

Szoftverhibák

Áldozatok:

- ❖ Az ember
- ❖ Egy cég
- ❖ A környezet

☞ *A hibák keresését minél előbb el kell kezdeni!*

Hiba eredménye:

- ❖ Anyagi veszteség
- ❖ Időveszteség
- ❖ Jó hírnév csorbulása
- ❖ Sérülés
- ❖ Halál

Hibák gyakori okai

- ✿ Szoros határidők
- ✿ Tapasztalatlan vagy nem megfelelően képzett résztvevők
- ✿ Nem megfelelő kommunikáció
- ✿ Félreértések
- ✿ Kód, tervezek, architektúra komplexitása
- ✿ Új, ismeretlen technológia
- ✿ Emberi tökéletlenség
- ✿ Környezeti tényezők
- ✿ ...

Okok feltárása



- ✿ Hibák okainak felkutatása rendkívül fontos!
 - ✿ Megfelelő és igazságos kezelési lépések
 - ✿ Későbbi problémák megelőzése
- ✿ **Eredendő ok:** legkorábbi körülmények vagy tevékenységek, amik hozzájárultak a hibához (root cause)
- ✿ A felhasználó általában csak a hatásokat látja

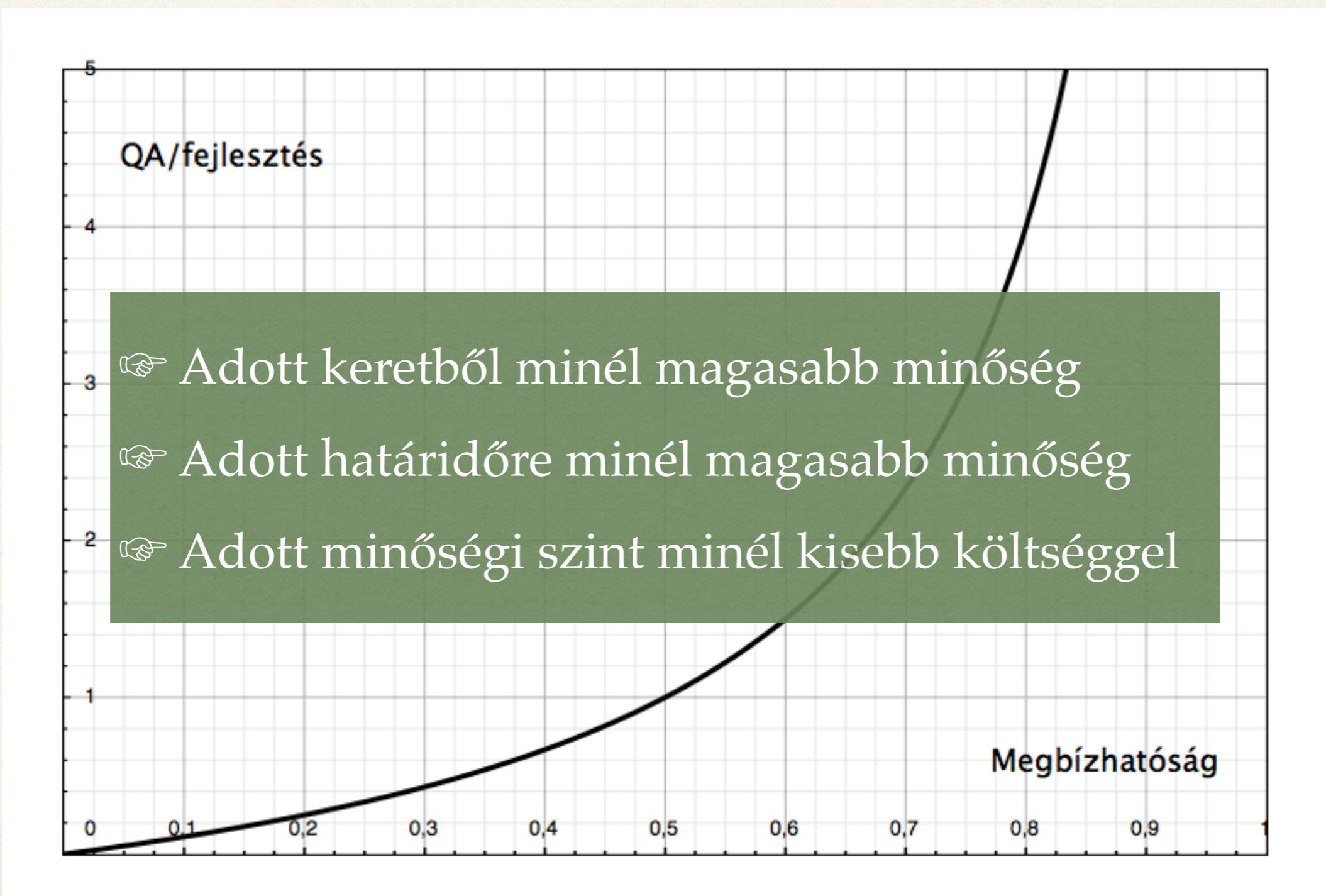
Minőség ára



- ❖ Miért maradhatnak defektusok a rendszerben???
- ❖ Mert **rosszul teszteltünk!**
 - ❖ Keveset teszteltünk
 - ❖ Eleve rossz típusú teszteket használtunk
- ❖ **Hibátlan fejlesztés nem létezik**
 - ❖ A fejlesztés emberi kreatív tevékenység, ami hibalehetőséget hordoz
 - ❖ A több és jobb teszt így indokolt célnak tűnik, de ez nem olyan egyszerű!

Kimerítő tesztelés:
Minden lehetséges
végrehajtás ellenőrzése

Tesztelés = kockázatcsökkentés



Kritikus rendszerek fajtái

Fajtái:

1. Biztonságosság-kritikus
atomerőmű

2. Küldetéskritikus
űrhajó

3. Üzletkritikus
számlavezető rendszer

Károk:



*Közvetlen
költségek*



*Közvetett
költségek*

Mi a tesztelés és mit csinál?

- ✿ A tesztelés nem csak tesztek futtatása, hanem egy nagyobb folyamat, többféle részfeladattal
- ✿ A tesztelés:
 - ✿ *Verifikáció* (“jól készítettük el a rendszert?”)
 - ✿ *Validáció* (“jó rendszert készítettünk el?”)

Mi a tesztelés és mit csinál?

*A tesztelés
a defektusok felfedezésével:*

- ✓ képes felmérni a szoftver aktuális minőségét,
- ✓ hozzájárul a kockázatok csökkentéséhez és ezáltal alapot ad a szoftver minőségének javításához

Mit és mennyit teszteljünk?

- ✿ Nem tesztelhetünk minden
- ✿ Minden rendszernek (és projektnek) megvannak a saját kockázati tényezői és minőségi követelményei
- ✿ Végesek az erőforrásaink
- ✿ DE! A cél: kockázat csökkentése (és nem a tökély elérése)

Tesztelés prioritálása

- ❖ Először a **fontos teszteket futtassuk**, így bármikor hagyjuk is abba, ez a konzervatív
- ❖ **Teljesítési / kilépési kritérium:** előre határozzuk meg, milyen feltételekkel tekintjük befejezettnék a tesztelést, például:
 - ❖ minden tervezett teszt lefutott
 - ❖ minden funkcionális követelményt legalább egyvalaki felülvizsgálta
 - ❖ minden funkcionális követelményt legalább egyszer leteszteltünk
 - ❖ minden ismert, kritikus hibát kijavítottunk
 - ❖ minden eljárást legalább egyszer futtattunk
 - ❖ az összes elágazás 80%-át futtattuk

Tesztelés gyakori céljai

- ✿ Munkatermékek kiértékelése
- ✿ Követelmények teljesülésének ellenőrzése
- ✿ Validáció
- ✿ A teszt tárgyába vetett bizalom kiépítése
- ✿ Hibák és meghibásodások felfedezése
- ✿ Döntés-előkészítés
- ✿ Kockázati szint csökkentése
- ✿ Jogszabályoknak, szerződéseknek, szabályozásnak való megfelelőség ellenőrzése

Tesztelés és hibakeresés

Debugging

- ❖ Két különböző tevékenység
- ❖ Mindkettő fontos és magasabb minőséghez vezet, de
 - ❖ még a debugging semmit nem mond a szoftver egészéről,
 - ❖ addig egy alapos tesztelés (és az azt követő javítások) után mondhatjuk, hogy a szoftver megfelel a követelményeknek

A tesztelés:

- ❖ a fejlesztőtől független
- ❖ defektusok felfedezésére és bejelentésére irányul
- ❖ szisztematikus tevékenység

A hibakeresés, nyomkövetés, hibajavítás (debugging):

- ❖ a fejlesztő végzi
- ❖ célja a defektusok okainak felderítése, kijavítása
- ❖ kevésbé szisztematikus, leginkább ad-hoc

Tesztelés mint szakterület

A tesztelés mint folyamat:

- ❖ A tesztelés sokkal több, mint szimplán a teszt végrehajtása
- ❖ Előkészítés
 - ❖ tervezés
 - ❖ környezet felállítása, stb.
- ❖ „Utómunkák”
 - ❖ jegyzőkönyv vezetése
 - ❖ teszt kiértékelése
 - ❖ teljesítési kritérium kiértékelése
 - ❖ jelentéskészítés, stb.

A teszt céljának meghatározása:

- ❖ Specifikációhelyesség igazolása
- ❖ A lehető legtöbb defektus felfedezése

Statikus és dinamikus tesztelés:

- ❖ Statikus: a kód működését nem vizsgáljuk: review-k
 - ❖ nem csak kódon
 - ❖ minél előbb
- ❖ Dinamikus: teszt futtatás tesztadatokon

Tesztelés mint szakterület

A tesztelés mint módszerek halmaza:

- ✿ Fontos, hogy a teszt effektív legyen, azaz megtalálja a szoftverhibákat
- ✿ Amelyik teszt nem talált hibát, annak nincs a szoftverhez hozzáadott értéke, csak fogyasztotta az erőforrásainkat
 - ✿ Ne örüljünk, nem a szoftver hibátlan...
- ✿ Hogyan lehet jó teszteket készíteni?
 - ✿ Gyakorlatban bizonyított technikák alkalmazásával
 - ✿ Számos alapelv van, amelyekre rengeteg technika épül

Teszt típusok (példák)

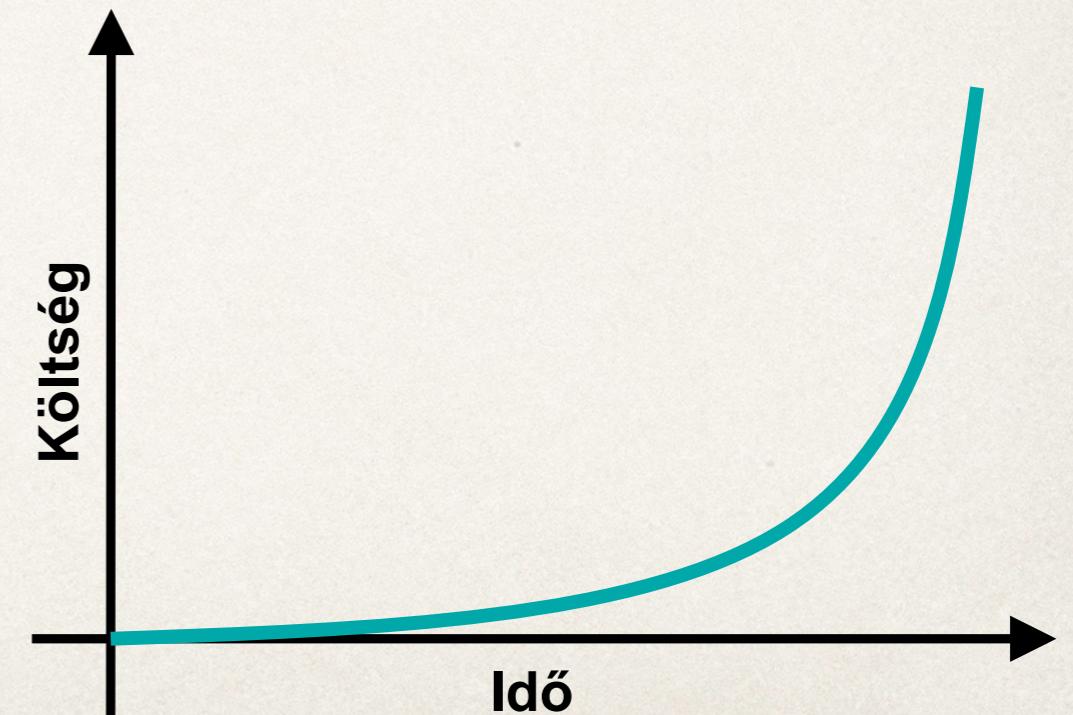
- ❖ Funktionális / nem-funkcionális
- ❖ Statikus / dinamikus
- ❖ Black-box / white-box
- ❖ Ellenőrző tesztelés (megismételt #1)
- ❖ Regressziós tesztelés (megismételt #2)
- ❖ *Terhelés-teszt (stressz-)*
- ❖ *Használhatósági teszt*
- ❖ *Biztonsági teszt*
- ❖ *Alfa / Béta teszt*
- ❖ *“Csimpánz” teszt*
- ❖ *Stb.*

Tesztelés hozzájárulása a sikerhez

- ❖ Megfelelő teszteléssel csökkennhető a hibásan átadott, üzemeltetett rendszerek gyakorisága
- ❖ De, csakis megfelelő szakemberekkel, megfelelő időben és módon!
- ❖ Példák:
 - ❖ Követelmény review: használhatóbb termék
 - ❖ Részvétel tervezéskor: jobb megértés miatt relevánsabb tesztek
 - ❖ Részvétel kódoláskor: jobb minőségű kód
 - ❖ Tesztelés átadás előtt: hibák megtalálása “házon belül”

Korai tesztelés

- ✿ Minél később kezdünk tesztelni, annál kevesebb időnk marad rá
- ✿ Amint a fejlesztési modell egyik fázisát befejeztük, a hozzá tartozó teszteket elkezdhetjük elkészíteni, és ezzel együtt a fázis eredményét is átnézzük
- ✿ Az adott fázisban elkövetett hibát még azelőtt észre kell venni, hogy az a következő fázisban is újabb defektusokat okozhatna
- ✿ Minél korábban el kell kezdeni a tesztelési tevékenységeket jól definiált célok mentén!



Hibafürtök megjelenése

- ✿ A defektusok eloszlása nem egyenletes a modulok között, mivel pl.:
 - ✿ komplexitás
 - ✿ gyakran változó kód
 - ✿ fejlesztők tapasztaltsága / tapasztalatlansága
- ✿ **Pareto-elv:** a problémák kb. 80%-a a modulok kb. 20%-ában található
 - ✿ A modulok kis aránya tartalmazza az átadás előtti tesztek során vagy az üzemeltetéskor fellépő meghibásodások nagy részét
 - ✿ Várható hiba gyakoriság alapján koncentráljuk a ráfordításokat



Rovarirtó jelenség

- ✿ Ugyanannak a tesztnak a futtatása nem fog újabb hibákat találni, „immúnissá” válik a rendszer
- ✿ Két jellemző ok:
 1. A változatlan/ellenőrizetlen tesztek elavulhatnak (pl. a specifikáció változása miatt)
 2. Ha a tesztelési módszerekben nincs változás, a fejlesztők “hozzászoknak” a tesztekhez, és az adott hibákat elkerülik, de attól még más hibákat ugyanúgy elkövethetnek
- ✿ A féregirtó jelenség elkerüléséhez tehát a teszteseteket folyamatosan frissíteni, újítani kell, újabb és különböző típusú teszteket kell készíteni a szoftver vagy rendszer különböző részeihez

A tesztelés környezetfüggő

- ❖ Különböző körülmények között különböző tesztelési stratégiákra lehet szükség
 - ❖ Máshogyan kell tesztelni egy információs weblapot mint egy online áruházat, egy légiforgalom-irányító rendszert mint egy hitelkalkulátort
- ❖ A kockázati tényező egy fontos szempont a teszt típusának meghatározásakor
- ❖ Minél nagyobb a várható veszteség, annál többet kell költeni a tesztelésre
- ❖ A szoftver biztonsági teszteléséhez például valószínűleg speciális szakértőre és/vagy eszközre lesz szükség

Csapatszemlélet

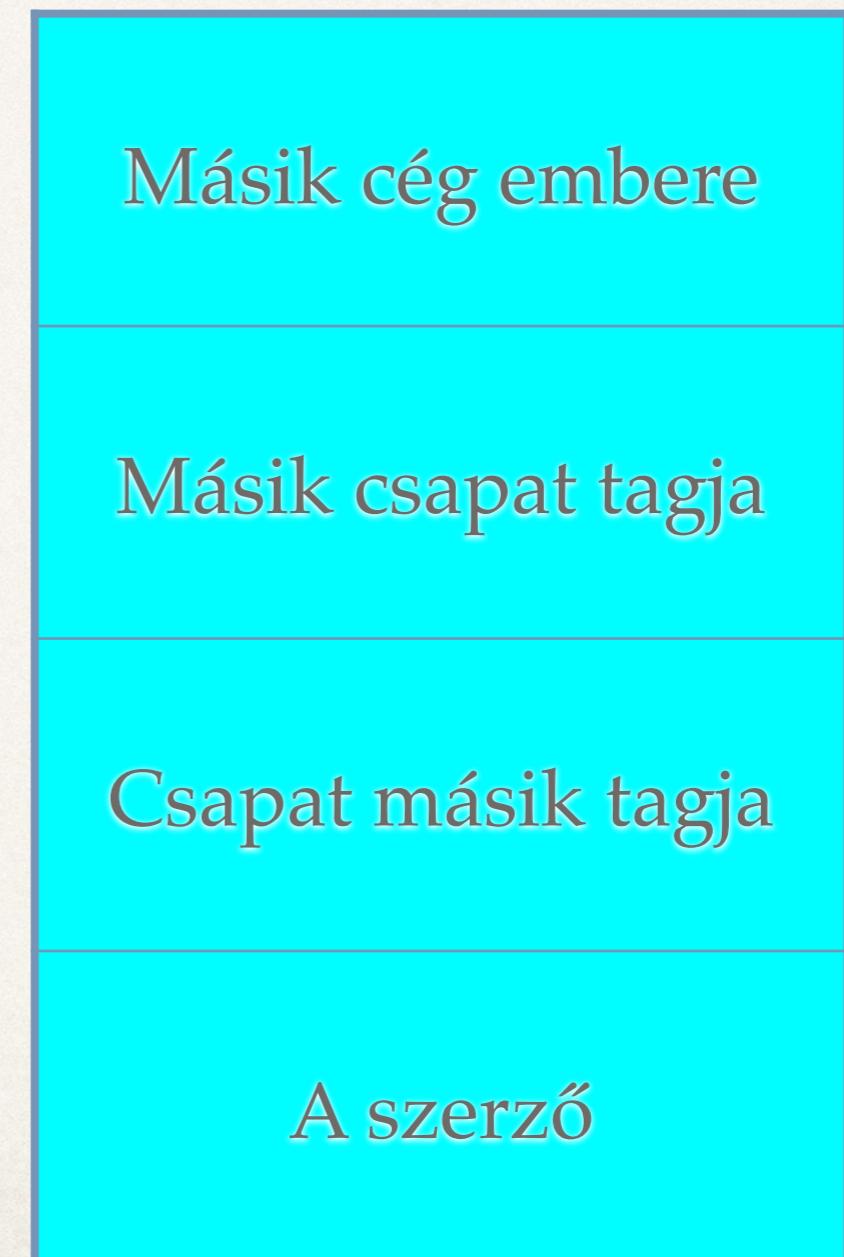
Whole team approach

- ✿ Keresztfunkcionális
 - ✿ “T” tudás modell
 - ✿ minden kompetencia jelen van
- ✿ Önszerveződő
 - ✿ Belső döntéshozatal, nincs “vezető”
- ✿ Csapat szintű felelősség
 - ✿ csapat célok vs. egyéni célok
- ✿ Szoros együttműködés



Független tesztelés

- ❖ A tesztelésbe sokféle ember bevonható
- ❖ Saját kódot tesztelni nem hatékony
 - ❖ A készítő számára a hibák láthatatlanok
 - ❖ A hibakeresés természetesen szükséges
 - ❖ Korai szakaszban vannak előnyei
 - ❖ A függetlenségnek szó szerint ára van



Tesztelési folyamat

A tesztfolyamatot befolyásoló környezeti tényezők:

- ✿ Életciklus-modell
- ✿ Projekt-módszertan
- ✿ Kockázatok
- ✿ Üzleti tevékenység
- ✿ Költségvetés, idő, szerződések, szabályok
- ✿ Szervezeti policy-k és gyakorlatok
- ✿ Belső és külső szabványok

Tesztelési folyamat:
tesztelési tevékenységek, melyek a
fejlesztési folyamatba ágyazódnak

Nyomonkölvethetőség

Traceability

- ❖ A tesztek és bármely munkatermék között definiálható (követelmény, kód, adatforrás, ...)
- ❖ Miben segít?
 - ❖ A változások hatásainak elemzésében
 - ❖ A tesztek auditálhatóvá tételeben
 - ❖ A teszt előrehaladási jelentések jobb megértésében
 - ❖ A tesztek technikai jellemzőinek minden résztvevő számára érthetővé tételeben

Szerepkörök a tesztelésben

1. Teszt menedzser

- Szervezői szerep: vezető, koordinátor (Agilis környezetben speciális)

2. Tesztelő

- Mérnöki, technikai szerep: elemző, végrehajtó

3. Egyéb szerepkörök

- Fejlesztő, üzemeltető, marketing, jog, stb.

További részletek: 5. fejezetben

Tesztelés eredménytermékei

Testware

- ✿ minden tevékenységnek jól definiált eredménytermékei (munkatermékei) kell, hogy legyenek (ez a "tesztver")
- ✿ Azok dokumentálása, tárolása, stb. esetenként változó
- ✿ Sok eredményterméket dedikált eszköz segítségével állítják elő
- ✿ Nyomon követhetőség fontos a tesztbázis és a munkatermék között

Folyamattervezés

- ❖ Mik a céljaink?
- ❖ Mit, hogyan és mikor fogunk tesztelni?
- ❖ Mik a kockázatok?
- ❖ Mik a feltételek (humán, SW, HW, ...)
- ❖ Mikor tekintjük teljesítettnek?
- ❖ A tesztelés befejezéséig folyamatosan zajlik

Munkatermékek:

- ❖ Teszt tervezek, tesztelési szintenként

Nyomonkövethetőség:

- ❖ Mik az egyes „döntések” okai, forrásai?

Teszt-felügyelet és irányítás

- ❖ A kontroll összehangolja a tervezett és a valós lépéseket, ha eltérnénk a tervtől
 - ❖ Az eredmények mérése és analízise
 - ❖ Az elvárt és valós folyamat összehasonlítása, a kilépési feltétel kiértékelése
 - ❖ Korrekciók végrehajtása, döntések, pl. újabb tesztek
- ❖ A tesztelés befejezéséig folyamatosan zajlik

Munkatermékek:

- ❖ Tesztjelentések (köztes, összefoglaló)

Teszt-elemzés

- ❖ Válaszok a „Mit teszteljünk?” kérdésre
- ❖ Fontosabb feladatok a lépés során:
 - ❖ Tesztbázis átnézése és elemzése
 - ❖ Tesztfeltételek meghatározása
 - ❖ Tesztvázlatokhoz feltételek gyűjtése
 - ❖ Nyomonkövethetőség meghatározása
 - ❖ Hibakeresés a tesztbázisban: kétértelműség, kihagyások, inkonziszencia, pontatlanságok, ellentmondások, felesleges részek

Tesztbázis:

- ❖ ami alapján a teszteket megtervezzük, pl.:
- ❖ specifikáció, terv, forráskód, tapasztalati tudás, stb.

Teszt-elemzés

Munkatermékek:

- ✿ Priorizált tesztfeltételek listája
- ✿ Tesztvázlat-kezdemények
- ✿ Tesztbázisban található hibák listája

Nyomonkövethetőség:

- ✿ Az egyes tesztek honnan, mely követelményekből, forrásokból erednek?

(Műszaki) Teszt-tervezés

- ❖ Válaszok a „Hogyan teszteljünk?” kérdésre
- ❖ Magasszintű tesztesetek, teszthalmazok és “tesztverek” meghatározása
- ❖ Fontosabb feladatok a lépés során:
 - ❖ Tesztesetek tervezése és priorizálása
 - ❖ Tesztadatok meghatározása
 - ❖ Teszkörnyezet és infrastruktúra megtervezése

Munkatermékek:

- ❖ Tesztesetek
- ❖ Teszteset-halmazok
- ❖ Tesztadatok

Nyomonkövethetőség:

- ❖ Tesztesetek, tesztfeltételek és tesztbázis közötti kapcsolatok meghatározása

Tesztesetek (előzetes)

Teszteset (Test case) - Előfeltételek, bemenetek, tevékenységek, elvárt eredmények és utófeltételek halmaza, a tesztfeltételek alapján.

1. **Tesztfeltételek** meghatározása
(*mit* fogunk tesztelni?)
2. **Tesztesetek** megtervezése valamely technika segítségével
(*hogyan* fogjuk tesztelni?)
3. **Teszteljárások** (szkriptek) kidolgozása
(implementáció és végrehajtásra való előkészítés)

Teszt megvalósítás

- ❖ Válaszok a „Minden megvan, hogy teszteket futtassunk?” kérdésre

- ❖ Főbb tevékenységek:

- ❖ Teszteljárások fejlesztése és prioritálása
- ❖ Automatizálás, szkriptek írása
- ❖ Automatikus futtatás ütemezése
- ❖ Tesztkészletek létrehozása, prioritálás
- ❖ Tesztkörnyezet kiépítése és beállítása
- ❖ Tesztadatok előkészítése és betöltése
- ❖ Tesztkörnyezet, tesztadatok ellenőrzése

Munkatermékek:

- ❖ Tesztver
- ❖ Teszt eljárások
- ❖ Tesztkörnyezet
- ❖ Szkriptek

Nyomonkövethetőség:

- ❖ Tesztbázis, tesztfeltételek, tesztesetek, teszteljárások, szkriptek közötti kapcsolatok meghatározása

Teszt végrehajtás

- ❖ A tesztek „futtatása”
- ❖ Főbb tevékenységek:
 - ❖ Tesztelemek és tesztek verziójának rögzítése
 - ❖ Tesztek végrehajtása (kézi vagy automatikus)
 - ❖ Elvárt és kapott eredmények összehasonlítása
 - ❖ Rendellenességek elemzése, hibák jelentése
 - ❖ Teszt eredmények naplózása
 - ❖ Tesztek ismétlése

Munkatermékek:

- ❖ Teszt logok
- ❖ Tesztesetek eredményei
- ❖ Hibák

Nyomonkövethetőség:

- ❖ Tesztbázis, tesztfeltételek, tesztesetek, a tesztelt szoftver és a teszteredmények közötti kapcsolatok meghatározása, frissítése

Teszt lezárás

- ❖ Projekt mérföldköveknél (pl. release) hajtjuk végre

- ❖ Főbb tevékenységek:

- ❖ Hibák állapotának ellenőrzése
- ❖ Összefoglaló tesztjelentés készítése
- ❖ Tesztkörnyezet, adatok, infrastruktúra archiválása

Munkatermékek:

- ❖ Összefoglaló tesztjelentés
- ❖ Változás-kérés
- ❖ Mentések
- ❖ Véglegesített tesztkörnyezet

- ❖ Tesztver átadása a karbantartó csapatnak
- ❖ Tesztfolyamat elemzése és fejlesztése

Tesztelestípusai (előzetes)

I. Funkcionális / nem-funkcionális teszt

2. Egyszeri / megismételt

3. Statikus / dinamikus

4. Fehérdoboz / feketedoboz teszt

- Terhelés-teszt
- Stressz-teszt
- Használhatósági teszt
- Biztonsági teszt
- Alfa / Béta teszt
- "Smoke" teszt
- Robosztussági, random teszt
- "Csimpánz" teszt
- Kombinatorikai módszerek
- Konfigurációs adatok tesztelése
- Teszt adatok tesztelése, "tesztek tesztelése", stb.

FEJLESZTÉS ÉS TESZTELÉS

- A tesztelés nem választható szét a fejlesztéstől
- Ritkán van teljesen különálló “fejlesztés” és “tesztelés” projekt
- A tesztelés mindenkor alkalmazkodik a választott fejlesztési modellhez
- Gyakran teljesen egybeolvad, pl. egységes tesztelés, teszt-alapú fejlesztés
- Emlékeztető: debugging szerepe

ÉLETCIKLUS MODELLEK

- A szoftverek fejlesztésének lépésein írják le

 Szekvenciális (lineáris) modellek

- Vízesés-modell

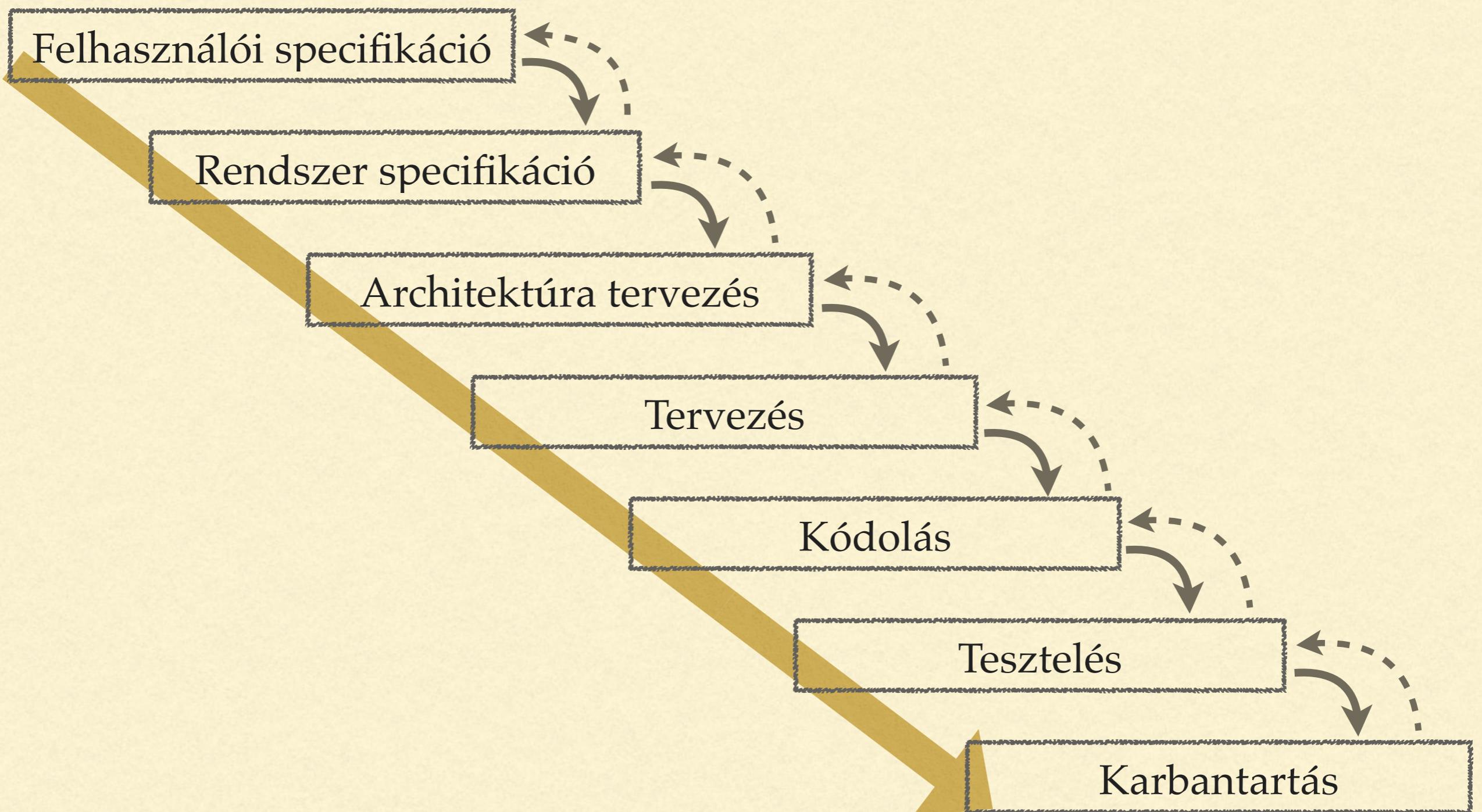
- V-modell, W modell

 Iteratív, inkrementális modellek

- Hagyományos, pl. RUP, Spirál

- Agilis, pl. Scrum, Kanban, XP

VÍZESÉS MODELL



VÍZESÉS MODELL

- **Minden munkatermék elkészül a következő lépés előtt**
- **Tesztelés a folyamat legvégén van csak**

■ Problémái:

- A szoftver nem megfelelő részeit nem lehet csak úgy “kihagyni”, a teljes szoftvert nem lehet csak úgy “eldobni” és minden előlről kezdeni
- Visszalépés nagyon költséges
- Ritkán ismert teljes egészében a specifikáció
- Kritikus rendszereknél alkalmazható: ha minden körülmény adott, nagyon robosztus rendszerek fejleszthetők így

- Szegecsek gyártása repülőgéptörzshöz:
A gyártási folyamat végén a hibás szegecsek egyenként kiszedhetőek
- A pilótafülke digitális műszereit, monitorait vezérlő program:
Felengedhető-e egy gép magasságmérő, vagy futómű visszajelzés nélkül?
Újra kell-e kezdeni minden a nulláról, ha csak 1-2 visszajelző működik hibásan?

VÍZESÉS MODELL KITERJESZTÉSE

Egy jó modellben a teljes folyamat alatt figyelni kell a minőségre

- minden fázis után ellenőrizzük az előállított munkatermékét
- Akkor léphetünk tovább, ha az előző eredménye rendben van
- Hibás munkatermék esetén az adott munkafázist meg kell ismételni

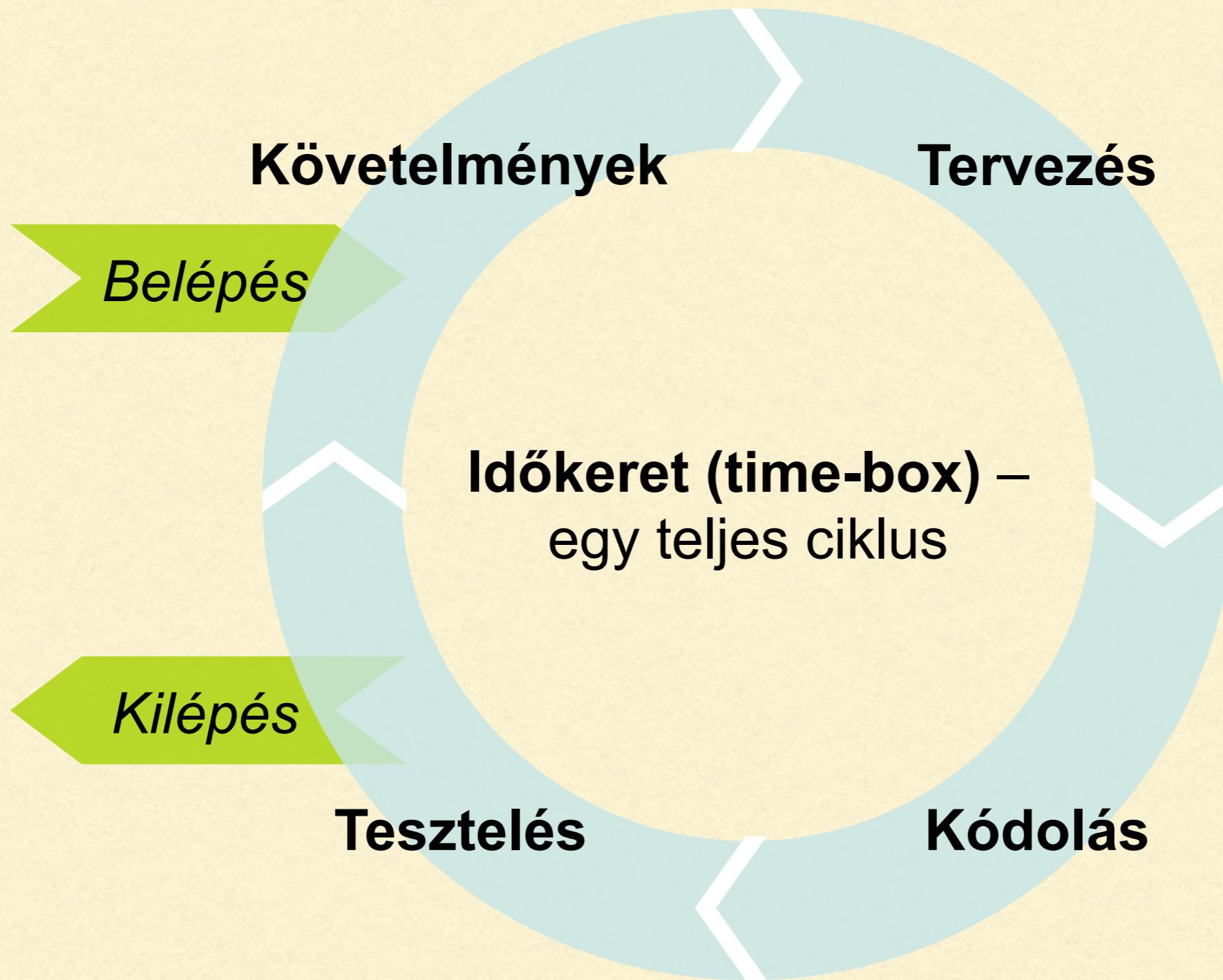
I. Átvizsgálás (review) minden munkafázis után (korai tesztelés!)

2. Teszt tervezés azonnal elkezdhető az adott fázis végén, minden munkatermék külön-külön tesztelhető (korai tesztelés!)



V-modell

ITERATÍV MODELLEK



ITERATÍV MODELLEK

- *Iteratív:*

a fejlesztési tevékenységek kisebb ciklusokban követik egymást

- *Inkrementális:*

a követelményeket nem szükséges az elején rögzíteni

az egyes iterációk egyre jobban finomítják, részletezik, kiegészítik

- A fejlesztés végét általában idő és költségkorlátok jelentik

ITERATÍV MODELLEK ELŐNYEI

- Egy-egy iteráció változó méretű lehet: napokban, hetekben, hónapokban, években is mérhető
- Nagy előnye a flexibilitás
- Kulcsszerep jut a *felhasználónak*, így egyszerre lehet verifikációt és validációt is megvalósítani
- Kisebb kockázat, hamarabb használható “inkremensek”, prototípusok, részlegesen kész termékek



ITERATÍV MODELLEK HÁTRÁNYAI

- A formális dokumentáció hiánya megnehezíti a tesztelést
- A változtatások dokumentálatlansága
- Nyomonkövethetőség hiánya követelmények és végtermék funkciói között
- Változtatások nagy mennyisége miatt kevésbé robosztus megvalósítás, több “műszaki tartozás”
- Megnövekedett igény a kommunikációra a fejlesztő, megrendelő és tesztelő között



*Modern módszertanok képesek a fentieket kiküszöbölni, pl.
TDD, regressziós tesztelés*

ITERATÍV MODELL PÉLDÁK

■ *RUP (Rational Unified Process)*

- Építőkövek (ki, mit, hogyan, mikor)
- Fázisok saját iterációval: kezdés, kidolgozás, kivitelezés, átmenet
- Hosszabb időkeretek
- Modellezési javaslatok

■ *Spirál*

- Kísérleti prototípusok, melyek a későbbi iterációban továbbfejleszthetők, vagy akár el is dobhatók

■ *eXtreme Programming*

- Első rövid ciklusú iteratív modell

ITERATÍV MODELL PÉLDÁK

- *Scrum (egy agilis modell)*
 - Pár hetes ciklusok (sprintek)
 - Napi maximum negyedórás megbeszélések
 - Önszerveződő fejlesztőcsapatok
 - Páros programozás
- *Kanban (egy agilis modell)*
 - Rögzített hosszúságú ciklusok, opcionálisak
 - Egyes fázisokban lévő tevékenységek számának korlátozása (Kanban board)
- *Lean (agilitás mintája)*
 - Projekt menedzsment

AGILIS TESZTELÉS

- Agile Manifesto: filozófiai elvek és értékek meghatározása
- Agilis tesztelési technikák:
 - Csapat: üzlet + fejlesztő + tesztelő
 - Test Driven Development
 - Acceptance Test Driven Development
 - Behaviour Driven Development
- Test Pyramid
- Testing Quadrants
- Planning Poker
- Agresszív regressziós teszt és folyamatos integráció
- Tesztautomatizálás
- Felderítő tesztelés, stb.

ÉLETCIKLUS KONTEXTUSA

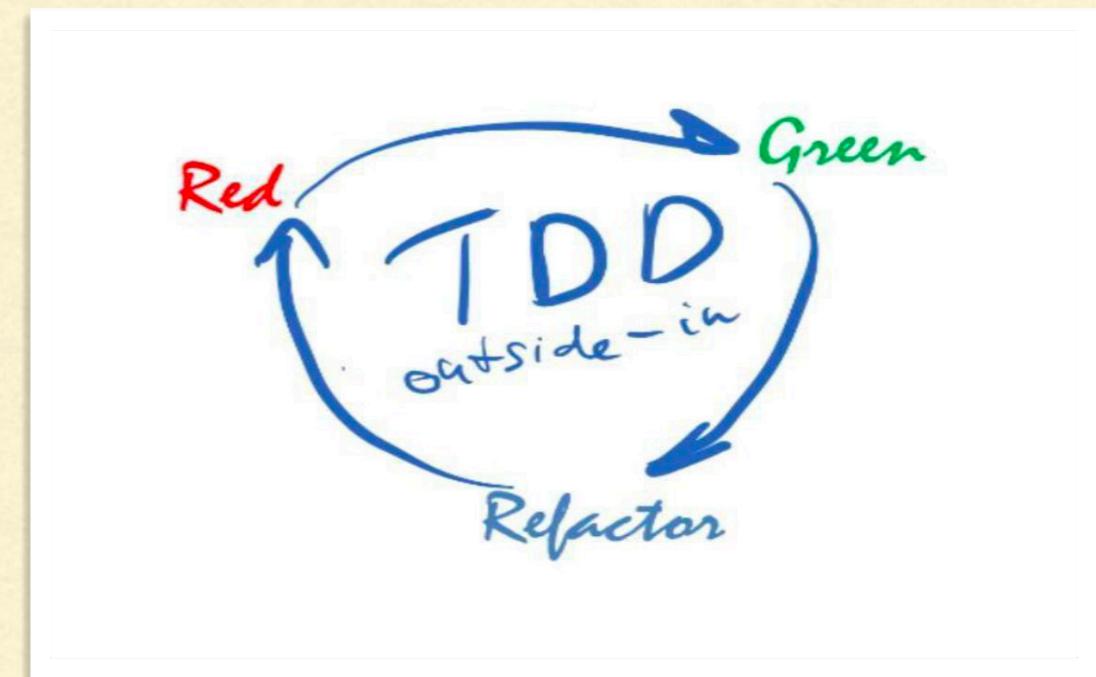
- Adott körülményekhez mérten kell a fejlesztési életciklus modellt megválasztani
 - A tesztelési megközelítés is ezt kell, hogy kövesse,
Id. kockázat alapú tesztelés
- Ugyanazon szervezeten belül előfordulhat többféle modell, pl.
 - V-modell a backend fejlesztéséhez és teszteléséhez
 - Egy agilis modell a UI frontend fejlesztéséhez és teszteléséhez
- Egyes területek különösen vegyesek, pl. IoT fejlesztés

JÓ TESZTELÉS TULAJDONSÁGAI

	SZEKVENCIÁLIS	ITERATÍV
Korai teszt tervezés	Specifikációs dokumentumok alapján a fázisok végén elkezdhető	Teszt alapú fejlesztés (TDD), egyéb xDD módszer
Munkatermékek tesztelése	Minden szinthez tartozik tesztelési tevékenység	Minden iteráció végén van tesztelés
Tesztelők bevonása	A dokumentumok átvizsgálásban részt vesznek	Különböző résztvevők tesztelhetnek, a felhasználó is
Tesztelés szintjei	V-modell szerint: felsőbb szintek fontossága	Teszt piramis: alsó szinteken nagy hangsúly, teszt autom.

TESZTELÉS-VEZÉRELT FEJLESZTÉS

- Iteratív fejlesztésnél (főleg Agilis) alkalmazható módszerek
 - A tesztek a fejlesztés “irányítói” (xDD)
 - Korai tesztelés és a “shift-left” elvek támogatása
 - A fejlesztés végén a tesztek megmaradnak
- *Test-Driven Development (TDD)*
 - Egység teszt szintű technika
 - A kódolást a tesztek írásával kezdjük
 - A kódot addig módosítjuk, amíg a tesztek át nem mennek
 - A végén refaktorálunk



TESZTELÉS-VEZÉRELT FEJLESZTÉS

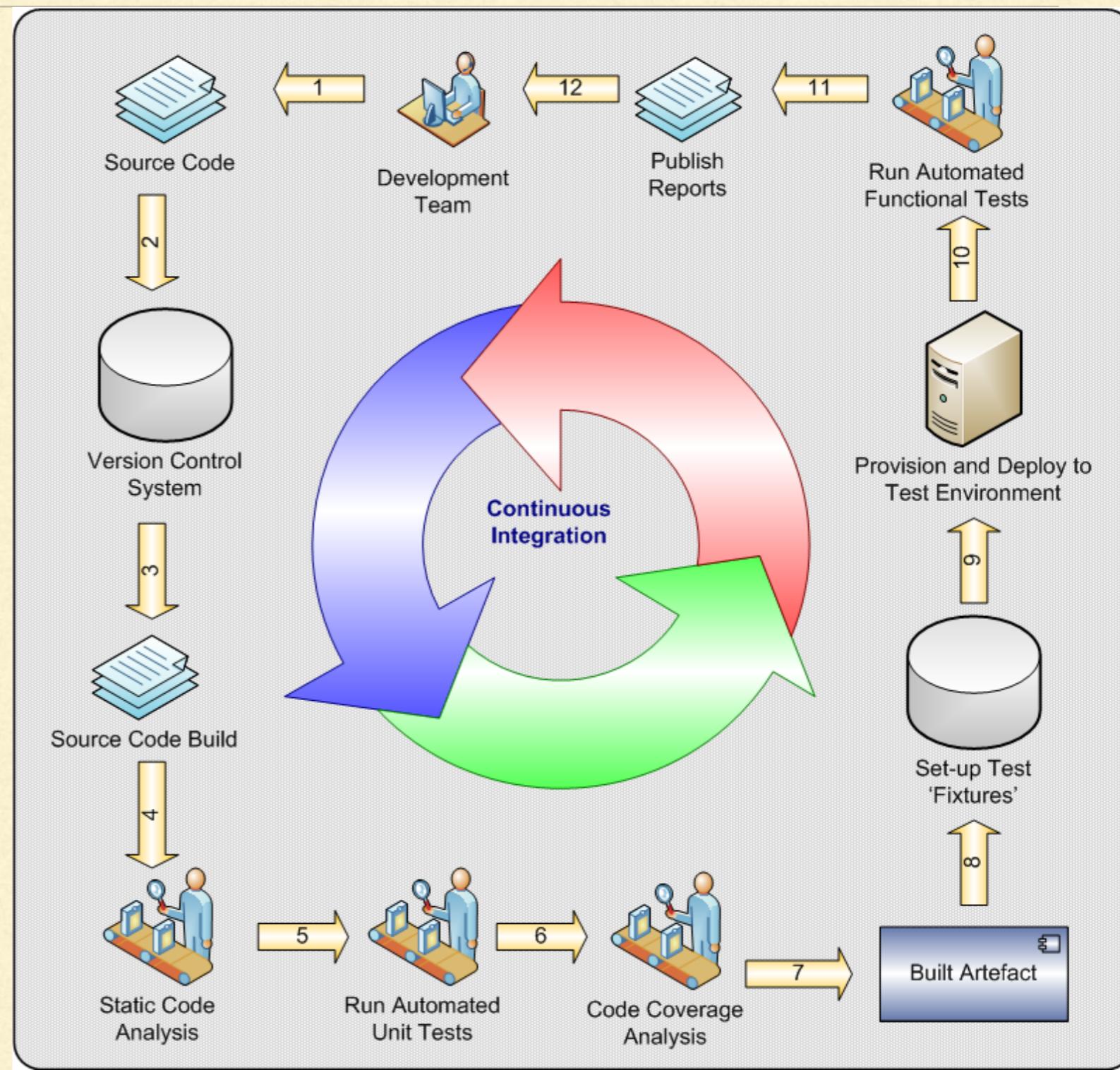
- *Acceptance Test-Driven Development (ATDD)*
 - Elfogadási kritériumokhoz írjuk a teszteket a tervezéskor
 - A funkcionális fejlesztése ezután kezdődik
- *Behavior-Driven Development (BDD)*
 - A viselkedés természetes nyelven van megfogalmazva (“given/when/then” formátum)
 - Végrehajtható tesztek automatikusan generálódnak

TESZTELÉS-VEZÉRELT FEJLESZTÉS

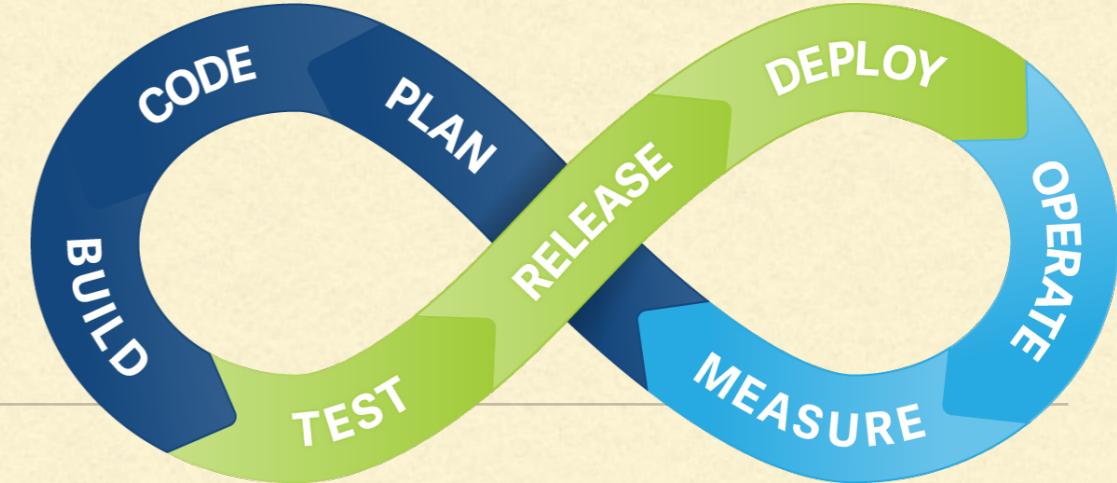
	TDD	ATDD	BDD
Definíció	Fejlesztési módszertan amely egyes funkciók megvalósítására fókuszál.	Fejlesztési módszertan amely a követleményeknek való megfelelésre fókuszál.	Fejlesztési módszertan amely a rendszer viselkedésére fókuszál.
Részttvevők	Fejlesztő	Fejlesztő, Ügyfél, Tesztelő	Fejlesztő, Ügyfél, Tesztelő
Használt nyelv	Programozási nyelvek	Hétköznapi nyelv (Angol)	Hétköznapi nyelv (Angol) Gherkin
Cél	Unit tesztek	Elfogadási tesztek készítése	Követelmények megértése
Eszközök	*Unit (J/N/PHP/...), TestNG, Jasmine, Spec Flow, RSpec	TestNG, FitNesse, EasyB, Spectacular, Concordion	Gherkin, Dave, Cucumber, JBehave, Spec Flow

CI/CD ÉS DEVOPS

- *Continuous Integration (CI)*
- Rendszerépítés kibővítése
- Nagyfokú integráció
 - Verziókövetővel
 - Fejlesztőkörnyezettel
 - Egyéb eszközökkel
- Időzített tevékenységek (“pipeline”)
- Pl. Jenkins, gitlab

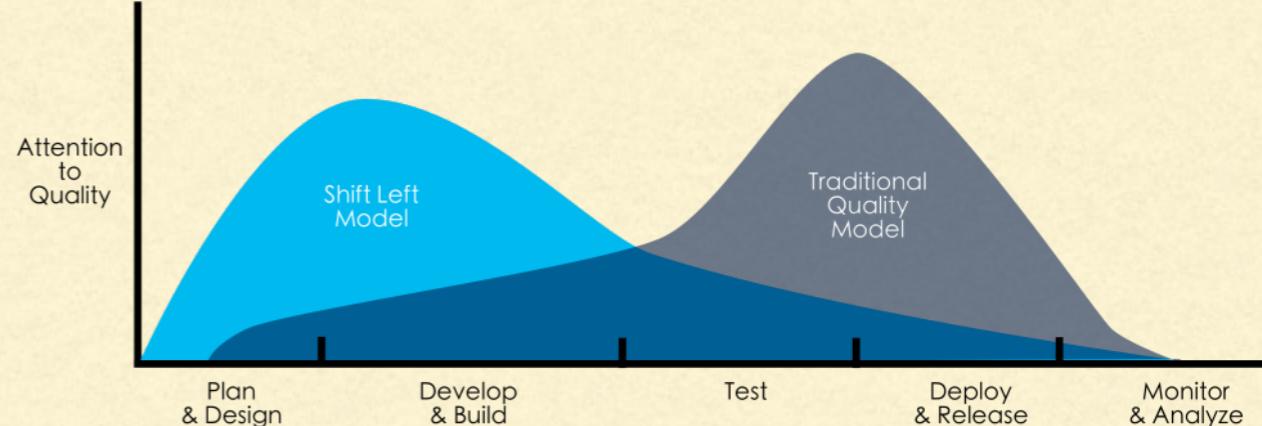


CI/CD ÉS DEVOPS



- *DevOps: Development-Operations*
 - Continuous Delivery / Continuous Deployment
- Fejlesztés és üzemeltetés integrációja
 - Fejlesztés, CI, kiadás, konfiguráció, üzemeltetés monitorozás
- Tesztelési szempontból:
 - Gyors visszajelzés a minőségről, korai tesztelés és “shift left” támogatása
 - Rgresszió minimalizálása
 - Nagyfokú automatizálás, virtualizáció, “dockerizáció” (technikai kihívások is!)

“SHIFT LEFT” ELV

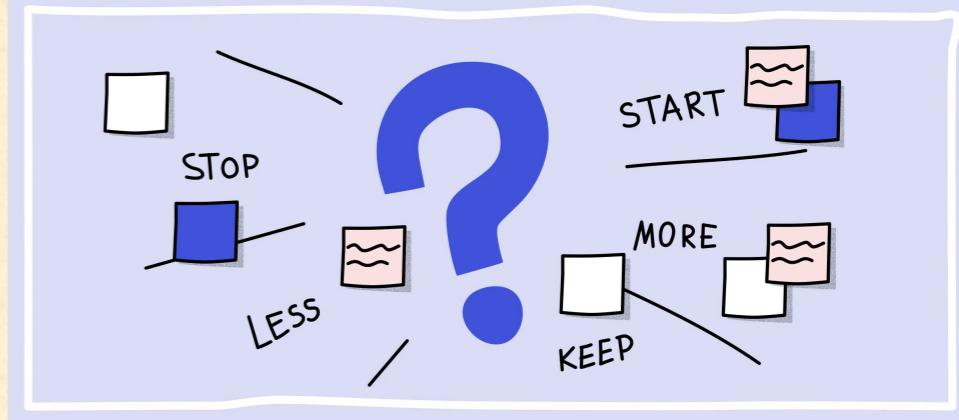


- Korai tesztelés: minél korábban teszteljünk (“balra” toljuk a tevékenységet)
 - “Fázis tartalmazás” elv: amikor keletkezik a hiba, akkor is javítsuk ki
- Kihívást jelenthet, mert komoly elköteleződés kell (“befektetés”)

Példák:

- *Specifikáció felülvizsgálata tesztelési szempontból*
- *xDD*
- *CI/CD/DevOps*
- *Statikus analízis dinamikus tesztelés előtt*
- *Minél több nem-funkcionális tesztet hozzunk korábbra*

RETROSPEKTÍVEK



- Projekt vagy iteráció végén tartott megbeszélések
- Különböző típusúak, agilis módszertanok definiálják
- A teljes csapat (nem csak a tesztelők) megvitatják:
 - Mi volt jó? Mi nem működött? Min lehet javítani?
- Javaslatokat kell megfogalmazni a fejlődéshez
 - Sokszor a teszt jelentés része

Előnyök:

- Hatékonyabb tesztelés
- Magasabb minőség
- Jobb csapatszellem
- Jobb tesztbázis
- Jobb kommunikáció fejlesztés / menedzsment / tesztelés között

EGYSÉGTESZTELÉS

Unit testing, component testing

- Célja a programkód egységeinek önmagukban, más egységektől függetlenül történő ellenőrzése
- Futtatás, inputok megadása és outputok ellenőrzése helyett gyakran maga a forráskód a teszt alapja, és ez nem könnyű (meghajtók, csonkok, mock-olás, virtualizáció, stb.)
- Általában maga a fejlesztő végzi
- A javított defektusoknak nem sok nyoma marad
- “Először teszteljünk” megközelítés (TDD)
- Iteratív fejlesztés tesztesetek kidolgozásával

Tipikus tesztbázis:

- ✿ Egységekre vonatkozó követelmények
- ✿ Részletes tervezet (design)
- ✿ Kód

Tipikus teszt objektumok:

- ✿ Komponensek, egységek
- ✿ Programok, kód és adatstruktúrák
- ✿ Osztályok
- ✿ Adatbázis modulok

INTEGRÁCIÓS TESZTELÉS

Integration testing

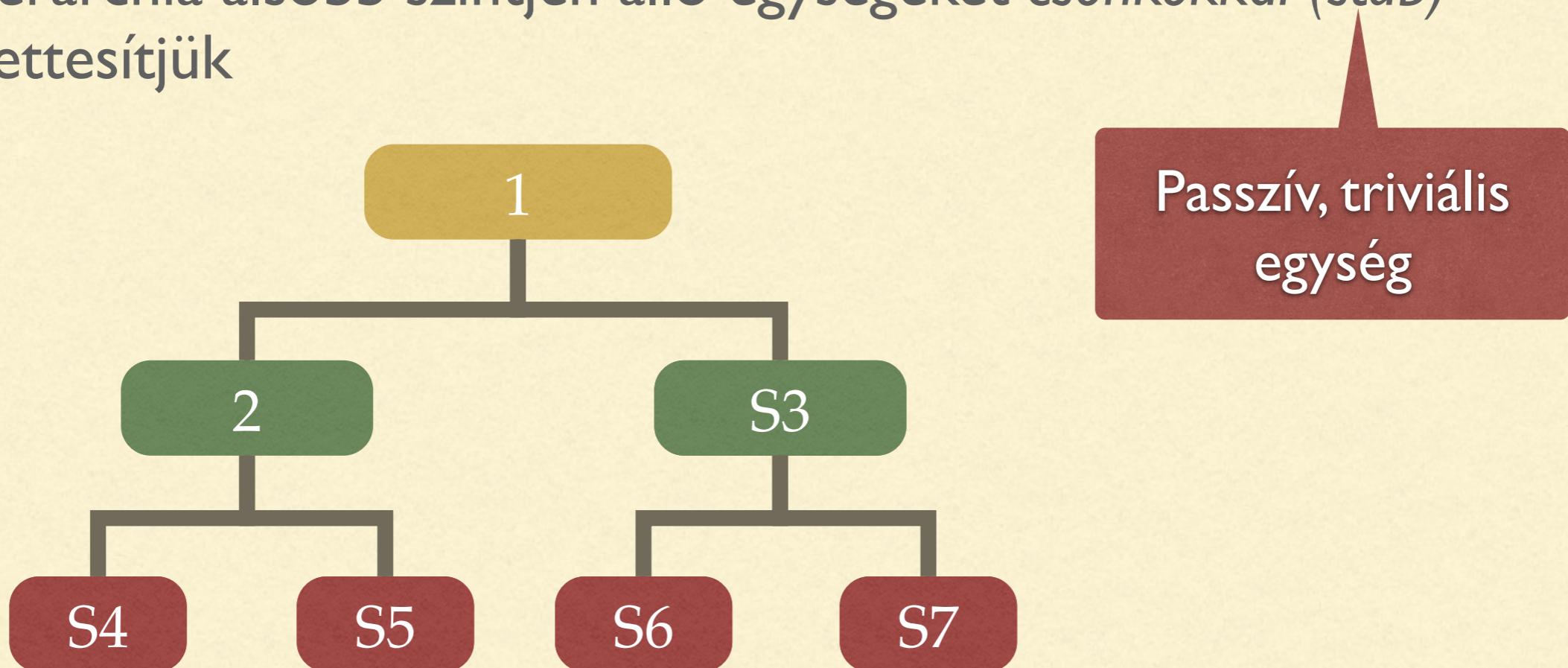
- Célja az egységek közötti, az interfészeken és az együttműködésben található defektusok kiszűrése
 - Operációs rendszer, fájlrendszer, hardver egységek, komponensek, alrendszerek közötti interfészek és kommunikáció
- Bonyolultabb, többszintű hierarchia esetén kell egy stratégia:
 - Nagy bumm (Big-bang) integráció
 - Felülről lefelé haladó (Top-down) integráció
 - Alulról felfelé haladó (Bottom-up) integráció

NAGY BUMM INTEGRÁCIÓ

- A teljes szoftvert összeállítjuk, minden részét integráljuk, és így végezzük el a teszteket
- Gyenge választás:
 - Csak az összes egység elkészülte után lehet elvégezni
 - Nehéz egy konkrét egységkapcsolat tesztelése
 - Nehéz a hibákat lokalizálni
 - Helyette valamely inkrementális megoldást kell választani

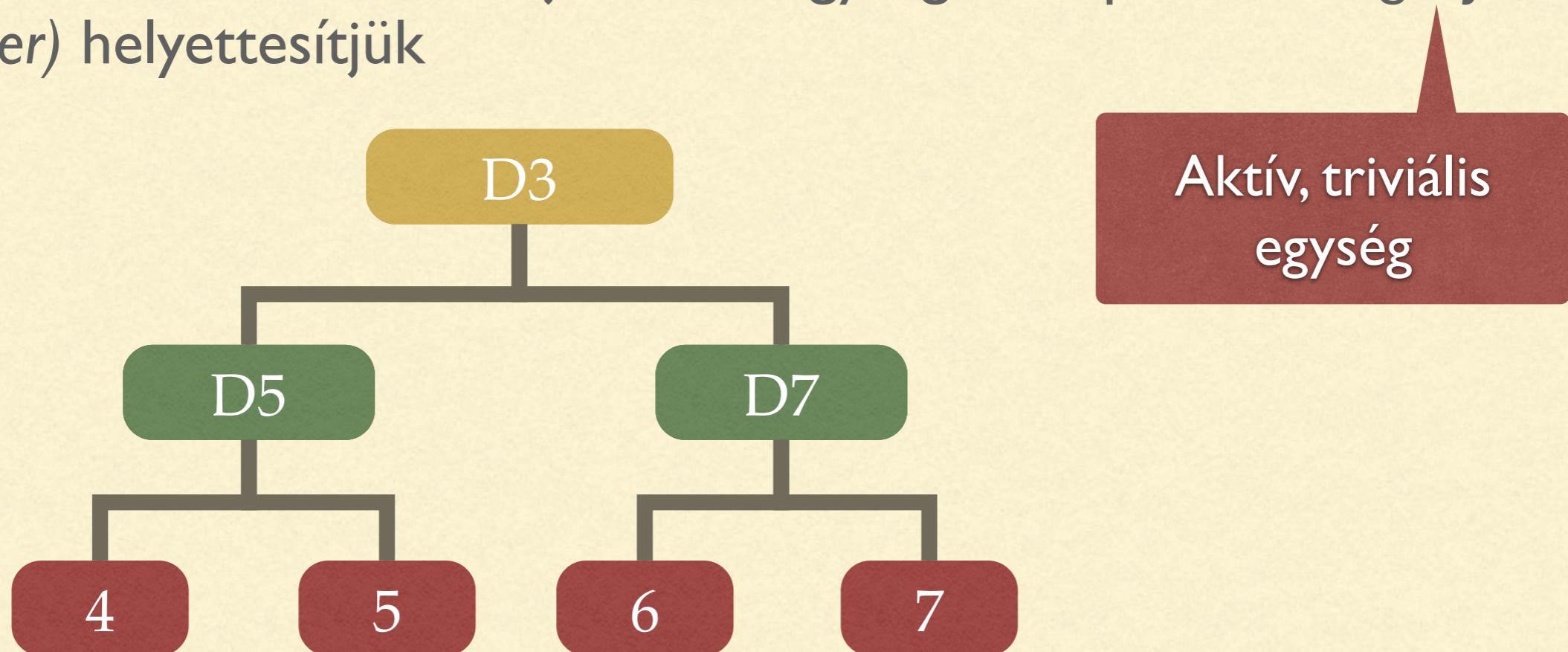
FELÜLRŐL LEFELÉ INTEGRÁCIÓ

- Az egységek hierarchiája szerint felülről kezdjük el ellenőrizni az egységek kapcsolatait
- A hierarchia alsóbb szintjén álló egységeket *csonkokkal (stub)* helyettesítjük



ALULRÓL FELFELÉ INTEGRÁCIÓ

- Az egységek hierarchiája szerint alulról felfelé kezdjük el ellenőrizni az egységeket
- A hierarchia felsőbb szintjein lévő egységeket speciális *meghajtókkal* (*driver*) helyettesítjük



INTEGRÁCIÓS SZINTEK

- Különböző szinteken és különböző méretű teszt objektumokon
- Rendszer architektúrája és külső rendszerek, szolgáltatások határozzák meg
- Integráció méretétől függően a hibák elszigetelése egyre nehezebb
- Nem-funkcionális követelmények (pl. teljesítmény) tesztelése integrációs és funkcionális teszteléskor is lehetséges

Komponens integrációs teszt

- Egy adott rendszer komponenseinek integrációja
- A rendszer egységeinek tesztelése (unit test) után következik (fejlesztők végzik)

Rendszer integrációs teszt

- Több rendszer együttműködésének ellenőrzése
- A rendszerek egyedi tesztje (system test) után következik (tesztelők végzik)

INTEGRÁCIÓS TESZT ELEMEI

Tipikus tesztbázis:

- ❖ Szoftver-, rendszertervezek
- ❖ Architektúra
- ❖ Munkafolyamatok (workflow)
- ❖ Használati esetek

Tipikus teszt objektumok:

- ❖ Alrendszerök, adatbázisok implementációja
- ❖ Infrastruktúra
- ❖ Interfészök, API-k
- ❖ Rendszer konfiguráció, konfigurációs adatok
- ❖ Mikroszolgáltatások

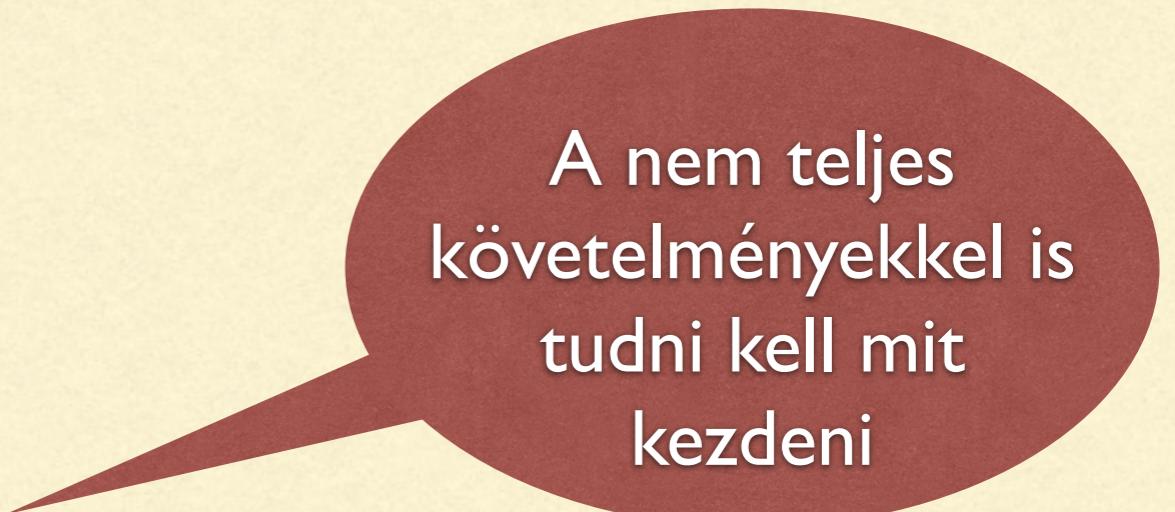
RENSZERTESZT

System testing

- Az eddigi tesztek tesztesetei nagyrészt alkalmatlanok a teljes rendszert átívelő kölcsönhatásokból, vagy a környezetből adódó hibák felderítésére
- Általában a fejlesztőktől független csapat végzi egy valós, reprezentatív környezetben
- A funkcionális specifikáción alapszik, amelyben le vannak írva a:
 - funkcionális követelmények,
 - nem-funkcionális követelmények,
 - adatok minőségi jellemzői.

RENDSZERTESZT TULAJDONSÁGAI

- A szükséges/ajánlott tesztelés mennyisége függ az előző fázisokban elvégzett tesztelés mennyiségétől, illetve a követelmények verifikációjától is
- A tesztelési környezet a lehető legjobban a végső környezetet kell, hogy tükrözze
- Rendszerteszt alapjául szolgálhat:
 - Kockázat és követelmény specifikáció
 - Üzleti folyamat leírások
 - Használati esetek
 - Egyéb magas szintű rendszer leírások és modellek



A nem teljes követelményekkel is tudni kell mit kezdeni

FUNKCIIONÁLIS/NEMFUNKCIIONÁLIS

Funkcionális követelmények/ tesztek

- A rendszer vagy rendszerek által megalósítandó funkciók részletes leírása
- “Mit fog csinálni a rendszer?”
- Általában *black-box*

Nemfunkcionális követelmények/ tesztek

- Konkrét funkciókhöz nem köthető, általánosan érvényes jellemzők, ld. ISO 25010
- “Hogyan fogja csinálni a rendszer?”
- Általában *black-box*

telepíthetőség, interoperabilitás, karbantarthatóság,
teljesítmény, terhelés-kezelés, stressz-kezelés, hordozhatóság,
helyreállás, megbízhatóság, használhatóság, ...

RENDSZERTESZT ELEMEI

Tipikus tesztbázis:

- ❖ Szoftver-, rendszerspecifikációk
- ❖ Használati esetek, user story-k
- ❖ Funkcionális leírás
- ❖ Kockázat elemzési adatok

Tipikus teszt objektumok:

- ❖ Rendszer
- ❖ Kézikönyvek (felhasználói, kezelői)
- ❖ Rendszer konfiguráció
- ❖ Konfigurációs adatok

ÁTVÉTELI TESZT

Acceptance testing

- Célja demonstrálni, hogy a rendszer az elvárt módon fog működni
- Mivel a követelmény-specifikáción alapszik, fontos, hogy az dokumentált és hibamentes legyen
- Ellenőrzi a rendszer készenléti állapotát az üzembe helyezéshez
- Az összes többi teszttől független
- Leggyakrabban a megrendelő végzi (*validáció*)

ÁTVÉTELI TESZT FAJTÁI

- Felhasználói elfogadási teszt (UAT)
 - A felhasználó teszteli a rendszert, hogy megfelel-e az üzleti igényeinek
- Üzemeltetési elfogadási teszt
 - A rendszer használati és karbantartási eljárásait is teszteli, pl: back-up lehetőségek, katasztrófavezetés, végfelhasználók képzése, karbantartási eljárások, adat terhelés és migrációs feladatok, biztonsági eljárások, stb.
- Szerződéses és szabályozott elfogadási teszt
 - Szerződésben vagy törvényben foglalt feltételeknek való megfelelés ellenőrzése, pl adatvédelmi törvény, bankbiztonsági rendelkezések
- Alfa és béta teszt
 - "Dobozos" termékeknél
 - A fejlesztőnél illetve a felhasználóknál végzett tesztek

ÁTVÉTELI TESZT ELEMEI

Tipikus tesztbázis:

- ❖ Felhasználó követelmények
- ❖ Rendszer követelmények
- ❖ Használati esetek
- ❖ Üzleti folyamat leírások
- ❖ Kockázat elemzési adatok

Tipikus teszt objektumok:

- ❖ Üzleti folyamatok az integrált rendszeren
- ❖ Üzemeltetési és karbantartási folyamatok
- ❖ Felhasználói eljárások
- ❖ Adatlapok, riportok
- ❖ Konfigurációs adatok

KÜLÖNBÖZŐ OSZTÁLYOZÁSOK

I. Funkcionális / nem-funkcionális teszt (Id. Rendszertesztelés)

2. Egyszeri / megismételt (Id. Változásokhoz kötött teszt)

3. Statikus / dinamikus (Id. Statikus módszerek, Teszt tervezés)

4. Fehérdoboz / feketedoboz teszt:

Fehérdoboz, white box, struktúra alapú teszt

- A rendszer felépítésére koncentrál
- Ez lehet maga a kód, vagy a rendszer architekturális leírása
- Megvalósítás alapján mérhető a tesztelés “mennyisége”

Feketedoboz, black box, specifikáció alapú teszt

- A rendszer működésére koncentrál
- Ezt minden valami külső információ határoz meg
- Specifikáció alapján mérhető a tesztelés “mennyisége”

VÁLTOZÁS TESZTELÉS

A szoftvert érintő változtatások esetén kétféle okból szükséges a tesztek újrafuttatása:

I. Ellenőrző teszt

- Confirmation testing, re-testing
- Az adott változtatás kijavította-e azt a hibát, amely miatt változtatni kellett a szoftvert?
- Csak az adott teszt futtatása

2. Rgressziós teszt (regression test)

- A (kódban vagy környezetben történt) változások okoztak-e olyan hibákat, amelyek a változtatás előtt nem voltak jelen a rendszerben?
- Nagyszámú teszteset újrafuttatása (szelekció?)
- Automatizálásra kiválóan alkalmas

KARBANTARTÁSI TESZTELÉS

Maintenance testing

- A legtöbb rendszer előbb-utóbb eljut odáig, hogy éles környezetben működjön, akár hosszú ideig
- A rendszer változtatására ezalatt is szükség lehet: újabb fejlesztésekre, adaptációkra, javításokra, stb.
- Egy működő rendszeren végzett tesztelést nevezünk **karbantartási tesztnak**
- Ellenőrző és regressziós tesztek

KARBANTARTÁS FAJTÁI

- *Korrektív*: Hibák, hiányosságok javítása
- *Adaptív*: Megváltozott környezetbe való illesztés
- *Perfektív*: A rendszer funkcionálitásainak módosítása, bővítése
- *Preventív*: Jövőbeli problémák minimalizálása

HATÁSANALÍZIS

- Egy teljes teszt ebben a fázisban megvalósíthatatlan vagy túlságosan költséges lenne
- A rendszer elemei közötti összefüggések ismerete sokat segíthet ezen a problémán
 - Hatásanalízis (Change impact analysis)
 - Bonyolult lehet a rendszer mérete és/vagy a dokumentáció hiánya/elavultsága miatt
 - Nyomonkövethetőség segít, de elemzési módszerek is léteznek
- Szelektív regressziós tesztelés valósítható meg így (**hatékonyság vs. kockázat!**)

Mi a statikus tesztelés?



TESZTELÉS KÓD VÉGREHAJTÁSA NÉLKÜL

- ❖ Nem csak kódon, hanem bármilyen dokumentumon végrehajtható, módszeres technikák
- ❖ Statikus teszteléssel egy hiba típus *minden előfordulása* megtalálható
- ❖ Statikus tesztelés átalában:
felülvizsgálat (review)
- ❖ Eszközzel támogatott statikus tesztelés:
statikus elemzés (általában forráskód)

Mire alkalmazható?

- ❖ Bármilyen írott anyag felülvizsgálható, pl.
 - ❖ Követelmény specifikáció, user story, ...
 - ❖ Rendszertervezek (architektúra, design)
 - ❖ Forráskód
 - ❖ Tesztelési terv, tesztesetek
 - ❖ Munkatervezek, szerződések, ...
 - ❖ Felhasználói kézikönyvek, weblapok, adatok, stb.

Érdekelt felek visszajelzései

Stakeholder feedback

- ❖ Korai és gyakori visszajelzés **minden** érdekelt fél részéről és számára!
 - ❖ Agilis környezetben elengedhetetlen
- ❖ Problémák időben történő kommunikálása
- ❖ Ha nincs, akkor a célok és elképzelések nem lesznek megvalósítva
 - ❖ drága átdolgozás, késés, felelősségre vonások, vagy a **projekt kudarca**
- ❖ Félreértések elkerülése, követelmények megváltozásának követése
- ❖ Fejlesztők jobban megértik a célokat, és a legnagyobb értéket tudják szállítani
 - ❖ kockázatok korai kezelése

Statikus és dinamikus tesztelés

- ❖ Különbség: defektus ill. meghibásodás a célkeresztben
- ❖ Célok hasonlóak (hibák megtalálása, minőség felmérése), de
- ❖ Statikus teszteléssel tipikusan másfajta hibák találhatók meg:
 - *a standarduktól, szabályzatuktól, törvényektől való eltérések,*
 - *biztonsági sérülékenységek,*
 - *követelmények problémái (többértelműség, hiányzó részek),*
 - *tervezek hibái (architektúra, interfészek),*
 - *kódolási problémák (nem inicializált változók, halott kód, stb.),*
 - *rossz belső minőség (karbantarthatóság),*
 - *eredménytermék nem felel meg az előző fázis kimenetével,*
 - *stb.*

Statikus és dinamikus tesztelés

Statikus tesztelés nem jó:

- ✿ Funkciók, üzleti logika tesztelésére
- ✿ Performancia tesztelésre
- ✿ Egyéb nemfunkcionális jellemzők (pl. használhatóság) tesztelésére
- ✿ Rendszer, illetve átvételi tesztelésre

Dinamikus tesztelés nem jó:

- ✿ Triviális kódolási problémák kiszűrésére
- ✿ Dokumentumok hibáinak kiszűrésére
- ✿ Belső minőség (pl. karbantarthatóság) tesztelésére
- ✿ Nehezen tesztelhető, kivételes esetek tesztelésére

Felülvizsgálatok folyamata

Sokféle lehet az alapján, hogy mennyire formálisan végezzük (szabályozottság és az elkészítendő felülvizsgálati dokumentáció határozza meg)

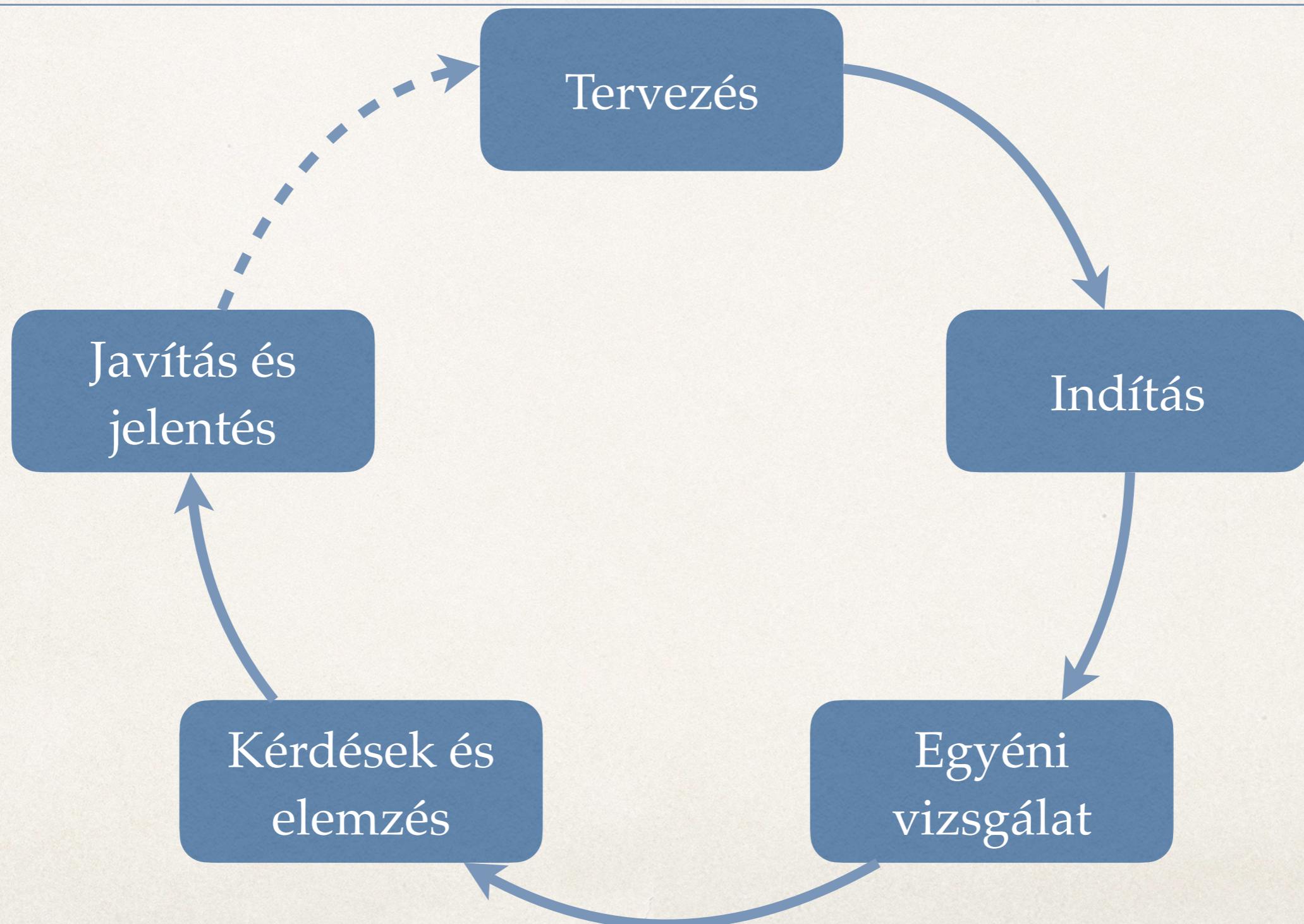
A formalitási szint kiválasztási szempontjai:

- ✿ A fejlesztés előrehaladottsága
- ✿ Törvényi és szabályozási követelmények
- ✿ Auditálási nyom (audit trail) szükségessége

Lehetséges célok:

- ✿ Defektusok felfedezése
- ✿ Munkatermékek megértése
- ✿ Vita generálása
- ✿ Konszenzusos döntés

Felülvizsgálatok folyamata



1. Tervezés

- ❖ Hatókör (scope) meghatározása
- ❖ Ráfordítás és időkeret becslése
- ❖ Megfelelő emberek kiválasztása (elsők között a felülvizsgálat vezetőjét!)
- ❖ Felülvizsgálat jellemzőinek meghatározása: szerepek kiosztása, tevékenységek, ellenőrzőlisták, stb.
- ❖ Be- és kilépési feltétel meghatározása
- ❖ Belépési feltétel ellenőrzése

2. Indítás

- ✿ Anyagok kiosztása
- ✿ Munkatermékek, segédletek
- ✿ Folyamat ismertetése
- ✿ Hatókör, célok, folyamat, szerepek, munkatermékek, ...
- ✿ A felülvizsgálattal kapcsolatban felmerülő kérdések megválaszolása

3. Egyéni vizsgálat

- ✿ Kiosztott anyag felülvizsgálata
 - ✿ Megadott paraméterek alapján
- ✿ Jegyzetek készítése
 - ✿ Lehetséges hibák
 - ✿ Javaslatok
 - ✿ Kérdések

4. Kérdések és elemzés

- ❖ Lehetséges hibák ismertetése és elemzése
- ❖ A megközelítést az indításakor el kell dönteni:
 - ❖ defektusok felsorolása, defektusok megvitatása, javítási javaslatok defektusok mellé, ... ?
- ❖ Ez a döntés több tényezőtől függ, pl. rendelkezésre álló idő, szerzők elérhetősége, felülvizsgálat típusa
- ❖ Minőség értékelése
- ❖ Felülvizsgálat eredményének értékelése

5. Javítás és jelentés

- ✿ Hibajelentések létrehozása
- ✿ Hibák kommunikációja
- ✿ Hibák javítása (frissített státuszt jelezni kell!)
- ✿ Metrikák gyűjtése (a termékről és a folyamatról is)
- ✿ Kilépési feltétel ellenőrzése
- ✿ Elfogadás

Szerepek és felelősség

- ❖ Általános defektuskeresés, de ...
- ❖ ... minden felülvizsgáló sajátos nézőpontból vizsgálja a dokumentumokat
 - ❖ Felhasználói nézőpont
 - ❖ Karbantartói nézőpont
 - ❖ Tesztelői nézőpont
 - ❖ Üzemeltetői nézőpont, stb.
- ❖ Formális felülvizsgálat esetén a folyamathoz kapcsolódó szerepek is vannak
- ❖ Checklist-ek használata nagy segítség lehet

Szerepek és felelősség

Szerző

- A felülvizsgálat tárgyát képző dokumentum(ok) szerzője, felelőse

Témavezető (Moderátor)

- Megbeszélés levezetése
- Különböző nézőpontok közötti közvetítő szerep
- A felülvizsgálat sikere nagyrészt rajta múlik

Menedzsment

- Felülvizsgálatok végrehajtásáról dönt
- Projekt ütemezéshez igazítja a felülvizsgálatokat
- Megvizsgálja, hogy a célok teljesültek-e

Felülvizsgálat vezető

- Övé az általános felelősség
- A résztvevőkről, időzítésről és helyről ő dönt

Szerepek és felelősség

Írnok (Jegyzőkönyvvezető)

- minden észrevételt, problémát és nyitott kérdést lejegyzetel és rögzít

Felülvizsgálók

- Megfelelő technikai és / vagy üzleti tudással rendelkező személyek
- Beazonosítják és leírják az észrevételeket
- Úgy kell megválasztani, hogy különböző nézőpontokat tudjanak képviselni

Felülvizsgálat típusok

INFORMÁLIS

ÁTVIZSGÁLÁS

TECHNIKAI FELÜLVIZSGÁLAT

INSPEKCIÓ

formalitás

Egy adott dokumentum többféle felülvizsgálaton is áteshet!

1. Informális vizsgálat

Informal review

- ❖ A legfőbb cél a defektusok megtalálása
- ❖ Nincs formális eljárás
- ❖ Dokumentáció nem szükséges
- ❖ Változó hatékonyság
- ❖ Nincsenek követelmények a vizsgáló személyére, ismereteire nézve
- ❖ Olcsó módja kis haszon megszerzésének
- ❖ Megvalósítható pl. páros programozással

2. Átvizsgálás

Walkthrough

- ✿ Fő cél a defektusok keresése, a szoftver javítása, alternatívák megbeszélése, megfelelősség értékelése
- ✿ Ezen kívül: résztvevők oktatása, véleménycsere, egyetértés
- ✿ A megbeszélést a szerző vezeti
- ✿ Írnok kötelező (általában a szerző maga)
- ✿ Egyéni felkészülés és ellenőrzőlisták használata opcionális
- ✿ Lehet informális és formális is
- ✿ Szcenáriók, „száraz futások”, szimulációk

3. Technikai felülvizsgálat

Technical review

- ❖ Célok: egyetértés megteremtése, hibák kimutatása
- ❖ Minőség értékelése, új ötletek generálása , alternatív megvalósítások, szerző motiválása
- ❖ Írnok kötelező (jó, ha nem a szerző)
- ❖ Egyéni felkészülés szükséges
- ❖ A megbeszélés, ellenőrzőlisták használata opcionális
- ❖ A felülvizsgálók szakértők a megfelelő területeken

4. Inspekción

Inspection, audit

- ✿ Fő cél a defektusok megtalálása, minőség értékelése, bizalom megalapozása, hasonló hibák megelőzése
- ✿ Képzett moderátor vezeti (\neq szerző)
- ✿ Írnok kötelező
- ✿ Jól definiált szerepek, ellenőrzőlisták, be- és kilépési feltételek
- ✿ Az egyéni felkészülés létfontosságú
- ✿ Jelentés készül a problémák részletes listájával
- ✿ A folyamatok javításához metrikákat gyűjtünk

Felülvizsgálati technikák

Ad hoc

- Kevés felkészülés, általában egyetlen végigolvasás során keresnek hibákat
- Sokat számítanak a felülvizsgálók képességei

Forgatókönyvek és száraz tesztelés (Scenarios and dry runs)

- Strukturált iránymutatás a felülvizsgálat módjáról
- A forgatókönyvek segítségével könnyebb hibákat találni mint szimpla ellenőrzőlistákkal

Ellenőrzőlista alapú (Checklist-based)

- A lehetséges defektusokhoz kapcsolódó kérdésekkel áll
- Szisztematikus technika
- A listákat karban kell tartani

Felülvizsgálati technikák

Szerep alapú (Role-based)

- Különböző érintett személyek szerepében vizsgálódnak
- User admin, rendszer admin, performancia tesztelő

Perspektíva alapú (Perspective-based)

- Hasonlít a szerep alapúhoz
- Különböző nézőpontokból elemzik az anyagot
- Végfelhasználó, marketinges, tervező, tesztelő
- A felülvizsgálók megpróbálják a megadott perspektíva szerint használni a terméket
- Ellenőrzőlisták használata elvárt

Felülvizsgálat sikeressége

Tesztelők kiemelt szereplők

- ✿ minden felülvizsgálatnak előre tisztázott célja kell, hogy legyen, és a cél eléréséhez a megfelelő embert kell választani
- ✿ minden defektus fontos; a defektusok objektíven legyenek kifejtve; a felülvizsgálat vezetője figyeljen az emberekre
- ✿ A munkatermékek típusának és szintjének megfelelő technikát alkalmazzuk
- ✿ A hatékonyság növelése érdekében ellenőrzőlisták és szerepek alkalmazhatóak
- ✿ A menedzsment támogatása szükséges
- ✿ Súlyt kell helyezni a tanulásra és az eljárás javítására

Felülvizsgálat sikeressége

- ❖ A felülvizsgálat a kölcsönös bizalom alapján történik
 - ❖ Az eredményt nem a résztvevők értékelésére használják
- ❖ Objektív mértékek
 - ❖ Megtalált defektusok száma
 - ❖ A felülvizsgálatra fordított idő
 - ❖ A költségvetési megtakarítás / túlköltekezés mértéke a projektben

Statikus elemzés (analízis) jelentősége

Static analysis

1. Kritikus rendszereknél fontos

- ❖ Repülőgépteknika
- ❖ Gépjárműipar, stb.

2. Biztonsági tesztelésnél gyakran használt

- ❖ Sérülékenységvizsgálat

3. Automatikus build és CI/CD rendszereknél, agilis módszereknél

- ❖ Regressziós tesztelés mellett sokszor “minőségi kapuként” szolgál minden build után (Quality Gate)

Statikus elemzés tulajdonságai

- ❖ Általában forráskóból dolgozik
 - ❖ Ritkábban modellből, egyéb leírásokból
- ❖ Elemzőszközök végzik, fordítás előtt (kereskedelmi, ingyenes)
- ❖ Tipikusan programozók használják
- ❖ Komponens tesztelés részeként is szokták használni
 - ❖ Konfigurációmenedzsment eszközbe való küldés előtt
- ❖ Nagy mennyiségű figyelmeztetést generálhat, amit megfelelően kell kezelni
- ❖ **Mindig a felülvizsgálat részeként kell kezelni!**

Statikus elemzés által megtalált hibák

Példák

- ✿ Definiálatlan változóra hivatkozás
- ✿ Modulok közötti kapcsolatok hibái, pl. elavult API
- ✿ Nem használt változók (nem jelent futásközbeni hibát)
- ✿ Elérhetetlen (halott) kód (ez sem)
- ✿ Programozási standardok megszegése, pl. JavaDoc hiánya, változók elnevezése, zárójelek tabulálása
- ✿ Sérülékenység, pl. SQL injection ellen nem védett mező
- ✿ Rossz nyelvhasználat, pl. nem megfelelő catch block

TESZT TERVEZÉS FOLYAMATA

Teszt feltételek

Tesztesetek

Teszt eljárások

Tesztfeltétel (Test condition) - A tesztbázis egy olyan nézőpontból való megközelítése, amely releváns a meghatározott tesztcélok eléréséhez.

- A tesztelési feltétel a komponens, program vagy szoftver bármely része, tulajdonsága, eleme vagy eseménye, amely egy vagy több teszteset segítségével verifikálható
- Pl. funkció, tranzakció, minőségi jellemző, strukturális elem, stb.

TESZT TERVEZÉS FOLYAMATA

Teszt feltételek

Tesztesetek

Teszt eljárások

Teszteset (Test case) - Előfeltételek, bemenetek, tevékenységek, elvárt eredmények és utófeltételek halmaza, a tesztfeltételek alapján.

Tesztbázis (Test basis) - Az a tudásanyag, ami a tesztelemzés és műszaki teszttervezés alapjául szolgál.

Tesztorákulum (Test oracle) - A teszt tárgyat képező rendszer várt és tényleges eredményeinek összehasonlítását támogató forrás.

TESZT TERVEZÉS FOLYAMATA

Teszt feltételek

Tesztesetek

Teszt eljárások

- A teszteset egy elvárás arra, hogyan viselkedjen a szoftver: adott állapotból adott bemenet hatására a szoftver milyen kimenetet ad és milyen állapotba kerül?
- A műszaki teszttervezés során a teszteset meghatározásához:
 - előállítunk bemeneteket és hozzájuk tartozó elvárt kimenet, és
 - definiáljuk, hogy az inputot a szoftver ilyen állapotban képes fogadni, és annak hatására milyen állapotba kerül.

TESZT TERVEZÉS FOLYAMATA

Teszt feltételek

Tesztesetek

Teszt eljárások

Teszteljárás (Test procedure) - Tesztesetek sorozata végrehajtási sorrendben, és minden olyan kapcsolódó tevékenység, amelyre szükség lehet a kezdeti előfeltételek felállításához vagy a végrehajtás után.

- A teszteljárás meghatározza sorrendben az összes tevékenységet, ami a tesztesetek futtatásához szükséges
- Egy teszteljárás több tesztesetet, és egyéb (pl. előkészítő) tevékenységeket is tartalmazhat
- A teszt eljárás specifikációt gyakran *teszt szkriptnek* (manuális teszt szkriptnek) nevezik

TESZT TERVEZÉS FOLYAMATA

Teszt feltételek

Tesztesetek

Teszt eljárások

- A fenti három lépés ismétlésével:
 - Meghatározzuk, hogy mit is akarunk tesztelni
 - A tesztbázis alapján megtervezzük a teszteseteket, ami ellenőrzi a tesztelési feltételt
 - Leírjuk, hogy hogyan tudjuk végrehajtani a tesztesetet, azaz jutunk el a megfelelő állapotba, adjuk meg az inputot, és ellenőrizzük le a kimenetet és a szoftver állapotát

TESZT LEFEDETTSEG

Test coverage

- Fontos a teszt minőségének és kiterjedésének méréséhez!
- Mennyiségi mérőszámot ad a már elvégzett tesztelésről
- Általa megbecsülhető a hátralévő tesztelési munka
- A lefedettség általános fogalom, pl.:
 - Követelmény lefedettség, állapot-átmenet lefedettség
 - Kód/utasítás/útvonal/elágazás lefedettség
 - Tesztfeltétel/teszteset lefedettség
- A lefedettség mérése **bármely szisztematikus** technikához alkalmazható

TESZT TERVEZÉSI KATEGÓRIÁK

- Tesztelés célja megbizonyosodni arról, hogy:
 - a rendszer megcsinál minden, amit a specifikáció előír, és
 - a rendszer semmi más nem csinál (ez a nehezebb).
- Mindezt a lehető leghatékonyabb módon, vagyis:
 - ne legyenek redundánsak a tesztek
 - a teszt legyen *hatékony* és *hatásos* egyszerre!
- Ebben segítenek a különböző technikák

TESZT TERVEZÉSI KATEGÓRIÁK

- A technikák használata az informálistól a nagyon formálisig terjedhet
- Több körülmény határozza meg a formalitás szintjét
- Sok teszt-tervezési technika van, a legfontosabb csoportok:
 - **Feketedoboz** (specifikáció-alapú) technikák: szisztematikus!
 - **Fehérdoboz** (struktúra-alapú) technikák: szisztematikus!
 - **Tapasztalat alapú** technikák: kevésbé szisztematikus

TECHNIKA KIVÁLASZTÁSI SZEMPONTJAI

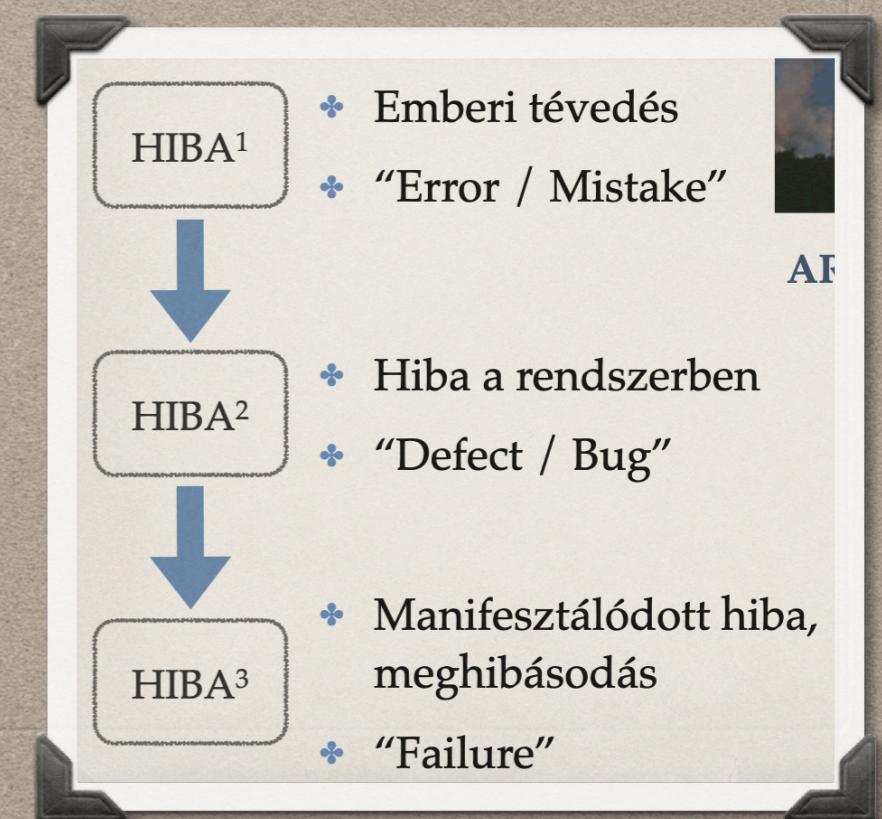
- A rendszer típusa
- Szabályozás, standardok
- Felhasználói követelmények
- A kockázat szintje
- A kockázat típusa
- A tesztelés célja
- Elérhető dokumentáció
- A tesztelők tudása
- Rendelkezésre álló idő és pénz
- A fejlesztési életciklus
- Használati eset modellek
- A megtalált defektustípusok ismerete
- Stb.

TECHNIKA KIVÁLASZTÁSI SZEMPONTJAI

- A gyakorlatban kombinációt alkalmaznak
- Tesztelési szintenként nagyban eltérő lehet
- Néhány szempont egyértelművé teheti a választást, pl. szabályozás, szerződésben leírt követelmények, bizonyos tesztelési célok
- Mások csak irányítanak valamerre, pl. kockázat mértéke és típusa, tesztelők tapasztalata
- Kevés a tiszta helyzet, az egyes szempontok a helyzettől függően lehetnek fontosabbak vagy kevésbé fontosak

EGYÜTTMŰKÖDÉSI TECHNIKÁK

- Az együttműködés (kollaboráció) minden sikeres projekt alapja
 - Agilis módszertanokban ez tudatos,
 - ezért specifikus együttműködés-alapú teszt technikák alakultak ki
- Ezek nem csak a hiba detektálására alkalmasak, de a hiba **elkerülésére** is!



USER STORY ÍRÁS

- A felhasználói történet (user story) alapvető elem az agilis tervezésben
- Leghatékonyabb, ha együttműködésben készül
- Tesztelőnek fontos szerepe van benne
- Brainstorming, mind mapping, stb.

Basic User Story Slide Template Example

User Story For: Positive Charge		Enable User Registration via Mobile App
USER STORY		ACCEPTANCE CRITERIA
As a new user, I want to register an account via the Positive Charge mobile app so that I can manage my EV charging sessions and track my usage conveniently.	1 The registration form must include fields for name, email, password, and vehicle details. 2 Users must receive a verification email upon registration. 3 The app must provide a confirmation message upon successful registration. 4 Users must be able to log in immediately after verification.	
PRIORITY  HIGH ESTIMATE 8 STORY POINTS	DESCRIPTION <p>This user story focuses on implementing a user-friendly registration process within the Positive Charge mobile app. The feature aims to streamline the onboarding process for new users, ensuring they can quickly set up their accounts and start using the app to manage their EV charging needs. Dependencies include the email verification system and secure storage for user data.</p>	

EXAMPLE

USER STORY ELFOGADÁSI KRITÉRIUMOK

Acceptance Criteria

- Olyan feltételek, amelyek megvalósítása elengedhetetlen ahhoz, hogy a user story-t elfogadják az érdekelt felek
- Emiatt ezek *teszt feltételként* is funkcionálnak:
 - Konszenzus kialakítása, megfelelő tervezés alapja
 - Pozitív és negatív, valamint nemfunkcionális teszteket is lefednek
- Formái:
 - Egyszerű lista
 - Szabályok, input-output párok
 - BDD és ATDD technikák alapja (Id. 2. fejezet)



FEHÉRDOBÓZ TESZT TERVEZÉS

Struktúra alapú, White box

- A program szerkezetének felderítése:
 - nem a helyes működés ellenőrzését szolgálja (a "külső" specifikció szerint),
 - hanem a teljes kód ellenőrzését (a "belső" megvalósítás szerint) biztosítja.
- Szerkezet lehet:
 - Programozási struktúrák komponens szinten (szekvencia, szelekció, iteráció)
 - Hívási függőségek integrációs szinten
 - Egyéb szerkezet rendszer szinten, pl. menü, üzleti folyamat, web oldal, stb.
- *Szisztematikus:* lefedettséget a kód különböző szintű elemei alapján mérhetünk

dinamikus tesztek által érintett elemek száma

összes elem száma

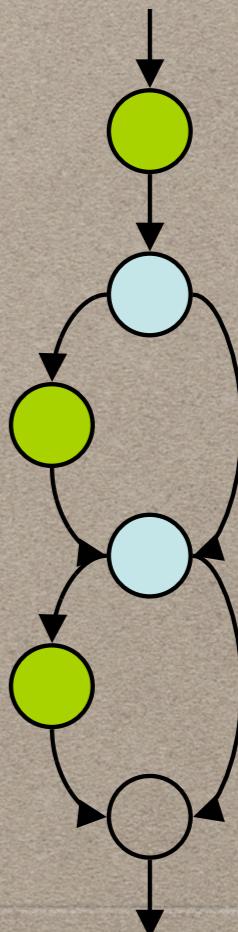
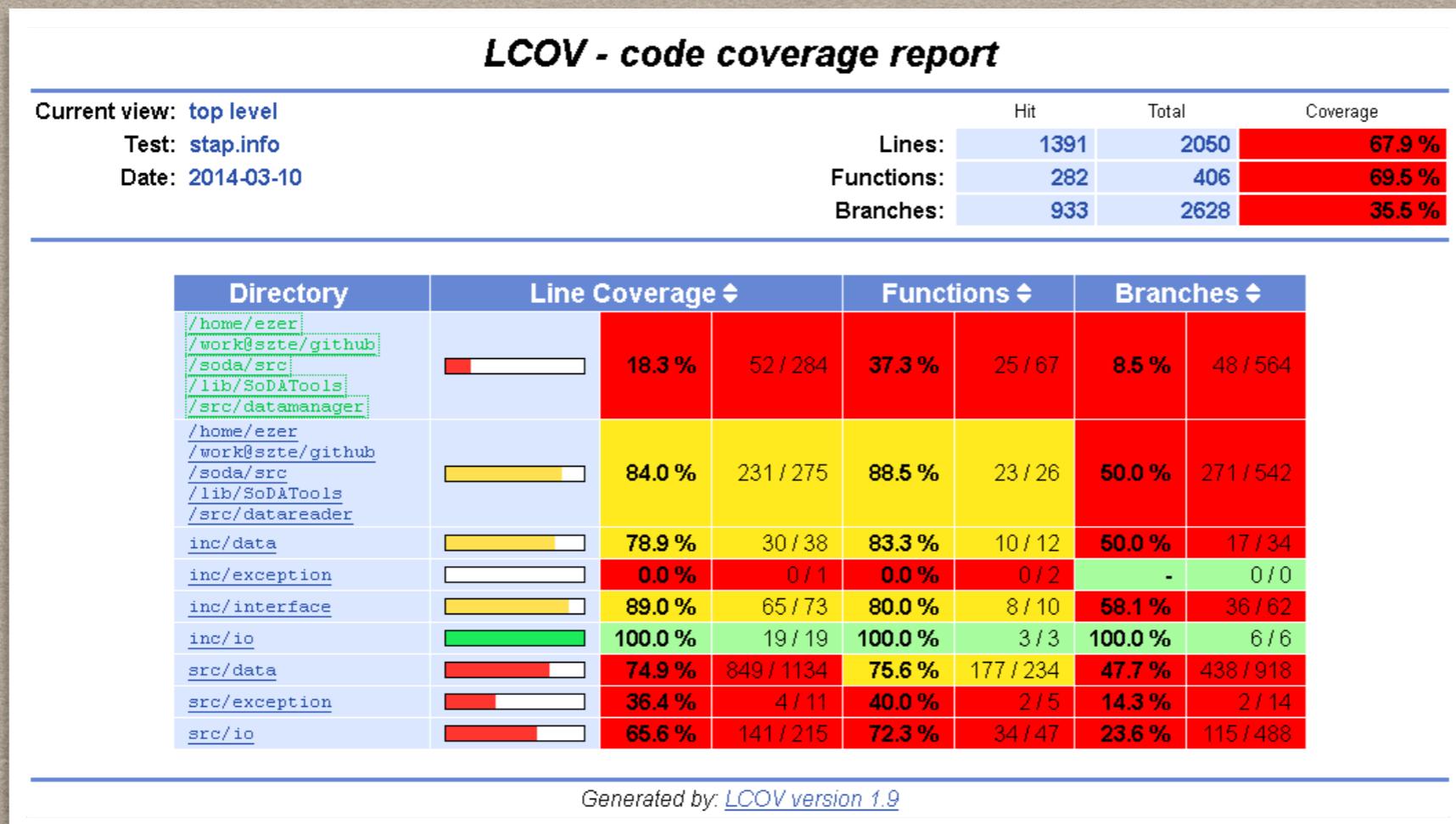
FEHÉRDOBÓZ TESZT TERVEZÉS

Struktúra alapú, White box

- Utasítás teszt és lefedettség
- Döntési teszt és lefedettség
- Egyéb technikák

```

1 Prints (int a, int b) {
2   int result = a+ b;
3   If (result> 0)
4     Print ("Positive", result)
5   Else
6     Print ("Negative", result)
7 }
```



FOLYAMATÁBRA

Flowchart

Utasítások

Döntések

Szekvencia

```
Read A  
Read B  
 $C = A + B$ 
```

Szelekció

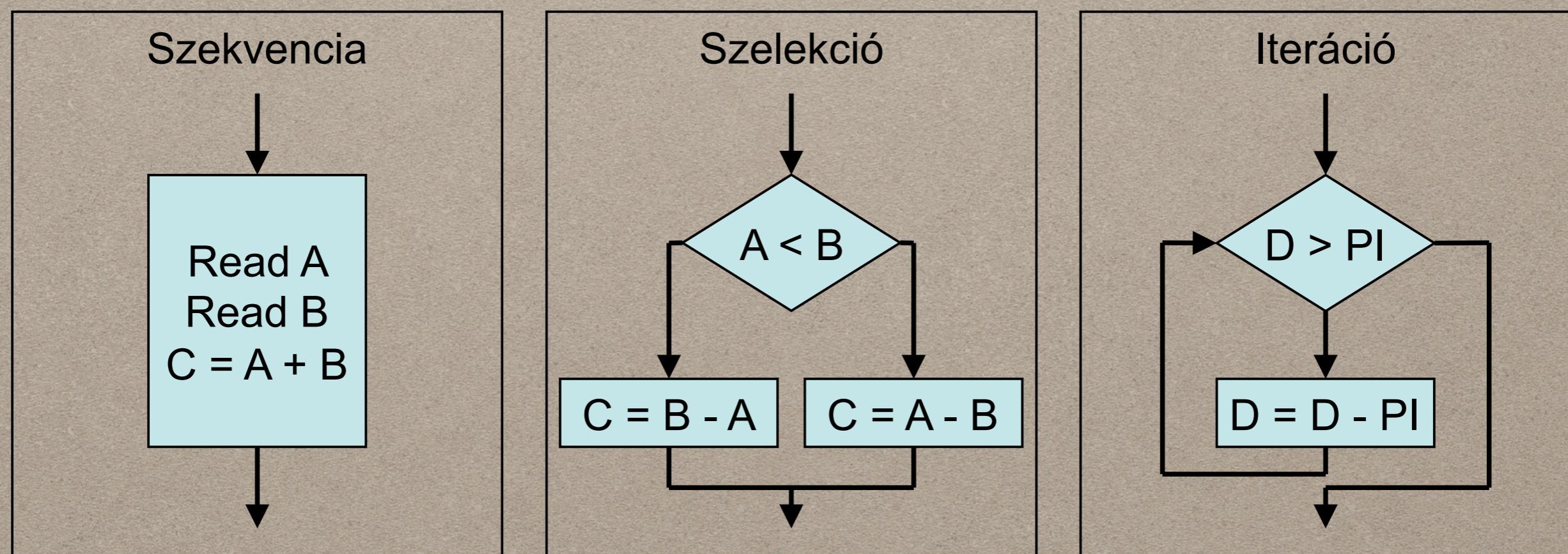
$A < B$

```
 $C = B - A$        $C = A - B$ 
```

Iteráció

$D > PI$

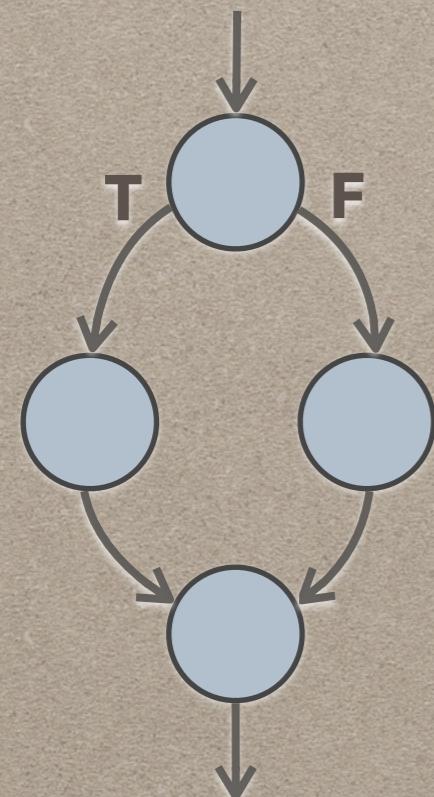
```
 $D = D - PI$ 
```



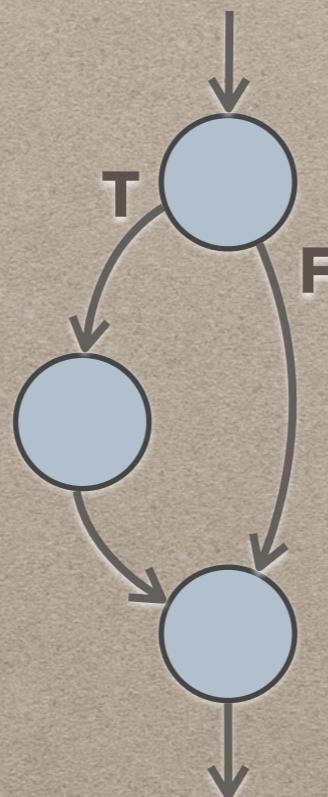
VEZÉRLÉSI FOLYAM GRÁF

Control Flow Graph (CFG)

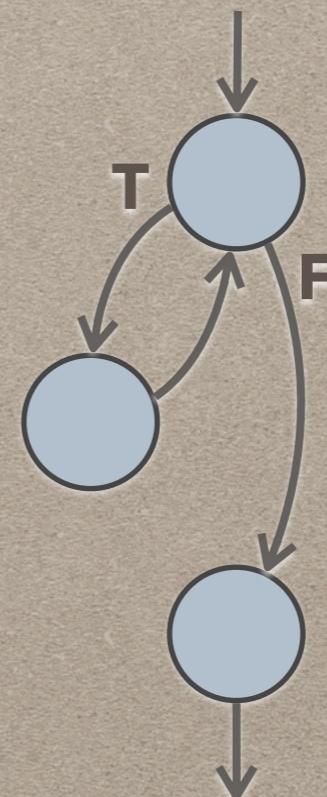
- A folyamatábrához képest csak a vezérlési folyamra hatással lévő pontokat jeleníti meg: valódi irányított gráf
- Hierarchikus is lehet: részgráfok rajzolhatók kisebb kódrészletekre
- Szekvenciális vezérlés (utasítások sorozata) egy blokk-ként kezelhető



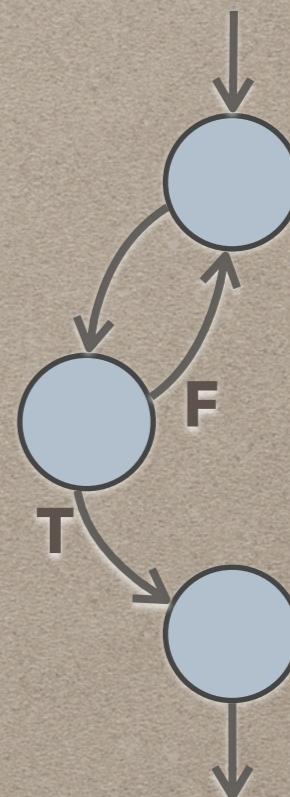
if-then-else



if-then



while-do



repeat-until

UTASÍTÁS LEFEDETTSÉG

Statement coverage

- Az utasítás teszt célja az utasítások módszeres vizsgálata
- Az utasítás lefedettség az átvizsgált (tervezett vagy végrehajtott) és az összes végrehajtható utasítás aránya
- Miért mérünk lefedettséget?
 - A 100%-nál kisebb lefedettséget produkáló teszt nem nevezhető teljesnek
 - De a 100% lefedettség önmagában nem sokat jelent

UTASÍTÁS LEFEDETTSÉG PÉLDA

Program CoverageExample

A, X: Integer

Begin

Read A

Read X

If A > 1 AND X = 2

Then

$$X = X/A$$

Endif

If A

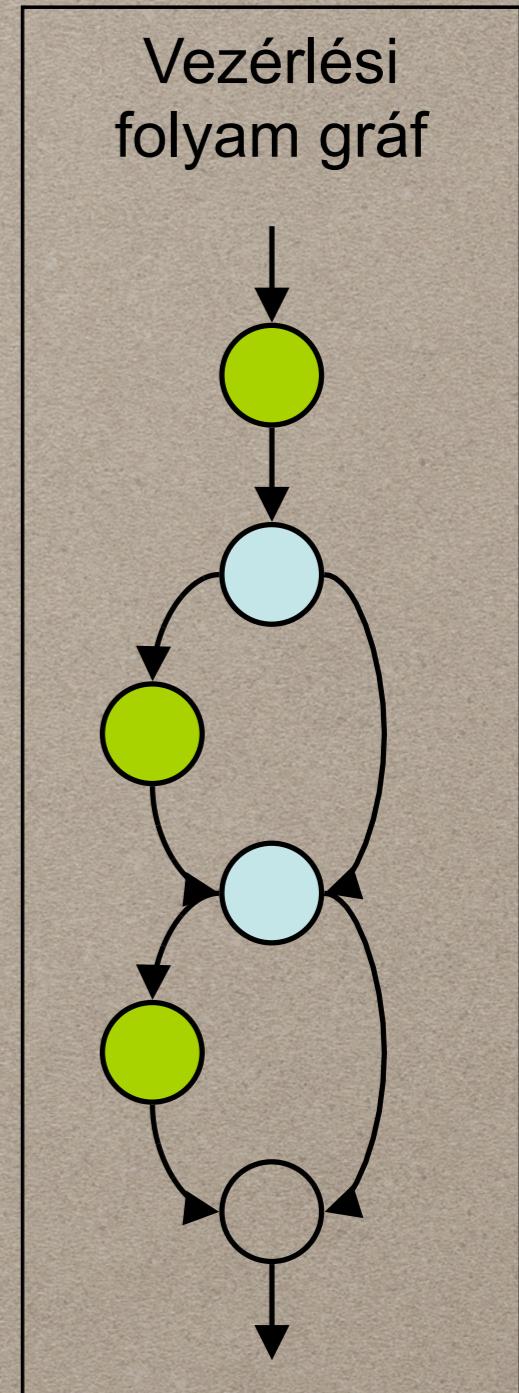
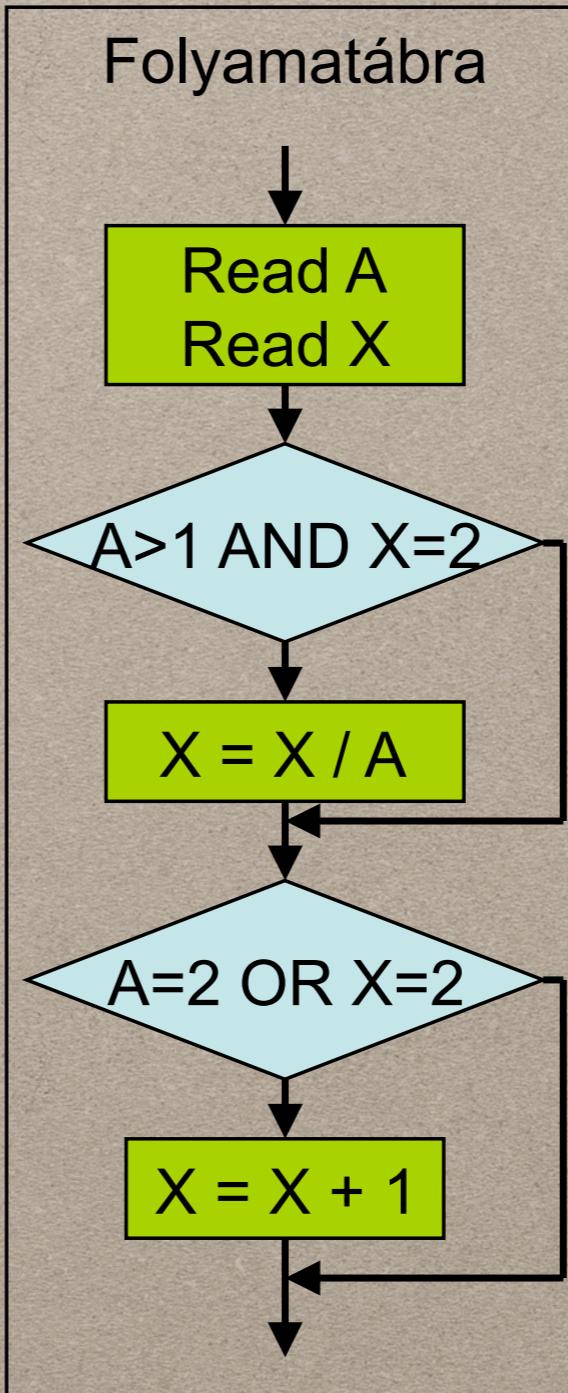
Then

3

X = X + 1

Endif

End



UTASÍTÁS LEFEDETTSÉG

- A legfontosabb a vezérlési szerkezetek megértése: „Dry run”
- Az előző példában egyetlen tesztesettel elérhető a 100% utasítás lefedettség: $A = 2, X = 2$
- Alkalmas-e ez a teszteset az alábbi hibák detektálására?
 - Első döntésben AND helyett OR kellene
 - Második döntésben $X > 2$ kellene
 - Valamelyik hamis ágban módosítani kellene X-et
- Összességében az utasítás-lefedettség még nem elég jó mérőszám

DÖNTÉSI LEFEDETTSÉG

Decision coverage

- A döntési teszt célja a döntések módszeres átvizsgálása
- minden döntési pont minden ágát érintenünk kell (ciklus: belépünk vagy kihagyjuk)
- A döntési lefedettség a tesztelt (végrehajtott) döntési eredmények és a lehetséges döntési eredmények aránya (más módon is lehet mérni)
- 100%-os döntési lefedettség \Rightarrow 100%-os utasítás lefedettség
(fordítva nem igaz)
- A döntési lefedettség meghatározásához jó kiindulópont lehet a vezérlési folyam gráf

DÖNTÉSI LEFEDETTSÉG PÉLDA

Program AgeCheck

 CandidateAge: Integer

Begin

 Read CandidateAge

If CandidateAge < 18

Then

 Print("Too young.")

Else

If CandidateAge > 30

Then

 Print("Too old.")

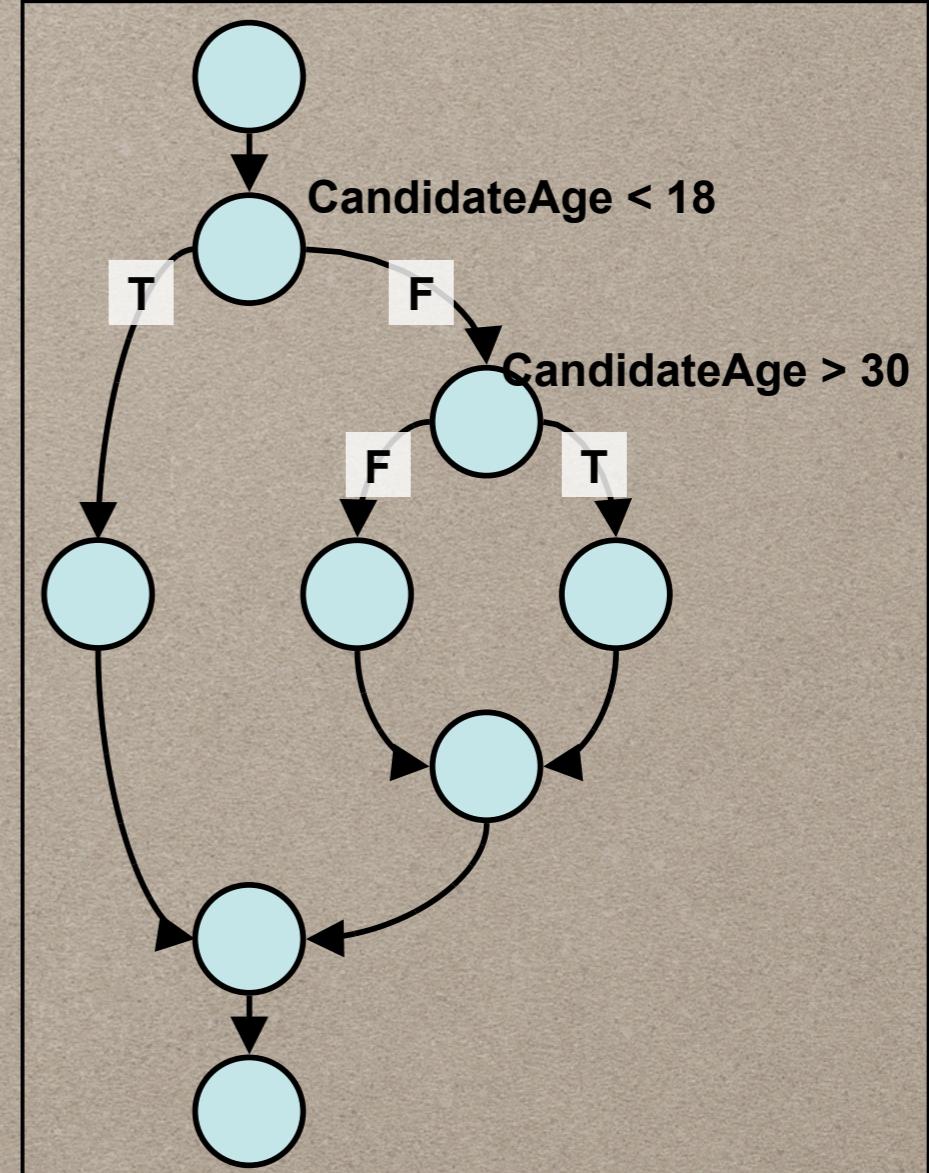
Else

 Print("May join to
club.")

Endif

Endif

End



**3 teszteset kell,
pl: 16; 21; 40**

DÖNTÉSI LEFEDETTSÉG PÉLDA

Program Check

Count, Sum, Index: Integer

Begin

Index = 0

Sum = 0

Read Count

Read New

While Index <= Count

Do

If New < 0

Then

Sum = Sum + 1

Endif

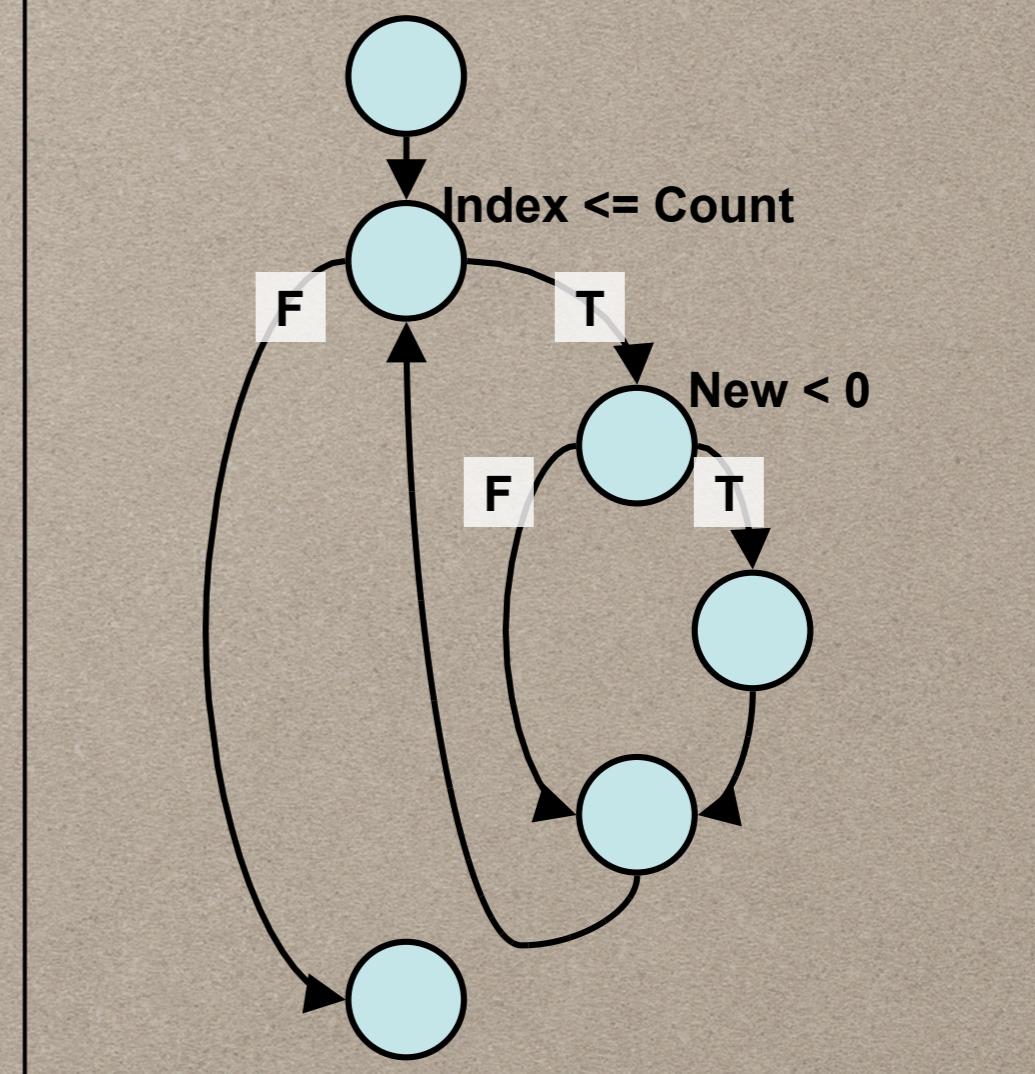
Index = Index + 1

Read New

Enddo

Print (Sum, " negative numbers")

End



A while miatt egyetlen teszteset futtatás elegendő az if minden ágának lefedéséhez

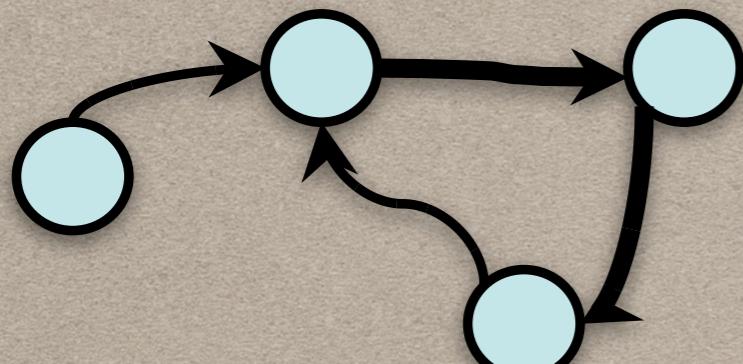
FEKETEDOBOZ TESZT TERVEZÉS

Specifikáció alapú, viselkedés alapú, Black box

- A specifikáció vagy valamilyen modell alapján dolgozik
- A teszt bázis nem mondhatja meg hogy hogyan működik a szoftver
 - Csak azt, hogy mit csinál (a „hogyan” és a „mit” szeparációja)
- A specifikáció lehet informális is, ilyenkor a tesztelő feladata megérteni és lemodellezni a követelményeket
- Nem-funkcionális követelményekre (pl. használhatóság, teljesítmény) ugyanúgy módszeresen kell teszteket gyártani
- *Szisztematikus:* lefedettséget a tesztbázis elemei és a kiválasztott technika alapján mérhetünk

FEKETEDOBOZ TESZT TERVEZÉS

Specifikáció alapú, Black box



Állapotátmenet teszt

	t2	t3	t4
P1	T	T	F
P2	T	F	-

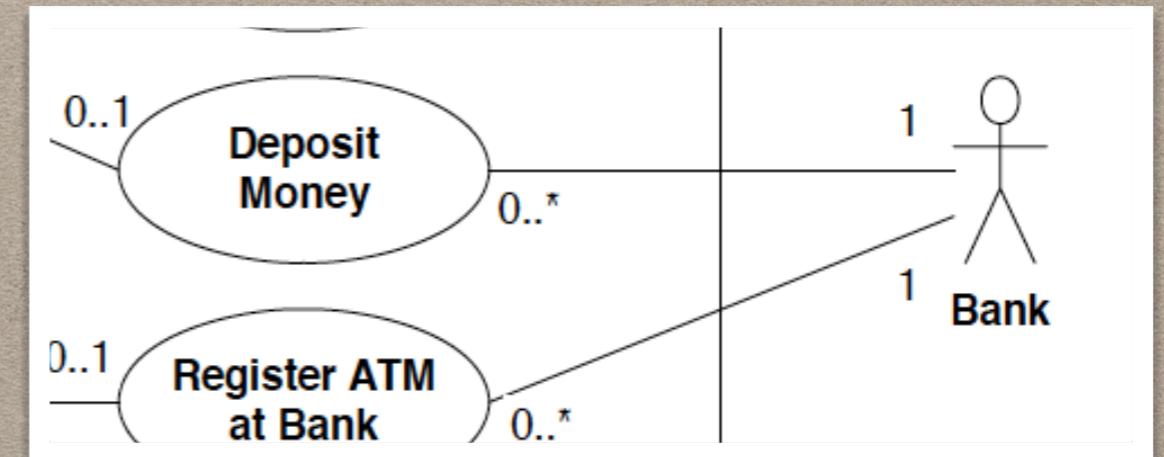
Döntési tábla teszt



Ekvivalencia partícionálás



Határérték analízis



Használati eset teszt

EKVIVALENCIA PARTÍCIÓK

Equivalence Partitioning - EP

- A valamelyen szempontból egyformán kezelt adatokat egy kalap alá vesszük
 - Ugyanabból a kalapból csak egyetlen esetet tesztelünk
- A partíciók diszjunktak
- Érvénytelen partíciókat egyenként teszteljünk
 - Pl. „nem lehet benne szám vagy ékezetes betű”
- A partíciók tovább bonthatók, ha szükséges

Miből lehet?

- * Bemenet
- * Kimenet
- * Belső értékek
- * Idő
- * Interfész paraméterek
- * Stb.

Partíciók fajtái:

- * Érvényes (valid)
- * Érvénytelen (invalid)

HATÁRÉRTÉK ANALÍZIS

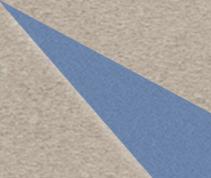
Boundary Value Analysis - BVA

- A hibák általában a specifikációban szereplő különböző határértékek környékén csoportosulnak (pl. adat intervallumok határai, különleges esetek, értékek)
- A határérték-hibák gyakran köthetőek bizonyos programozási struktúrákhoz
 - Különféle relációs jelek felcserélése
 - Ciklusindítás helyes érték ± 1 -ről, stb.
- Határértékek
 - Érvényes határérték
 - Érvénytelen határérték
 - „Belső szomszéd” (2-értékű vs. 3-értékű!)



Az ekvivalencia partíciókkal együtt kiválóan használható, annak kiterjesztése

DÖNTÉSI TÁBLA TESZT



A döntési tábla az összes bemeneti feltétel és kimeneti művelet felsorolásával, valamint az összes lehetséges döntési kombináció megjelenítésével elősegíti, hogy ne maradjon ki egyetlen lehetséges üzleti szabály sem a tesztelésből

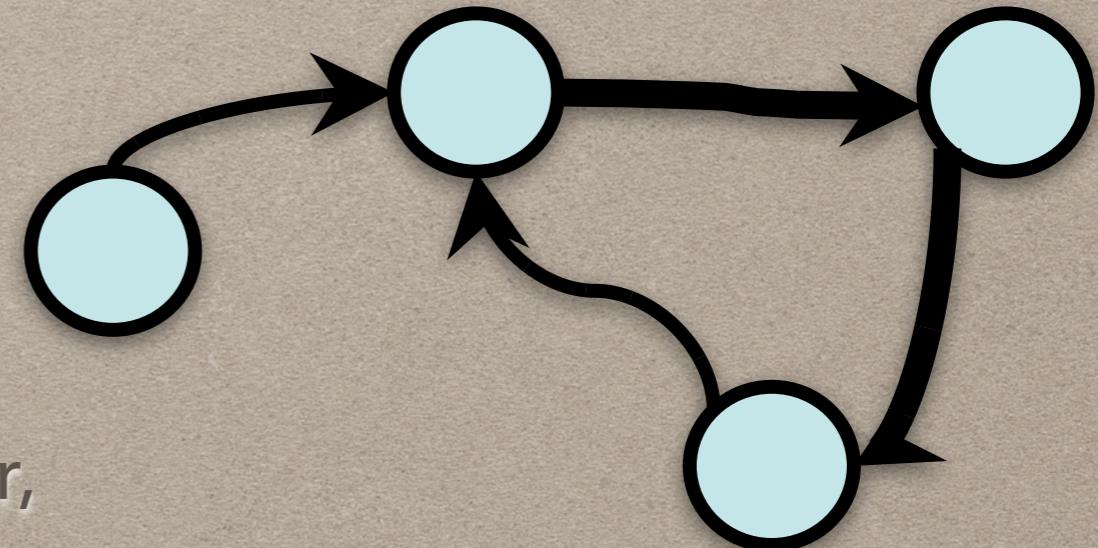
- A specifikáció gyakran tartalmaz üzleti logikát a rendszer egyes funkcióihoz rendelve
- Az egyedi döntések egyszerűek, de a kombinációik nem
- A feladat az összes kombináció tesztelése

ÁLLAPOTÁTMENET TESZT

- Olyan rendszerekhez használatos, ahol a kimenetet a bemeneti feltételeinek vagy a rendszer állapotának a változása váltja ki
 - Vezérlő rendszerek, beágyazott rendszerek, de akár web-oldalak, menüszerkezetek is
- Állapot-átmeneti diagramot használ
 - Az állapotokat körrrel jelöljük (nagyon eltérő, hogy mi lehet egy)
 - A lehetséges átmeneteket nyilak jelzik
 - A nyilakat eseményekkel (bemenet), őrfeltételekkel és kimenetekkel lehet címkézni

ÁLLAPOTÁTMENET TESZT

- Tesztesetek hogyan generálhatók szisztematikusan?
- Több stratégia
 - minden állapot,
 - minden érvényes átmenet,
 - minden érvényes átmenet-pár,
 - minden érvényes N-hosszú átmenetsorozat,
 - minden érvényes és érvénytelen átmenet (táblázatban minden cella)



HASZNÁLATI ESET TESZT

- A használati esetek „forgatókönyvek”, amik a rendszer funkcionálitását írják le különböző felhasználók szemszögéből
- Egy használati eset egy adott felhasználó és a rendszer olyan kölcsönhatását írja le, amelynek eredménye értékes a felhasználó számára
- A használati esetek (scenario) jó kiindulási alapot jelentenek a teszteléshez

TAPASZTALAT ALAPÚ TECHNIKÁK

- Ha nincs elég idő vagy pénz
- Ha nincs vagy hiányos a teszt bázis (specifikáció)
- Szisztematikus tesztek kiegészítésére
- Technikák: Hibasejtés, Hibatámadás, Ellenőrző listák, Felderítő tesztelés, Hiba taxonómiák, "Test charter", stb.

Az ad hoc teszteléstől a felderítő tesztelésig minden ilyen technika a tesztelők tudása és tapasztala segítségével deríti fel a rendszer kritikus részeit, ahelyett, hogy szisztematikusan feldolgozná a rendelkezésre álló specifikációt



- * Más módszerekkel fel nem fedezhető hibák
- * Gyors, olcsó
- * "Szórakoztató"



- * Teljesség nem ellenőrizhető
- * Komolyan kell menedzselni
- * Csak valóban tapasztalt tesztelőkkel érdemes végezni

HIBASEJTÉS

Error guessing

- A tesztelő tudása, megérzése és tapasztalata alapján készítünk teszteket
 - A módszeres tesztekkel esetleg nehezen felfedezhető hibák detektálására
 - A szoftver ismert vagy várható, esetleg múltban már előfordult hibáinak tesztelésére (régi hibák, tipikus fejlesztői hibák, hasonló szoftverek hibái)
- Hátránya a tesztelő tapasztalatától függő változó hatékonyság
 - Több tesztelő/felhasználó által gyártott közös lista a lehetséges hibákról segíthet
 - A többek által összeállított hasonló eszközökben előforduló defektusok és meghibásodások listája is jó tesztbázis lehet

FELDERÍTŐ TESZTELÉS

Exploratory testing

- A tesztelő tapasztalatát egyesíti a módszerességgel
- Hiányzó vagy rossz specifikáció és időhiány esetén jól használható
- *Defektus csoportosulás* elve jól hasznosítható
- Jellemzői:
 - Tesztelési célok köré szerveződik
 - Időkeretek, „test charter”
 - Párhuzamos teszt tervezés, végrehajtás, kiértékelés, folyamatos „tanulás”

ELLENŐRZŐLISTA ALAPÚ TESZT

Checklist-based testing

- Egy listában található (magas szintű) tesztfeltételekhez készítünk teszteseteket
- A lista alapja lehet:
 - Tapasztalat
 - Felhasználói szempontok
 - Jellemző hibák
 - Nemfunkcionális jellemzők
 - ...



Tesztmenedzser feladatai

A legfontosabb: mindenki tisztában legyen a feladatával, a feladatok időben és a költségkereten belül elkészüljenek

- ❖ A teszt policy és tesztstratégia kialakítása vagy felülvizsgálata
- ❖ A tesztelési tevékenységek tervezése:
 - ❖ tesztelési megközelítések kiválasztása, időtartam-, ráfordítás- és költségbecslés, erőforrások beszerzése, tesztszintek és ciklusok meghatározása, hibamenedzsment tervezése
 - ❖ Tesztterv(ek) készítése és frissítése
 - ❖ Teszttervek koordinációja az érdekelt felekkel
 - ❖ Teszt tevékenységek kezdeményezése, monitorozás, kilépési feltételek ellenőrzése
 - ❖ Teszt státuszjelentés és összefoglaló jelentés elkészítése

Tesztmenedzser feladatai

- ❖ Teszt folyamatok irányítása
- ❖ A hibamenedzsment, konfigurációmenedzsment tervezése
- ❖ Megfelelő metrikák bevezetése
- ❖ A tesztelési folyamatot segítő eszközök kiválasztásának és bevezetésének menedzsmentje
- ❖ Döntéshozatal a tesztkörnyezet kialakításáról
- ❖ A tesztelők, a teszt csapat és a tesztelői hivatás segítése és támogatása a szervezeten belül
- ❖ A tesztelők képességeinek javítása
- ❖ A szervezet fejlettségi szintjének javítása

Tesztelő feladatai

- ❖ Teszttervezek felülvizsgálata és részvétel a kidolgozásukban
- ❖ Tesztbázis analízise, felülvizsgálata, kiértékelése
- ❖ Tesztfeltételek azonosítása, nyomonkövethetősége
- ❖ Teszkörnyezet(ek) tervezése, kialakítása
- ❖ Tesztesetek és eljárások tervezése és megvalósítása
- ❖ Tesztadatok előkészítése, felvétele
- ❖ Részletes teszt ütemezés elkészítése
- ❖ Tesztek végrehajtása, az eredmények értékelése
- ❖ Teszteszközök használata
- ❖ Nemfunkcionális jellemzők értékelése
- ❖ Mások által kifejlesztett tesztek felülvizsgálata

További tesztelői erőforrások

- ✿ Ha specialisták nem állnak rendelkezésre, különféle tesztelési szintekhez plusz erőforrások rendelhetők:
 - ✿ Komponens és integrációs teszteléshez fejlesztő
 - ✿ Rendszer és felhasználói elfogadási teszthez felhasználó
 - ✿ Működési elfogadási teszthez üzemeltető
- ✿ Különleges szervezeti modellek:
 - ✿ Outsourcing, insourcing, crowdsourcing

 Ld. még Csapatszemlélet (1. fejezet)

Teszt tervezés és becslés

- ❖ A teszttervezés a tesztmenedzser legfontosabb tevékenysége:

- ❖ Feladatok lista
- ❖ Mérföldkövek, határidők
- ❖ A tesztelés „formája” és „mérete”

- ❖ Folyamatos tevékenység:

- ❖ Változik a szoftver, változnak a kockázatok
- ❖ A tesztelési tervezek fixálva vannak
- ❖ Az életciklusba épített *project change process* biztosíthatja, hogy a fixált dokumentáción csak jogos és dokumentált változtatásokat lehessen elvégezni



Teszt-tervezési tevékenységek

- ❖ A tesztstratégia megalkotása, a tesztelési szintek, be- és kilépési feltételek meghatározása
- ❖ A tesztelési tevékenységek beillesztése a szoftverfejlesztés folyamatába
- ❖ A tesztelendő elemek meghatározása, szerepkörök meghatározása és kiosztása, tevékenységek ütemezése, kiértékelés módjának és a kilépési feltételnek a meghatározása
- ❖ Erőforrások tevékenységekhez rendelése
- ❖ Miből áll majd a tesztelés dokumentációja?
- ❖ A menedzsment felé kommunikálandó információk meghatározása; a teszt előkészítés, végrehajtás, defektus javítás és kockázati kérdések felügyelete
- ❖ A tesztek megismételhetőségének biztosítása

A teszt-stratégia



A teszt-stratégiák

- ❖ **Analitikus:** egy adott tényező részletes elemzésén alapul, pl. kockázat alapú
- ❖ **Modell alapú:** a termék jellemzőire épülő modell alapján, pl. üzleti folyamat modellek, megbízhatósági modell
- ❖ **Módszeres:** tesztek, tesztfeltételek szisztematikus használata, pl. hiba alapú tesztelés, ellenőrzőlisták
- ❖ **Folyamat szerinti:** szabvány alapú, ágazatspecifikus előírások, pl. repülésbiztonsági szabvány
- ❖ **Irányított:** konzultatív, tanácsadókon alapuló, pl. külső biztonsági szakértők bevonása
- ❖ **Regresszió-kerülő:** automatizálás, teszt újrafelhasználás, pl. CI regressziós tesztek
- ❖ **Reaktív:** a rendszer működésére, eseményeire reagál, pl. felderítő tesztelés

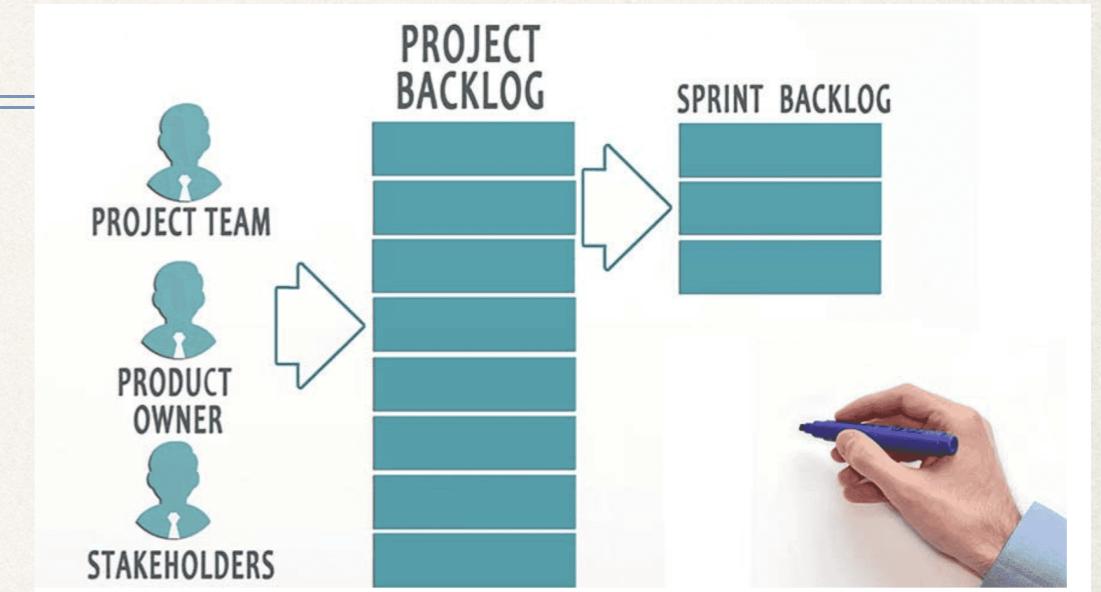
A stratégiák kombinálhatók, kockázat, tapasztalat, szabályozás, stb. alapján

Tesztstratégia és megközelítés

- ❖ *A tesztelési irányelvek:* a tesztelési stratégia rövid kivonata, általános filozófiai elvek leírása
- ❖ *A tesztelési megközelítés:* a tesztelési stratégia megvalósítása egy bizonyos projektben
- ❖ A teszttervezés és a műszaki teszttervezés során határozzák meg és finomítják
- ❖ A tesztelési célok és a kockázatelemzés kimenete határozza meg
- ❖ Az alkalmazandó technikák, belépési és kilépési kritériumok alapja
- ❖ Megválasztása sok tényezőn múlik, pl. kockázatok, kritikusság, erőforrások, képzettség, technológia, rendszer típusa, tesztelés célja, szabályozás, stb.

Tervezés iteratív folyamatokban

- ✿ Iteratív fejlesztési életciklusokban (főleg agilisban) a fő tervezési egységek:



- ✿ *Kiadás tervezés.* Tesztelő szerepe: tesztelhető követelmények, kilépési feltételek, kockázatelemzés, teszt erőforrásbecslés és tervezés
- ✿ *Iteráció tervezés.* User story kockázatelemzés, tesztelhetőség elemzés, feladat lebontás (főleg tesztelési), erőforrás becslés



Belépési feltétel

Entry Criteria, Definition of Ready

- ✿ Meghatározza mikor kezdhető el a tesztelés...
 - ✿ egy tesztelési szint elején,
 - ✿ amikor a tesztek egy halmaza végrehajtásra kész

Tipikusan például:

- Teszt környezet elérhetősége és készenléte
- Tesztelési eszköz készenléte a környezetben
- Tesztelendő kód elérhetősége
- Tesztelési adatok elérhetősége



Kilépési feltétel

Exit Criteria, Definition of Done

- ❖ Bármely tesztelési tevékenységhez rendelhető, pl. teszttervezéshez, a specifikáció vagy végrehajtás egészéhez, illetve szintjeihez külön-külön
- ❖ Célja, hogy a döntésekhez elegendő informaciót szolgáltasson
- ❖ A fontosabb folyamattervekben lennie kell kilépési feltételnek
- ❖ Minél korábban meg kell határozni, bár úgyis változik

Tipikusan például:

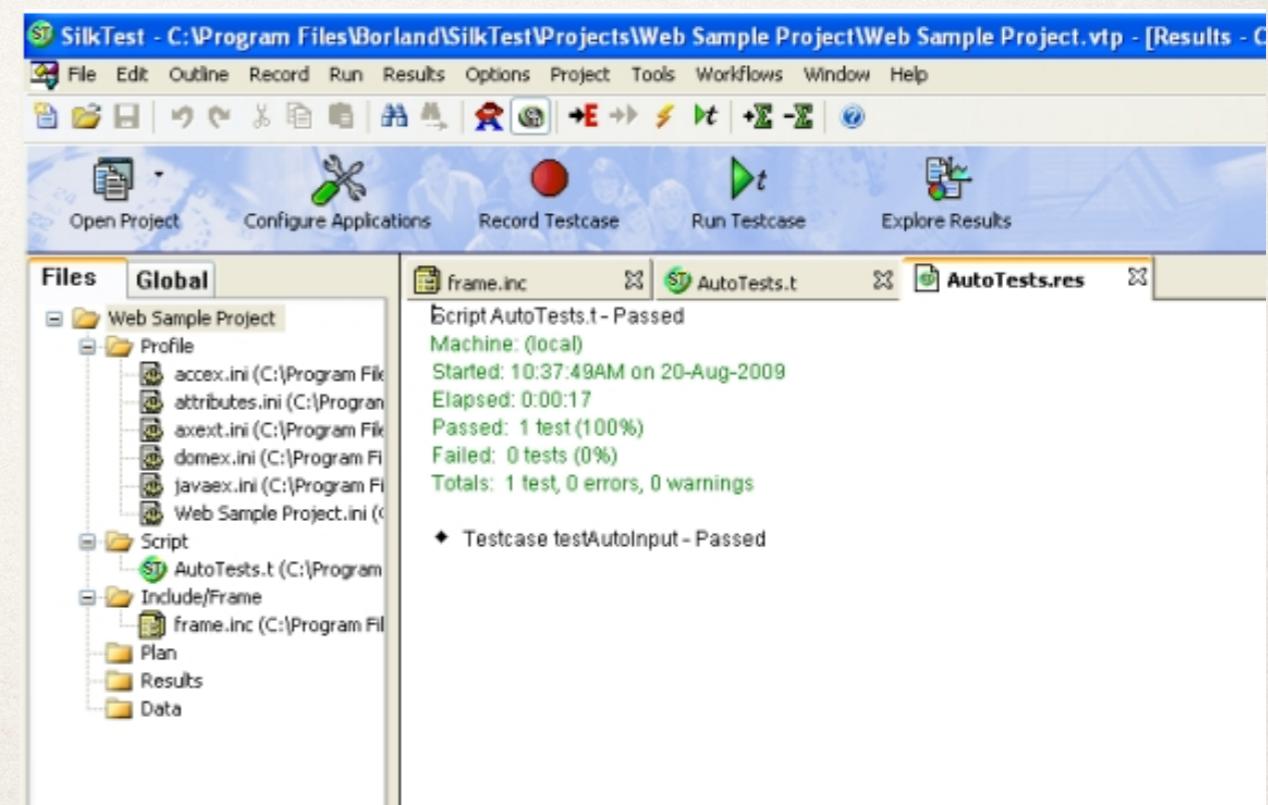
- Minden tervezett teszt lefutott
- Bizonyos szintű lefedettség elérése
- Nem maradt súlyos defektus
- minden magas kockázatú terület teljesen tesztelve lett
 - (Elfogyott a pénz, lejárt az idő)

Teszt végrehajtás

1. Teszt eljárás implementációja
2. Végrehajtás (kézi vagy automata)
3. Kiértékelés, elvárt eredmény ellenőrzése
4. (Hiba esemény rögzítése)
5. Kilépési feltétel ellenőrzése
6. Ismétlés

Teszt automatizálás:

- Egységesztelés
- Interfész tesztelés
- Protokoll tesztelés
- Adatvezérelt tesztelés
- “Capture and replay”, stb.



Végrehajtási ütemterv

- ✿ A tesztcsomagok (*test suite*) futási sorrendjét határozza meg
- ✿ Általában a magasabb prioritású kerül előre
- ✿ Függőségekre figyelni kell
- ✿ Ellenőrző és regressziós tesztek is kellenek
- ✿ Szelektív regressziós teszt priorizálás
- ✿ “Smoke”, “nightly”, “weekly” tesztek

Priorizálási stratégiák

1. *Kockázat alapú*: kockázat elemzés az inputja (ld. később)
2. *Kódlefedettség alapú*: fehérdoboz technika esetén: a legnagyobb abszolút vagy inkrementális kódlefedettség (pl. utasítás) szerint
3. *Követelmény alapú*: amennyiben van nyomon követhetőség, akkor azok prioritása szerint (üzlet mondja meg)

Becslést befolyásoló tényezők

- ❖ Termék jellemzői:
 - ❖ Tesztbázis mérete, elérhetősége, minősége
 - ❖ Végtermék komplexitása
 - ❖ Nem funkcionális, biztonsági követelmények száma
 - ❖ Szükséges dokumentáció mérete
- ❖ A tesztelés eredménye:
 - ❖ Hibák száma és jellege
 - ❖ Szükséges átdolgozások száma
- ❖ Fejlesztési folyamat jellemzői:
 - ❖ Időkeretek mérete
 - ❖ Rendelkezésre álló pénz
 - ❖ A fejlesztők és tesztelők képzettsége
 - ❖ Használt eszközök
- ❖ Az emberek jellemzői:
 - ❖ Tapasztalat
 - ❖ Képességek
 - ❖ Csapatszellem, vezetés

Becslési stratégiák

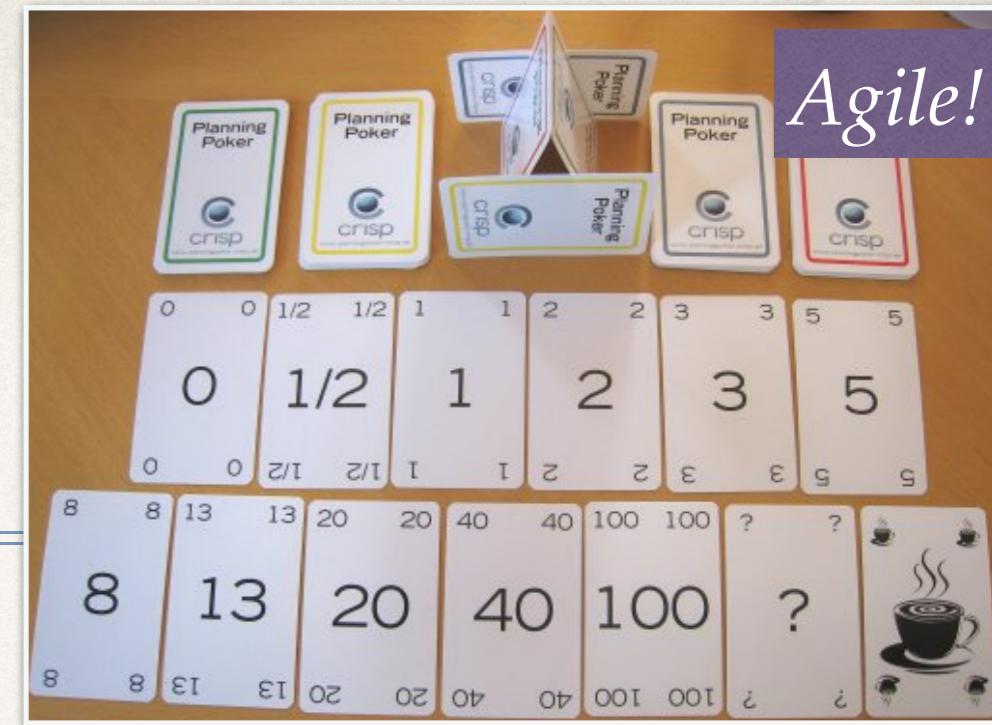
Metrikákon alapuló

- ❖ Információ más projektekből, pl. tesztelési feltételek, megírt vagy végrehajtott tesztesetek, megtalált defektusok száma, tesztesetek fejlesztésére, végrehajtására fordított idő, környezetből adódó leállások száma és hossza
- ❖ Hasonló projekteknél meglehetősen pontos
- ❖ Fontos az aktuális projekt teszteléséhez felhasznált idő és pénz pontos rögzítése, pl. burndown charts

Szakértői

- ❖ Emberek tapasztalatai alapján: üzleti szakértők, tesztelési konzulensek, fejlesztők, üzemeltetők, tervezők, de alapvetően bárki, akinek tesztelői tapasztalata van
- ❖ Hogyan becsüljünk? Saját részfeladatok becslésének összegzése, az egész feladatra vonatkozó egyedi becslések megvitása
- ❖ Egy projekten belül akár vegyesen is használhatóak
- ❖ Pl. Planning poker, Wideband Delphi

Becslési technikák



1. *Arány alapú*: korábbi projektekből meghatározzuk a fejlesztés / tesztelés arányt és azt vesszük alapul
2. *Extrapoláció alapú*: a projekt elejétől mérjük a tesztelési ráfordítást, és abból becsüljük a továbbiakat
3. *Wideband Delphi*: fokozatos közelítés elvén működő iteratív módszer szakértőkkel (Planning Poker is ilyen)
4. *Hárompontosbecslés*: pessimista+reális+optimista becslés súlyozott átlaga hibahatárral

Monitorozás és kontroll

- ❖ Célja a kész folyamattervben rögzített folyamat előrehaladásának mérése
- ❖ A szükséges adatok manuálisan vagy automatikusan is gyűjthetők
- ❖ A gyűjtött adatok a kilépési feltétel kiértékeléséhez is felhasználhatók
- ❖ **Teszt metrikák** használata a tervezett ütemezéshez való összehasonlításra

Folyamat-metrikák

- ❖ Fejlesztési tevékenységek jellemzői, konkrét projekt vagy folyamat általában
- ❖ Honnan gyűjthető?
 - ❖ Dokumentációból
 - ❖ Támogató adatbázisokból
- ❖ Gyakran relatív: *idő, méret*
- ❖ (*Termék metrikák: termék, forráskód jellemzői, pl. komplexitás*)

Példák:

Mean Time To Change (MTTC)

Mean Time Between Failures (MTBF)

Defect Removal Efficiency (DRE)

Defect Detection Percentage (DDP)

Defects / kLOC

% Automated tests

kLOC / Person Month

Általános teszt metrikák

- ✿ Elkészült és tervezett tesztesetek aránya
- ✿ Tesztkörnyezet kialakítására fordított munka százalékos aránya
- ✿ Teszt futtatási adatok (futtatott/hátralévő/pass/fail)
- ✿ Defektus adatok (hibasűrűség, javított, ellenőrző tesztek)
- ✿ Követelmény/kockázat/kód lefedettség
- ✿ A tesztelő szubjektív véleménye
- ✿ Tesztelési mérföldkövek dátumai
- ✿ Tesztelési költségek



A tesztjelentés

- ❖ Fontos tisztázni, hogy kinek szól a jelentés
- ❖ A jelentés tartalma lehet:
 - ❖ A vizsgált időszak eseményei
 - ❖ További döntéseket segítő metrikák: hátralévő defektusok megbecslése, a teszt gazdasági előnyei, kiemelkedő kockázatok, mennyire tűnik jónak a szoftver?
 - ❖ A gyűjtött információ tovább hasznosítható a folyamat javítására, pl. jók-e a tesztelés kitűzött céljai, a stratégia megfelelő-e, tesztelés hatékony-e?



Teszt kontroll

- ❖ A tesztprojekt állapotáról gyűjtött információk alapján hozunk a tesztelést / fejlesztést érintő döntéseket
- ❖ A cél a kilépési feltétel teljesítése

Lehetséges döntések:

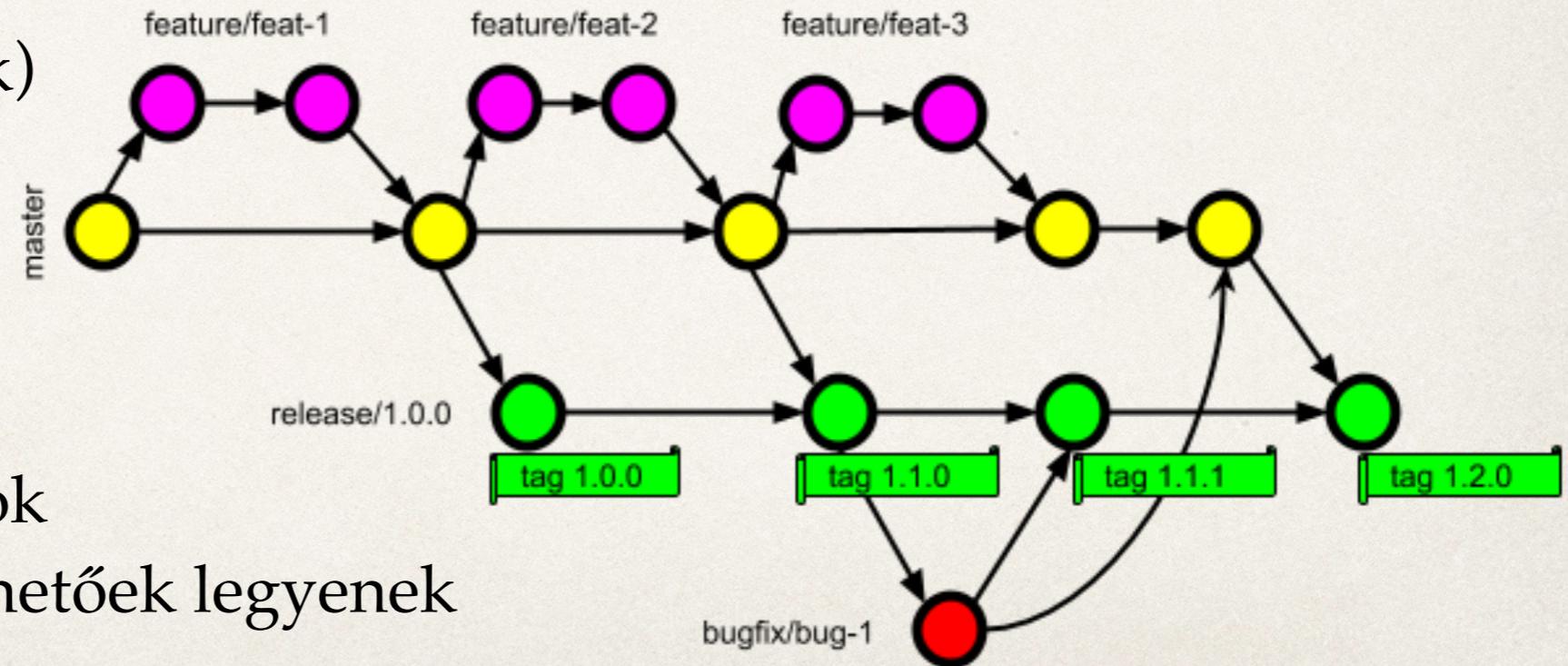
- Tesztek újrapriorizálása, ha egy projekt kockázati elem „bejön”
- Ütemezés módosítása a tesztkörnyezet elérhetősége miatt
- Hibajavítás elfogadásának előzetes teszteléshez kötése
- Kockázatok újraértékelése
- Tesztcélok módosítása, késői követelményváltozások miatt

Nem kifejezetten teszt menedzseri döntések:

- Funkcionalitás csökkentése
- Release elhalasztása
- Szállítás utáni folyamatos tesztelés

Konfiguráció-menedzsment

- ❖ Célja a termékek (dokumentáció, forráskód, adat, stb.) integritásának megőrzése a fejlesztési folyamat során
- ❖ A fejlesztett termék változásait, verzióit egyértelműen kell tudni azonosítani
- ❖ A tesztelés elemei (pl. folyamat, tesztesetek) szintén változhatnak, azokat szintén verziókövetni kell
- ❖ Fontos, hogy a változások utólag is nyomon követhetőek legyenek



Konfiguráció-menedzsment

- ❖ A termék és a tesztelési elemek kapcsolatát egyértelműen kell megadni
 - ❖ Pl. egyértelmű hivatkozások a szoftver-elem vagy dokumentáció megfelelő verziójára
- ❖ A legtöbb esetben a konfiguráció menedzsmentről maga a projekt gondoskodik, a tesztelés csak felhasználja azt
- ❖ Ha mégsem így van, akkor a teszt vezetőnek kell beterveznie azt

Kockázat és tesztelés

- ❖ Ha nem lenne kockázat, nem kellene tesztelni
- ❖ A teszt vezetőnek kétféle kockázattal kell számolnia:
 - ❖ *Projekt kockázat:* a tesztelés egészét fenyegető „külső” kockázatok
 - ❖ *Termék kockázat:* a tesztelés célja ezen kockázatok csökkentése

Kockázat (risk):

egy hatással és valószínűséggel jellemzett tényező (esemény, veszély, fenyegetés), amelynek a jövőben negatív következményei lehetnek

Kockázat szintje (level of risk):
valószínűség × hatás

Projekt-kockázatok, példák

- ❖ *Projekt*: feladatok csúszása, késői változások, pontatlan becslések
- ❖ *Szervezeti*: kevés tapasztalat vagy ember, személyes konfliktusok, elérhetőségek
- ❖ *Politikai*: elégtelen visszajelzés a tesztelőktől, fejlesztők nem figyelik a visszajelzéseket, rossz hozzáállás, túlzott elvárások
- ❖ *Technikai*: rosszul definiált követelmények, tesztkönyezet csúszása, gyenge minőségű fejlesztési munkatermékek, rossz hibakezelés
- ❖ *Szállítói*: harmadik fél késve, rossz minőséget szállít, csőd, szerződések csúsznak



Fontos a kockázatcsökkentés



Lehetséges kockázatokat dokumentálni kell

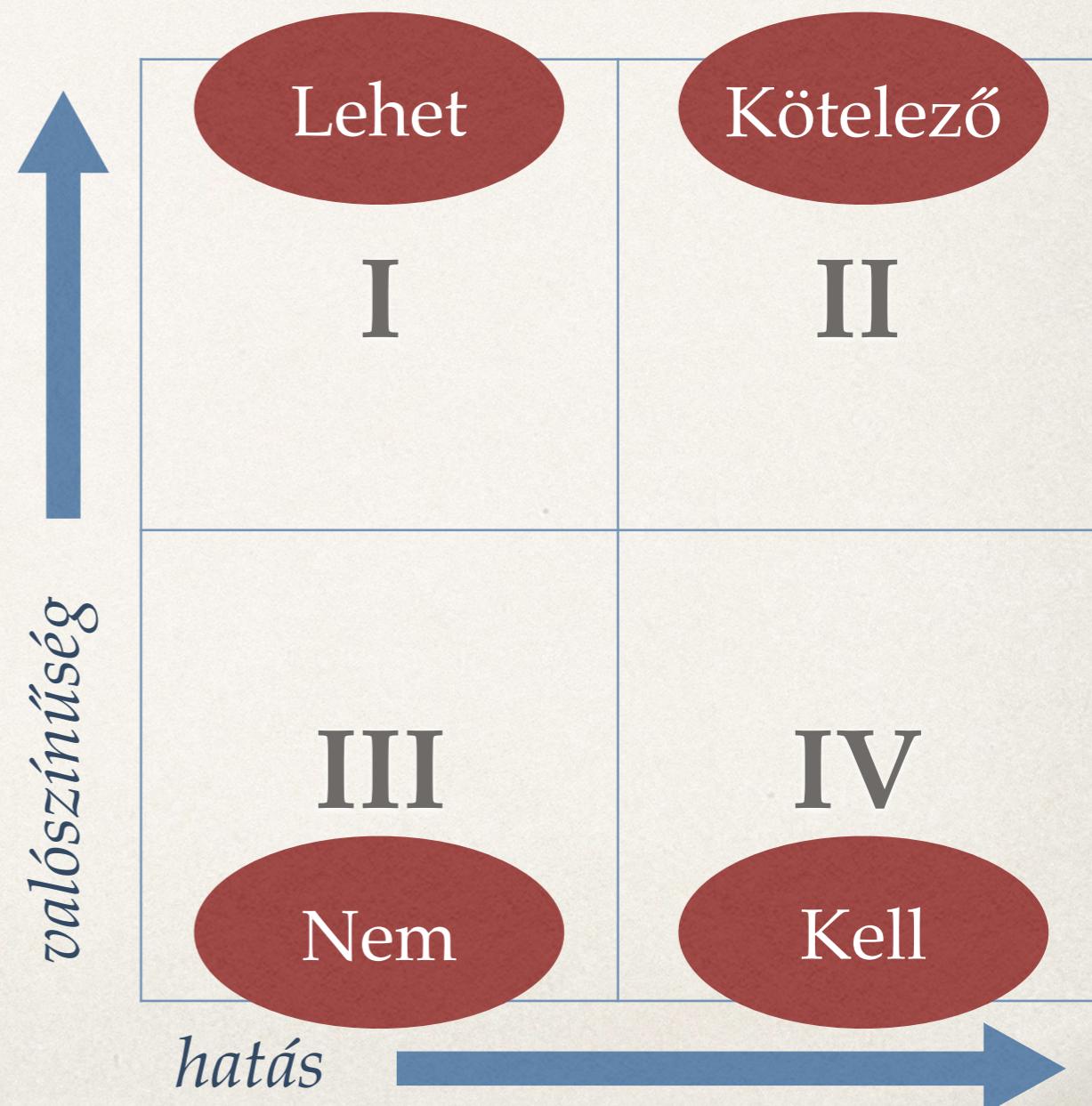
Kockázat alapú tesztelés

- ❖ A kockázatok:
 - ❖ meghatározhatják a használt technikát és / vagy a tesztelés mennyiségét
 - ❖ segíthetnek a tesztesetek priorizálásában
 - ❖ meghatározhatják a kockázatcsökkentéshez szükséges nem tesztelési tevékenységeket
- ❖ Az összes résztvevő ismereteit felhasználjuk a kockázatok meghatározásához és minimalizálásához

Kockázatelemzés

Kockázat = bekövetkezés valószínűsége × hatás mértéke

- Tesztelés = kockázat kezelése
- Mindkét szempontra 2-5 szintű skálák:
 - Alacsony, közepes, magas
 - Elhanyagolható, mérsékelt, súlyos, stb.
- minden tevékenységünket ez vezérelje!



Kockázatmenedzsment

- ❖ A kockázatmenedzsment szabályai:
 - ❖ A kockázatfelmérés folyamatos legyen a teljes fejlesztési cikluson keresztül
 - ❖ A kockázatokról mindenkorban tudni kell, melyik mennyire fontos
 - ❖ A kockázatokat különféle módszerekkel csökkenteni kell
 - ❖ Kellenek kezelési tervezet!
- ❖ A tesztelés is egy kockázatkezelő tevékenység
- ❖ **De nem csak az létezik! ➡ Általános QA**

Kockázatmenedzsment lépései

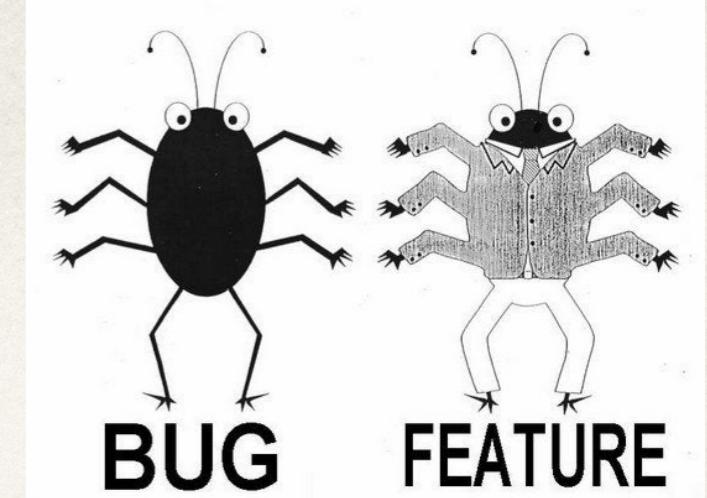
1. Kockázatelemzés

- ❖ *kockázat beazonosítása*: kreatív technikák, pl. ötletelés, interjúk, okozat gráfok, stb.
- ❖ *kockázat kiértékelése*: osztályozás valószínűség és hatás szerint, prioritálás, kezelési lehetőségek

2. Kockázatkontroll

- ❖ *kockázat kezelése*: kockázati szint csökkentése, beazonosított tevékenységek végrehajtása (termék-kockázat esetén **megfelelő tesztelés elvégzése!**)
- ❖ *monitorozás*: kezelés hatékonyságának mérése, kiértékelés javítása, új kockázatok azonosítása, régiek felülvizsgálata

Hibamenedzsment



- ✿ Az aktuális és tervezett eredmények közötti eltéréseket ki kell vizsgálni, hiszen defektusra utalhatnak
- ✿ A tesztelőknek meg kell próbálniuk kiszűrni a „nem valódi” hibákat
- ✿ Megfelelő intézkedéseket kell foganatosítani, hogy a hibák kezelve legyenek
- ✿ A hibák követve kell, hogy legyenek: a felfedezésüktől és osztályozásuktól kezdve, a javításukig és az ellenőrzésig
- ✿ A szervezetnek fel kell állítania a hibamenedzsment folyamatait és az osztályozás szabályait

Incidens

Incidens:

minden olyan nem tervezett esemény, ami további vizsgálatokat igényel

- ❖ Incidens menedzsment folyamata magában foglalja:
 - ❖ Események feljegyzése
 - ❖ Események osztályozása
 - ❖ Esemény hatásának beazonosítása
 - ❖ Észleléstől a javításig nyomon követhetőség biztosítása
 - ❖ Újratesztelés
 - ❖ Lezárás

*Nagyon fontos a szervezetek **incidens menedzsment** folyamatának dokumentálása*

*Pl. kinek van jogosultsága lezárni egy **incidenst** (nem annak, aki "kijavította" a hibát!)*

Incidens

Incidens:

minden olyan nem tervezett esemény, ami további vizsgálatokat igényel

- ✿ Nem minden incidens jelent hibát! Pl.:

- ✿ Rosszul értelmezett teszteset
- ✿ Más futtatási környezet
- ✿ Nem reprodukálható
- ✿ Nem szoftverhiba, hanem konfiguráció vagy adat
- ✿ Stb.



A hibajelentés

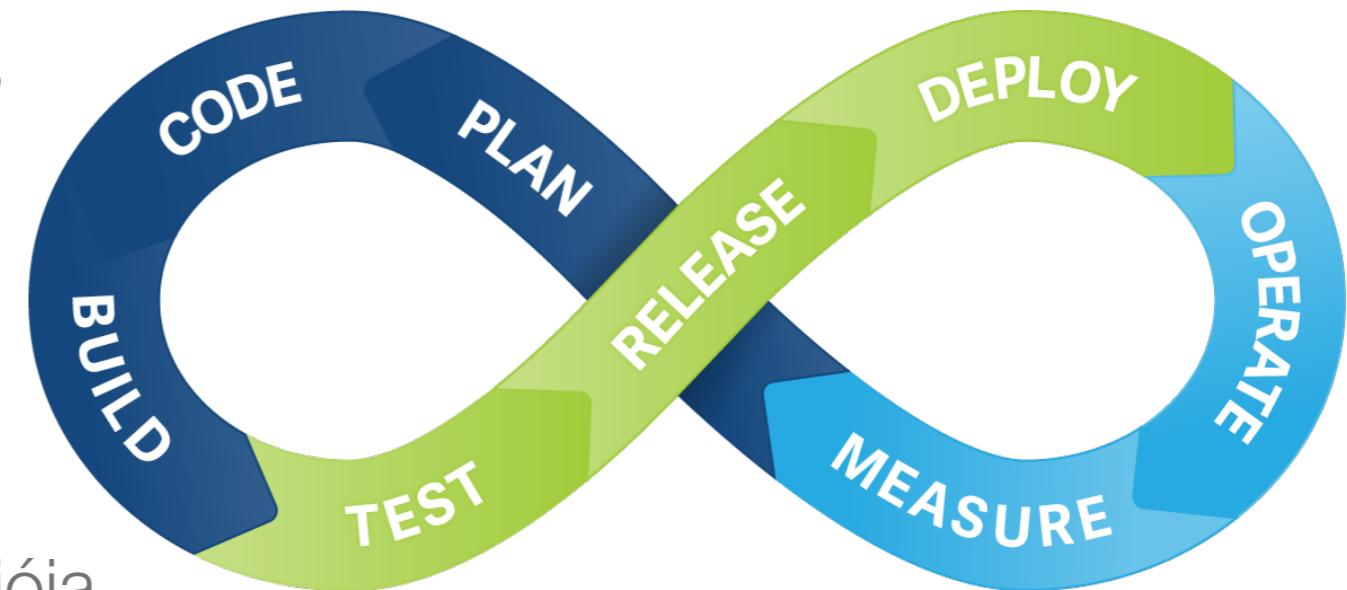
- ❖ Céljai:
 - ❖ Elegendő információ nyújtása az adott probléma reprodukálásához, beazonosításához, elkülönítéséhez, kijavításához
 - ❖ A teszt vezető képet kap a szoftver minőségéről (hibák száma, prioritása, javítások)
 - ❖ Ötleteket adhat a tesztelési folyamat javításához (mely részekre kell fókuszálni?)

CASE eszközök osztályozása

1. Követelményelemzés
2. Modellezés
3. Implementáció,
kódgenerálás
4. Rendszerépítés,
fordítóprogramok
5. Nyomkövetés, **tesztelés**,
hibamenedzsment
6. Dokumentációkészítés
7. Folyamatirányítás
8. Kommunikációs
eszközök, kollaboráció
9. Produktivitást növelő
eszközök
10. Infrastruktúra támogatás,
virtualizáció

Trendek

- DevOps: Development-Operations
 - Continuous Delivery
 - Continuous Deployment
- Fejlesztés és üzemeltetés integrációja
 - Fejlesztés, CI, kiadás, konfiguráció, üzemeltetés monitorozás
- Építés, tesztelés és kiadás nagyon gyakori (percek!) és hatékony
- Virtualizáció, konténerizáció
- Erős eszköztámogatás, de szemléletváltás is



facebook

NETFLIX

twitter

A teszteszköz

Teszteszköz (Test tool):

Egy vagy több tesztelési tevékenységet támogató szoftver vagy hardver.

- Gyakorlatilag bármilyen, ami könnyebbé, gyorsabbá, pontosabbá teszi a tesztelést
- *Előnyök:*
 - Hosszútávú erőforrás-spórolás
 - Az emberi tényező kiküszöbölésével csökken a hibák valószínűsége
 - Egyszerűbb a statisztikai információk beszerzése (pl. defektusok száma)
- *Kockázatok:*
 - Az eszköz várható hasznának túlbecslése
 - Elégtelen kezdeti befektetés

Támogatott tevékenységek

- **Közvetlenül teszteléskor használt eszközök:**
 - Teszt végrehajtás
 - Teszt-adat generálás
 - Eredmény összehasonlítás
- **Felderítéskor használt eszközök:**
 - Alkalmazás aktivitásának monitorozása
- **Tesztelési folyamatot támogató eszközök:**
 - Teszt eredmények, adatok, követelmények, incidensek, defektusok, stb. menedzselése
 - Teszt végrehajtás riportálása és monitorozása
- **Bármely más eszköz, ami segít:**
 - Táblázatkezelő, e-mail, stb.

Teszteszközök::Osztályozás::Céljai

Eszköztámogatás céljai

1. Tesztelési feladatok hatékonyságának növelése, pl. idő csökkentése
2. Manuális tesztelési tevékenységek támogatása, pl. tesztesetek nyilvántartása
3. Ismétlődő, fárasztó manuális tevékenységek kiváltása, pl. tesztadatok előkészítése és feltöltése
4. Kézzel nem végrehajtható tevékenységek, pl. load tesztek
5. Tesztelés megbízhatóságának növelése, pl. eredmények összehasonlítása

Kategorizálási szempontok

- Kereskedelmi/ingyenes/nyílt/stb.
- Különálló/integrált
- Módosítja-e a működést
(pl. kódlefedettség mérés)?
- Alkalmazott technológia
- Milyen típusú tesztelést támogat?
- Ki használja? (fejlesztő, elemző, ...)
- A tesztfolyamat mely részét segítik? ← *most e szerint*



Teszteszközök::Osztályozás::Menedzsment

Eszközök projektmenedzsment támogatására

- Cél: projektmenedzsment hatékony és átlátható megvalósítása
- Nem kifejezetten szoftverfejlesztésre specifikus

Általános irodai eszközök:

- szövegszerkesztés
- táblázatkezelés
- prezentációkezelés

Kommunikációs eszközök:

- e-mail
- instant messaging
- web konferenciák

Dedikált eszközök:

- ütemezés tervezés
- feladatkövetés
- munkaóra menedzsment
- dokumentumtár eszköz

Eszközök funkciói

1. Munkaidő nyilvántartás

- Ki, mikor, mely feladaton dolgozott

2. Feladat kezelés

- Jegyek létrehozása, hozzárendelése, stb.

3. Kommunikációs felület, közös dokumentumtárral

- Pl. wiki oldalak

4. Fájl szerver



GitLab



trac

Integrated SCM & Project Management



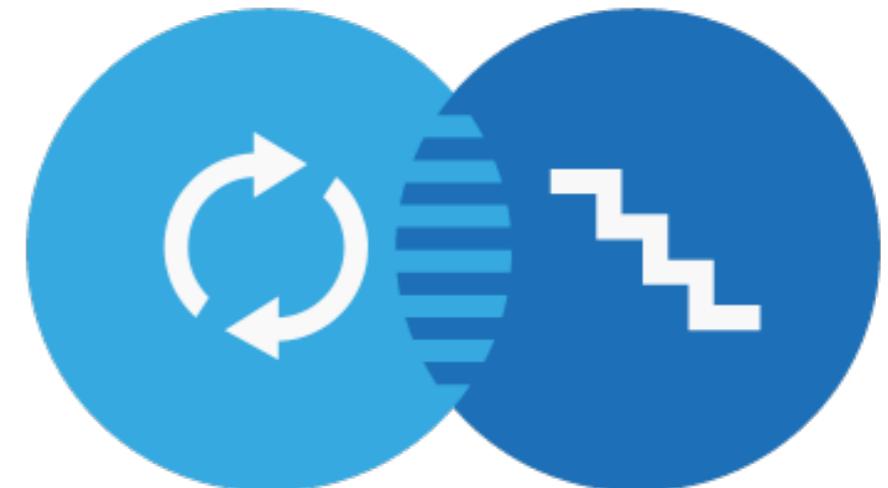
Lineáris és agilis folyamatok

1. Általános megoldások: minden folyamattípusra

2. Lineáris folyamatok, pl. vízesés, általános iteratív

AGILE

WATERFALL



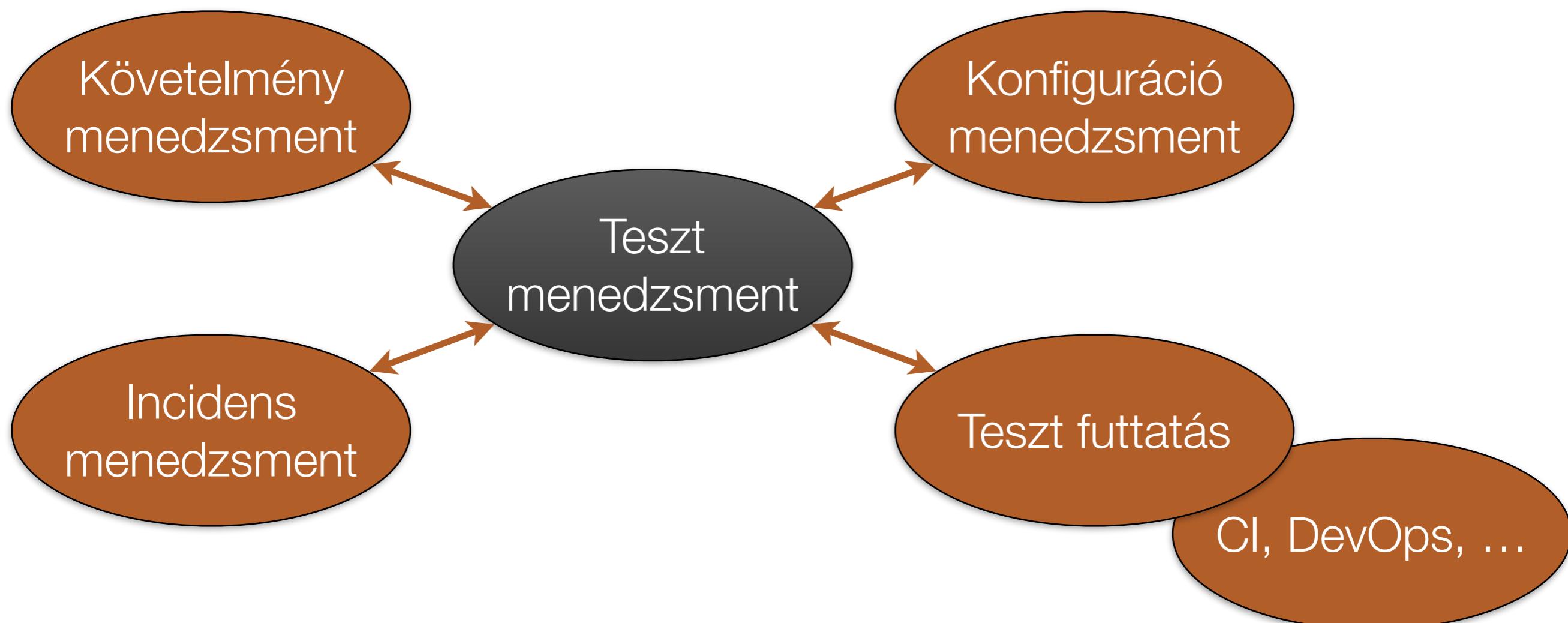
- Klasszikus projekttervezés, pl. gantt
- Hierarchikus projektmenedzsment

3. Agilis folyamatok

- Önszerveződő munkamegosztás, közösségi tudásmegosztás
- Közös felületek, pl. Kanban board, burndown chart, fehér tábla és sticker

Teszt menedzsment támogatás

- „Saját” működésén túl vezérlőegységként funkcionál a különféle eszközök között
 - Külön tool-ok interfészeken keresztül kommunikálnak
 - Integrált rendszerben almodulként működnek



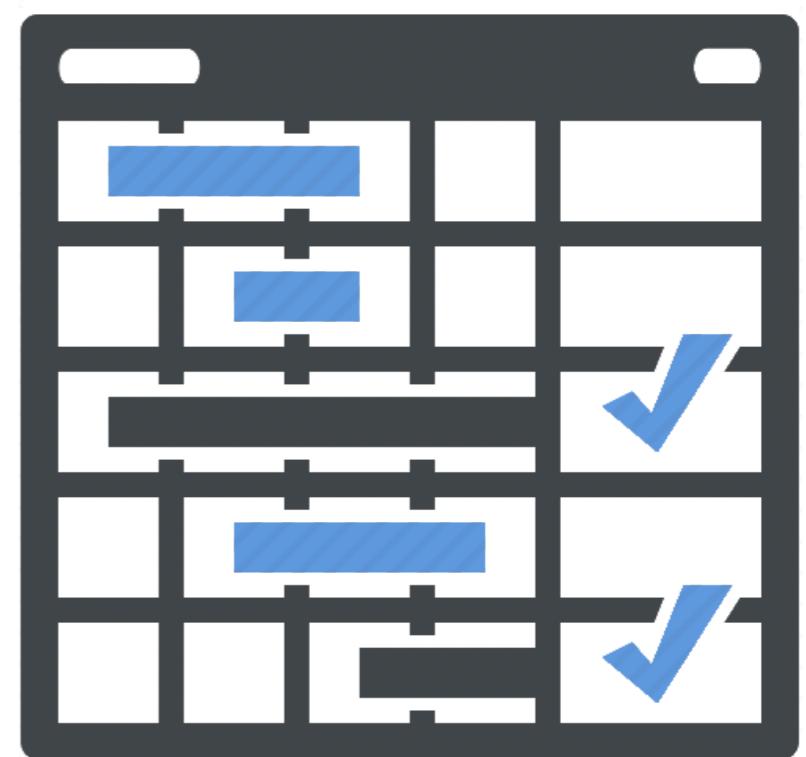
Statikus tesztelés támogatása

Review-k támogatása:

- A review eljárás információinak karbantartása: szabályok és ellenőrző listák
- Defektusok és kommentek rögzítése, közreadása
- Lehetőséget biztosít az átnézendők javítására, változások naplázására
- Átadandók közötti kapcsolatok rögzítése, karbantartása
- Webes elérhetőség a világ bármely pontjáról
- Minél formálisabbak a review-k, annál inkább megéri egy ilyen eszköz használata

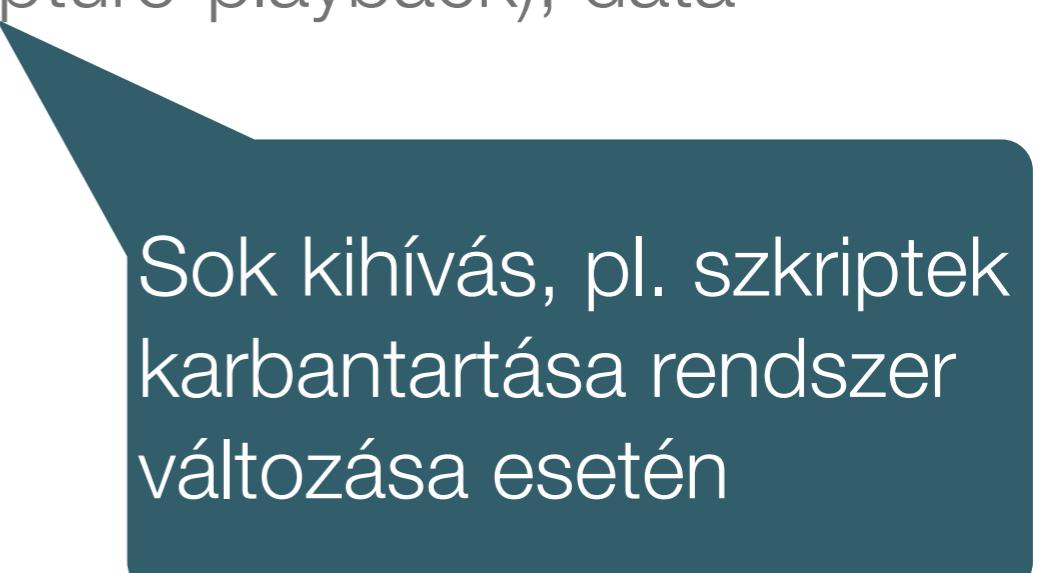
Teszttervezés és megvalósítás

- Teszttervező eszközök
 - Teszteset specifikáció és menedzsment
- Modellező eszközök
 - Modellek készítése, megjelenítése
- Tesztadat-előkészítő eszközök
 - Adatbázisok, fájlok manipulációja, adat-előállítás
- TDD, ATDD és BDD eszközök
 - Keretrendszer, gyakran végrehajtás és naplózás is



Teszt futtatás és naplózás

- Teszt végrehajtó, futtató eszközök
 - Rögzítési, GUI teszt végrehajtás (capture-playback), data driven, keyword driven
 - Lefedettségmérés (kód, követelmény, ...)
 - Nyelvspecifikus megoldások
 - (Egység-) teszt keretrendszerök
 - xUnit, gTest, ...
 - Általában szaktudást igényelnek



Sok kihívás, pl. szkriptek karbantartása rendszer változása esetén

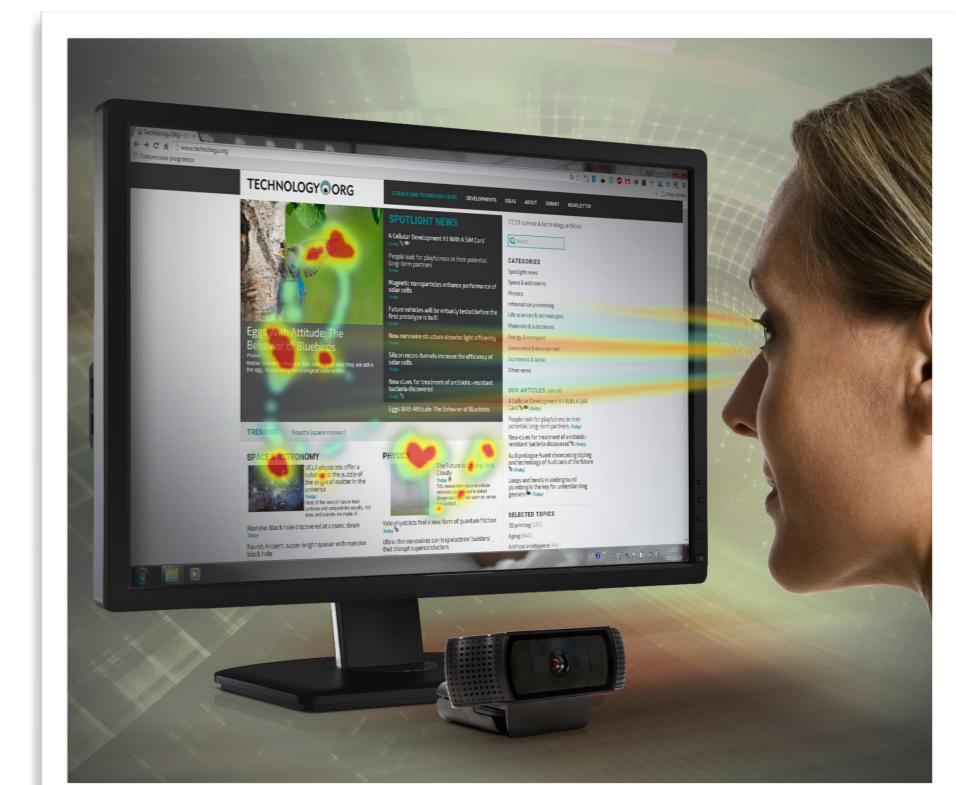
Performancia mérés és dinamikus elemzés

- Teljesítményteszt eszközök
 - Tesztek tömeges, időzített végrehajtása
- Felügyeleti eszközök
 - CPU, memória, háttértár műveletek, hálózati forgalom figyelése, watchdog
- Dinamikus elemzők
 - Profiling, memóriaszivárgás
- E feladatokat általában nehéz vagy lehetetlen kézzel elvégezni



Speciális tesztelési igények kiszolgálása

- Adatminőség-értékelés
- Adatkonverzió, migráció: adatbázis szeletelés, adatmaszkolás
- Használhatósági teszt: kamera, képernyőfelvétel
- Lokalizációs teszt
- Biztonsági teszt: speciális keretrendszerök és eszközök
- Hordozhatósági teszt
- Stb.



Lehetséges előnyök

Ismétlődő tevékenységek csökkennek:

- Regressziós tesztek végrehajtása
- Teszt környezet felállítása, leállítása
- Ugyanazon teszt-adatok újra-felvitele
- Kódolási szabványok ellenőrzése

Objektív kiértékelés:

- Statikus mérőszámok
- Lefedettség mérés
- Jelentéshez, monitorozáshoz

Jobb konzisztencia és megismételhetőség

- Eszközzel ugyanolyan sorrendben ugyanolyan gyakorisággal lehet a teszteket megismételni
- Követelményekből származtatott tesztek

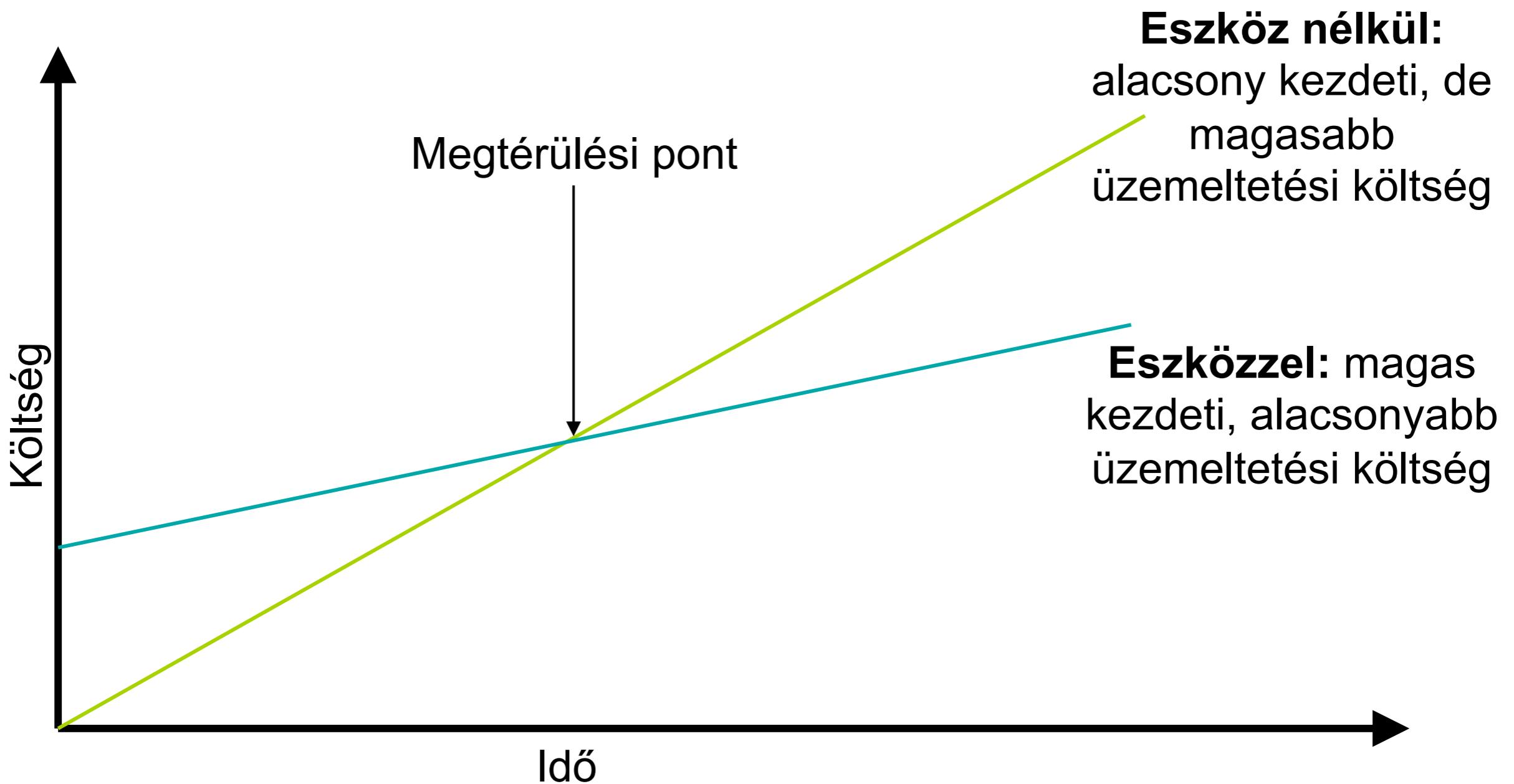
Könnyebb információszerzés a tesztelés folyamatáról

- Statisztikák és grafikonok az előrehaladásról
- Incidens és defektus előfordulási arányok

Lehetséges kockázatok

- Valótlan elvárások, túlzott bizalom az eszközzel szemben
- Bevezetés költségének és időtávlatának alulbecslése
- Bevezetésből származó előnyök alulbecslése
- A bevezetésből származó „javak” karbantartásának alulbecslése
- Az eszköz verzió-követésének figyelmen kívül hagyása
- Egyéb eszközökkel való együttműködés hiánya
- Az eszköz szállítójának megszűnése, átalakulása
- Gyenge követés és támogatás a szállító részéről
- Nyíltkódú, ingyenes eszköz projektjének kockázata
- Előre nem látható kockázatok, pl. új platform támogatottságának hiánya

Az eszközök használatának megtérülése



Automatizáló, teszt végrehajtó eszközök



- Előnyök eléréséhez jelentős befektetésre van szükség
- Az előállít szkriptek instabilak lesznek nem várt eseményekkor
- Technikai tudás szükséges (szkript nyelv ismerete)

- ***Adat-vezérelt tesztelés (data-driven)***

- **Teszt adatok szeparálása az általánosabb teszt szkripttől**
- **Adatok generálása különböző algoritmusokkal**

- ***Kulcsszó alapú teszt (keyword-driven)***

- **A szeparált táblázat kulcsszavakat is tartalmaz az adatok mellett, amelyek azt mondják meg, mit kell csinálni**

- ***Modell-alapú tesztelés (MBT)***

- **Rendszer leírása modell formájában (pl. aktivitás diagram)**
- **Tesztesetek generálhatók belőle**

Teszt menedzsment eszközök



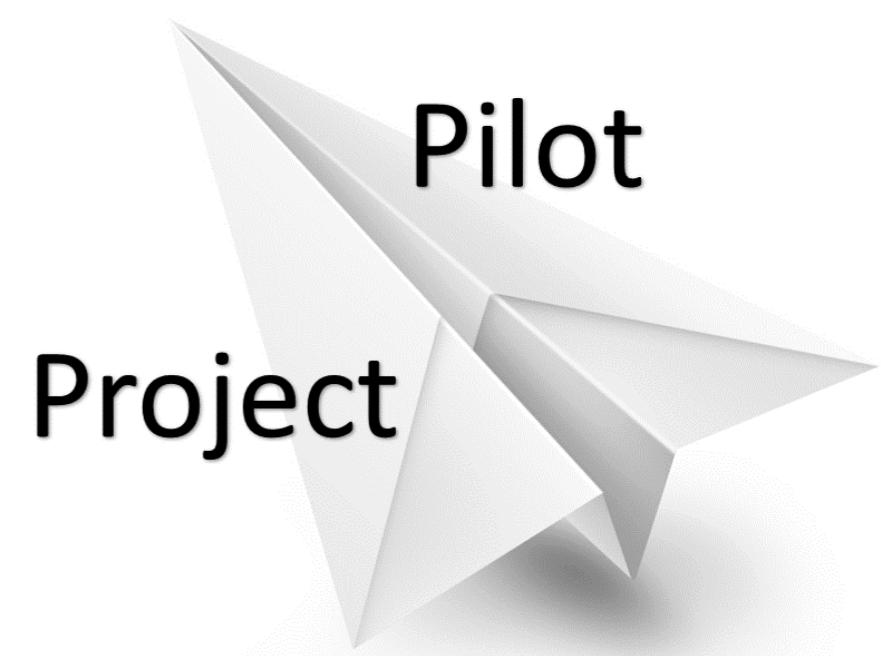
- Szoros integrációra van szükség más eszközökkel
- A szervezet számára megfelelő formátumú és tartalmú információkat kell, hogy szolgáltasson
- Nyomonkövethetőség biztosítása fontos
- Eszköz beszerzése önmagában még nem garancia a hatékonyabb szervezetre!
 - Sokszor folyamat átalakítások kellenek előbb

Bevezetés lépései

- *Analízis:*
 - Elég érett-e a tesztfolyamat eszközök bevezetéséhez?
 - Technológiai elvárások megértése (pl CI)
- *Alternatív megoldások:* már alkalmazott eszközök, folyamat javítása?
- *Követelmények és megkötések:* eszközre és terjesztőre is!
- *Kiértékelés és válogatás:*
 - Többkörös szűkítése a listának
 - Valós környezetben (trial verzió)
- *Három valószínű eredmény:*
 - Egyik eszköz sem éri meg
 - Az egyik eszköz jobb a többinél
 - Több potenciális jelölt is megfelelő
- Tárgyalás a terjesztővel, pl. az árról

A pilot projekt

- Céljai
 - Kell-e változtatni a jelenlegi teszfolyamaton?
 - Az eszköz használatának részletes kidolgozása
 - Számszerűsíteni kell az eszköz hasznát (ROI, *Id. megtérülési pont*)
 - Az eszköz lehetőségeinek, korlátainak kitapasztalása
- Demonstrálja, hogy az eszköz használható
 - „Proof-of-concept”
 - Előnyök és költségek demonstrálása
 - Használat során gyűjtött metrikák megértése



A sikeres bevezetés tényezői



- Pilot projekt tapasztalatainak megvalósítása
- Felhasználói segédlet megírása
- Az eszköz többlépcsős bevezetésére azokra a területekre, ahol a leghatékonyabban használható
- A teszt eljárás és az eszköz használatának szinkronizálása
- Megfelelő oktatás, „ügyfélszolgálat” biztosítása
- Felmerült problémák és megoldásaik feljegyzése, tárolása
- Az eszköz hatékonyságának mérése, megtakarítások
- Technikailag és szervezetileg is integrálva van a fejlesztési környezetbe
- Elegendő erőforrás biztosítása az eszköz megfelelő implementálásához