indíthat el.

A szoftverben hibák vannak, mert: A szoftvert emberek írják, és “tévedni emberi”, Komplex feladatok elvégzésénél az emberek követnek el hibákat, ez elkerülhetetlen. Tapasztalt programozók átlagban minden 7-10 forrássorban vétenek 1 hibát. Ezen hibák felét a gépnyelvre történő fordításkor kijavítják. A tesztelés során további hibák is kijavulnak, de a hibák 15%-a bent marad az ügyfélnek való átadáskor.

Itt feltehetjük a kérdést, hogy ez sok vagy kevés? Természetesen ezt nem lehet így általánosságban megmondani, mert ez mindig az adott helyzettől, szoftvertől függ. A megoldás, hogy teszteljünk minél több ideig, minél korábban, minél több elemet. Sajnos, a gyakorlatban nem lehetséges 100%-ban letesztelni. Mert vagy nincs idő, vagy nincs pénz. Ezért a **tesztelés célját, tartalmát, mértékét, időtartamát** és egyéb feltételeit mindig az aktuális helyzet ismeretében kell meghatározni!

**Fogalmazzon meg 2 lehetséges teszt célt.**

* Az adott határidő előtt fedjük le a használati esetek 80%-át
* Igazoljuk, hogy adott hardverkonfiguráció esetén a webszolgáltatás képest 2000 embert párhuzamosan kiszolgálni hiba nélkül
* Növeljük a bizalmat szoftver iránt

**Miben különbözik a szoftvertesztelés a hibakereséstől és a minőségbiztosítástól?**

Kapcsolódik a tesztelés és a minőségbiztosítás, de nem ugyanaz: a minőségmenedzsment fogja őket össze. A minőségmenedzsment sok folyamatot ölel fel, például a minőségirányítást is. A teszttevékenységek a minőségirányítás részei, a teljes fejlesztés vagy karbantartási folyamat részeit képezik.

# Alapfogalmak. Ismertesse a következő tesztelési alapfogalmakat: emberi eredetű hiba, meghibásodás, hibaok, hibamaszkolás, stressz-teszt. Mi a különbség a teszt eset és a teszt adat között? Melyek egy teszt eset legfontosabb elemei?

**Alapfogalmak**

**Hiba (defect)**: Egy projekttermék azon tökéletlensége vagy hiányossága, ami nem felel meg a követelményeknek vagy a specifikációnak.

**Emberi eredetű hiba (error):** Egy ember elkövethet hibát (tévedhet) → szoftverkódban vagy valamely kapcsolódó munkatermékben hiba megjelenése → kiválthat egy olyan emberi eredetű hibát, ami pedig egy kapcsolódó munkatermékben egy másik hiba megjelenését eredményezi

Például: téves követelményértelmezés → követelményhiba→ programozási hiba → hiba a kódban → ha lefuttatjuk: meghibásodás (de – sajnos - nem feltétlenül minden esetben)

**Emberi eredetű hibák lehetséges okai:**

* Szoros határidők
* Emberi tévedhetőség
* Tapasztalatlan vagy nem megfelelően képzett projektrésztvevők
* Nem megfelelő kommunikáció a projektrésztvevők között, beleértve a követelményekkel és a tervekkel kapcsolatos kommunikációs hibákat
* A kód, a terv, az architektúra, a megoldandó probléma és/vagy a használt technológia komplexitása
* Félreértések a rendszerbeli és a rendszerközi interfészekkel kapcsolatban, különösen, amikor nagy a rendszerbeli, illetve a rendszerközi kölcsönhatások száma
* Új, ismeretlen technológiák

**Meghibásodás (failure)**: Előidézheti hibás kód, környezeti viszonyok (pl. sugárzás, elektromágneses mezők, a szennyezés…). Nem minden váratlan teszteredmény meghibásodás.

**Hamis pozitív eredmények:** hibásan végrehajtott tesztek eredményeként, vagy a tesztadatokban, a tesztkörnyezetben, más tesztverben lévő hibák miatt, illetve egyéb okokból. Hibaként számontartva, de igazából nem azok.

**Hamis negatív eredmények:** olyan tesztek, amik nem detektálják a hibákat, melyeket kellett volna.

**Hibák, kiváltó okok és hatások**

* A hibák kiváltó okai: **azok a legkorábbi tevékenységek vagy feltételek, amelyek hozzájárultak a hibák kialakulásához.**
* A hibákat lehet elemezni a kiváltó okok azonosítása érdekében vagy azért, hogy csökkentsék a hasonló hibák kialakulását a jövőben.
* A legjelentősebb kiváltó okokra fókuszálva, a kiváltó ok elemzés olyan folyamatfejlesztéseket eredményezhet, amelyek jelentős számú, a jövőben előforduló hibák megjelenését előzhetik meg.

A hibák lehetnek:

* Folyamattal kapcsolatos hibák: A minőségbiztosítás hivatott a megjelenésüket meggátolni
* Termékkel kapcsolatos hibák: A tesztelés hivatott őket megtalálni (a szemle / inspekció is tesztelés! (pl: a specifikáció szemlézése…))
* Fontos a hibák „jó” (egyértelmű, értelmes, minden érdekelt fél által elfogadott…) besorolása!

**Hibamaszkolás**

olyan állapot, amikor az egyik hiba megakadályozza a másik hiba megtalálását. → defect masking, fault masking

**Stressz-teszt**

Terheléses teszt esetén, ha a terhelés a normálisan elvárt határ fölé emelkedik, akkor stressz tesztről beszélünk. (Szokták még teljesítmény- teszt –értelmében is emlegetni).

* Egy rendszer vagy entitás stabilitásának megvizsgálásra használják
* Normál terhelés fölötti terhelés („szakítópont”) hatásának vizsgálatára használják
* A stressz teszt a terheléses teszt egyik formája

**Mi a különbség a teszt eset és a teszt adat között?**

**Teszt eset**: önálló identitása van, és a program egy bizonyos viselkedésével / működési módjával kapcsolatos. Minden teszt esetnek van egy input halmaza és egy elvárt output halmaza.

**Teszt adat:** Olyan adat, amely a teszt előtt is létezik (például egy adatbázisban) és amely kölcsönhatásban van a tesztelés alatt álló rendszerrel, vagy a rendszerkomponenssel. Tipikusan pl. teszt eset inputja.

**Melyek egy teszt eset legfontosabb elemei?**

**Input** – alapvetően kétféle:

* Előfeltétel (precondition): a teszt eset végrehajtása előtt már fennálló körülmény
* Aktuális input: valamilyen tesztelési módszer / technika segítségével azonosított érték(ek)

**Elvárt output** - alapvetően kétféle:

* Utófeltétel (postcondition): a teszt végrehajtását követően fennálló körülmény
* Aktuális output: a teszt végrehajtását követően elvárt érték

**Példa**

* Egy repülőgép optimális útvonalát meghatározó szoftver
	+ Input: a légifolyosó terheltsége, időjárási körülmények az adott napon / idő-intervallumban
	+ Output: optimális útvonal (az adott, létező körülmények esetében)
* Honnan tudhatjuk, hogy az eredményül kapott útvonal ténylegesen az optimális-e?
	+ „mindent tudó orákulum” kialakítása
	+ Referencia tesztelés: szakértő bevonása a teszt outputjának értékelésében

**Dinamikus tesztelés** (A szoftver működés közbeni vizsgálata)

* Funkcionális elvárások alapján: funkcionális / fekete doboz tesztelésnek nevezzük. A teszt eseteket a specifikáció alapján határozzuk meg.
* A szoftver belső szerkezetének alapján: strukturális / fehér doboz tesztelésnek nevezzük. A teszt eseteket a kód alapján határozzuk meg.

Széleskörűen alkalmazott értelmezés szerint: a statikus tesztelés szemlézést, inspekciót, elemzést jelent. A „teszt eset” kifejezést inkább dinamikus tesztelés esetén használják.

**Tesztelés:** A szükséges előfeltételek meghatározása, input értékek megadása a teszt esetekre, az output megfigyelése, az output összehasonlítása az elvárt értékekkel. Mindezek alapján tudjuk megállapítani, hogy a teszt sikeres volt-e.

**Teszt esetek azonosítása:**

Elvárás (általában):

* Teszt eset azonosító:
* Cél:
* Előfeltételek:
* Input adatok (értékek):
* Elvárt output (értékek):
* Utófeltételek:
* Teszt eset végrehajtásának története: (Dátum, Eredmény, Verzió, Ki végezte)

**Hogyan érhető el, hogy a tesztelés megfelelő legyen ?** Megfelelő megközelítés és megfelelő teszt esetek kiválasztásával.

# Ismertesse a tesztelés 7 alapelvét

1. **alapelv – Hibák látszólagos hiánya**

Bár a tesztelés kimutathatja a hibák jelenlétét, de azt nem képes igazolni, hogy nincsenek hibák. A teszteléssel csökken annak az esélye, hogy a szoftverben felfedezetlen hibák maradnak, de ha nem találnak hibát, az nem annak a bizonyítéka, hogy a rendszer tökéletes (értsd: valóban nincs benne hiba).

1. **alapelv – Nem lehetséges kimerítő teszt**

Kimerítő teszt – azaz mindenre (a bemenetek és előfeltételek minden kombinációjára) kiterjedő tesztelés – a triviális eseteket leszámítva – nem lehetséges. A kimerítő teszt helyett kockázatelemzést és prioritásokat kell alkalmazni, ezáltal növelve a teszttevékenységek összpontosításának hatékonyságát.

1. **alapelv – Korai tesztelés**

A tesztelést a szoftver vagy rendszerfejlesztési életciklusban a lehető legkorábban el kell kezdeni, és előre meghatározott célokra kell összpontosítani.

1. **alapelv – Hibafürtök megjelenése**

A tesztelést a feltételezett, illetve a megtalált hibák eloszlásának megfelelően kell koncentrálni. A kiadást megelőző tesztelés során a megtalált hibák többsége néhány modulban van, vagy legalábbis ezen modulok felelősek a működési hibák többségéért.

1. **alapelv – A féregirtó paradoxon**

Ha mindig ugyanazokat a teszteket hajtjuk végre, akkor az azonos tesztkészlet egy idő után nem fog új hibákat találni. A „féregirtó paradoxon” megjelenése ellen a teszteseteket rendszeresen felül kell vizsgálni, és új, eltérő teszteseteket kell írni annak érdekében, hogy a szoftver vagy a rendszer különböző részei kerüljenek futtatásra, és ezáltal további programhibákat találhassanak.

1. **alapelv – A tesztelés függ a körülményektől (körülményfüggés)**

A tesztelést különböző körülmények esetén különbözőképpen hajtják végre. Például egy olyan rendszert, ahol a biztonság kritikus szempont, másképp tesztelnek, mint egy e-kereskedelmi oldalt.

1. **alapelv – A hibamentes rendszer téveszméje**

A hibák megtalálása és javítása hasztalan, ha a kifejlesztett rendszer használhatatlan, és nem felel meg a felhasználók igényeinek, elvárásainak.

# A tesztelés pszichológiája. Miben különbözik a tesztelői viselkedés a szoftverfejlesztői viselkedéstől? Melyek a tesztelő legfontosabb képességei? Mit tud a tesztelő hozzáadni a fejlesztendő rendszerhez? Mit jelent a megerősítési torzítás?

A tesztelés pszichológiája a tesztelő feladatait, a hibák kommunikálási módját, a szoftverfejlesztői és a tesztelői gondolkodásmódot foglalja magába.

**A tesztelés pszichológiája**

* A tesztelés és a felülvizsgálat a szoftverfejlesztéstől eltérő gondolkodásmódot kíván.
* Jó, ha van független tesztelői csapat.
	+ Ha van, általában az integrációs teszttől kezdődően.
	+ A tesztelők **hasznos szempontokat tudnak bevinni a fejlesztésbe, már a tervezés** szakaszában. Vonjuk be őket!
* A tesztelő hibákat keres, közölnie kell, ha hibát talál. Ez konfliktusforrás lehet.
* A hibák, programhibák konstruktív szemléletű közlésével elkerülhetők a tesztelők és elemzők, tervezők és fejlesztők közötti ellentétek.
* Tárgyilagos megfogalmazás fontos. Állapítsuk meg és dokumentáljuk a hibát. Ne minősítsük a fejlesztőket!

**Miben különbözik a tesztelői viselkedés a szoftverfejlesztői viselkedéstől?**

(A rossz hír közlőjét nem kedvelik)

**Fejlesztői és tesztelői gondolkodásmód**

* A fejlesztők helyesnek vélik a kódjukat
* A tesztelő a hibákat keresi
* A fejlesztés konstruktív, a tesztelés destruktív?
* De: ugyanaz a cél! Ezt kell hangsúlyozni!
* Fejlesztők is tesztelhetik saját kódjukat (Tesztelői szemléletmóddal!)

**Jó kommunikáció: kulcsfontosságú**

* Együttműködéssel indítani – hadakozás helyett
* A tesztelés előnyeinek hangsúlyozása
* Semlegesen, tárgyilagosan kell kommunikálni az eredményeket és az egyéb találatokat, a hibás elem megalkotójának kritizálása nélkül
* Meg kell próbálni megérteni a másik személy érzéseit és az okokat, amiért negatívan reagálhatnak az információkra.
* Meg kell győződni afelől, hogy a másik személy megértette az elmondottakat és fordítva

**Melyek a tesztelő legfontosabb képességei?**

**Tesztelői képességek**

* Éles megfigyelőképesség
* Részletekre figyelés
* Destruktív kreativitás
* A terméket úgy felfogni, mint részeinek integrációját
* Vevő-központú hozzáállás
* Cinikus, de odafigyelő hozzáállás
* Szervezettség, rugalmasság, türelem
* Objektív & semleges hozzáállás

**Mit tud a tesztelő hozzáadni a fejlesztendő rendszerhez?**

A tesztelő legfontosabb feladata a megfelelő tesztelési technikák kiválasztása és kombinálása. Ezen felül igen komoly domén tudásra kell szert tegyen, ami az idő előrehaladtával egyre értékesebb lesz.

**Mit jelent a megerősítési torzítás?**

A fejlesztők helyesnek vélik a kódjukat, a megerősítési torzítás miatt nehéz elfogadniuk a kód helytelenségét. A megerősítési torzítás azt jelenti, hogy azokat az információkat részesítjük előnyben, amik igazolják a saját igazunkat. Továbbá egy gyakori emberi jellemvonás, hogy a rossz hír hozóját okolják, és a tesztelés által szolgáltatott információ gyakran tartalmaz rossz hírt.

Megerősítési torzítás („Confirmation bias”): „Az embereknek azon hajlandósága, hogy azokat az információkat részesítik előnyben, amelyek igazolják saját prekoncepciójukat (előfeltételezéseiket) vagy hipotézisüket, tekintet nélkül az információ igaz vagy nem igaz voltára. Ennek eredményeképpen, az emberek szelektíven gyűjtenek bizonyítékokat és hívják elő emlékeiket a memóriából, nemkülönben torzított módon interpretálják az információkat. A torzítások különösen az emocionálisan jelentős kérdéseknél jelennek meg, és a már megerősödött, igaznak tartott kognitív tartalmak esetén.”

**Egyéb kognitív torzítások**

* Mindent feketén vagy fehéren látunk
* Túláltalánosítás
* Negatív szűrés
* pozitív dolgok figyelmen kívül hagyása
* Nagyítás vagy kicsinyítés (hibák felnagyítása, pozitívumok leértékelése)
* Korai és/vagy önkényes következtetés
* Érzelmi érvelés: egy negatív érzésünket tényként kezeljünk
* Perszonalizáció („te vagy a hibás”)
* Címkézés (egy hiba miatt az egész rendszer rossz…)

# A tesztelési folyamat. Ismertesse, hogyan jelenik meg a tesztelés az alapvető szoftverfejlesztési életciklus modellekben. Miről szól az ISO 12207 szabvány? Hogyan kapcsolódik a V-modellhez? Hogyan használható az ISO 12207 egy szoftverfejlesztési életciklus modell kialakításában?

**A tesztelési folyamat.**

**Tesztfolyamat**: Kölcsönös kapcsolatban lévő tevékenységek halmaza, melynek részei a:

* teszttervezés
* tesztfelügyelet és irányítás
* tesztelemzés
* műszaki teszttervezés
* tesztmegvalósítás
* tesztvégrehajtás
* tesztlezárás

**Ismertesse, hogyan jelenik meg a tesztelés az alapvető szoftverfejlesztési életciklus modellekben.**

* A tesztelés szoftverfejlesztési életciklus minden fázisában megjelenik.
* Minden fejlesztési tevékenységhez tartozik egy teszttevékenység.
* A tesztelés a fejlesztési életciklus elején elkezdődik és a teljes életciklus alatt tart!
* Függetlenül attól, hogy melyik szoftverfejlesztési életciklusmodellt választjuk, a teszttevékenységeket az életciklus korai szakaszában kell kezdeni, a korai tesztelés elvének betartásával.
* Minden tesztszint rendelkezik az adott szintre jellemző tesztcéllal
* A tesztek elemzése és tervezése az adott tesztszinthez tartozó fejlesztési tevékenység során kezdődik
* A tesztelőket be kell vonni a követelmények és a tervezés kialakításába és finomhangolásába, valamint a dokumentációk felülvizsgálatába is minél előbb, vagyis amint a fejlesztési életciklusban elkészültek az első munkatermékek (pl. követelmények, műszaki tervek, felhasználói történetek).

**Miről szól az ISO 12207 szabvány?**

Az ISO 12207 összefoglalóan tartalmazza és jól leírja a szoftverfejlesztés során felmerülő összes tevékenységet. Egységes fogalmi keretet hoz létre a szoftveréletciklus folyamatokra olyan jól meghatározott **terminológiát** használva, amely a szoftveripar számára **kiindulásul** szolgálhat. Olyan **folyamatokat**, **tevékenységeket** és **feladatokat** tartalmaz, amelyek szoftvert tartalmazó rendszerek, különálló szoftvertermékek és szoftverszolgáltatások beszerzése, valamint szoftvertermékek szállítása, fejlesztése, üzemeltetése és karbantartása során alkalmazandók.

**Hogyan kapcsolódik a V-modellhez?**

Kompatibilis azzal. (Az ISO 12207 nem tartalmaz konkrét életciklus modellt - így a V-modellt sem).

**Hogyan használható az ISO 12207 egy szoftverfejlesztési életciklus modell kialakításában?**

A szabvány olyan folyamatot is tartalmaz, amelyet a szoftveréletciklus folyamatok létrehozására, ellenőrzésére és javítására lehet használni. Gyakorlatilag a szoftverfejlesztés minden elemére terminológiát és leírást ad. Emellé pedig támogató folyamatot, amivel bárki szoftverfejlesztési életciklust alakíthat ki.

# Az alapvető tesztelési folyamat. Ismertesse az alapvető tesztelési folyamatot az ISTQB ajánlása alapján.

**A tesztelési folyamat**

A **tesztfolyamat** az alábbi fő tevékenységcsoportokból áll:

* Teszttervezés
* Tesztfelügyelet és -irányítás
* Tesztelemzés
* Műszaki teszttervezés
* Teszt megvalósítása
* Tesztvégrehajtás
* Tesztlezárás

(A tesztelés tervezése, végrehajtása, kiértékelése dokumentált folyamat!)

Ezek a szoftverfejlesztési életciklus modellbe integrálódnak. Bármelyik életciklus modellbe! (Bár logikailag egymás után következnek, a folyamat tevékenységei átfedhetik egymást, és párhuzamosan is folyhatnak. Általában szükséges ezen fő tevékenységeknek a projekten, illetve a rendszeren belüli testreszabása.)

**Teszttervezés**

Azokat a tevékenységeket foglalja magában, amelyek definiálják a tesztelés céljait, illetve ezen célok teljesítésének módját a kontextus (pl. megfelelő teszttechnikák és feladatok specifikálása, teszt ütemtervének kialakítása a határidők betartása érdekében) által meghatározott korlátozásokat figyelembe véve. A teszttervek szükség esetén újragondolandók a felügyeleti és irányítási tevékenységek visszajelzései alapján.

**Tesztfelügyelet és -irányítás**

A tesztfelügyelet magában foglalja az aktuális előrehaladás és a tervezett előrehaladás folyamatos összehasonlítását, a teszttervben definiált felügyeleti metrikák felhasználásával.

Kilépési feltételek kiértékelése támogatja, amire néhány szoftverfejlesztési életciklus modellben a „Kész“ definíciójaként utalnak (DoD- Definition of Done).

**Tesztelemzés**

A tesztelemzés során a tesztbázist elemzik annak érdekében, hogy azonosítsák a tesztelhető funkcionalitásokat és meghatározzák a kapcsolódó tesztfeltételeket. A tesztelemzés meghatározza azt, hogy „mit teszteljünk“, mérhető lefedettségi feltételekkel megadva.

**Fő tevékenységek:**

* Az aktuális tesztszintnek megfelelő tesztbázis elemzése
	+ Pl. követelmények, design, komponens, kockázati lista stb.
* A tesztbázis és a tesztelemek kiértékelése a különböző típusú hibák azonosítása érdekében, melyek lehetnek:
	+ Kétértelműség
	+ Kihagyások
	+ Inkonzisztenciák
	+ Pontatlanságok
	+ Ellentmondások
	+ Felesleges utasítások
	+ Tesztelendő funkcionalitások és funkcióhalmazok meghatározása
* Minden egyes funkcióhoz a tesztfeltételek definiálása és priorizálása a tesztbázis elemzése alapján, illetve funkcionális, nemfunkcionális és strukturális jellemzők, valamint más üzleti és technikai tényezők, kockázati szintek figyelembevétele
* A tesztbázis minden eleme és a hozzá kapcsolódó tesztfeltételek közötti kétirányú nyomonkövethetőség meghatározása

**Műszaki teszttervezés**

A műszaki teszttervezés során a tesztfeltételekből magasszintű teszteseteket, magasszintű tesztesethalmazokat és más tesztvereket alakítanak ki. A műszaki teszttervezés azt válaszolja meg, hogy “hogyan teszteljünk?” (**Tesztver**: A tesztelési folyamat alatt előállított projekttermékek, melyeket a tervezés, műszaki teszttervezés, végrehajtás, kiértékelés és jelentéskészítés során használnak.) A műszaki teszttervezés az alábbi fő tevékenységekből áll:

* Tesztesetek és tesztesetek halmazainak tervezése és priorizálása
* A tesztfeltételek és tesztesetek támogatásához szükséges tesztadatok meghatározása
* A tesztkörnyezet megtervezése, valamint a szükséges infrastruktúra és eszközök azonosítása
* A tesztbázis, tesztfeltételek és a tesztesetek közötti kétirányú nyomon követhetőség meghatározása

**Tesztmegvalósítás**

A tesztmegvalósítás során létrehozzák és/vagy befejezik a tesztvégrehajtáshoz szükséges tesztvert, beleértve a tesztesetek teszteljárásokba történő szervezését. Amíg a műszaki teszttervezés arra a kérdésre ad választ, hogy “hogyan teszteljünk?”, addig a tesztmegvalósítás azt válaszolja meg, hogy “rendelkezésünkre áll minden, ami a tesztek futtatásához szükséges?

**A tesztmegvalósítás az alábbi fő tevékenységekből áll:**

* Teszteljárások fejlesztése és priorizálása, valamint potenciálisan automatizált tesztszkriptek készítése
* Tesztkészletek létrehozása a teszteljárásokból és (ha van) az automatizált tesztszkriptekből
* A tesztkészletek tesztvégrehajtási ütemtervbe történő szervezése olyan módon, hogy hatékony tesztvégrehajtást eredményezzen
* Tesztkörnyezet (beleértve potenciálisan a teszttámogató szoftverkörnyezetet, szolgáltatás virtualizációt, szimulátorokat és más infrastruktúra elemeket) kiépítése és annak ellenőrzése, hogy megfelelően lett beállítva minden, ami szükséges
* Tesztadatok előkészítése és annak biztosítása, hogy a tesztkörnyezetbe megfelelően vannak betöltve
* A tesztbázis, tesztfeltételek, tesztesetek, teszteljárások és tesztkészletek közötti kétirányú nyomonkövethetőség verifikálása és frissítése

**Tesztvégrehajtás**

A tesztvégrehajtás során a tesztkészleteket a tesztvégrehajtási ütemtervben meghatározott sorrendben futtatják.

**Fő tevékenységek:**

* A tesztelem(ek) vagy a teszt tárgyának, a teszteszköz(ök) és a tesztver azonosítójának, illetve verziójának rögzítése
* A tesztek manuálisan vagy tesztvégrehajtási eszközök segítségével történő végrehajtása
* A várt és a tényleges eredmények összehasonlítása
* A rendellenességek elemzése a valószínű okuk meghatározása céljából
* Hibák jelentése a megfigyelt meghibásodások alapján
* A tesztvégrehajtás kimenetének naplózása (pl. sikeres, elbukott, blokkolt)
* Teszttevékenységek ismétlése vagy egy rendellenesség esetén végzett tevékenységek eredményeként, vagy a tervezett tesztelés részeként (pl. egy kijavított teszt végrehajtása, ellenőrző tesztelés és/vagy regressziós tesztelés)
* A tesztbázis, tesztfeltételek, tesztesetek, teszteljárások és teszteredmények közötti kétirányú nyomon követhetőség verifikálása és frissítése

**Tesztlezárás**

A tesztlezárás tevékenységei során a tapasztalatok, a tesztver és az egyéb releváns információk véglegesítése érdekében adatokat gyűjtenek a befejezett teszttevékenységekből. A tesztlezáráshoz kapcsolódó tevékenységeket projektmérföldkövek elérésekor hajtják végre.

(például amikor kiadnak egy szoftverrendszert, befejeznek (vagy visszavonnak) egy tesztprojektet, egy Agilis projektiterációt, egy tesztszinten való tesztelést, vagy egy karbantartó frissítést.)

**Fő tevékenységek:**

* Annak ellenőrzése, hogy minden hibajelentést lezártak, változtatáskérések vagy termék-teendőlista elemek felvétele a tesztvégrehajtás végén megoldatlan hibák kezelésére
* Összefoglaló tesztjelentés készítése az érintett felek tájékoztatása céljából
* A tesztkörnyezet, a tesztadatok, a teszt infrastruktúra és más tesztver véglegesítése és archiválása későbbi felhasználásra
* A tesztver átadása a karbantartást végző csapatoknak, más projektcsapatoknak és/vagy más érintetteknek, akiknek előnyére válhat a használata
* A befejezett teszttevékenységek tanulságainak elemzése a jövőbeli iterációkhoz, kiadásokhoz és projektekhez szükséges változtatások meghatározására
* A gyűjtött információ felhasználása a tesztfolyamat érettségének fejlesztésére

# Tesztszintek. Ismertesse az ISTQB ajánlásaiban szereplő teszt szinteket.

Az ISTQB 4 szintet különböztet meg (főleg ISQTB sillabuszból):

* Komponens tesztelés (unit teszt)
	+ **célja:** A kockázat csökkentése. Verifikálás, hogy a komponens funkcionális és nem funkcionális viselkedése megfelel-e a specifikációnak és a terveknek. A bizalom megteremtése a komponens minősége iránt. A komponensben levő hibák megtalálása. Megakadályozni, hogy a hibák magasabb tesztszintre bekerüljenek.
	+ **teszt bázisa:** A részletes tervek, a kód, adatmodellek, a komponens.
	+ **teszt tárgya:** Komponensek, egységek vagy modulok; kód és adatstruktúrák; osztályok; adatbázis modulok.
	+ **jellemző hibák:** Nem megfelelő funkcionalitás, adatfolyam problémák, helytelen kód és logika.
	+ **megközelítés, felelősség:** Tipikusan fejlesztő végzi (akár egymásnak). TDD-ben a teszteset írása megelőzi a kód írását.
* Integrációs tesztelés (itt derülnek ki a legköltségesebb hibák)
	+ **célja:** Kockázatok csökkentése. Verifikálás, hogy az interfészek funkcionális és nemfunkcionális viselkedése megfelel a tervekben és specifikációkban foglaltaknak. Bizalom megteremtése az interfészek minőségével kapcsolatban. Hibák megtalálása (amelyek magukban az interfészekben vagy a komponenseken, illetve rendszereken belül vannak). Megakadályozni, hogy a hibák egy magasabb tesztszintre kerüljenek
	+ **teszt bázisa:** Szoftver- és rendszerterv; Szekvencia diagram; Interfész- és kommunikációs protokoll specifikációk; Használati esetek; Komponens- vagy rendszerszintű architektúra; Munkafolyamatok; Külső interfész leírások
	+ **teszt tárgya:** Alrendszerek, adatbázisok, infrastruktúra, interfészek, API-k és mikroszolgáltatások.
	+ **jellemző hibák:** Helytelen vagy hiányzó adatok. Az interfészhívások helytelen sorrendezése vagy időzítése. Interfész hibás illesztése. Komponensek közötti kommunikáció meghibásodásai. Nem, vagy rosszul kezelt kommunikációs meghibásodás a komponensek között
	+ **megközelítés, felelősség:** Az integrációs tesztelés a komponensek vagy rendszerek közötti kölcsönhatásokra összpontosít. A **komponens integrációs** tesztelés az integrált komponensek közötti kölcsönhatásokra és interfészekre összpontosít. **A rendszer integrációs** tesztelés a rendszerek, csomagok és mikroszolgáltatások közötti kölcsönhatásokra és interfészekre összpontosít. A **komponens integrációs** tesztelés leggyakrabban a fejlesztők felelőssége. A **rendszer integrációs** tesztelés általában a tesztelők felelőssége.
* Rendszertesztelés
	+ **célja:** Kockázatcsökkentés; Annak verifikálása, hogy az interfészek funkcionális és nemfunkcionális viselkedése megfelel a tervekben és specifikációkban foglaltaknak; Annak validálása, hogy a rendszer teljes és az elvártaknak megfelelően működik; A rendszer egészének minősége iránti bizalom megteremtése; A hibák megtalálása; Megakadályozni, hogy a hibák magasabb tesztszintekre vagy a termékbe kerüljenek
	+ **teszt bázisa:** Rendszer- és szoftverkövetelményspecifikáció (funkcionális és nemfunkcionális), kockázatelemzés-jelentés, használati esetek, epic-ek és felhasználói történetek, rendszerviselkedési modellek, állapotdiagramok, rendszer- és felhasználói kézikönyvek
	+ **teszt tárgya:** Alkalmazások, hardver/szoftver rendszerek, operációs rendszer, tesztelt rendszer (SUT, rendszerkonfiguráció és konfigurációs adatok)
	+ **jellemző hibák:** Helytelen számítások; A rendszer helytelen vagy váratlan funkcionális- vagy nemfunkcionális viselkedése; Helytelen vezérlés és/vagy adatfolyam a rendszeren belül; A rendszer nem képes megfelelően végrehajtani a végponttól végpontig terjedő funkcionális feladatokat; Meghibásodás az adott rendszerkörnyezet(ek)ben; A rendszer nem a rendszer- és felhasználói kézikönyvekben leírtak szerint működik
	+ **megközelítés, felelősség:** A rendszertesztelésnek a rendszer teljes, végponttól végpontig terjedő viselkedésére kell összpontosítania, mind funkcionális, mind nemfunkcionális tekintetben. A felhasználóknak is érteni kell.
* Elfogadási tesztelés
	+ **célja:** Bizalom kialakítása a teljes rendszer minőségével kapcsolatban; Annak validálása, hogy a rendszer teljes és az elvártaknak megfelelően fog működni; Annak verifikálása, hogy a rendszer funkcionális és nemfunkcionális specifikációkban meghatározottak szerint készült-e el.
	+ **teszt bázisa:** Üzleti folyamatok; Felhasználói vagy üzleti követelmények; Szabályozások, jogi szerződések és szabványok; Használati esetek és/vagy felhasználói történet teszt; Rendszerkövetelmények; Rendszer- vagy felhasználói dokumentáció; Telepítési eljárások; Kockázatelemzési jelentések
	+ **teszt tárgya:** Tesztelt rendszer (SUT); Rendszerkonfiguráció és konfigurációs adatok; A teljesen integrált rendszer üzleti folyamatai; Helyreállítási rendszerek és hot site-ok (az üzletfolytonossági- és katasztrófa utáni helyreállítási tesztekhez); Üzemeltetési és karbantartási folyamatok; Űrlapok; Jelentések; Létező és átalakított, az éles rendszerből származó adat
	+ **jellemző hibák:** A rendszer munkafolyamatai nem felelnek meg az üzleti- vagy felhasználói követelményeknek. Az üzleti szabályokat nem megfelelően valósították meg. A rendszer nem felel meg a szerződéses vagy szabály alapú követelményeknek.
	+ **megközelítés, felelősség:** Az elfogadási tesztelés gyakran a vevők, az üzleti felhasználók, a terméktulajdonosok vagy a rendszer üzemeltetőinek a felelőssége, és más érintettek is részt vehetnek benne. Az elfogadási tesztelést gyakran úgy tekintik, mint az utolsó tesztszintet a szekvenciális fejlesztési életciklusban

**Tesztszintek (diasorból, kivéve a már beírt infókat)**

Minden tesztszint esetén a következőket kell tudni meghatározni:

* a teszt általános célja,
* a tesztesetek készítésekor meghivatkozott projekttermékek (pl.a tesztbázis),
* a teszt tárgya (mit tesztelnek),
* a megtalálandó hibák és meghibásodások,
* a teszttámogató szoftverkörnyezet követelményei és az eszköztámogatás,
* a jellemző megközelítések és felelősségi körök.

**Komponens (Unit) teszt / Egységteszt**

* **Komponens**: a legkisebb önállóan tesztelhető szoftver egység
* Az egységeket / komponenseket teszteljük a specifikációval szemben. Egy egység: egy v. több eljárás, függvény
* A forráskód egy adott egységének helyességét vizsgáló tesztmódszer
* Alapötlet: készüljön teszt eset a modul minden nem triviális függvényére, funkciójára vagy metódusára, úgy, hogy minden teszt eset önálló, a többitől különválasztható legyen

**Integrációs teszt**

* Az egységeket integráljuk, és az integrált részek interfészeit teszteljük a specifikációval szemben.
* Általában az egységtesztet követi és a rendszertesztet előzi meg
* Az egységteszten sikeresen „átesett” modulokat veszi alapul
* Célja, hogy a nagyobb elemekkel szemben támasztott, funkcionalitásra, teljesítményre, megbízhatóságra vonatkozó követelményeket ellenőrizze
* A modulok közötti interfészek kerülnek kipróbálásra
* A modulok / alrendszerek közötti együttműködést szimulálva történik a tesztelés
* Alapötlet: mindig tesztelt, ellenőrzött alapelemekből építkezünk

**Rendszerteszt**

* A teljes rendszert teszteljük a specifikációval szemben
* Input a teszthez: integrált, sikeresen tesztelt (al)rendszer(ek)
* A rendszertesztet a felhasználónak is értenie kell – gyakran szorosan kapcsolódik az átvételi teszthez

**Elfogadási teszt**

* Egy egész rendszer vagy termék viselkedésére és képességeire összpontosít
* Általános formái:
	+ felhasználói elfogadási tesztelés
	+ üzemeltetési elfogadási tesztelés
	+ szerződéses és szabály alapú elfogadási tesztelés
	+ alfa és bétatesztelés

# Funkcionális és nemfunkcionális jellemzők tesztelése. Mutassa be a nemfunkcionális szoftvertermék jellemzők tesztelésére alkalmazható megközelítést.

**Funkcionális tesztek:** Olyan teszteket foglal magában, melyek kiértékelik a rendszer által végzendő funkciókat.

* 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.
* A funkcionális teszteket minden tesztszinten el kell végezni
* A funkcionális tesztelés a szoftver viselkedését veszi figyelembe, így a feketedoboz technikák felhasználhatók a komponens vagy rendszer funkcionalitására vonatkozó tesztfeltételek és tesztesetek származtatására
* A funkcionális tesztelés alapossága a funkcionális lefedettséggel mérhető.
	+ A funkcionális lefedettség azt mutatja meg, hogy a funkcionalitást milyen mértékben teszteljük le, mindez az érintett elem típusának százalékos arányában kifejezve.
	+ Például a tesztek és a funkcionális követelmények közötti nyomon követhetőséget alkalmazva kiszámítható a tesztelt követelmények aránya, ez potenciálisan azonosítja a lefedettségi réseket. A funkcionális teszttervezés és -végrehajtás speciális készségeket vagy ismereteket igényelhet, mint például az adott üzleti problémát kezelő szoftveres megoldás ismerete (pl. az olaj- és gázipar geológiai modellezési szoftvere).

**Nemfunkcionális tesztek**

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

* A nemfunkcionális tesztelés során azt vizsgáljuk, hogy miként viselkedik a rendszer.
* A nemfunkcionális tesztelés minden tesztszinten elvégezhető
* A feketedoboz technikákat használhatjuk a tesztfeltételek és a nemfunkcionális tesztelés teszteseteinek meghatározásához
	+ Például a határérték-elemzés segítségével meghatározhatók a teljesítménytesztek stresszfeltételei.
* A nemfunkcionális tesztek alapossága a nemfunkcionális lefedettséggel mérhető. Azt mutatja meg, hogy a nemfunkcionális elemek bizonyos típusait milyen mértékben teszteljük le, mindezt az érintett elem típusának százalékában kifejezve.

Funkcionális és nemfunkcionális tesztelés a funkcionális és nemfunkcionális követelményeken alapszik.

**Nemfunkcionális követelmények (Ahogyan a rendszer csinálja)**

* Minőségi követelményeknek is szokás nevezni őket
* Expliciten dokumentálni kell őket. Különös tekintettel az alábbiakra:
	+ Teljesítmény, Biztonság, Megbízhatóság, Használhatóság, Karbantarthatóság, Hordozhatóság

**Megbízhatóság**: Azon attribútumok összessége, amelyek hatással vannak a szoftver azon képességére, hogy teljesítményének szintjét - bizonyos feltételek mellett, meghatározott időre – megőrizze.
**Használhatóság**: Azon attribútumok összessége, amelyek hatással vannak a szoftver használatához szükséges erőfeszítésre, és a használat bizonyos meghatározott vagy feltételezett felhasználócsoport általi értékelésére.
**Hatékonyság**: Azon attribútumok csoportja, amelyek hatással vannak az elért eredmény és a felhasznált erőforrás viszonyára, megadott feltételek mellett.
**Karbantarthatóság**: Bizonyos, meghatározott módosítás elvégzéséhez szükséges erőfeszítésre hatást gyakorló attribútumok összessége.
**Hordozhatóság**: Olyan attribútumok összessége, amelyek befolyásolják a szoftvernek azt a képességét, hogy adott környezetből másikba átvihető legyen.

 **Példa nemfunkcionális követelményekre**

* Folyamatos, kritikus menedzsment döntések
* a szervezetnek a szoftverterméktől való erős függősége
	+ megbízhatóságra / elérhetőségre (availability) vonatkozó követelményeket eredményez
* Sok tapasztalatlan felhasználó
* alacsony iskolázottság
	+ könnyen tanulhatóság követelményét eredményezi
* sok menedzser van a felhasználók között
* sok a tapasztalatuk
	+ fontossá teszi a rövid válaszidőket (időfüggő viselkedés)

**Megbízhatóság**: A szoftver azon képessége, hogy teljesítményét bizonyos feltételek mellet megőrizze

* **érettség**: A rendszer azon képessége, hogy “ne omoljon össze” a szoftver hibái miatt hogyan mérhető: Mean Time Between Failures = működési idő / meghibásodások száma
* **hibatűrés**: A szoftver azon képesége, hogy bizonyos szinten működőképes marajdon hibák megjelenése esetén is hogyan mérhető. Input Error Detection Ratio = Azonosított bemeneti hibák száma az összes inputhoz viszonyítva

**Visszaállíthatóság**: A szoftvertermék azon képessége, hogy hiba esetében visszaállítson egy meghatározott működési szintet és a hiba által érintett adatokat. Hogyan mérhető: Mean Down Time = A rendszer teljes működésképtelenségének ideje / az azonosított meghibásodások

**Elérhetőség**: A szoftvertermék azon képessége, hogy egy adott funkciót egy adott pillanatban biztosan végrehajtson hogyan mérhető: Relative Availability Percentage = A szoftver elérhetőségének és szükségességének aránya

# Karbantartási tesztelés. Ismertesse a karbantartási tesztelés legfőbb típusait. Beszéljen a regressziós és a progressziós tesztelés jelentőségéről.

**Karbantartási tesztelés**

Átadott rendszert karban kell tartani

* A különböző típusú változtatások
	+ az üzemeltetés során felfedezett hibák javítása
	+ új funkciók hozzáadása
	+ a már leszállított funkcionalitás törlése, módosítása.
* Megőrizni vagy javítani a komponens vagy rendszer élettartalma során annak nemfunkcionális minőségi jellemzőit, különösen a teljesítmény hatékonyság, kompatibilitás, megbízhatóság, biztonság, és hordozhatóság tekintetében.

Amikor a karbantartás részeként bármilyen változtatás történik, akkor karbantartási tesztelést kell végezni annak érdekében, hogy kiértékeljék az eszközölt változtatások sikerességét, valamint, hogy ellenőrizzék a rendszer változatlanul maradt részeiben bekövetkező esetleges mellékhatásokat.

**Tervezett és nem tervezett kiadások**

* Egy karbantartás kiadás esetén a karbantartási tesztre több tesztszinten és különböző teszttípusokat felhasználva is szükség lehet a hatókör alapján.

**A karbantartási teszt hatóköre függ:**

* A változtatás kockázatának mértékétől, például annak mértéke, hogy a szoftver megváltoztatott része milyen mértékben kommunikál más komponenssel vagy rendszerrel
* A meglévő rendszer méretétől
* A változtatás méretétől

**Karbantartási tesztek típusai**

* Bármilyen technika, bármilyen szinten
* Regressziós és progressziós tesztek fontosak

**Regressziós** teszteléssel bizonyítjuk, hogy ami az előző build-ben működött, a jelenlegi build-ben is működik. (Tapasztalat: minden ötödik módosítás új hibát visz a rendszerbe.)

**Progressziós** tesztelés: feltételezzük, hogy az integrációs teszt rendben lefutott, és az új funkciókat tesztelhetjük.

Leggyakoribb regressziós teszt: megismételjük a rendszertesztet

* Egyszerűsíthetünk, ha tesztelési vezérfonalakat választunk, a progressziós és regressziós teszt céljainak függvényében
* Progressziós teszttel új területet tesztelünk, több hibára számítunk, a hibát lokalizálni kell- emiatt rövidebb vezérfonalak választása javasolt
* Regressziós tesztben kevesebb hibára számítunk, kevésbé valószínű, hogy a hibákat lokalizálni kell- választhatunk hosszabb vezérfonalat

**Karbantartást kiváltó okok**

**Módosítások**

* a betervezett bővítések (például: kiadás alapon)
* javító és szükséges változtatások
* éles környezetváltozások
* a kereskedelmi dobozos szoftverek (COTS) frissítése
* hibák javítása és sebezhetőségek

**Migráció**

* egyik platformról a másikra váltás, adat konverziók tesztelése, amikor az adat másik alkalmazásból átmigrálásra kerül a karbantartott rendszerbe
* Visszavonultatás
	+ Amikor egy alkalmazás vagy rendszer visszavonultatásra kerül, akkor szükség lehet adat migrációs tesztre vagy archiválásra hosszú adatmegőrzési idő esetén.
* Visszaállási / visszakeresési eljárások
* Regressziós tesztelésre lehet szükség a használatban maradt funkciók működésének biztosítása érdekében.

**Karbantartási hatáselemzés**

* Kiértékeli azon változtatásokat, amelyek a karbantartás kiadás során mentek végbe annak érdekében, hogy azonosítsa a várható következményeit, valamint a változás várható és lehetséges mellékhatásait, továbbá, hogy azonosítsa azon részeit a rendszernek, amelyek érintettek a változás által.
* A hatáselemzés segít azonosítani a változtatás kihatását a meglévő tesztekre.
* A regressziós kockázat miatt a változtatás mellékhatásait és érintett rendszer területeit tesztelni kell a regressziók miatt, várhatóan azt követően, hogy frissítésre kerültek a változtatás által érintett tesztek.

A gyenge és erős ekvivalencia osztály alapú tesztelés hasznos a progressziós és regressziós teszt közötti különbség megértésében.

# Tesztek típusai. Ismertessen a tesztelésben alkalmazott teszt típusok közül legalább 3 lehetséges csoportosítást és elemeiket. A csoportok közül egyik legyen a tesztelési technikák szerinti csoportosítás.

**Tesztek típusai**

Sokféle csoportosítás (átfedések, ismétlések) Pl:

* Tesztelt elem szerint
* Az életciklusban elfoglalt hely szerint
* Tesztelő személye szerint
* A teszt eredményének felhasználása szerint
* Alkalmazott technikák szerint

**Az életciklusban elfoglalt hely szerint**:

* Pre-alfa
* alfa
* béta
* akár tovább

**A tesztelt elem szerint**:

* Komponens (egység/ unit) -teszt
* Integrációs teszt
* Rendszerteszt

**Teszt típusok az alkalmazott technika szerint**

* Statikus tesztelés
* Dinamikus tesztelés
	+ Strukturális tesztelés
	+ Funkcionális tesztelés

A szoftver futtatása nélkül: a szoftver belső szerkezetének / felépítésének vizsgálata: **statikus tesztelés**

A szoftver működés közbeni vizsgálata: **dinamikus tesztelés**

Széleskörűen alkalmazott értelmezés szerint: a **statikus tesztelés** szemlézést, inspekciót, elemzést jelent. A „teszt eset” kifejezést inkább **dinamikus tesztelés** esetén használják.

Általában a statikus és dinamikus, az utóbbin belül pedig a strukturális és funkcionális technikákat megfelelő arányban kombinálva alakul ki az adott esetben jó tesztelési módszer. A jó tesztelési módszert „kikeverni” nem egyszerű feladat! Szaktudást igényel!

**Különbségek statikus és dinamikus tesztelés között**

* Lehetnek azonos céljaik – kiegészítik egymást
* Statikus tesztelés: hibák közvetlen megtalálása
	+ Dinamikus tesztelésben: meghibásodásokat találunk
* Statikus teszteléssel könnyebben és olcsóbban megtalálható hibák:
	+ Követelmény hibák
	+ Tervezési hibák
	+ Kódolási hibák
	+ Szabványoktól eltérés
	+ Helytelen interfész specifikációk
	+ Biztonsági sebezhetőségek
	+ Hiányok vagy pontatlanságok a tesztbázis nyomonkövethetőségében vagy a lefedettségben

# Ismertesse a tesztelési folyamat alapvető dokumentumait. Mutasson rá, hogy melyik dokumentum tipikusan melyik teszt fázisban kerül kidolgozásra.

**Legmagasabb szinten értelmezve a kérdést**

* Teszt tervezési dokumentumok
* A tesztek bizonylatolása
* A hibák bizonylatolása

**Részletesebben (ezeket vettük, de amúgy nincs teljesen egységes szabvány)**

* **Tesztelési irányelvek**
	+ Magas szintű dokumentum, amely a szervezet elveit, megközelítésmódját, valamint céljait mutatja be a tesztelésre vonatkozóan. Általános leírás; vállalati érettség függvénye, hogy mikortól létezik.
* **Tesztterv (és ennek részletezése)**
	+ A teszt hatáskörét, megközelítését, erőforrásait, valamint a tevékenységek tervezett ütemezését tartalmazó dokumentum.
	+ Ezen kívül meghatározza a tesztelemeket, a tesztelendő funkciókat, feladatokat, a tesztet végrehajtó személyek függetlenségét, a tesztkörnyezetet, a műszaki teszttervezési technikákat, a belépési és kilépési feltételeket, valamint kockázatokat. A teszttervezési folyamat meghatározó dokumentuma [IEEE 829 alapján].
	+ A tesztelés kezdetétől, folyamatosan karbantartva a tevékenység végéig. Szervezete válogatja, hogy pontosan miket rögzítenek bele és meddig tartják karban.
* **Teszteljárás / forgatókönyv (Tesztszkript)**
	+ A teszt futtatásának tevékenységsorozatát rögzítő dokumentum. Tesztszkript illetve manuális tesztszkript néven is ismert. [IEEE alapján].
	+ Legtöbbször teszteljárás specifikációra használt kifejezés, elsősorban automatizált teszt esetén.
	+ **Manuális tesztszkript:** manuálisan létrehozott teszt esetek sorozata, több tesztadattal, mely segít a kevésbé hozzáértőknek is a teszt elvégzésében.
	+ **Automatizált tesztszkript:** programozott teszt esetek, különböző tesztadat kombinációkkal, melyeket eszközök végeznek.
	+ Nyilvánvalóan a végrehajtás előtt a műszaki teszttervezés és a tesztmegvalósítás során
* **Teszteset**
	+ Egy tesztelemre vonatkozó, a teszteseteket leíró dokumentáció (cél, bemenetek, teszttevékenységek, elvárt eredmények, végrehajtás előfeltételei) [IEEE 829 alapján].
	+ Műszaki teszttervezés és a tesztmegvalósítás során
* **Tesztadat**
	+ Olyan adat, amely a teszt előtt is létezik (például egy adatbázisban) és amely kölcsönhatásban van a tesztelés alatt álló rendszerrel, vagy a rendszerkomponenssel.
	+ input; előfeltételek; output; utófeltételek
	+ műszaki teszttervezés és a tesztmegvalósítás során
* **Tesztnapló**
	+ A tesztvégrehajtáshoz kapcsolódó részletek időrendi rögzítése.
	+ Teszt végrehajtás során
* **Hibajelentés**
	+ Olyan dokumentum, amely leírja a szoftver azon hibáit, amelyek a program elégtelen működéséhez vezethetnek.
	+ Teszt végrehajtás és teszt lezárás során
* **Összefoglaló tesztjelentés**
	+ A teszttevékenységeket és eredményeket tartalmazó dokumentum. Ebben a dokumentumban található a kilépési feltételeknek megfelelően ellenőrzött tesztelemek kiértékelése is.
	+ Teszt lezárás során
* **Teszt kiértékelési jelentés**
	+ A tesztfolyamat végén készített dokumentáció, amely összegzi az összes teszttevékenységet és eredményt. Ezen kívül tartalmazza a tesztfolyamat kiértékelését és a teszt során szerzett tapasztalatokat.
	+ Teszt lezárás során

# Mutassa be, hogy a szoftverfejlesztési életciklusban általában milyen tesztelési technikák alkalmazása szokásos az egyes fázisokban. Javaslat: vegye alapul a V-modellt.

Ez több tétel elegye. Itt talán az segít a legtöbbet, ha ezt érted:

* A lényeg, hogy a V modell lemenő ágán az elemeknek megvan a tipikus párja a felmenő ágon
* Bármilyen munkaterméket lehet statikusan tesztelni, de inkább a követelményektől a kódolásig jellemző. Eleinte inkább szemle, kódolásnál már kódelemzés is játszhat.
* Unit tesztet tipikusan fejlesztők írják (strukturális jellemző)
* Integrációs teszten a (feltételezetten) sikeresen tesztelt unitokat tesztelik dinamikus módszerekkel, de inkább strukturális (fehér dobozos) a jellemző
* A rendszer tesztet úgy készítik, hogy azt egy mezei felhasználó is megérthesse. Dinamikus módszereket alkalmaznak, de inkább a funkcionális (fekete dobozos) a jellemző



# Statikus tesztelés. Mutassa be a statikus tesztelés jellemzőit és legfontosabb, az előadásokon vizsgált technikáit. Milyen munkatermékekre alkalmazhatunk statikus tesztelést? Melyek a statikus tesztelés előnyei, és milyen hátrányai lehetnek?

* A tesztelésnek az a formája, amikor magát a szoftvert nem használjuk (nem futtatjuk)
* Statikus tesztelési módszerek: kódszemlézés, inspekció, „walkthrough”, statikus kódelemzés
* Mindegyik emberi „megfigyelésen” alapul
* Bármilyen munkatermékre alkalmazható
* Hatásos
* Alapelv: futtatással nem tudunk mindent lefedni – viszont pl. a teljes kódot át lehet nézni
* Cél nemcsak a hibák megtalálása, hanem okaik felderítése és további előfordulásuk meggátolása (pl. a folyamat javításával)
* Egy komponens vagy rendszer tesztelése specifikáció, vagy implementáció szinten a szoftver futtatása nélkül. Például felülvizsgálat vagy statikus forráskód elemzés (HTB).

**Statikus teszteléssel vizsgálható munkatermékek**

* Specifikációk, beleértve az üzleti követelményeket, funkcionális követelményeket, és biztonsági követelményeket
* Leírások, felhasználói történetek, és elfogadási feltételek
* Architekturális és tervezési specifikációk
* Kód
* Tesztver, beleértve tesztterveket, teszteseteket, teszteljárásokat és automatizált tesztszkripteket
* Felhasználói kézikönyvek
* Weboldalak
* Szerződések, projekt tervek, időbeosztások, és költségvetés tervezetek
* Konfiguráció terv és infrastruktúra terv
* Modellek

**Statikus teszteléssel megtalálható tipikus hibák**

* Követelmény hibák (például következetlenségek, többértelműségek, ellentmondások, kihagyások, pontatlanságok és ismétlődések)
* Tervezési hibák (például nem hatékony algoritmusok vagy adatbázis struktúrák, erős kapcsolódás, kismértékű kohézió)
* Kódolási hibák (például változók értékadás nélkül, deklarált, de fel nem használt változók, elérhetetlen kód, ismétlődő kód)
* Szabványoktól való eltérés (nem követi például a kódolási konvenciókat)
* Helytelen interfész specifikációk (eltérő mértékegységek használata például a hívó rendszerben és a hívott rendszerben)
* Biztonsági sebezhetőségek (például a puffer túlcsordulás lehetőségei)
* Hiányok vagy pontatlanságok a tesztbázis nyomonkövethetőségében vagy a lefedettségben (például hiányzó tesztek egy elfogadási feltétel vizsgálatára)

**Statikus tesztelés előnyei**

* Hibák korai megtalálása, javítása
* Olyan hibák azonosítása, amelyek nem találhatók meg könnyen dinamikus teszteléssel
* Hibák megelőzése a tervezésben vagy a kódolásban
* A fejlesztés termelékenységének növelése
* A fejlesztési költségek és időigény csökkentése
* A tesztelési költségek és időigény csökkentése
* A szoftver élettartama során a minőség teljes költségének csökkentése azáltal, hogy kevesebb meghibásodás kerül elő az életciklus későbbi szakaszában, vagy a működésbe állítás után
* A felülvizsgálatokban való részvétel javítja a csoporttagok közötti kommunikációt

**Statikus tesztelés hátrányai**

* Nem könnyű rávenni a szervezetet (érettséget és szervezést igényel)
* Emberi „megfigyelésen” alapul – sokat számíthat a tapasztalat

**Statikus analízis technikák**

* a program belső szerkezetének vizsgálata
* a program teljességének és konzisztenciájának vizsgálata előre meghatározott szabályok alapján
* a program összehasonlítása a specifikációjával vagy dokumentációjával.

Statikus kódelemzés

* A szoftver forráskódjának elemzése azzal a céllal, hogy megértsük, mit csinál a szoftver. Ugyanakkor, helyességi kritériumokat is felállítunk.
* Többféle elemzési technika van, egy részüket eszköz is támogatja
	+ A „veszélyesnek látszó” elemeket keressük
	+ Formális módszerek, amelyekkel a program szemantikája matematikailag leírható

Statikus analízis

* Formális módszer-család, amellyel egy szoftver viselkedéséről automatikusan információ nyerhető
* A statikus analízis egyik alkalmazása pl. egy debuggoló, amely a futási időben történő hibákat segít megtalálni
* A statikus analízis eredménye nem egyértelmű: tulajdonképpen semmilyen módszerrel nem lehet eldönteni, hogy futtatás során megjelennek-e hibák

Átvizsgálás (walkthrough)

* kód felosztása részekre, célhoz kapcsolódóan, majd átvizsgálása

**Felülvizsgálat**

formálistól az informálisig

* informális felülvizsgálat
* átvizsgálás
* technikai felülvizsgálat
* inspekció

Szerepek lehetnek:

* szerző - létrehozza a munkaterméket
* management - megtervezi, időkeretet teremt
* moderátor - hatékony levezetése a felülvizsgálatot
* felülvizsgálati vezető - általános felelősség a felülvizsgálat végrehajtásáért, résztvevők meghatározása
* Felülvizsgáló - szakértők, jártas személyek
* írnok - jegyzőkönyvvezető

Felülvizsgálati technikák:

* ad hoc - nincs iránymutatás
* ellenőrzőlista alapú - előnye, hogy szisztematikusan lefedi a gyakori hibákat
* forgatókönyvek és száraz tesztelés (elméleti futtatás) - strukturális iránymutatást ad, pl user storyk alapján
* perspektíva alapú - különböző érintett felek szempontjaiból (marketing, tervezői, üzemeltetői)
* szerep alapú - különböző érintett felek, hasonló a perspektívához, főként különböző user szemszög (tapasztalt, gyerek, idős)

# Dinamikus tesztelési technikák. Röviden mutassa be a dinamikus tesztelésben a teszt esetek meghatározásához alkalmazott két alapvető megközelítést. Vannak tiltott technikák bizonyos szoftverek esetében?

* Funkcionális elvárások alapján: funkcionális / fekete doboz tesztelésnek nevezzük. A teszt eseteket a specifikáció alapján határozzuk meg.
* A szoftver belső szerkezetének alapján: strukturális / fehér doboz tesztelésnek nevezzük. A teszt eseteket a kód alapján határozzuk meg.

**Dinamikus tesztelés**

* A rendszer / modul futtatását feltételezi
* A rendszer próbafuttatása, teszt környezetben, teszt adatokkal

Kétféle lehet:

* Fekete doboz teszt (funkcionális teszt)
* Fehér doboz teszt (strukturális teszt)

**Funkcionális tesztelés**

* Funkcionális elvárások alapján
* A teszt eseteket a specifikáció alapján határozzuk meg
* Nem feltételezi a kód belső szerkezetének ismeretét
* A halmazelmélet és függvényelmélet alapfogalmait felhasználjuk a funkcionális tesztelésben
* Bármely program tekinthető egy függvénynek, amely az input értékeket (értelmezési tartomány) az output értékekre (képtér) képezi le
* A funkcionális tesztelés alapvetően az input értékek felhasználásával kialakított teszt esetek halmazát jelenti. A teszt esetekkel az output értékek viselkedését figyeljük.

Funkcionális tesztelés előnyei:

* Független a szoftver implementálási módjától
* A teszt esetek fejlesztése a szoftver fejlesztésével egy időben történhet (rövidül a projekt átfutási ideje)

Funkcionális tesztelés hátrányai:

* A teszt esetek között lényeges redundancia fordulhat elő
* Nagy arányban maradhat teszteletlen a szoftver
	+ tehát kombinálni kell más technikákkal

**Strukturális tesztelés**

* A forráskódon alapszik
* „Abszolút” alapokon nyugszik
* Technikáiban pontos definíciókat, matematikai analízist alkalmaz
	+ Az alkalmazott technikák már a ’70-es években léteztek
* Egészen pontos méréseket tesz lehetővé

Strukturális tesztelés előnyei

* Szilárd elméleti alap: a gráfelmélet elemeit használva egészen pontosan leírható, hogy mit tesztelünk
* Pontos lefedettségi metrikákat tesz lehetővé. Ezekkel pontosan mérhető, hogy egy szoftver elemet milyen mértékben teszteltünk.

Strukturális tesztelés hátrányai

* Befektetést igényel, hogy a tesztelők az elméleti alapokat elsajátítsák
* A szilárd elméleti alap nélkül nem lehet jól csinálni
* Nagyon hangsúlyozza, hogy a tesztelés komoly „szakmai” feladat

**Vannak tiltott technikák bizonyos szoftverek esetében?**

Nincsenek. A cél szentesíti a „technikát”. Bármilyen technikával lehet tesztelni, maximum nem lesz hatékony.

# Funkcionális és strukturális tesztelés. Mutassa be, milyen alapvető különbség van a két megközelítésmód között. Kombinálható-e a két megközelítés? Mondjon példát a funkcionális és strukturális tesztelés hatékony kombinálására.

**Lásd 14-es tétel. Egyébként Csanád:**

fekete dobozos (funkcionális)

* teszt esetek specifikáció alapján vannak meghatározva
* nem feltételei a kód belső ismeretét

fehér doboz (strukturális)

* a rendszer belső ismeretét feltételezi

A jó tesztelési módszer a technikákat megfelelő arányban kombinálva alakul ki.

Hitviták alakultak ki a kérdéskör körül:

* Funkcionális tesztelés előnyei
	+ független a szoftver implementálási módjától
	+ a teszt esetek a szoftver fejlesztésével egyidőben történhetnek, rövidebb idő
* Funkcionális tesztelés hátrányai
	+ nagy arányban maradhat teszteletlen a szoftver
* Strukturális tesztelés előnyei
	+ gráfelméletet használva pontosan leírható, hogy mit tesztelünk
	+ lefedettségi metrikákat tesz lehetővé
* Strukturális tesztelés hátrányai
	+ elméleti alap nélkül nem lehet megcsinálni, szaktudást igényel

**Kombinálhatók-e?**

Igen

**Hatékony kombináció**

Integrációs tesztelésnél strukturális módszerekkel eldöntöm, hogy mely komponenseket választom be egy tesztbe és ezeket az (immáron integrált) komponenseket tesztelem funkcionálisan.

# Funkcionális tesztelési technikák. Ismertesse a tananyagban szereplő 5 alapvető funkcionális tesztelési technikát. Hogyan viszonyulnak ezek egymáshoz a kialakított teszt esetek száma és a tesztelési ráfordítás szempontjából?

**5 alapvető funkcionális tesztelési technika**

**Határérték tesztelés**

* A tesztesetek az input tartomány határán található értékeket használják – tipikusan a ciklusokban használt feltételeknél figyelték meg a feltételek hibáit.
	+ min, min+, nom, max-, max
	+ n változó esetén 4n+1 teszteset
* feltételezés, hogy egy megjelenő belső hiba, csak nagyon ritka esetben egynél több belső hiba eredménye.
* egyszerre csak egy változó értékét módosítjuk, ez veszi fel az értékeket. A többi nominális
* Nem veszi figyelembe a függvény típusát, sem a változók jelentését - kezdetleges
* Akkor alkalmazható hatékonyan, ha program független változók függvénye, és ezek a változók „korlátos” fizikai változókat jelölnek.
* Robosztusság tesztelés: hozzávesszük még a min- és max+ értékeket is.
	+ felhívja a figyelmet a különleges helyzetekre
* Worst case test: nem csak 1 változót változtatunk, minden változónál mind az 5 érték keresztszorzatát nézzük
	+ 5n teszt eset
	+ ennek is van robosztus változata
* jól használható hibaüzenetek figyelésére
* minden szinten alkalmazható
* fedettség: (tesztelt határértékek száma) / (összes azonosított határ érték) százalékban
* tipikus találat: hibás adatkezelés / adatok a határokon

**Ekvivalencia partícionálás**

* Szeretnénk „teljes” tesztelést. Nem akarunk redundáns adatokkal tesztelni
* Ekvivalencia osztályokat hozunk létre
	+ egy halmaz egy partícióját alkotják
	+ a partíciók diszjunktak, uniójuk a teljes halmaz
	+ minden osztályból legyen egy input teszt adat: teljesség
	+ a diszjunktság miatt nincs redundancia
* „egyszerre egy hiba” elv (nem valószínű, hogy a hibákat több változó együttesen okozza, valószínűbb, hogy minden hibát egyetlen változó okoz)
	+ ez a gyenge normál ekv. oszt. alapú tesztelés
* Több változó is lehet hibás feltételezés
	+ erős normál ekv. oszt. alapú tesztelés
	+ az osztályok descartes szorzatának minden elemére szükség van
* gyenge robosztus ekv. oszt. alapú tesztelés
	+ egyszerre 1 hibát feltételezünk
	+ érvénytelen értékekere is kipróbáljuk
* erős robosztus ekv. oszt. alapú tesztelés
	+ egyszerre több hibát feltételezünk
	+ az osztályok descartes szorzatának minden eleme + érvénytelen inputok
	+ redundáns!
* Az ekvivalencia osztály alapú tesztelést erősíti a határérték analízissel kombinált megközelítés
* Ekvivalencia osztályokat akkor érdemes létrehozni, amikor az input adatokat intervallumként vagy diszkrét értékek halmazaiként adják meg
* minden szinten alkalmazható
* fedettség: (legalább 1 értékkel tesztelt partíciók száma) / (összes partíciók száma) százalékban
* tipikus találat: hibás adatkezelés, hibás adatok

**Döntési táblák**

* A logika szigorán alapszik
* akkor javasolt, ha különböző akciók kombinációra kerül sor, változó feltételrendszerek mellett
	+ if-then-else
	+ inputok közt logikai kapcsolat van (**ha függnek egymástól a változók**)
* Szabályokat hozunk létre
	+ feltételek
	+ akciók
* A döntési tábla teljessége ellenőrizhető, így az is, hogy minden lehetőséget fedtünk tesztesetekkel
* Kiterjesztett döntési tábla: a (N/A) helyett T és F esetet is figyelembe veszünk
* Előfordulhat, hogy ugyanahhoz a bemenethez többféle akció is társul. Emiatt a tábla inkonzisztens, ezeket az eseteket meg kell szüntetnie a tesztelőnek
* a jó tábla iteratívan alakul ki
* fedettség: (legalább 1 eset által fedett döntési szabályok száma) / (összes döntési szabályok száma) százalékban
* tipikus találat: üzleti folyamat / szabályok hibái

**Állapotátmenet tesztelés**

Egy állapotátmenet diagram megmutatja a lehetséges szoftverállapotokat, illetve azt, hogy a szoftver hogyan lép be egy állapotba, hogyan lép ki onnan, valamint az állapotok közötti átmeneteket. Az állapotátmeneti tábla tartalmazza az állapotokat, az összes köztük érvényes és érvénytelen átmenetet, az eseményeket, illetve az érvényes átmenetekhez tartozó végrehajtandó műveleteket.

A tesztek tervezhetők:

* az állapotok egy tipikus sorozatának lefedésére
* minden állapot végrehajtására
* minden átmenet végrehajtására
* átmenetek meghatározott sorozatának végrehajtására
* érvénytelen átmenetek tesztelésére
* Tipikusan beágyazott szoftverek, menü alapú alkalmazásokhoz
* fedettség: (tesztelt azonosított állapotok v. átmenetek száma) / (összes azonosított állapot v átmenetek száma) százalékban
* tipikus találat: állapotok közti átmenetek hibái

**Használati eset tesztelés**

* Olyan tesztelési technika, amiben műszaki tesztterveket különböző használati eset forgatókönyvek futtatására tervezték – tehát a tesztesetek használati esetekből származtathatók.
* A használati eseteket aktorokhoz és tárgyakhoz kötjük
	+ aktor: emberi felhasználó, külső hardver, másik komponens
	+ tárgy: komponens vagy rendszer
* a használati esetek egy-egy viselkedést határoznak meg, amit a tárgy hajt végre együttműködésben az aktorokkal
* tehát meghatározott viselkedések végrehajtására tervezik a teszteket
* fedettség: (tesztelt használati eset viselkedések száma) / (összes használati eset viselkedés számával) százalékban
* tipikus találat: üzleti folyamat / szabályok hibái
* agilis környezetben ez user story alapján megy

**Hogyan viszonyulnak ezek egymáshoz a kialakított teszt esetek száma és a tesztelési ráfordítás szempontjából?**




Tehát a könnyen alkalmazható technikák sok esetet generálnak, amiket sokáig is tart lefuttatni.

**Összegzés**

* Ha fizikai változókról van szó vagy a független változókról van szó:
	+ BVA tesztelés és ekvivalencia osztály alapú tesztelés hatékony
* Ha a változók függnek egymástól:
	+ döntési táblákat használjunk
* Ha „egyszerre egy belső hiba” valószínű:
	+ a határérték tesztelés és robosztussági tesztelés a kívánatos
* „Több hiba együttesen” esetében:
	+ legrosszabb eset, robosztus legrosszabb eset és döntési táblák ajánlatosak
* Ha a programban lényeges a kivételkezelés:
	+ robosztussági teszt és döntési táblák
* Ha a változók logikaiak:
	+ ekvivalencia osztály alapú tesztelés és döntési táblák használata ajánlott

# Strukturális tesztelési technikák: utasításalapú, döntésalapú, adatfolyamalapú tesztelés.

Strukturális tesztelés:

* forráskódon alapszik, tehát fehér dobozos
* pontos definíciókat, matematikai analízist használ
* pontos méréseket tesz lehetővé

**utasításalapú**

* A programnak megfeleltetünk egy gráfot
	+ két csomópont között a gráfban akkor van él, ha a nekik megfelelő utasítások vagy utasítás részletek közvetlenül egymás után lehet végrehajtani
	+ ez a CFG – Control Flow Grap, vezérlésfüggés gráfja
* törekedni kell az egyszerű CFG-re
	+ el kell dönteni, hogy utasítás vagy utasítás részlet lesz csomópont
	+ típusdeklarációhoz teszünk-e vagy sem
	+ könnyen generálódhat annyi út, amit képtelenség tesztelni
* a kódban lévő potenciálisan végrehajtható utasításokat hajtja végre
* a lefedettség: (tesztek által végrehajtott utasítások) / (az összes utasítás) százalékban

**döntésalapú tesztelés**

* a kódban lévő döntéseket hajtja végre
* minden lehetséges logikai (if-then-else, case) ágat letesztel (ezek lesznek a tesztesetek)
* döntés-döntés útvonalak alapján (DD-Path)
	+ szekvenciális sorozat elágazás nélkül
* a DD-Path gráf egy irányított gráf, amelyben a csomópontok a program gráfjának DD-Path-jai, az élek pedig egymást követő DD-Path-ok közötti vezérlést mutatják
* egy programhoz több programgráf is társítható, de ezek mindegyike egy egységes DD-Path gráfra redukálható
* kb. 100 soros kódig érdemes DD-Path gráfokat létrehozni, efölött már automatizáló eszközökre érdemes támaszkodni
* fedettség: (tesztelt döntési kimenetek) / (összes döntési kimenet) százalékban

100% döntési lefedettség 100%-os utasításlefedettséget jelent, de ez fordítva nem igaz

**adatfolyam alapú tesztelés**

* azokra a pontokra fókuszál, ahol a változók értéket kapnak és azokra, amelyekben ezeket az értékeket felhasználják
* felfedezhető hibák:
	+ változó, amit sehol nem használunk
	+ használt változó, amely nincs definiálva
	+ többször definiált változó
* Alfajai:
	+ define / use tesztelés: programgráf segítségével vizsgáljuk a programot. az alapja, hogy a gráf információja, hogy egy változót mely utasításokban / utasítás részletekben definiálnak és használnak
	+ szeletelés: programgráfot kisebb szeletekre bontjuk (változóra vonatkozva)

# A különböző tesztelési szintek technikái. Mennyire kapcsolódnak a tesztelési szintekhez bizonyos technikák? Vannak olyan technikák, amelyek tiltottak egy tesztelési szinten?

Lásd 12-es tétel (80% egyezés)

**Mennyire kapcsolódnak a tesztelési szintekhez bizonyos technikák. Tiltások?**

Lazán. Vannak jellemzők, de nincs szentírás, a cél szentesíti a „technikát”. Nincsenek tiltások szintekhez kapcsolódóan. Nem a legelső ötlet lenne adatfolyam alapon rendszer tesztelni, de ha egyébként a rendszerteszt így működik és ez kell a tervben kitűzött célok eléréséhez, akkor ez kell.

# Strukturális technikák integrációs tesztelésben. Ismertessen legalább 2 példát arra, hogy integrációs teszteléshez hogyan lehet teszt eseteket létrehozni strukturális technikák alkalmazásával.

Alapprobléma integrálni kell részeket, de miket?

**Funkcionális dekompozíció**

Kétféleképpen lehet hozzáállni:

**Top-down**

* A fő programmal kezdünk, teszteljük a következő szintre vonatkozó hívásokat
	+ majd haladunk lejjebb a funkciókkal
* az alacsonyabb szint funkcióját mindig csonkkal (stub) helyettesítjük, ami kidobható és a valódi kódot helyettesíti
* csonkokat írni időigényes. érdemes kódként kezelni, konfiguráció kezelni
* ATM rendszert alapul véve: 1. core funkció, aztán terminál IO, tranzakció kezelés stb

**Bottom up**

* A legalsó szinttel kezdjük (példánál maradva kártya behelyezés, pin megadás)
	+ majd haladunk feljebb a funkciókkal
* a magasabb szint kódját, mindig egy driverrel helyettesítjük, ami a magasabb szintű hívásokat emulálja
* kevesebb drivert kell írni, mint csonkot kellene fordítva, de a driverek komplexebbek

Ezeken kívül lehet még kombinálni, és az egészet egyben is tesztelni.

A következő kérdés, hogy hány elemet rakunk össze ezekből egy-egy tesztre? (Az biztos, hogy már tesztelt komponensek integrációját teszteljük.) Ha sokat, akkor kevés esetünk lesz, de nehezen találjuk meg az esetleges hiba helyét. Ha csak két elemet integrálunk egyszerre, akkor könnyű lesz megtalálni a hibákat, de sok teszteset lesz.

**Pro:**

* intuitív
* tesztelt egységekkel dolgozunk
* hiba esetén a tesztelt egységet kell vizsgálni (van kiindulásunk a feltáráshoz)
* az előrehaladást a dekompozíciós fával könnyű követni

**Kontra:**

* ez mesterséges és feltételezzük, hogy a jól működő részeket integrálva működő rendszert kapunk
* csonkok és driverek előállítása lényeges ráfordítás

**Hívási gráf alapú integráció**

* hívási gráfot készítünk, ahol a csomópontok az eljárások az élek pedig a hívások köztük
* párokra figyelő integráció: egy test session egy gráf elempárra vonatkozik (gráf éle)
	+ nincsenek csonkok és driverek
	+ de páronként megy a tesztelés, sessionök száma nagy
* szomszédsági integráció
	+ gráfbeli szomszédosság alapján
	+ a hívásban a megelőző szomszéd és a követő szomszédokat válogatjuk integrációra
	+ itt sincsenek csonkok és driverek
	+ kevés sessionből megvan
	+ de nehéz a hiba helyét meghatározni

**Pro:**

* tisztán funkcionális megközelítés helyett strukturális irány
* nincsenek csonkok és driverek
* jól illeszkedik a buildeket létrehozó fejlesztésekhez
	+ pl. szomszédság alapján hozzuk létre

**Kontra:**

* nehéz a hiba megtalálása, főleg nagy szomszédságok esetén
* ha a hiba olyan csúcsban (egységben) van, amelyik több szomszédságnak is eleme, akkor megváltozik a kód -> további tesztek kellenek, nem elég egyet újra ellenőrizni.

# A tesztelés menedzsmentje. Ismertesse a legfontosabb teszt menedzsment tevékenységeket. Mit jelent a „defect leakage” és mire alkalmazható?

**Tipikus tesztmenedzser feladatok**

* A szervezeti tesztstratégia és tesztterv kialakítása vagy felülvizsgálata
* A tesztelési tevékenységek tervezése a körülmények figyelembevételével és a tesztcélok, kockázatok megértése.
* Ide tartozhat a tesztelési megközelítések kiválasztása, a tesztelés időtartamának, ráfordításainak és költségeinek becslése, erőforrások beszerzése, tesztszintek és ciklusok meghatározása, a hibamenedzsment tervezése
* Tesztterv(ek) készítése és frissítése
* Teszttervek koordinációja projekt menedzserekkel, termék tulajdonosokkal és másokkal
* A tesztelési perspektívák megosztása más projekttevékenységekkel, mint például az integráció tervezés
* A tesztek elemzésének, előkészítésének, kivitelezésének és végrehajtásának kezdeményezése, a tesztfolyamatok és eredmények monitorozása, a kilépési feltételek (vagy a „kész” definíció) állapotának ellenőrzése és a tesztelés elvégzésének segítése
* Az összegyűjtött információk alapján a tesztstátuszjelentés és az összefoglaló jelentés előkészítése és átadása
* A tervezés átalakítása a teszteredmények és a tesztelés előrehaladása alapján (amit gyakran tesztstátuszjelentésekben, vagy a projekt során már elvégzett tesztfeladatok összefoglaló jelentésében dokumentálnak), a tesztirányításhoz szükséges lépések megtétele.
* A hibamenedzsment rendszer kialakításának támogatása és a tesztver megfelelő konfiguráció menedzsmentjének létrehozása
* Megfelelő metrikák bevezetése a tesztelés állapotának mérésére, valamint a tesztelés és a termék minőségének értékelésére
* A tesztelési folyamatot segítő eszközök kiválasztásának és bevezetésének támogatása
* Döntéshozatal a tesztkörnyezet(ek) kialakításáról
* A tesztelők, a teszt csapat és a tesztelői hivatás segítése és támogatása a szervezeten belül
* A tesztelők képességeinek és karrierlehetőségeinek javítása

**Mit jelent a defect leakage?**

Maga a fogalom azt a jelenséget takarja, hogy bizonyos hibák észrevétlenül a következő fázisba haladnak tovább. Ugyanezen a néven hatékonysági metrika is, mellyel mérhető egy szervezet QA hatékonysága. Gyakorlatilag az átvételi tesztek alatt talált hibák száma, osztva a QA által korábban megtalált hibák számával.

# Kockázatkezelés, konfigurációkezelés, mérés a tesztelés szolgálatában. Milyen mérőszámokat szokás alkalmazni a tesztelés hatékonyságának figyelésére?

**Csanád féle**

Általánosan akkor jó a tesztelés

* ha a hibák minél korábban derülnek ki
* kód alapú lefedettség mérhető
* elkészült termék az elfogadási kritériumnak megfelel
* vevő elégedett

Konfigurációmanagement

* célja a munkatermék integritásának biztosítása, a konfiguráció azonosításával, ellenőrzésével, állapotának követésével
* mindent, ami fontos konfigurációkezelés alá kell vonni
* lehetnek alapverziók, amik már jóvá lettek hagyva
* a verziókezelés fontos eleme, ha lehet ezt automatizáljuk (pl SVN, Git)

Kockázatmanagement

* kockázat: annak a valószínűsége, hogy előre nem látható esemény fordul elő
* kockázatmanagement: tevékenységek összessége, melyek elősegítik, hogy egy tevékenység tényleges eredménye a lehető legjobban közelítse az elvárt eredményt
* **termékkockázat**: a projekttermék nem felel meg a felhasználók igényeinek, a minőségi jellemzők nem megfelelőek
* **projektkockázat**: olyan helyzetek, melyek negatív hatással lehetnek a projekt céljainak elérésére, pl. késedelem, pontatlan becslés, késői változtatás
* lépések: kockázat azonosítása, elemzése, tényezők priorizálása, intézkedések összeírása végrehajtása, eredmények követése
* risk based testing

Hibamegtalálási százalék (DDP)

* (ebben a tesztelésben megtalált hibák / összes hiba beleértve azokat is amiket csak később fogunk megtalálni) \* 100
* ez előrejelzésre is jó, ha tudjuk, hogy eddig milyen %-ok voltak a korábbi release-ek hasonló fázisában (feltéve, hogy nagyságrendileg ugyanarról a fejlesztő közegről van szó)

Hibamegtalálási arány (DDR)

* bármely fázisban megtalált, összes hibák száma/abban a fázisban eltöltött tesztelési, szemlézési idő

Defect leakage:

* Lásd 20-as tétel

# Kockázatkezelés, konfigurációkezelés, mérés a tesztelés szolgálatában. Milyen kockázatokkal foglalkozunk a szoftvertesztelés során?

A kockázatmenedzsment olyan tevékenységek összegzése, amelyek elősegítik azt, hogy egy tevékenység tényleges eredménye a lehető legjobban közelítse meg az elvárt eredményt.

* **termékkockázat**: a projekttermék nem felel meg a felhasználók, érdekelt felek igényeinek. A termék sajátos minőségi jellemzőivel kapcsolatos kockázatokat minőségi kockázatoknak is nevezik.
	+ pl nem a specifikációnak megfelelően működik a szoftver, vagy nem támogatja kellőképpen a nemfunkcionális követelményeket (UX, teljesítmény stb)
	+ termékkockázatokat a tesztelés hivatott megoldani
* **projektkockázat**: olyan helyzetek, melyek negatív hatással lehetnek a projekt céljainak elérésére, pl. késedelem, pontatlan becslés, késői változtatás, szervezeti és politikai problémák, technikai problémák

**Kockázatmenedzsment**

* Elkerülés
	+ ne indítsuk el a veszélyeztetett tevékenységet (aminek a kimenetele kétséges)
* Csökkentés
	+ a tevékenység megkezdése előtt tegyünk lépéseket, hogy a kockázati elem hatása minimálisra (elfogadható nagyságúra) csökkenjen
* Kompenzálás
	+ fogadjuk el a kockázati tényező negatív hatását a tevékenységre, de egyéb tényezőkre figyelve igyekezzünk ezt a negatív hatást elfogadható mértékűre csökkenteni
* Megegyezés
	+ tételezzük fel, hogy a kockázati tényező kifeji hatását és készüljünk fel a negatív hatás kezelésére

**Agilis környezetben**

Szedjük össze a súlyozott esélyeket és a súlyozott hatásokat egy mátrixban. helyezzük el a kockázatokat és kezdjük azokkal, ahol a súlyértékek szorzata a legnagyobb és haladjunk a legkisebbek felé.



# Miben különbözik a teszt menedzsmentben használt teszt terv a műszaki teszt tervtől? A hibák bizonylatolásán kívül mit szükséges még bizonylatolni a tesztelés végrehajtása során?

**Menedzsmentben használt tesztterv**

* A tesztelés tárgyának, a céloknak és a kockázatoknak a meghatározása
* A tesztelés általános megközelítésének definiálása
* A teszttevékenységek koordinálása és beépítése a szoftver életciklusába
* Döntéshozatal arról, hogy mit tesztelünk, mely személyek, illetve erőforrások szükségesek a teszttevékenységek végrehajtásához, és hogy ezen tevékenységeket hogyan kell végrehajtani
* A tesztelemzési, tervezési, megvalósítási, végrehajtási és értékelési tevékenységek ütemezése adott időpontra (például a szekvenciális fejlesztésben) vagy az egyes iterációkhoz kötődően (például az iteratív fejlesztésben)
* Metrikák kiválasztása a tesztfelügyelethez és irányításhoz
* Teszttevékenységek költségvetésének meghatározása
* A tesztdokumentáció részletességének és struktúrájának meghatározása (például sablonok vagy mintadokumentumok rendelkezésre bocsátása).

**Műszaki teszttervezés (lásd 6-os tétel)**

A műszaki teszttervezés során a tesztfeltételekből magasszintű teszteseteket, magasszintű tesztesethalmazokat és más tesztvereket alakítanak ki. A műszaki teszttervezés azt válaszolja meg, hogy “hogyan teszteljünk?” (**Tesztver**: A tesztelési folyamat alatt előállított projekttermékek, melyeket a tervezés, műszaki teszttervezés, végrehajtás, kiértékelés és jelentéskészítés során használnak.) A műszaki teszttervezés az alábbi fő tevékenységekből áll:

* Tesztesetek és tesztesetek halmazainak tervezése és priorizálása
* A tesztfeltételek és tesztesetek támogatásához szükséges tesztadatok meghatározása
* A tesztkörnyezet megtervezése, valamint a szükséges infrastruktúra és eszközök azonosítása
* A tesztbázis, tesztfeltételek és a tesztesetek közötti kétirányú nyomon követhetőség meghatározása

**A hibák bizonylatolásán kívül mit szükséges még bizonylatolni a tesztelés végrehajtása során**

A teszteket és a végrehajtást (tesztesetek, szkriptek, tesztnapló, összefoglaló tesztjelentés). A tesztelés tervezése, végrehajtása, kiértékelése dokumentált folyamat!

# Az agilis tesztelés elemei. Az agilis tesztelés és a hagyományos tesztelés kapcsolata. Agilis tesztelési módszerek. Miben különbözik az agilis tesztelői szervezet a hagyományostól?

**Agilis tesztelés**

* Az agilis tesztelés a lehető legkorábbi tesztelést hangsúlyozza a szoftverfejlesztési életciklusban
* Megköveteli a nagyon hangsúlyos vevői részvételt a tesztelésben is, amint a kód hozzáférhetővé válik
* A kódnak elég stabilnak kell lennie ahhoz, hogy lehetővé tegye a rendszertesztelést
* Fontos a regressziós tesztelés
* A kommunikáció a csapaton belül és a csapatok között kulcsfontosságú!

Tehát a hagyományos teszteléssel szemben itt nincs meg előre minden információ, és nem a fejlesztés végén vonjuk be a tesztelőket, sőt. **A tesztelés alapja / fókusza a user story**, és az iterációk során bekerülő elemeket fokozatosan és folyamatosan integrálják a kódbázisba és tesztelik le. Minden új kód integrálása során automatizált tesztek ellenőrzik, hogy a rendszerbe való illesztés során okozott e valamilyen hibát az új kódrészlet és ennek eredményeként a lehető leghamarabb visszajelzést ad az integráció eredményéről.

* Folyamatosan vannak rövid iterációk a tervezés, kódolás és tesztelés tevékenységekre
* A tesztelési tevékenységek is iteratív módon, folyamatosan kerülnek végrehajtásra
* Az agilis tesztelők
	+ Korai fázistól kezdve dolgoznak
	+ A projekt teljes idején be vannak vonva a tervezésbe

Egy XP (extreme programming) tesztelési megközelítés a TDD (Test Driven Development) melynek során először teszteket írnak meg és csak aztán a funkcionalitást.

# A TMMi modell. Ismertesse röviden a TMMi modell célját, szerkezetét, folyamatait!

**Test Maturity Model Integration**

* A tesztelési folyamat fejlesztésére dolgozták ki
* Teljesen illeszkedik a CMMI modellhez
* Lépcsős folyamatfejlesztési modell
* Tanúsítható

A TMMI segítségével:

* szervezetek költséghatékonyan javíthatják a tesztelési folyamataikat
	+ lépcsőzetesen haladva egyre magasabb érettségi szint felé. Ez kvázi jó sorrendet ad, hogy milyen folyamatokkal érdemes foglalkozni a fejlődés útján. (és mikkel nem feltétlenül éri meg még foglalkozni alacsonyabb szinten)
	+ minden szint folyamatokat, tevékenységeket ír le
* összehasonlíthatóvá válnak a certifikált szervezetek tesztelési szempontból

**TMMI szintek és jellemző tevékenységek, fogalmak**

1. Initial (kezdeti)

-

2. Managed (menedzselt)

* Tesztelési irányelvek és stratégia
* Teszt tervezés
* Tesztfelügyelet és -irányítás
* Teszt tervezés és végrehajtás
* Teszt környezet

3. Defined (meghatározott)

* Teszt szervezet
* Teszt tréning program
* Tesztelés életciklus és integráció
* Nem funkcionális tesztelés
* Szemlézés (peer review)

4. Measured (mért)

* Tesztelés mérése
* Szoftverminőség kiértékelés
* Haladó szemlézések, inspekciók

5. Optimization (optimalizálás)

* hiba megelőzés
* Teszt folyamat optimalizálása
* Minőségirányítás

