



MISKOLCI EGYETEM
GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

Szoftvertechnológia gyakorlatok

GEIAL316-B2

Szoftverfejlesztési modellek

Dr. Tompa Tamás

egyetemi adjunktus

Általános Informatikai Intézeti Tanszék

Miskolc, 2025

A szoftverkrízis

- **Szoftverkrízis (NATO-Szoftverfejlesztés Konferencia, 1968)**
 - a szoftver projektek jelentős része sikertelen
 - Sikertelen mert:
 - vagy a tervezettnél drágábban készül el (over budget),
 - vagy a tervezettnél hosszabb idő alatt (over time),
 - vagy nem az igényeknek megfelelő,
 - vagy rossz minőségű / rossz hatásfokú / nehezen karbantartható,
 - vagy anyagi / környezeti / egészségügyi kárhoz vezet,
 - vagy átadásra sem kerül

→ a tesztelés szükségességét a szoftverkrízis húzta alá

→ a tesztelés a minőségi problémákra ad választ, a károkozás megelőzésében segít

Tesztelés a szoftverélelciklus folyamatában

- **Szoftver élelciklusa** (Software Development Life Cycle, SDLC)
 - A szoftver átadása után újabb igények merülnek fel
 - szoftver továbbfejlesztése
 - ciklikusan megújul
 - ez az élelciklus
 - az élelciklus fázisait a **módszertanok** határozzák meg
 - új igény felmerülése
 - első és utolsó lépés → ciklikusság
 - egy hasznos szoftver élelciklusa
 - elvileg végtelen
 - gyakorlatilag előregszik (programozási nyelv, technológia, környezet stb.)

Szoftver életciklus



Módszertanok

- **Feladata**
 - szoftver **életciklus lépéseinek és azok sorrendjének meghatározása**
 - egyfajta szabálykönyv
 - milyen dokumentumokat kell előállítani
 - milyen szoftvert és hogyan stb.
 - a **projekt céljaitól és feladataitól függ** az adott módszertan kiválasztása

- Vízesés modell
- V-modell
- Prototípus modell
- Inkrementális fejlesztés
- Gyors alkalmazásfejlesztés
- Agilis fejlesztés
- Extrém programozás

Vízesés modell

- A szoftverfejlesztés folyamatának első publikált modellje, más tervezői modellekből származik
- egyszerű
- fázisok, eredménye egy dokumentum
- egy fázis csak akkor indulhat, ha az előző befejeződött
- rugalmatlan, korai szakaszokban komoly döntések

- **1. fázis: követelmények elemzése és meghozása**
 - a rendszer felhasználóival való konzultáció alapján kialakul a:
 - rendszer szolgáltatásai,
 - megszorításai,
 - célja

- **2. fázis: rendszer - és szoftverterv**
 - a rendszer átfogó architektúrájának kialakítása

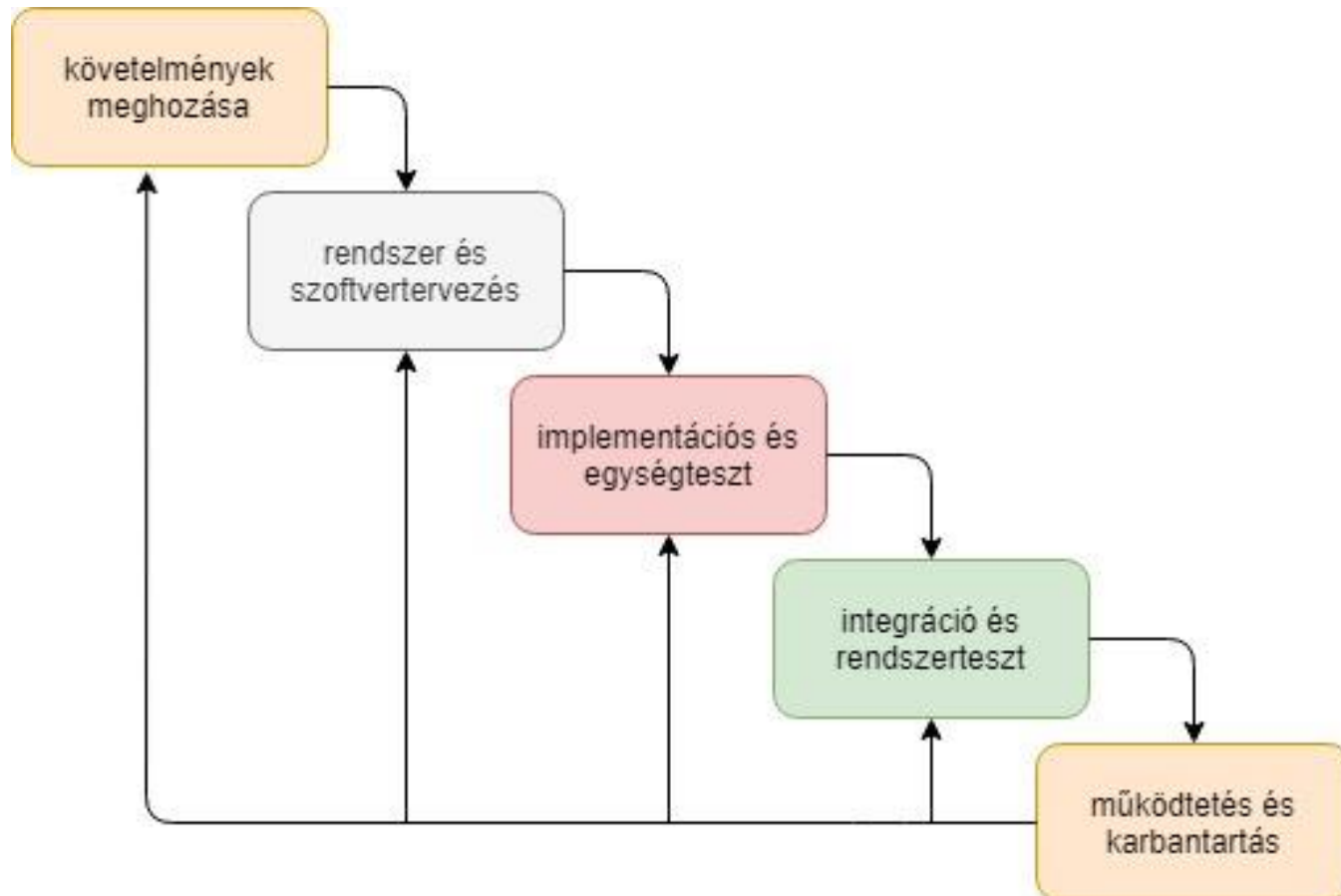
Vízesés modell

- **3. fázis: implementáció és egységteszt**
 - ebben a szakaszban megvalósul a szoftverterv

- **4. fázis: integráció és rendszerteszt**
 - a különálló programegységek, programok integrálása
 - teljes rendszerként való tesztelése

- **5. fázis: működtetés és karbantartás**
 - a később kiderült hibák javítása
 - a rendszeregységek implementációjának továbbfejlesztése
 - új követelmények léphetnek fel, így szükséges lehet a rendszer szolgáltatásainak továbbfejlesztése.

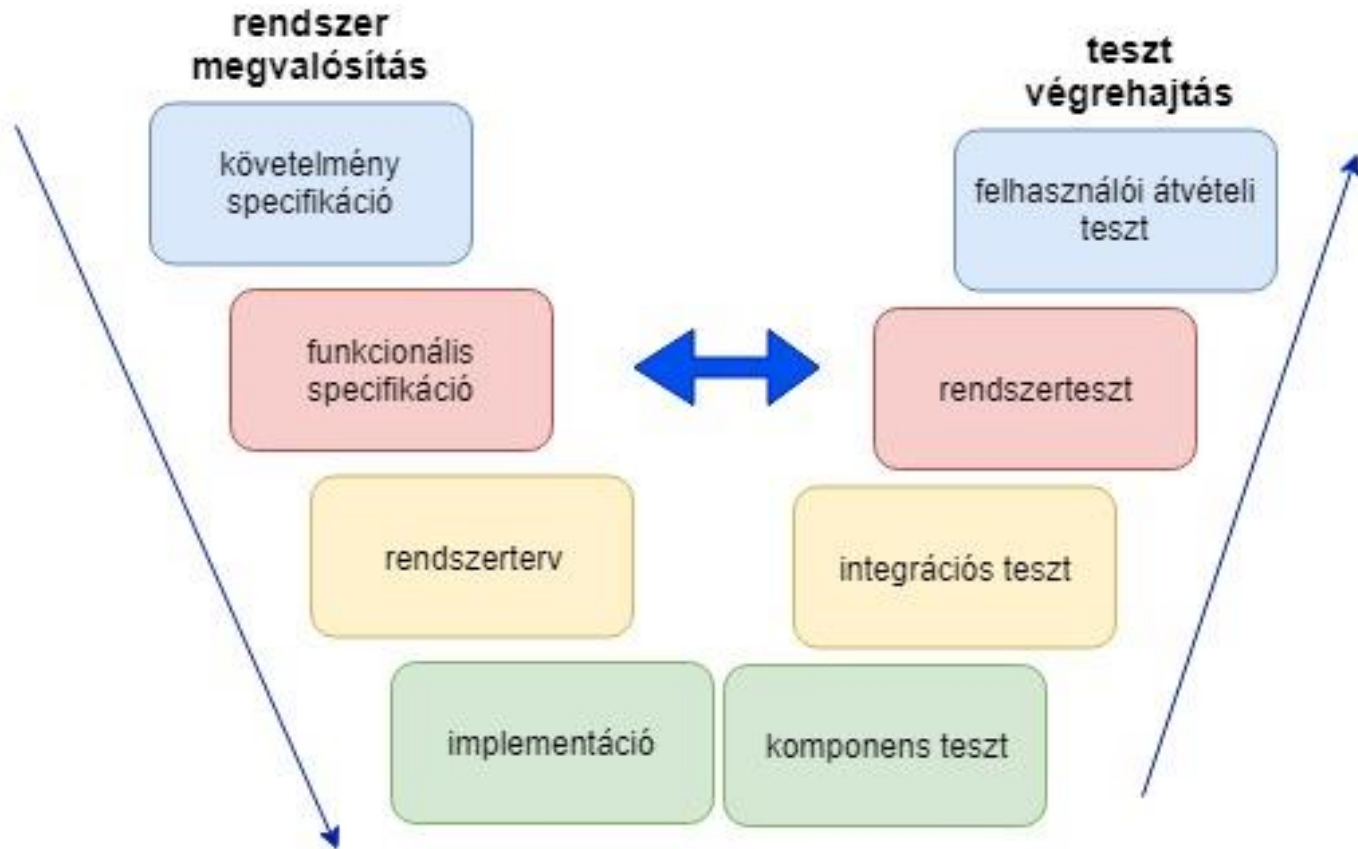
Vízésés modell



V-modell

- Két szár \rightarrow V betűhöz hasonló alak
 - **fejlesztési rész \rightarrow vízesés modell**
 - **+ tesztelési rész**
 - a vízesés modell kiegészítése teszteléssel
 - fejlesztés majd tesztelés
 - ha a tesztelés hibát ad akkor vissza a fejlesztéshez
 - az egy szinten lévő fejlesztési és tesztelési lépések összetartoznak
 - a tesztelési lépés a fejlesztési lépés során létrejött dokumentumokat használja, vagy a létrejött terméket teszteli
- elterjedt de nagyon merev
- ha nem változnak a követelmények a fejlesztés alatt akkor jó módszertan
- ha változnak akkor más módszer inkább \rightarrow iteratív vagy agilis módszertan

V-modell



V-modell

○ Kritikák

- a vízesés modellből ered
- rugalmatlan, képtelen jól alkalmazkodni a változásokhoz
- helytelenül, lineáris képet fest a szoftver fejlesztés folyamatáról
- a tesztelést is rugalmatlan folyamatként kezeli, ahelyett, hogy a tesztelőkre bízná a legjobb módszer felkutatását az adott probléma vizsgálatához
- sok változatának köszönhetően sok a bizonytalanság

V-modell - Teszt

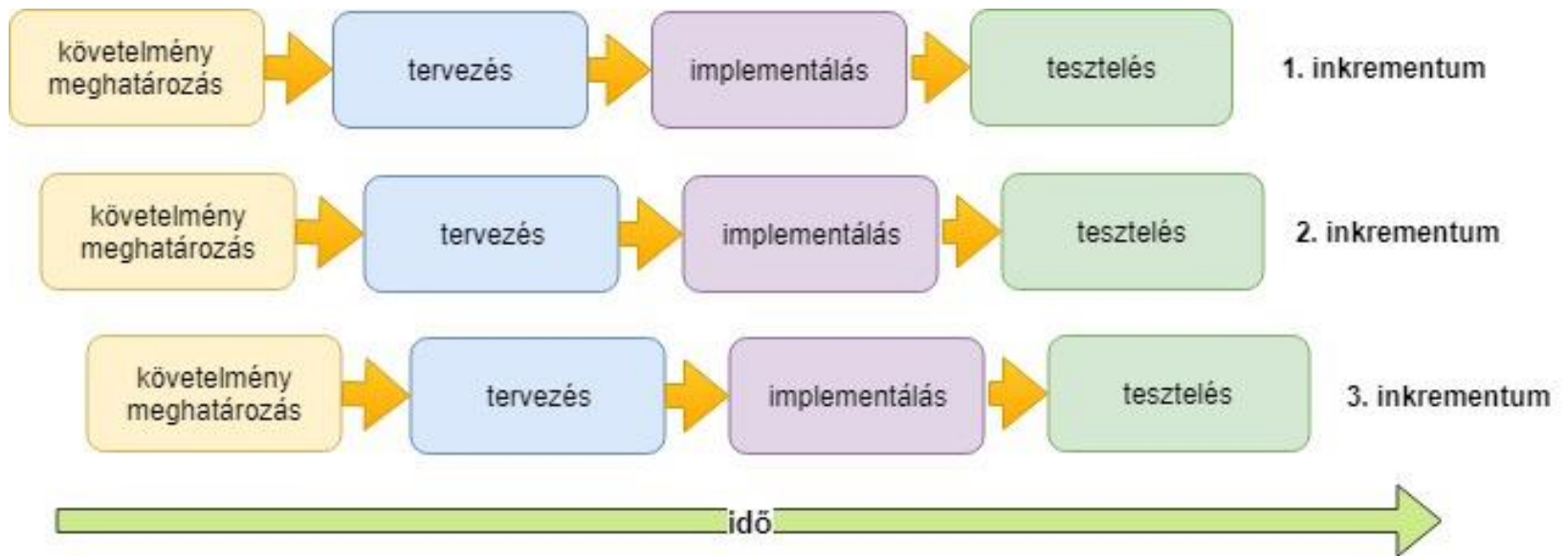
○ **Tesztelés**

- teszt végrehajtása az adott szint megvalósítása után
- de nem jelenti azt, hogy a tesztelők munkája a fejlesztések lezárultával kezdődik
- amint előállnak a funkcionális és nem-funkcionális követelmények, kezdődhet az átvételi tesztesetek és a megfelelőségi elvárások tervezése
- programegységek esetében pedig az interfészeik specifikációja alapján egységtesztek készítése

Iteratív-inkrementális módszertanok

- **a fejlesztés kisebb iterációk sorozata**
- **minden iterációban tervezés és implementáció**
- életciklus lépései nem egymás után jönnek, mint a strukturált módszertanok esetén, hanem időben átfedik egymást
- **folyamatos finomítás**
 - minden iteráció kiegészíti a már kifejlesztett prototípust
- a folyamatra teszik a hangsúlyt azaz az iterációra
- kiegészítés hozzáadásával növekvő részrendszer jön létre, amelyet tesztelni kell
- mai módszertanok nagy része ebbe a családba tartoznak
 - prototípus modell
 - gyors alkalmazásfejlesztés (RAD)
 - Rational Unified Process (RUP)
 - agilis fejlesztési modellek

Iteratív-inkrementális módszertanok



Iteratív-inkrementális módszertanok - Teszt

- **V-Modellhez hasonlóan**
 - az idő elteltével egyre mélyebb szinten a teszttervezési és konkrét tesztelési feladatok
- ellenőrzés, visszajelzés és javaslattétel megjelenése
- tesztelők bevonása be a fejlesztési folyamatba amint lehet
- építés az ellenőrző munkájukra a fejlesztés minden szintjén
- bizonyított megoldásokat hozhatnak a projekt korai fázisaiban
 - tapasztalat alapján
- de: folyamatos változások → rosszul strukturált rendszer

Prototípus modell

○ **Vízesés modell esetében:**

- **a rendszerrel a felhasználó csak a projekt végén találkozik**
- gyakran csak ekkor derült ki, hogy félreértették egymást
- probléma orvosolása → **prototípus modell**

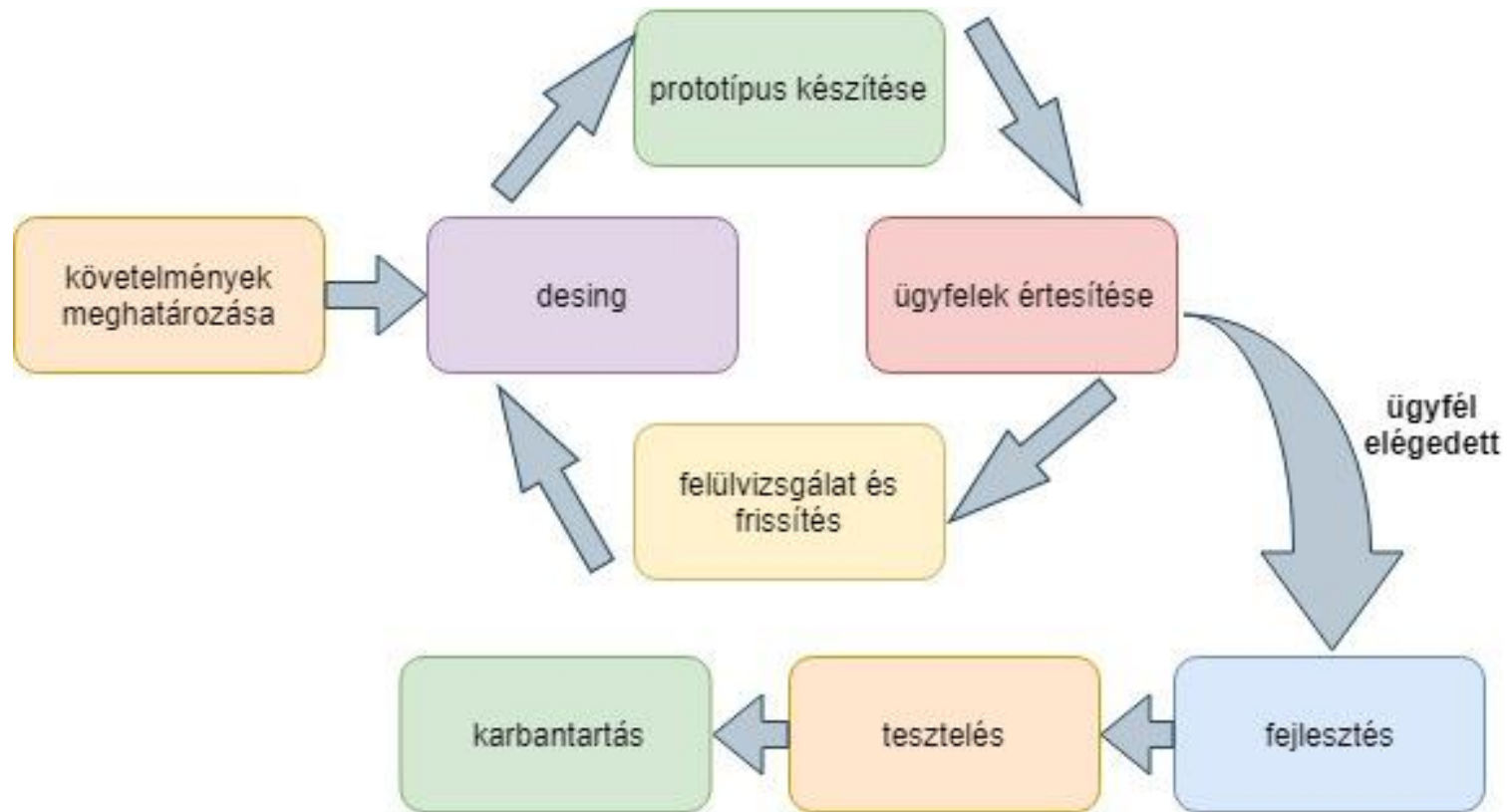
○ **Prototípus modell**

- a megrendelő üzleti folyamatai, követelményei nem ismerhetők meg teljesen
- ezek idővel változnak
- követelmények finomítása prototípusok segítségével
- prototípus használta után annak megfogalmazása, hogy miért nem felel meg a követelményeknek
- ezt követően finomítás
- a rendszer és a felhasználó között sok lesz a párbeszéd

Prototípus modell

- **1. lépés: alapkövetelmények meghatározása**
- **2. lépés: kezdeti prototípus kifejlesztése:** Csak a felhasználói felületeket, a mögötte lévő funkciók nem
- **3. lépés: Bemutatás:** Ez egyfajta felhasználói átvételi teszt.
- **4. lépés. A követelmények pontosítása:** A visszajelzéseket felhasználva pontosítjuk a követelmény specifikációt. Ha még mindig nem elég pontos a specifikáció, akkor a prototípust továbbfejlesztjük és ugrunk a 3. lépésre.

Prototípus modell



Agilis szoftverfejlesztés

- **a jó szoftver előállításának módja (90-es évek nézete)**
 - gondosan tervezett projekt
 - minőséggel szemben támasztott követelmények megismerése
 - elemzési és tervezési módszerek használata
 - irányított, precíz folyamatok
 - → rengeteg többletmunka
 - → több idő a tervezés mint a tényleges fejlesztés és tesztelés
- → **ezen okokból**
 - új, gyors (agilis) módszerek bevezetése
 - **a fejlesztőcsapat a szoftver fejlesztésére koncentrál és nem a tervezésre, dokumentálásra**
 - kulcsszavak: gyors iterációk, ügyfélközpontúság, rugalmasság

Agilis szoftverfejlesztés

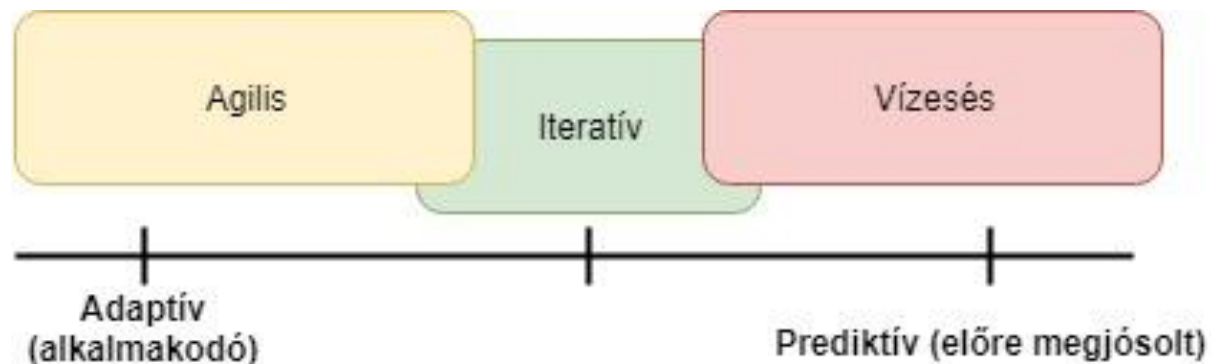
- **iteratív szoftverfejlesztési módszer**
- **2001 (Agile Manifesto kiadvány): alapelvek rögzítése**
- szó jelentése: **fürgeség, gyorsaság**
 - olyan adottság, amely képes reagálni a változásokra
- **résztevők alkalmazkodása a projekthez**
- **Szemlélet szerint:**
 - értékesebbek az **egyének** és interaktivitás szemben a **folyamatokkal** és az eszközökkel
 - értékesebb a **működő szoftver** szemben a terjedelmes **dokumentációval**
 - értékesebb az **együtműködés** a megrendelővel szemben a szerződéses **tárgyalásokkal**
 - értékesebb az **alkalmazkodás** a változásokhoz szemben a **terv követésével**

Agilis szoftverfejlesztés

○ Alapelvek

- legfontosabb a megrendelő igényeinek kielégítése
- a követelmények kései változtatása sem okoz problémát
- működő szoftver / prototípus átadása rendszeresen, a lehető legrövidebb időn belül
- napi együttműködés a megrendelő és a fejlesztők között
- a leghatékonyabb kommunikáció a szemtől-szembeni megbeszélés
- az előrehaladás alapja a működő szoftver
- folyamatos figyelem a technikai kitűnőségnek
- egyszerűség, a minél nagyobb hatékonyságért
- önszervező csapatok készítik a legjobb terveket

Módszertanok - összehasonlítás



- Átfedés a módszerek között
- Agilis ↔ vízésés
- Adaptív
 - hosszútávú tervezés, nincs előre jóslás
 - követlen problémákra koncentrálnak
 - „mit fogunk a héten csinálni”
- Prediktív
 - megtervezett lépések
 - minden lépés az egészre összpontosítva

Agilis szoftverfejlesztés

○ Pl.

- Scrum
- Extrém Programozás (XP)
- Kanban

- Közös jellemzők
 - kevesebb dokumentáció
 - növekvő rugalmasság, csökkenő kockázat
 - könnyebb kommunikáció, javuló együttműködés
 - a megrendelő bevonása a fejlesztésbe
 - hónapok helyett hetek
 - fontosak a határidők
 - nem flexibilisek

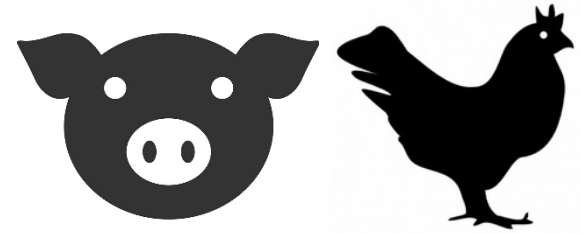
Scrum

- Rögiből átvett kifejezés
 - jelentése: visakodik, összecsap
 - 1990
- megmaradt sport szemlélet
 - Pl. váltófutás
 - stafétát a szoftver
 - a fázisok a futók
 - ha egy futó rosszul teljesít akkor az egész csapat veszít
- Alapgondolat
 - fázisok erősen átlapoltak
 - kisebb csoportok, más területek emberiből
 - az összes fázisban együtt dolgoznak

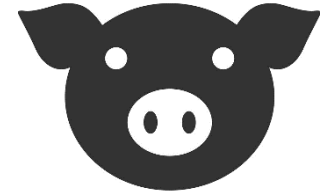
Scrum - jellemzők

- Adaptív tulajdonságok
- Nincs „forgatókönyv”
- A fejlesztő csapat egyszerre kezd el dolgozni
 - a csapat felelős a végeredményért
 - Scrum Team
- Kommunikáció
 - különböző szakterületek
- Átlátható, világos
 - ki, miért és milyen határidővel felelős

Scrum - szerepkörök



- A disznó és a csirke mennek az utcán
- a csirke megszólal:
 - „Te, nyissunk egy éttermet!”
- disznó:
 - „Jó ötlet, mi legyen a neve?”
- a csirke erre gondolkozik, majd azt feleli:
 - „Nevezzük Sonkás-tojásnak!” (Ham and eggs)
- a disznó erre:
 - „Nem tetszik valahogy, mert én biztosan mindent beleadnék, te meg éppen csak hogy részt vennél benne.”

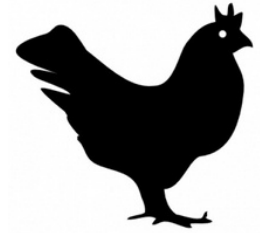


Scrum – „Disznók”

- „vérüket” adják a projekt sikeréért
- elkötelezettek a szoftver projekt sikerében

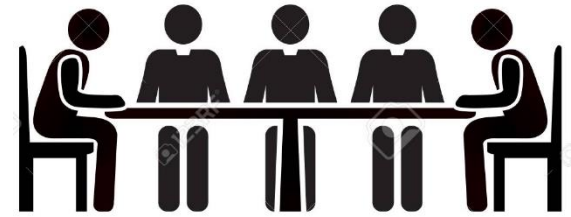
- Disznók
 - Scrum mester (Scrum Master)
 - működtetés, folyamat felügyelése
 - Terméktulajdonos (Product Owner)
 - megrendelő, felelős azért, hogy a csapat mindig azt a részét fejlessze a terméknek, amely éppen a legfontosabb
 - Csapat (Team)
 - 5-9 fős csapat, különböző területekről
 - feladatok megvalósítása
 - csapatjáték fontos → „passzolgatás”

Scrum – „Csirkék”



- közvetetten részei a folyamatnak
- Csirkék
 - Üzleti szereplők (Stakeholders)
 - megrendelők, forgalmazók
 - Menedzsment (Managers)
 - környezet felállítása a csapatok számára
 - megfelelő, legjobb környezet

Scrum – megbeszélések

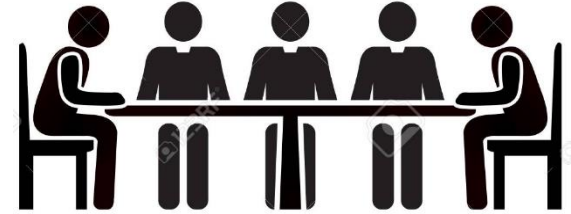


- Sprint Planning Meeting (futamtervező megbeszélés)
 - ki mennyi munkát tud elvállalni, majd ennek tudatában dönti el a csapat, hogy mely sztorikat vállalja be a következő sprintre

- Backlog Grooming/Backlog Refinement
 - a Product Backlog finomítása a Teammel együtt, előfordulhat például, hogy egy taszk túl nagy, így story lesz belőle, és utána taszkokra bontva lesz feldolgozva

- Daily Meeting/Daily Scrum
 - a sprint ideje alatt minden nap kell tartani egy rövid megbeszélést, ami maximum 15 perc
 - Mit csináltál a tegnapi megbeszélés óta?
 - Mit fogsz csinálni a következő megbeszélésig?
 - Milyen akadályokba ütköztél az adott feladat megoldása során?

Scrum – megbeszélések



- Sprint Review Meeting (Futam áttekintés)
 - minden sprint végén összeülnek a szereplők, és megnézik, hogy melyek azok a sztorik, amelyeket sikerült elkészíteni, illetve az megfelel-e a követelményeknek

- Sprint Retrospective (Visszatekintés)
 - az egyik legfontosabb meeting. A Scrum egyik legfontosabb funkciója, hogy felszínre hozza azokat a problémákat, amelyek hátráltatják a fejlesztőket a feladatmegoldásban → alkalmazkodás a feladatokhoz



Scrum – dokumentumok

- Story
 - lényegi leírás a megrendelőktől
- Product backlog (termék teendő lista)
 - Story feldolgozása, priotásokkal
- Sprint backlog (futam teendő lista)
 - a konkrét feladatok feltüntetése az adott sprintre (ki, mit, milyen határidővel vállalt be)
- Burn down chart (Napi Eredmény Kimutatás)
 - diagram, amely segít megmutatni, hogy az ideális munkatempóhoz képest hogyan halad a csapat az aktuális sprinten belül

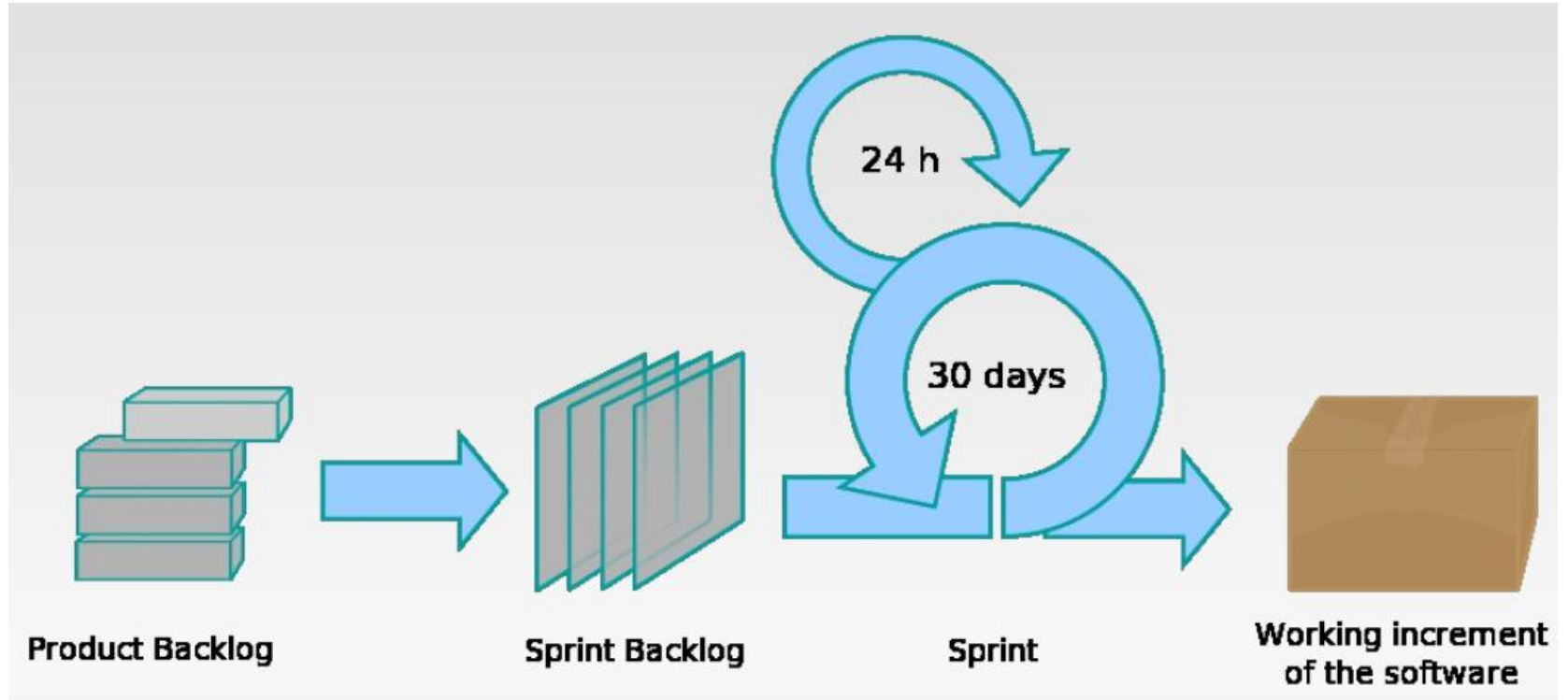


Scrum – fogalmak

- Sprint (futam)
 - előre megbeszélte hosszúságú fejlesztési időszak, általában 2-4 hétig tart
 - iterációs ciklus → addig kell ismételni amíg el nem tűnnek a megoldásra váró felhasználói szotrik
 - minden sprint végére leszállítható szoftver előállítása

- Akadály (Impediment)
 - olyan gátló tényező, amely a munkát hátráltatja
 - csak és kizárólag munkahelyi probléma tekinthető akadálnak (a csapattagok magánéleti problémái nem azok)
 - akadály például, hogy lejárt az egyik szoftver licence
 - a Scrum Masternek kell elhárítani az akadályokat

Scrum



Extrém programozás



- Extreme Programming, XP
- agilis módszertan
- az eddigi módszertanokból átveszi a jól bevált technikákat
 - és azokat nem csak jól, hanem extrém jól alkalmazza
 - majd minden mást feleslegesnek tekint
- ≠ „programozunk összeesésig” módszerrel
 - 24 órás vagy akár 48 órás programozó verseny

Extrém programozás



○ 4 tevékenység

● Kódolás

- legfontosabb, előjönnek a nehézségek, kommunikációs a fejlesztők között → mindenki ugyanazt érti alatta

● Tesztelés

- addig nem lehetünk benne biztosak, hogy egy funkció működik, amíg nem teszteltük. Az extrém felfogás szerint kevés tesztelés kevés hibát talál, extrém sok tesztelés megtalálja mind

Extrém programozás



○ 4 tevékenység

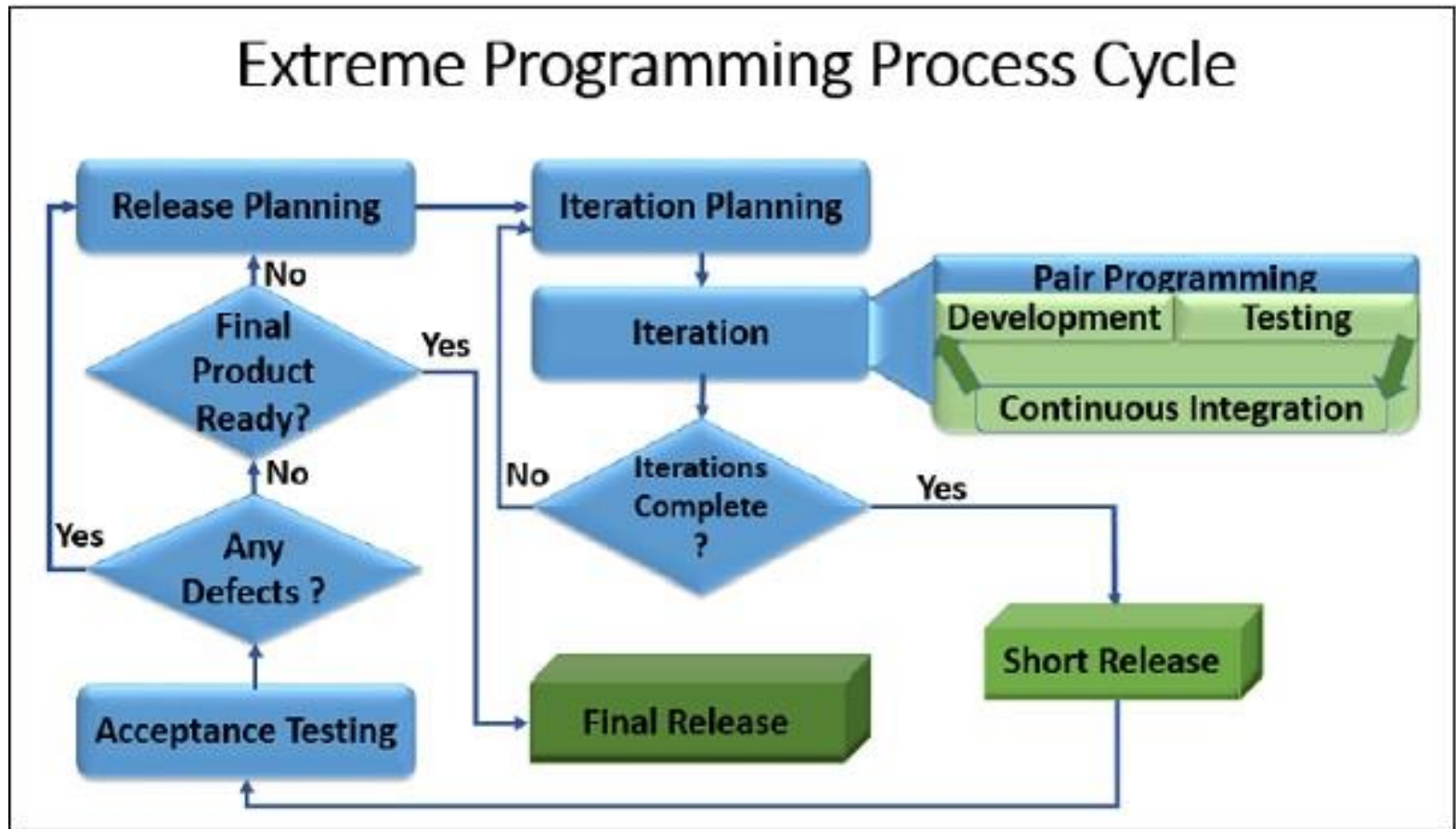
● Odafigyelés

- a fejlesztőknek oda kell figyelniük a megrendelőkre, meg kell érteniük az igényeiket

● Tervezés

- tervezés nélkül nem lehet szoftvert fejleszteni, mert az ad-hoc megoldások átláthatatlan struktúrához vezetnek. Mivel fel kell készülni az igények változására, ezért úgy kell megtervezni a szoftvert, hogy egyes komponensei amennyire csak lehet függetlenek legyenek a többitől.

Extrém programozás



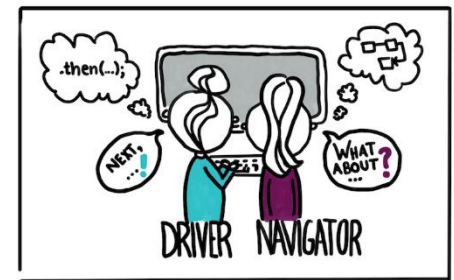
Extrém programozás



○ Jellemző technikák

- **Páros programozás (pair programming)**
 - ketten egy kódot, ír ↔ figyel, ha hibát lát szól
- **Teszt vezérelt fejlesztés (test driven development)**
 - először egységteszt írás majd metódus implementálás
- **Forráskód átnézés (code review)**
 - vezető fejlesztő átnézi a kódot → hogyan kell jobban csinálni
- **Folyamatos integráció (continuous integration)**
 - verziókövető rendszerbe bekerült kódok integrációs tesztje
- **Kódszépítés (refactoring)**
 - a már letesztelt, működő kódot lehet szépíteni, de funkcionalitás nem változhat

Extrém programozás

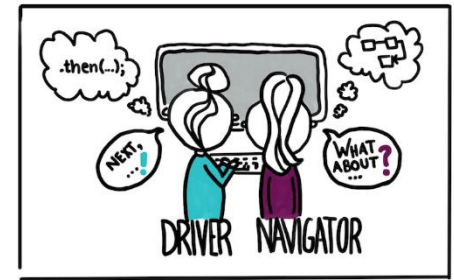


○ Jellemző technikák

● Páros programozás (pair programming)

- ketten közösen írnak egy kódot
 - ír ↔ figyel, ha hibát lát szól
- két szerep:
 - Vezető (driver): ő írja a kódot
 - Navigátor (navigator): figyeli a kódot, hibákat keres, és stratégiát javasol
- folyamatos kommunikáció: a fejlesztők folyamatosan megbeszélik az esetleges problémákat

Extrém programozás



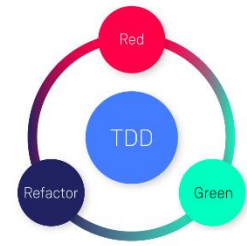
○ Jellemző technikák

● Páros programozás (pair programming)

○ Következmények:

- **jobb minőségű kód:** a valós idejű ellenőrzés csökkenti a hibák számát
- **tudásmegosztás:** fejleszti a csapat tudását és tapasztalatát
- **gyorsabb problémamegoldás:** két ember könnyebben találhat optimális megoldást
- **de mentálisan fárasztóbb:** intenzív koncentrációt igényel, ezért rendszeres szünetek ajánlottak
- alkalmazása hasznos komplex problémák megoldására és junior fejlesztők mentorálására

Extrém programozás



○ Jellemző technikák

● Teszt vezérelt fejlesztés (test driven development)

- először egységteszt írás majd metódus implementálás
- → a kódolás tesztek írásával kezdődik
- Red-Green-Refactor ciklus:
 - red: írd meg a tesztet, ami először sikertelen lesz
 - green: írd meg a minimális kódot a teszt sikeressé tételéhez
 - refactor: optimalizáld a kódot a funkcionalitás megőrzése mellett
- folyamatos visszacsatolás: a tesztek azonnali visszajelzést adnak a kód minőségéről
- jobb kódminőség: modulárisabb és karbantarthatóbb kódot eredményez
- kevesebb hiba: a fejlesztés korai szakaszában kiszűri a hibákat⁴¹

Extrém programozás



○ Jellemző technikák

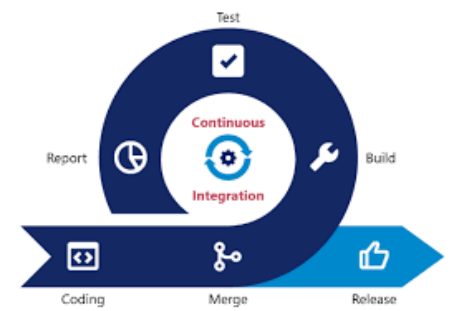
● Forráskód átnézés (code review)

- vezető fejlesztő átnézi a kódot → hogyan kell/lehet jobban csinálni
- a csapattagok is ellenőrizhetik egymás kódját

○ Céljai:

- hibák és biztonsági rések kiszűrése
- kódminőség és olvashatóság javítása
- legjobb gyakorlatok és konvenciók betartása („mi így szoktuk”)
- kis méretű, könnyen átlátható módosítások ellenőrzése
- konkrét és építő jellegű visszajelzések adása

Extrém programozás

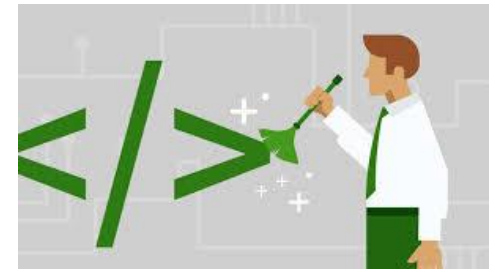


○ Jellemző technikák

● Folyamatos integráció (continuous integration, CI)

- verziókövető rendszerbe bekerült kódok integrációs tesztje
- kódmódosítások egy közös kódbázisban, ahol automatikus tesztek és ellenőrzések
- automatizált build és tesztelés: a kód minden módosítása után automatikusan lefutnak a tesztek és lefordul a kód (ha le tud)
- CI eszközök: Jenkins, GitHub Actions, GitLab CI/CD, CircleCI
- előkészítés a folyamatos szállításra (CD): a CI az alapja a folyamatos szállításnak (Continuous Delivery, CD), kódmódosítások automatikus előkészítése a telepítésre

Extrém programozás



○ Jellemző technikák

● Kódszépítés (refactoring)

- a már letesztelt, működő kódot lehet szépíteni, de funkcionalitás nem változhat
- meglévő kód átalakítása anélkül, hogy annak funkcionalitása megváltozna
- célja a kód olvashatóságának, karbantarthatóságának és hatékonyságának javítása
- nincs új funkció: a refaktorálás nem változtatja meg a kód viselkedését, csak a szerkezetét javítja.
- jobb olvashatóság: tisztább, érthetőbb kódot eredményez
- karbantarthatóság növelése: könnyebb a jövőbeni módosítások és hibajavítások elvégzése
- kódduplikáció csökkentése: a redundáns kódrészek eltávolítása

Kanban



- **Vizuális alapú, folyamatos fejlesztési (Lean, Agile) módszer**
- Csoportos munkák menedzselése
- Jellemzők:
 - Folyamatos fejlesztés (Continuous Delivery)
 - nincs előre meghatározott iteráció, a munka folyamatosan áramlik
 - Vizualizáció
 - a feladatokat egy Kanban táblán kezelik, amely oszlopokra van bontva (pl. „Teendő”, „Folyamatban”, „Kész”)
 - WIP (Work In Progress) korlátozás
 - meghatározza, hány feladat lehet egyszerre egy adott szakaszban, így csökkenti a torlódást

Kanban



○ Jellemzők:

- Pull-rendszer
 - a csapatok húzzák a feladatokat, amikor készen állnak rájuk, nem pedig előre kiosztják őket
- Mérhető teljesítmény
 - Pl. leadási idő (cycle time), átfutási idő (lead time) alapján optimalizálható a folyamat

○ Alkalmazása

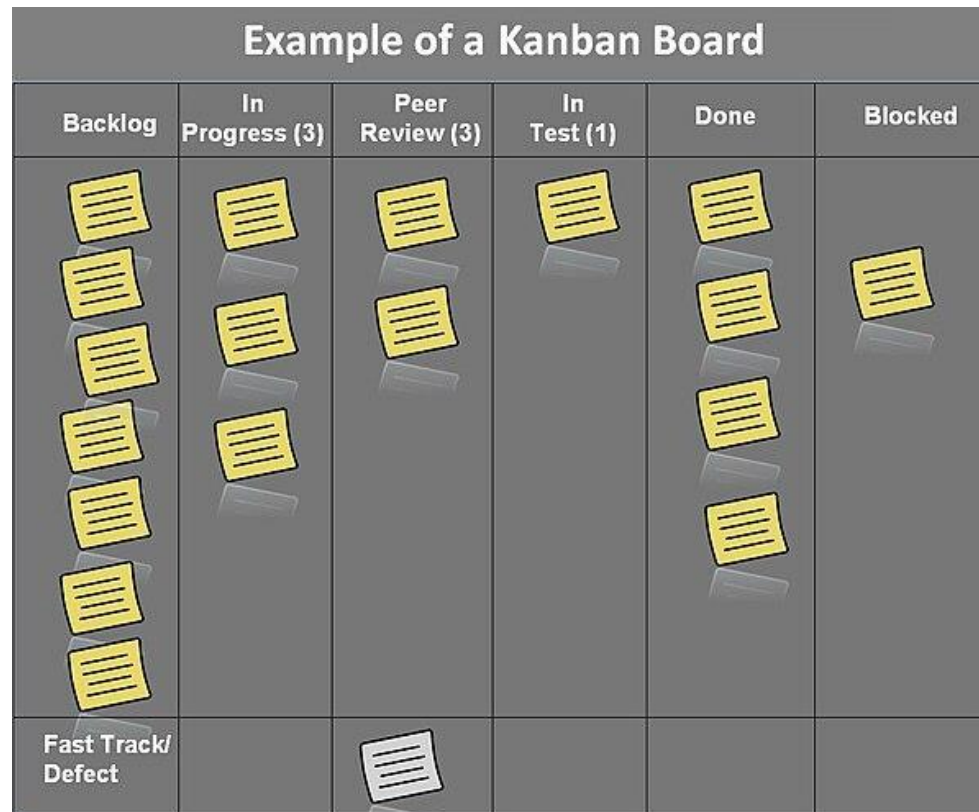
- ha folyamatos fejlesztésre van szükség
- ha vizuális átláthatóság kell a feladatok kezeléséhez
- ha a munkaterhelés változó, és rugalmasabb folyamat kell, mint a Scrum sprintjei

Kanban



○ Kanban tábla

- a csapat a feladatokat balról jobbra mozgatja a folyamat előrehaladtával



Scrum vs. Kanban

Jellemzők	Kanban	Scrum
Iteráció hossza	folyamatos	1-4 hetes sprint
Munka menedzselése	pull-alapú	sprint backlog
Prioritás változás	rugalmas	sprint közben nehézkes
Szerepkörök	nem kötöttek	kötöttek

Lean



- Toyota **Lean gyártási filozófiájából** ered
- Célja a
 - szoftverfejlesztési folyamat hatékonyságának növelése
 - felesleges folyamatok kiküszöbölése, pazarlás minimalizálása
 - ügyfélérték maximalizálása
 - Toyota, Amazon, Google
- Fő elvek
 - **Pazarlás csökkentése (eliminate waste)**
 - felesleges munka megszüntetése
 - felesleges kód és funkciók (pl. amit az ügyfél nem használ)
 - túl sok dokumentáció, amit senki nem olvas
 - hosszú várakozási idők (pl. tesztelés, jóváhagyások)
 - kommunikációs problémák a csapatok között

Lean



○ Fő elvek

- **Gyors visszajelzés és döntéshozatal (create knowledge)**
 - folyamatos visszacsatolás az ügyfelektől és a csapaton belül
 - a fejlesztőknek önálló döntéshozatali lehetőségeik vannak
 - az MVP (Minimum Viable Product) elv használata: először egy minimális működő verziót építenek
- **Minőség beépítése a fejlesztésbe (best quality in)**
 - automatizált tesztelés és tesztvezérelt fejlesztés (TDD)
 - hibák javítása már a fejlesztési fázisban, nem utólag
 - kódatvizsgálás (Code Review) a problémák korai észlelésére

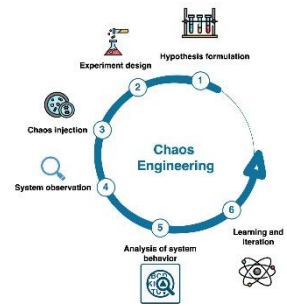
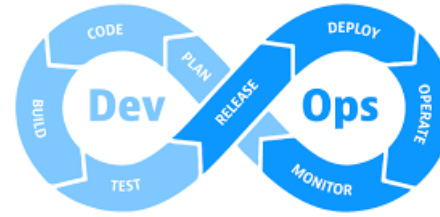
Lean



○ Fő elvek

- **Késleltetett döntéshozatal (delayed in decision making)**
 - nem hoznak elhamarkodott döntéseket, hanem gyűjtik az adatokat, és csak akkor döntenek, amikor elegendő információ áll rendelkezésre
 - nem kötik le az erőforrásokat túl korán
- **A csapat tisztelete (empower your team)**
 - az alkalmazottak autonómiát kapnak, hogy saját döntéseket hozzanak
 - önszerveződő csapatok, amelyek gyorsan reagálnak a változásokra
- **Egész rendszer optimalizálása (optimize the whole)**
 - nem csak a fejlesztésre, hanem az üzleti célokra és a teljes folyamatra figyelnek
 - a teljes szervezet működését optimalizálják, nem csak az egyes részfolyamatokat

További modellek



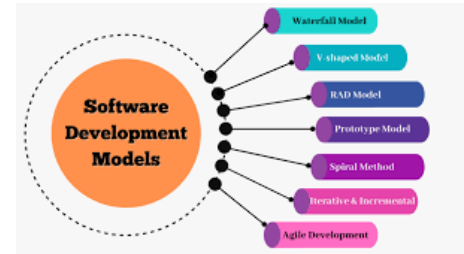
○ DevOps

- **Development + Operations**
- célja a fejlesztési (Dev) és üzemeltetési (Ops) csapatok összehangolása
- a fejlesztők és az üzemeltetők együtt dolgoznak, automatizált eszközöket használnak a folyamatok gyorsítására

○ Chaos Engineering

- **szimulált meghibásodásokkal teszteli a rendszerek ellenállóképességét**
- „Mi történik, ha egy szerver leáll? Mi van, ha az adatbázis hirtelen elérhetetlenné válik?”
- ahelyett, hogy ezt a való életben kellene kideríteni, **mesterségesen idéznek elő meghibásodásokat**, hogy felkészítsék a rendszert az ilyen helyzetekre

Gyakorlati alkalmazás



- **A gyakorlatban ezen módszertanok alkalmazása keveredik**
 - ha gyorsan kell reagálni a változásokra → Agilis
 - ha folyamatos frissítés kell → CI/CD, DevOps
 - ha nagy, skálázható rendszer → Mikroszerviz
 - ha fontos a biztonság → DevSecOps
 - ha a hatékonysága fő → Lean fejlesztés
- **Néhány nagy cég**
 - Google: agilis, DevOps, CI/CD
 - Amazon: DevOps, CI/CD, Lean
 - Microsoft: DevOps, CI/CD, Agilis, Lean,
 - Facebook: CI/CD, DevOps, Agilis, Mikroszervíz
 - Tesla: Agilis, DevOps, Lean, TDD

Szoftver etikai kódexe



- A szoftverek tesztje során a résztvevők bizalmas információkhoz juthatnak hozzá
 - az etikai kódexre azért van szükség, hogy az információkat ne használják fel illetéktelenül
- A kódex pontjai a következők:
 - **Közérdek**
 - a képezett tesztelőknek következetesen a közérdeknek megfelelően kell tevékenykedniük

Szoftver etikai kódexe



○ **Megrendelő és munkaadó**

- a képezett szoftvertesztelőknek úgy kell tevékenykedniük, hogy a megrendelőik, illetve munkaadóik igényeit legjobban kiszolgálják, ugyanakkor a közérdekkel ne kerüljenek szembe

○ **Termék**

- a képezett szoftvertesztelőknek biztosítaniuk kell, hogy az általuk tesztelt, átadásra kerülő termék, vagy rendszer megfelel a legmagasabb szakmai szabványoknak

○ **Vélemény**

- a képezett szoftvertesztelőknek feddhetetlennek és függetlennek kell maradniuk a szakmai véleményalkotáskor

Szoftver etikai kódexe



○ Menedzsment

- a képezett szoftver tesztmenedzsereknek és vezetőknek azonosulniuk és támogatniuk kell az etikai kódexet a szoftvertesztelés menedzsmentje felé

○ Szakma

- a képezett szoftvertesztelőknek elő kell segíteniük a szakma hírnevét és feddhetetlenségét a közérdeknek megfelelően

Szoftver etikai kódexe



○ Munkatársak

- a képezett szoftvertesztelőknek korrektül és támogatóan kell fellépni a munkatársaikkal szemben és segíteniük kell a szoftverfejlesztőkkel való együttműködést

○ Személyes

- a képezett szoftvertesztelőknek életük végéig tanulniuk kell a szakmájukban és munkájuk végzése során figyelniük kell arra, hogy az etikai kódex a munkájuk részévé váljon



Köszönöm a figyelmet!

thank you 😊