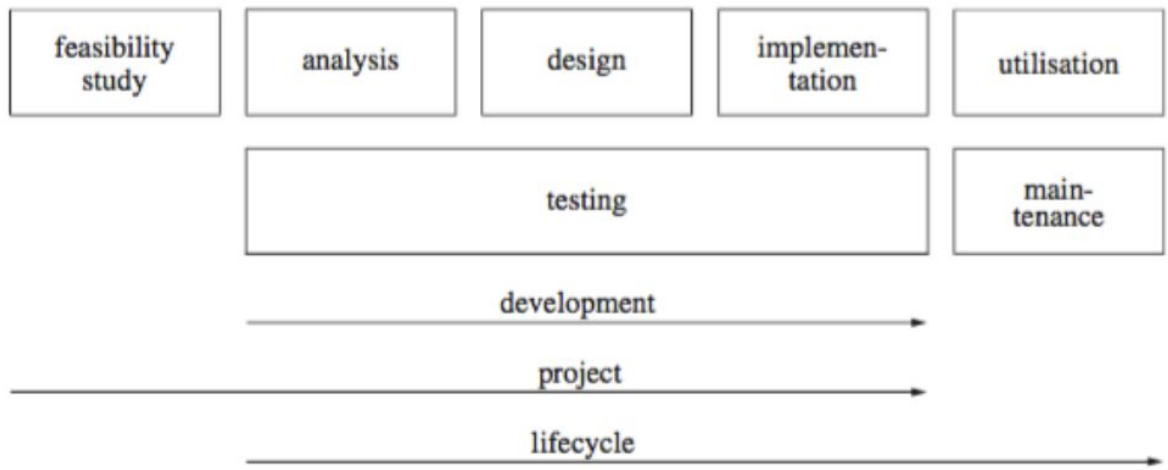
Szoftverfejlesztési módszerek és paradigmák 2016/2017 2. félév

Fejlesztési módszertanok

1. A szoftverfejlesztési folyamat tevékenységei. Foglald össze és röviden mutasd be a szoftverfejlesztési aktivitásokat!

- Követelményelemzés:
    - felderítés, egyeztetés, priorizálás, dokumentálás
    - természetes nyelven, formális modellek formájában
    - eredménye képezi a projekt alapját
- Specifikáció készítés
- Tervezés
- Implementálás
- Tesztelés
- Debuggolás
- Telepítés / Üzembe helyezés
- Karbantartás



2. Mutasd be a szoftverrendszerek, alkalmazások értékeit!

- Karbantarthatóság
- Folyamatos továbbfejleszthetőség és folyamatos szállítás képessége
- Új szoftverrendszerek fő jellemzője: folyamatosan változó követelmények
- Értékek a fejlesztésben
    - Csapatmunka
    - Kódminőség
    - Szoftver életciklus, fenntartható fejlesztési folyamat

3. Mutasd be a szoftverprojektek életciklusának elemeit, azok jellemzőit!

(Minden, ami az egyes kérdésben szerepel a lifecyclehöz)

4. Mutasd be a szoftverfejlesztési folyamat lépéseit, röviden elemezd őket!

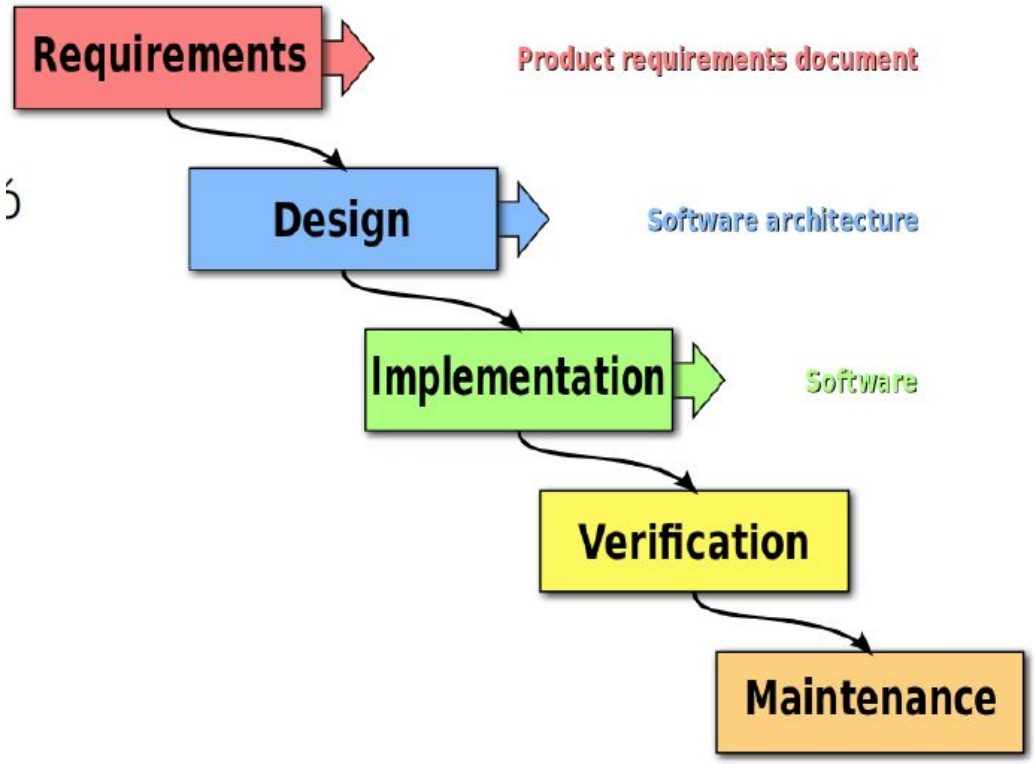
(Minden, ami az egyes kérdésben szerepel a developmenthez)

5. Mutasd be a szoftvertermék elemeit!

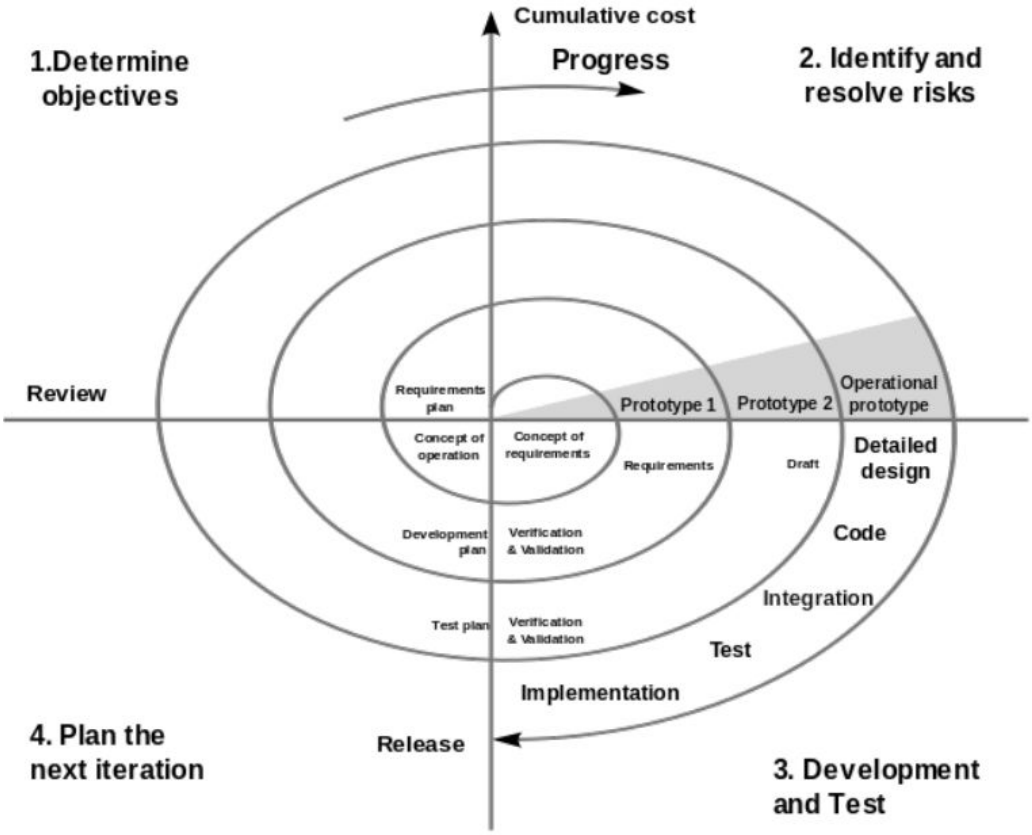
- A szoftverrendszer jól azonosítható és koherens komponensek halmaza, melyek egymással szorosan együttműködnek egy adott cél elérése érdekében.
- A szoftvertermék elemei:
    - alkalmazások, melyek a termék funkcionalitását kínálják;
    - adatstruktúrák, melyek az alkalmazások futásához szükséges információkat teszik elérhetővé
    - dokumentáció, amely összefoglalja hogyan installáljuk, használjuk és tartsuk karban az alkalmazásokat programokat.

6. Hasonlítsd össze a vízesés, a spirál és az iteratív fejlesztési módszereket!

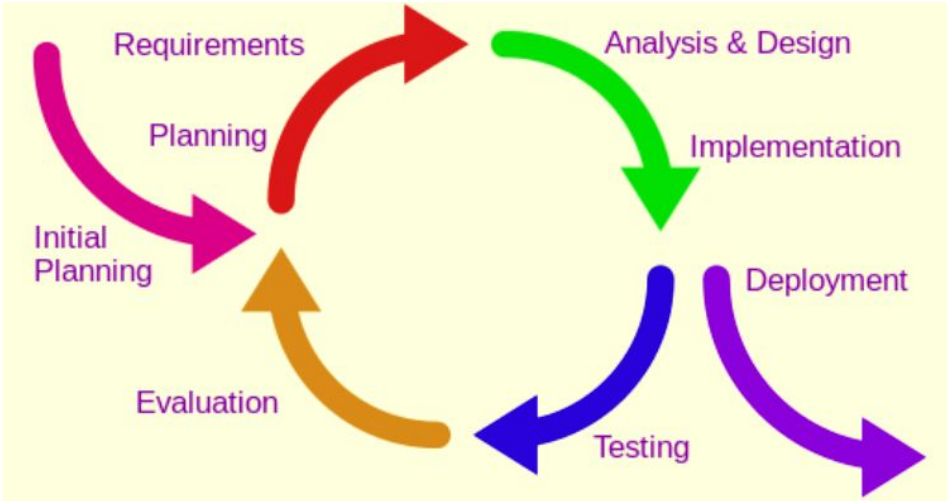
- **Vízesés modell:** Az elmúlt 50 év leggyakrabban használt modellje. Lépései: **Követelményspecifikáció** (túl korai és hosszú dokumentum, ebből lesz minden probléma)**, Szoftvertervezés, Megvalósítás és integráció, Tesztelés és ellenőrzés, Telepítés, Karbantartás**
    - Nem az készül el, amire szükség van, mivel gyakran nem kapunk visszajelzéseket a termékről addig, amíg az el nem készült
    - A folyamatosan megjelenő új követelmények koncepcionális változásokat indukálnak, ami újratervezést igényel
    - A tervezők gyakran nincsenek tisztában az implementációs részletekkel
    - A tervező eszközök nem integrálódnak jól az implementációt segítő eszközökkel
    - Túl hosszú lehet a teljes fejlesztési fázis



- **Evolúciós modellek:** Több ciklus, mindegyik a termék egy részéért felel: ez lehetővé teszi a követelmények módosítását
    - Rövidebb ciklusok csökkentik a hibás ágak scope-ját, rövidebb visszajelzési ciklus lehetőséget ad a rendszeres beavatkozásra, mindez segíti a folyamatosan irányban tartást, a végső sikerhez vezető közvetlen úton járást
    - Ilyenek pl. a spirál és a különböző iteratív fejlesztési módszerek
- **Spirál:** Négy fő aktivitást ismétel ciklikusan (spirál formájában): **Tervek kialakítása, Kockázatelemzés, Implementáció, Következő iteráció tervezése**
    - Kockázat menedzsment a fejlesztés megfelelő fázisában: a projekt egyedi kockázatai alapján veszi át más folyamatmodellek elemeit (inkrementális, vízesés, evolúciós modellek)



Iteratív és inkrementális fejlesztés: A rendszer ismétlődő ciklusok (iteratív) és egy időben kisebb részek kidolgozásával (inkrementálisan) fejlődik. Minden iterációban frissül a terv és új funkciók kerülnek kidolgozásra.



7. Mutasd be az iteratív és inkrementális fejlesztési módszerek jellemzőit!

(Minden, ami az előző kérdésben elhangzott ehhez és az evolúciós modellekhez)

8. Sorolj fel és röviden mutass be szoftverfejlesztési módszerekhez kapcsolódó legjobb gyakorlatokat!

- RUP - IBM Rational Unified Process (RUP): Legjobb gyakorlatok átgondolt gyűjteménye:
    - Több ezer projekt alapján
    - Használjuk a mások által sikerrel alkalmazott módszereket, folyamatokat
    - Projekt sablonok (felépítés, erőforrások, mérföldkövek, leszállítandók), dokumentumok

A 6 best practice:

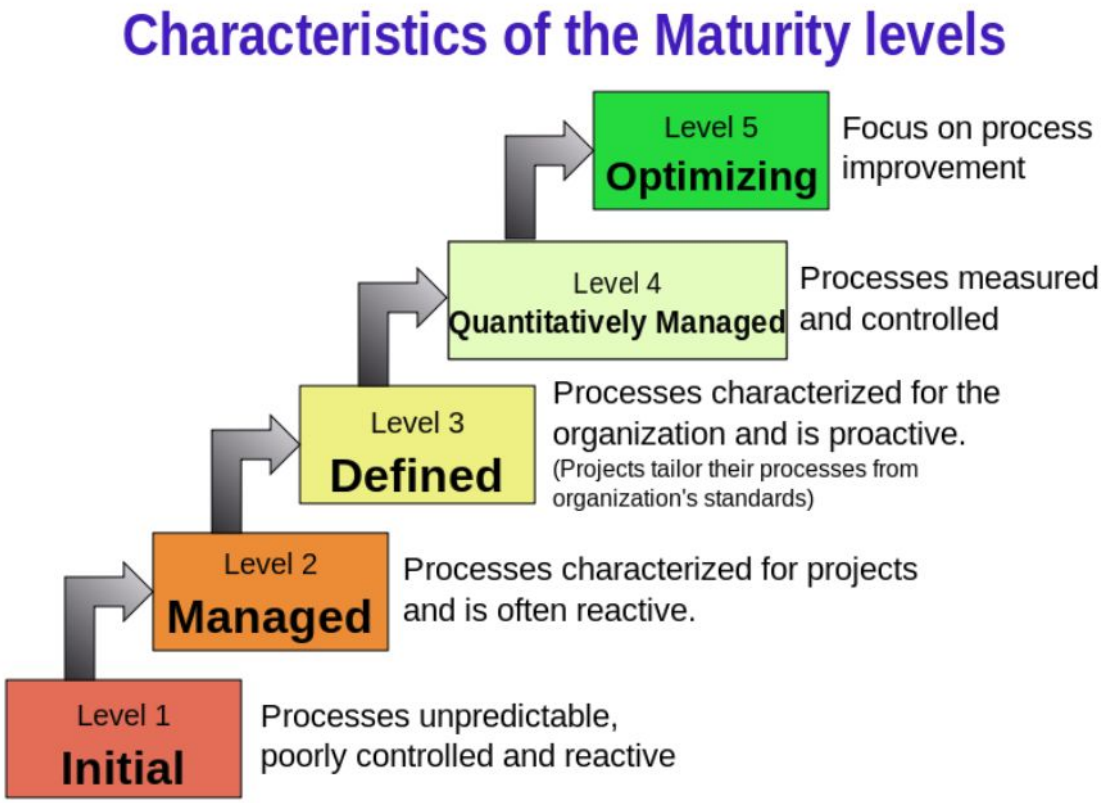
- **Develop iteratively (Fejlesszünk iteratívan):** A legjobb minden követelményt előre ismerni, de ez gyakran nem így van.
- **Manage requirements (Kezeljük a követelményeket):** Mindig tartsuk szem előtt, hogy a követelményeket a felhasználók határozzák meg.
- **Use components (Használjunk komponenseket):** A projekt komponensekre bontása segíti a tesztelést, újrafelhasználást, átlátást, menedzselést.
- **Model visually (Vizuálisan modellezünk):** Használjunk diagramokat a fő komponensek, felhasználók, interakciók leírására (szoftver modellezés, UML, DSL).
- **Verify quality (Ellenőrizzük a minőséget):** A tesztelés mindig kapjon kitüntetett szerepet a projekt minden fázisában.
- **Control changes (Kövessük és ellenőrizzük a változásokat):** Számos projektet több különböző csapat fejleszt, gyakran különböző helyszínen, más-más platformok használatával. Folyamatosan gondoskodjunk a változások szinkronizálásáról és verifikálásáról.

9. Mutasd be a Capability Maturity Model Integration (CMMI) különböző területeinek céljait és jellemzőit!

- A **CMMI** egy folyamatokat javító modell, amely teljesítménybeli kérdések javításban segít, tetszőleges iparágra adaptálható. Célja segíteni az üzleti célok azonosításában és elérésében (mérhető teljesítménybeli célok).
- 3 területen érhető el:
    - The CMMI for Development (CMMI-DEV)
        - Termékfejlesztési folyamatok javítása
        - Hatékonyság, teljesítmény minőség ösztönzése
    - The CMMI for Services (CMMI-SVC)
        - A végfelhasználók számára szolgáltatásokat létrehozó és fenntartó szervezetek folyamatait segíti
    - The CMMI for Acquisition (CMMI-ACQ)
        - Beszerzés, szállítmányozás, termék és szolgáltatás integrálását végző folyamatok javítása
- Különböző iparágakat fed le: szoftverfejlesztés, gyártás, pénzügy, hardver, telekommunikáció, biztonság…

10. Mutasd be a Capability Maturity Model Integration (CMMI) különböző szintjeit és azok jellemzőit!

- A **CMMI** egy folyamatokat javító modell, amely teljesítménybeli kérdések javításban segít, tetszőleges iparágra adaptálható. Célja segíteni az üzleti célok azonosításában és elérésében (mérhető teljesítménybeli célok).
- A CMMI folyamatok felosztása két szempont szerint történik: Maturity Level wise és Category wise



- Category wise folyamat területek: Project Management, Engineering, Process Management, Support

11. Adj kritikát a vízesés és az evolúciós szoftverfejlesztési modellekre!

Vízesés modell: Az elmúlt 50 év leggyakrabban használt modellje. Lépései: Követelményspecifikáció (túl korai és hosszú dokumentum, ebből lesz minden probléma), Szoftvertervezés, Megvalósítás és integráció, Tesztelés és ellenőrzés, Telepítés, Karbantartás

- Nem az készül el, amire szükség van, mivel gyakran nem kapunk visszajelzéseket a termékről addig, amíg az el nem készült
- A folyamatosan megjelenő új követelmények koncepcionális változásokat indukálnak, ami újratervezést igényel: ilyenkor jelentős a kidobott idő, energia
- A tervezők gyakran nincsenek tisztában az implementációs részletekkel: néha egyszerű tervváltoztatás helyett bonyolult implementáció készül el
- A tervező eszközök nem integrálódnak jól az implementációt segítő eszközökkel
- Túl hosszú lehet a teljes fejlesztési fázis

Evolúciós modellek: Több ciklus, mindegyik a termék egy részéért felel: ez lehetővé teszi a követelmények módosítását
Rövidebb ciklusok csökkentik a hibás ágak scope-ját, rövidebb visszajelzési ciklus lehetőséget ad a rendszeres beavatkozásra, mindez segíti a folyamatosan irányban tartást, a végső sikerhez vezető közvetlen úton járást

- Megrendelő oldalról költség és határidő szempontjából nehezen tartható, valamint az ő részvéte is gyakrabban szükséges
- Alkalmazás előállításának költségei tipikusan nagyobbak, utólag nehéz elszámolni vele a sok módosítás és iteráció miatt (nincs fix fejlesztési terv az elejétől, sokat változik)
- A megrendelő nincs rákényszerítve a specifikáció végiggondolására (vízesésnél előre elkészül)

12. Mit gondolsz az alkalmazásban megjelenő rugalmasságról?

- Kétfajta szerződéses modell van, ami esetében ezt a kédést vizsgálhatjuk: project alapú és költség alapú
    - Projekt alapú: Szerződésben rögzített specifikáció (feature set), és ellenérték
    - Költség alapú: Szerződésben fejlesztői munka/erőforrások időarányos ára (óradíj)
- A **projekt alapú**nál szükséges előre elkészíteni a **feature set**-et: egy kellően részletes tervet kell előre elkészíteni, amely alapján a szerződés létrejön. Ez a **klasszikus vízesés modell** módszertanra épít jobban. Ahogyan a vízesés módszer is kellően **rugalmatlan**, úgy ez a szerződés típus is, de puffer-el javítható.
- **Költség alapú**nál a megrendelő nincs rákényszerítve, hogy előre átgondolja a követelményeket, specifikációt, valamint menet közben kérhet bármilyen **változtatás**t amely akár architektúrális jellegű is lehet, tehát ez sokkal közelebb áll az **evolúciós (agilis) módszertan**hoz. Kellően **rugalmas** mind a módszertan mind a szerződés.

Követelményelemzés és szoftvertervezés

13. Foglald össze az Structured Systems Analysis And Design Method (SSADM) módszer lényegét, fázisait, előnyeit és hátrányait!

- A módszer elkülönült egységekre osztja fel az információs rendszer fejlesztésének munkáit és hajlékonyan idomul a különböző feladatokhoz: egyfajta vízesés módszer, dokumentum-vezérelt megközelítés
- **3 meghatározó techinka:** Más-más nézőpontot biztosítanak ugyanahhoz a rendszerhez, mindegyik szükséges a teljes képhez és egymásra hivatkoznak.
    - Logical Data Modeling: A rendszerrel kapcsolatos adatelvárások azonosítása, modellezése és dokumentálása
    - Data Flow Modeling: A rendszerben történő adatmozgások azonosítása, modellezése és dokumentálása. Elemzi az adatfeldolgozási folyamatokat, adattárolást, külső entitásokat, és adatfolyamokat

    - Entity Behavior/Event Modeling: Entitások viselkedésének modellezése. Az entitásokra hatással lévő események azonosítása, modellezése és dokumentálása
- SSADM alapú alkalmazásfejlesztési projektek **5 modul**ra bomlanak, melyek további lépésekből, feladatokból állnak. Minden lépést a korábban befejezett lépés eredményére épít.
    - Feasibility Study – az üzleti terület elemzése azzal a céllal, hogy a megvalósításra kerülő rendszer mennyire tudja költséghatékonyan támogatni az üzleti igényeket
    - Requirements Analysis – a kifejlesztendő rendszerrel szembeni követelmények azonosítása, az aktuális üzleti környezet modellezése (folyamatok, adatstruktúra)
    - Requirements Specification – részletes funkcionális és nem funkcionális követelmények azonosítása, az elvárt feldolgozási képesség és adatstruktúra definiálása
    - Logical System Specification – technikai rendszer opciók, a működés logikai modellje
    - Physical Design – adatbázis terv, konkrét alkalmazásspecifikáció
- Előnyök: Az ad-hoc megközelítésekkel szemben, az üzlet (vagy az üzlet egy részének) metodológián alapuló tanulmányozása, az elvárások szerint mélyebb, rendszerezettebb ismereteket ad az üzleti folyamatokról, adatok szerkezetéről. Az elvárásaink szerint mindez egy kerekebb, korrektebb rendszerhez vezet.
- Hátrányok:Az SSADM esetén minden fázis elkezdését meg kell előzze a korábbi fázis befejezése.
    - "analysis paralysis“: az üzlet és a folyamatai folyamatosan változnak, melynek következtében folyamatosan frissíteni kell az elemzési és a tervezési fázisok eredménytermékeit. Mindez plusz feladatokat és így gyakran késést eredményez.

14. Foglald össze a követelményelemzés céljait, módszereit, azok előnyeit és hátrányait!

- A Requirements engineering az a folyamat, amely során kialakítjuk, dokumentáljuk és karbantartjuk a szoftverrendszerrel kapcsolatos elvárásokat.
    - A vízesés modellben ez a fejlesztés első fázisa, későbbi módszertanokban a rendszer fejlesztése alatt végig.

- Stakeholder azonosítás és interjúk
    - Stakeholder bárki, aki kapcsolatban van a rendszerrel
    - Az interjú jól bevárt eszköz a követelményelemzéshez
    - Az interjúk személyes jellege egy oldott környezetet teremt, ahol fókuszált, hatékony a kommunikáció
- Szerződés jellegű követelménylista
    - Egy hagyományos módja a követelmények dokumentálásának
    - Összetett rendszerek esetén nagyon sok oldalas dokumentumot jelent
    - Erősségei
        - A követelmények egy kipipálható listáját adja, szerződést formál a projekt szponzor és a fejlesztők között

        - Nagy rendszerek esetén egy magas szintű leírást jelent, amelyből az alacsonyabb szintű követelmények származtathatók
    - Gyengeségek
        - Nagyon hosszú, nehezen olvasható
        - A részleteket elfedi, a teljes követelményhalmaz felfedése eltolódik a fejlesztés és a tesztelés fázisokra
        - Egy lista elkészítése nem garantálja a teljességet
- Use-case-ek
    - A rendszerrel szembeni funkcionális elvárások dokumentálásának egy struktúrája
    - Forgatókönyvek egy halmaza: a rendszer és a felhasználó, valamint a rendszer és más rendszerek közti interakció leírása
    - Nem tartalmaz technikai zsargont, a végfelhasználó, a szakterület nyelvezetét használja
    - Gyakran a követelményelemző és a stakeholder-ek együtt szerkesztik őket
    - Egyszerű eszközei a rendszerrel szembeni elvárások leírásának: nem definiálnak belső működést, implementációs részleteket. A feladat elvégzéseshez szükséges lépésekre koncentrálnak.
    - A felhasználó célja alapján készül
- Prototípus készítése
    - példa alkalmazás, segíti az új rendszer céljainak demonstrálását
    - segíti az követelmények pontosítását és véglegesítését
    - Wireframe, Mockup és Prototype készítés: bevált módszerek, de nem ugyanaz a 3!
    - Támogatja a tervezési döntéseket
    - Az alkalmazás korai megismerése csökkenti a későbbi változtatások számát

15. Mutasd be a követelményelemzés során végzett aktivitásokat és azok jellemzőit!

- **1. Requirements inception / requirements elicitation**: követelmények felderítése
- **2. Requirements identification:** Követelmények azonosítása
- **3. Requirements analysis and negotiation:** követelmények elemzése és egyeztetése, konfliktusok feloldása
- **4. Requirements specification (Software Requirements Specification)** – követelmények dokumentálása
- **5. System modeling:** a rendszer modellezése (különféle jelölésrendszerekkel, pl. UML)
- **6. Requirements validation:** követelmények validálása, annak ellenőrzése, hogy a dokumentált elvárások és a modellek konzisztensek, valamint, hogy teljesítik a stakeholder-ek elvárásait
- **7. Requirements management:** követelménymenedzsment, a változási kérések kezelése a fejlesztés és az üzembe helyezés során

16. Mutasd be a követelményelemzés során előtérbe kerülő különböző követelménytípusok jellemzőit!

- **Functional requirement:** Describes a functionality to be made available to the users of the system.
    - **Coherent:** there are no contradictions among its elements
    - **Complete:** considers all the necessities that the client wishes to see satisfied
- **Non-functional requirement:**
    - **appearance:** the visual aspect and the aesthetics of the system, namely the graphical interface;
    - **usability:** the easiness of utilization of the system and everything that permits a more friendly user experience;
    - **performance:** aspects of speed, real-time, storage capacity, fault tolerance, scalability, and execution correction;
    - **operational:** characteristics about what the system must do to work correctly in the environment where it is inserted;
    - **maintenance and support:** attributes that allow the system to be repaired or improved and new functionalities to be anticipated;
    - **security:** issues related to access, confidentiality, protection, and integrity of the data;
    - **cultural:** factors related to the stakeholders culture and habits;
    - **legal:** laws, rules, and standards that apply to the system so that it can operate.
- **User requirement:** A **functionality** that the system is expected to provide to its users or a **restriction** that is applicable to the operation of that system
- **System requirement:** Constitutes a **more detailed specification** of a requirement; is an **intermediary stage between user requirements and system design**

17. Mutasd be a követelményelemzés során előtérbe kerülő kihívásokat!

- felhasználók különféle módon akadályozzák a követelmények összegyűjtését
    - Nem tudják mit szeretnének vagy nincs egy tiszta képük az elvárásaikról
    - Nem tudnak elköteleződni egy definiált, leírt funkcióhalmaz mellett
    - Új követelményeket erőltetnek annak ellenére, hogy a költség és az ütemezés rögzített
    - A felhasználókkal való kommunikáció lassú
    - Nem vesznek részt az interjúkon, egyeztetéseken
    - Technikailag alulképzettek
    - Nem értik meg a fejlesztési folyamatot
    - Nem ismerik az aktuális technológiát
- Mindez olyan helyzetekhez vezethet, ahol a felhasználói követelmények folyamatosan változnak, még akkor is, amikor a rendszer vagy termékfejlesztés már elkezdődött

18. Foglald össze az üzleti elemzés céljait!

- Célja az üzleti igények azonosítása és megértése, az üzleti problémákhoz megoldások keresése, stratégiai irányok segítése, stratégiai célok elérésének támogatása
    - A megoldás lehet rendszerfejlesztés, folyamatjavítás, szervezeti változtatás, szabályzat fejlesztés
- Vállalat/szervezet elemzés
    - Üzleti architektúra készítése és karbantartása
    - Megvalósíthatósági tanulmányok készítése
    - Új üzleti lehetőségek azonosítása
    - Üzleti lehetőségekhez kapcsolódó scope definiálás
    - Üzleti forgatókönyvek előkészítése
    - Kockázati elemzések készítése
- Célok:
    - Megoldások létrehozása
    - Elegendő eszköz rendelkezésre bocsátása a robosztus project menedzsmenthez
    - Hatékonyság javítása, a veszteség csökkentése
    - Alapvető dokumentáció létrehozása
    - Projekt hatékenységának növelése, időre befejezés

19. Foglald össze az üzleti elemzés területeit, technikáit és jellemzőit!

Az üzleti elemzés területei:

- Követelménytervezés és menedzsment
    - Magas prioritású követelmények azonosítása
    - Változáskezelés
- Követelmények összegyűjtése
    - Brainstorming
    - Dokumentum elemzés
    - Fókuszált csoportmunka
    - Interfészelemzés
    - Interjúk
    - Workshopok
    - Felmérések
    - Felhasználói feladatok felmérése
    - Folyamatok leképzése
- Elemzés
    - Követelmények specifikálása olyan részletességgel, hogy azok sikeresen implementálhatók legyenek
    - Architektúra elemzés
    - Üzleti folyamatok elemzése
    - Struktúra elemzés
    - Tárolás, adatbázis és adattárház elemzés
- Dokumentálás

    - Szöveges- pl. user story-k, amelyek bizonyos információkat összefoglalnak
    - Mátrix – pl. követelmények táblázata a prioritások megjelölésével
    - Diagramok - pl. adatfolyam az egyik struktúrából a másikba
    - Wireframe – pl. egy weboldal tartalma
    - Modellek
- Követelmények kommunikálása
    - Technikák, amelyek segítenek a stakeholderekkel megérteni a követelményeket és, hogy azok hogyan kerülnek implementálásra
- Értékelés és validálás
    - A javasolt megoldás helyességéről való meggyőződés
    - A megoldás implementálásának támogatása
    - Az implementálás során a hiányosságok kezelése

Lehetséges technikák:

- A **SWOT** elemzés 4 attribútuma
    - **Erősségek** (Strengths) – Mik az előnyök? Mit végzünk jelenleg jól?
    - **Gyengeségek** (Weaknesses) – Mely dolgok javíthatók? Mit végzünk rosszul?
    - **Lehetőségek** (Opportunities) – Milyen jó lehetőségek kínálkoznak a szervezet részére?
    - **Veszélyek** (Threats) – Milyen nehézségekkel néz szembe a szervezet? (pl. kulcs terület, ahol a versenytárs jól teljesít)
- **PESTLE**: külső környezeti elemzés, amely vizsgálja a különféle, a szervezetre hatással bíró külső faktorokat
- **5 Whys**:
    - Iteratív kérdés-válasz technika, célja a probléma körüli ok és okozat felderítés
    - Az elsődleges cél a gyökér (alap) probléma/hiba felderítése
- **MoSCoW**: a követelmények priorizálása, a követelmények egymáshoz viszonyításával
    - Must have – egyébként a teljesítés hibás
    - Should have – egyébként alternatív megoldást (workaround) kell keresni
    - Could have – hogy növeljük az elégedettséget
    - Would like to have in the future – de most nem

Szoftver tervezési módszerek

20. Mutasd be a szoftvertervezés során alkalmazott alapelveket és koncepciókat!

A folyamat, amely során a szoftvertermék specifikációja készül el, mindazon aktivitás, amely a követelményspecifikáció elkészítése és a fejlesztés elkezdése között van.

- A szoftvertervezés eredményei modellek és dokumentáció (terméktípustól függően más és más jellegű és mennyiségű)
- A tervezés lehet platform-függő vagy platformfüggetlen
- A tervezés egy megoldás kidolgozása egy adott problémára a rendelkezésre álló képességek/tudás figyelembe vételével
- A célkörnyezettől függően más-más terv születhet ugyanarra a feladatra

Alapelvek:

- A szoftvertervezés folyamat és modell is egyben
- A tervezési folyamat lépések sorozata, mely során a tervező a szoftvertermék minden aspektusát részletesen kidolgozza
- A tervezési folyamat nem egyszerűen egy recept, kreativitás, jelentős tapasztalat, a "jó szoftverhez" való érzék/ráérzés, a minőség iránti elköteleződés mind kritikus tényezői a kompetens tervezésnek
- A tervmodell különböző absztrakciós szinteken mutatja be a rendszert más és más aspektusból
- **The design process should not suffer from "tunnel vision."** Alternatív lehetőségek vizsgálata, azok erőforrás igényeinek, és a kapcsolódó kapacitások figyelembe vételével
- **The design should be traceable to the analysis model.** A tervmodell egy eleme gyakran több követelményhez is tartozik, a követelmény-terv kapcsolatokat igazolni kell
- **The design should not reinvent the wheel.** Meglévő tervezési minták használata.
- **The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.** A szoftverterv struktúrája, amennyire az lehetséges, imitálja a szakterület struktúráját
- **The design should exhibit uniformity and integration.** Egységes szabályok, stílus használata: mintha egy kollega készítette volna az egészet.
- Változás elfogadásának tervezése
- A nem várt körülményekre is fel kell készülni, lassú degradáció
- A tervezési absztrakciós szint magasabb, mint a fejlesztési
- Számos eszköz, koncepció segít megmérni a terv minőségét, használjuk!
- Időt kell szánni a terv ellenőrzésére! A koncepcionális kérdéseken kell legyen a hangsúly!

21. Mi a szoftvermodellezés szerepe a szoftvertervezés során? Milyen módszereket, technikákat ismersz?

- Modellezés: segít megérteni, egyeztetni és kommunikálni a felhasználói igényeket. A követelményeket diagramok egy halmaza fedi le, más-más aspektusra koncentrálnak
- Inkonzisztencia csökkentése
    - A különböző szereplők emberi természetüknél fogva is máshogy látják az üzleti világot. Az első feladat ezen eltérések feloldása
    - A modell építése során számos üzleti kérdés felmerül, amelyek átbeszélése egységesíti a képet, közelíti a nézőpontokat

Modelltípusok és felhasználásuk

- **Dependency graphs – Függőségi gráf:** A kód felépítését és a kódon belüli kapcsolatokat segít áttekinteni. Függőségek, struktúra megértését segíti
- **Layer diagrams – Rétegezett diagram:** Segíti az alkalmazás struktúrájának definiálását: rétegek, blokkok, explicit függőségek. A layer diagram és a kód által definiált függőségek (automatikusan) összehasonlíthatók. A függőségi konfliktusok elkerülését/felderítését segíti

- **UML model – UML modell:** Számos nézet: osztály, komponens, use case, aktivitás, szekvencia diagramok. Testre szabható, így illeszthető az alkalmazás szakterületéhez
- **Sequence diagrams – Szekvencia diagramok:** Segít vizualizálni, hogy a kód hogyan implementál egy adott metódust/funkciót.
- **Domain-specific language (DSL) – Szakterületi nyelv:** Egy jelölésrendszer speciális célokra.

22. Mutasd be a szoftvertervezés során előkerülő tervezési szempontokat (megfontolásokat)!

Koncepciók, tervezési megfontolások:

- **Compatibility** – más termékekkel, korábbi verziókkal való együttdolgozás
- **Extensibility** – az új elemek/funkciók hozzáadása a meglévők jelentős módosítása nélkül tehető meg
- **Fault-tolerance** – a szoftver hibatűrő képessége, mennyire tud talpra állni az egyes komponensek hibáit követően
- **Maintainability** – a karbantarthatóság mértéke: hibajavítás, funkcionális módosítás
- **Modularity** – a moduláris szoftverek jól definiált, független komponensekből állnak. Jobb karbantarthatósághoz vezet. Az egyes komponensek külön-külön fejleszthetők és tesztelhetők
- **Reusability** – kevés módosítással adhatók hozzá új funkciók
- **Robustness** – stressz alatt is jól teljesít, nem várt bemenetekre fel van készülve. Pl. nem dől össze kevés memória esetén
- **Security** – rosszindulatú támadásokkal szemben ellenálló
- **Usability** – a felhasználói felület használható a célfelhasználók számára, pl. alapértelmezett értékek ki vannak töltve
- **Performance** – a felhasználók által elfogadható válaszidőn belül elvégzi a feladatát
- **Portability** – ugyanazon szoftver használható más-más környezetekben
- **Scalability** – a megnövekedett felhasználószámhoz vagy adatmennyiséghez jól adaptálódik

23. Miért használunk a modellezés során eltérő absztrakciós szintű modelleket? Mutass rá példákat!

A különböző szereplők emberi természetüknél fogva is máshogy látják az üzleti világot. Az első feladat ezen eltérések feloldása.

High-level design:

- **Understanding the Requirements – A követelmények megértése:** A tervezés kiindulópontja a felhasználói igények teljes megértése.
- **Architectural Patterns – Arcitektúrális minták:** A rendszer megvalósításához választott alap technológiák és a rendszer architektúrális elemei.

- **Components and their Interfaces – Komponensek és interfészek:** A rendszer fő komponenseinek definiálása (komponens diagram), a köztük lévő, az interakciót szolgáló interfészek
- **Interactions between Components – A komponensek közti interakció:** Minden use case, esemény, beérkező üzenet alapján készíthető szekvencia diagram, amely szemlélteti a rendszer fő komponenseinek interakcióját
- **Data Model of the Components and Interfaces – A komponensek és az interfészek adatmodellje:** Az információt szemléltető osztálydiagramok

Fejlesztő eszközök hatékony használata

24. A fejlesztői eszköz kiválasztásánál milyen szempontokat érdemes figyelembe venni és miért?

- Kik használják? Egyedül vagy egy egész csapat (Miért? Támogatja-e a közös munkát a szoftver), ők milyen meglévő ismeretekkel rendelkeznek (Miért? Hasonló szoftver gyorsíthatja a betanulást), milyen az eszköz tanulási görbéje, új kolléga mennyi idő alatt tanul bele, milyen a piaci ismertsége (Miért? Mert ez már jelent valamit…)
- Milyen időtávon használjuk? (Miért? Mekkora súlyú döntést kell hoznunk, mennyi időt töltsünk a választással)
- Ki az eszköz gyártója? Nagy cég esetén lassan történnek a változások, de megbízható, kis cég gyors innovációt ad, de könnyen csődbe mehet, opensource esetén pedig nem biztos a support
- Integrálódás más eszközökkel? Sosem 1 eszközt használunk, így meg kell vizsgálni milyen milyen másik eszközzel tud integrálódni (import, export, élő kapcsolat…)

25. Mik az előnyei a verzió kezelő eszközök használatának? Milyen verzionálási modelleket (nem elágaztatási stratégiát!) ismersz, mik ezek előnyei, hátrányai?

- Potenciális ütközések kezelése
- Biztonsági mentés
- Visszaállás adott verzióra
    - Például kiadott éles verzió javítása
    - Félresikerült módosítások visszavonása – rövid/hosszú távon
- Változások követése
- Felelősök követése
- Hozzáférés szabályozás
- Elágazás és összefésülés

- Verzionálási modellek:
    - Zárolás (pesszimista): Lock-Modify-Unlock
        - Nagy csapatnál kényelmetlen, várni kell
        - Offline működés nem támogatható
        - Mindig a legfrissebb verzióval dolgozunk, a változásokat le kell tölteni szerkesztés előtt
        - Bináris, office fájlokhoz ideális lehet

    - Zárolás nélkül (optimista): Copy-Modify-Merge
        - Merge-elni kell
        - Nem kell várni, bármi módosítható
        - Offline működik
        - Nem kell a változásokat beszinkronizálni szerkesztés előtt

26. Ismertesd a központosított és elosztott verziókezelés előnyeit, hátrányait!

Központosított:
- Szerver – kliens működés
- Minden változtatás a központ repositoryn keresztül megy
- Egyszerű
- Teljes kontroll a hozzáférés fölött
- Több, jobban működő kliens
- Bizonyos műveletek szervert igényelnek

Eloszott:
- Minden felhasználónak van a saját repositoryja
- Az önálló repositoryk hálózatként működnek
- A megosztáson kívül minden helyi művelet, nincs szerver függőség (SPoF)
- A teljes history helyben lehet
- Egyszerűbb elágaztatás
- Bonyolult
    - Felhasználóknak nehezebb, kevés jó eszköz

27. Milyen elágaztatási stratégiákat ismersz? Jellemezd őket röviden!

Ág típusok:
- Main
    - A Development és Release ágak találkozása
    - Mindig fordul, a QA csapat ezt használja
- Development
    - Független a többi ágtól, a fejlesztés normál ütemben folyhat
    - Gyakori merge javasolt
- Release
    - Kiadott verziók támogatása
    - A hotfixeket vissza kell vezetni a Main/Dev ágakba

Stratégiák:
- Main only
    - Egyszerű
    - Címkék jelzik a fejlesztés diszkrét állapotait
        - A címkék helyéről bármikor indítható új ág ha szükséges
- Development isolation
    - Egy vagy több párhuzamos ág fut Main mellett
        - Akár minden fejlesztőnek egy külön ág….
    - Ezek lehetnek feature fejlesztések, bugfixek stb.
    - Mindig egy támogatott release verzió van, a Mainből
- Release isolation

        - Több release ág, párhuzamos release támogatás
        - A release ágakon ritkán van változtatás
            - Legfeljebb hotfix
- Development and release isolation
    - A két előző stratégia kombinálása
        - Párhuzamos release-ek karbantartása
        - Izolált fejlesztési ág(ak)
- Feature isolation
    - A feature-ök külön ágat kapnak
    - Akkor mozgatható a Mainbe, amikor készen van
        - Vagy amikor a Megrendelő kéri
- Code promotion
    - Vízesés jellegű, minőség alapú ágak
    - Hosszú teszt ciklus

Branching alternatívák:

- Shelving / stashing
    - “Elmentjük későbbre”
- Labeling / tagging
    - Megjelöljük az állapotot
    - Később is lehet belőle leágazni

Tesztelés

28. A tesztelésbe milyen szerepkörű résztvevők vannak bevonva? Mi a tipikus feladatuk a teszteléssel kapcsolatban? Jellemezd, hogy a projekt különböző fázisaiban mik a módosítások költségei!

Tesztelésben résztvevő szereplők:

- **Megrendelő:**
    - A kiadás előtt az alfa/béta tesztelésben vesz részt
    - Átadás/átvételi tesztelést végzi
- **Üzleti elemző/tervező:**
    - Funkcionális specifikációt ír, ami alapján a teszt terv készül
    - Ellenőrzi a teszt tervet
- **Fejlesztő:**
    - Megírja a kódot amit tesztelni tudunk
    - Átnézi a teszt tervet
    - Unit teszteket készít
- **Tesztmérnök:**
    - Teszt tervet készít
    - Teszt eseteket ír le és végrehajtja őket
    - Jóváhagyja a kiadni kívánt terméket
- **Végfelhasználó:**
    - Részt vehet béta tesztelésben vagy átvételi tesztben

- Kiadás után, használat közben visszajelzéseket ad

A javítás költsége attól függően, hogy hol és mikor találjuk meg a hibát

	Mikor ismerjük fel a hibát...				
Hol van a hiba	**Spec. elemzés**	**Tervezés**	**Megvalósítás**	**QA**	**Kiadás után**
Követelmény	1x	3x	5-10x	10x	10-100x
Terv	-	1x	10x	15x	25-100x
Megvalósítás	-	-	1x	10x	10-25x

29. Mi az ellenőrzés és mi a validáció? Mit tartalmaz a teszterv? Röviden jellemezd az elemeket! Mit értünk tesztelési stratégián?

- **Validáció** (Validation): "A megfelelő szoftvert készítjük?"
  - Tényleg erre van szüksége a felhasználónak? Nincs hiba a követelményekben?
  - Elfogadási teszt szól a validációról (többi csak ellenőrzés)
- **Ellenőrzés** (Verification): "Jó irányban haladunk a szoftverrel?"
  - Jól működik-e a kód?

- **Teszterv**: A teszteléshez tartozó követelményrendszer
  - Scope-ot definiál: mit tesztelünk és mit nem?
  - Teszteseteket ír le: egy lista, előfeltételek (infrastruktúra, architektúra, input adatok, külső függőségek pl. szolgáltatás), tesztelési stratégiák (pl. Milyen keretrendszer?) és sikerkritériumok

- **Tesztelési stratégia:**
  - Leírja a használni kívánt tesztelési módszert: black/gray/white box teszt
  - Megadja milyen módszerrel végezzük és milyen keretrendszert használunk hozzá

30. Ismertesd a tesztelés szintjeit!

- Unit tesztelés (white/gray)
  - A kód egy kis részét ellenőrzi
  - Jellemzően white-box teszt
- Integrációs tesztelés (white/gray)
  - A komponensek közti interfészt ellenőrzi
  - Iteratív folyamat amíg a teljes rendszer ellenőrzésre kerül
- Rendszer tesztelés (black)
  - A rendszereket önmagukban ellenőrzi a követelmények alapján Integrációs teszt
- Rendszer integrációs teszt (black)
  - Már mindegyik rendszer átment a „Rendszer teszten"

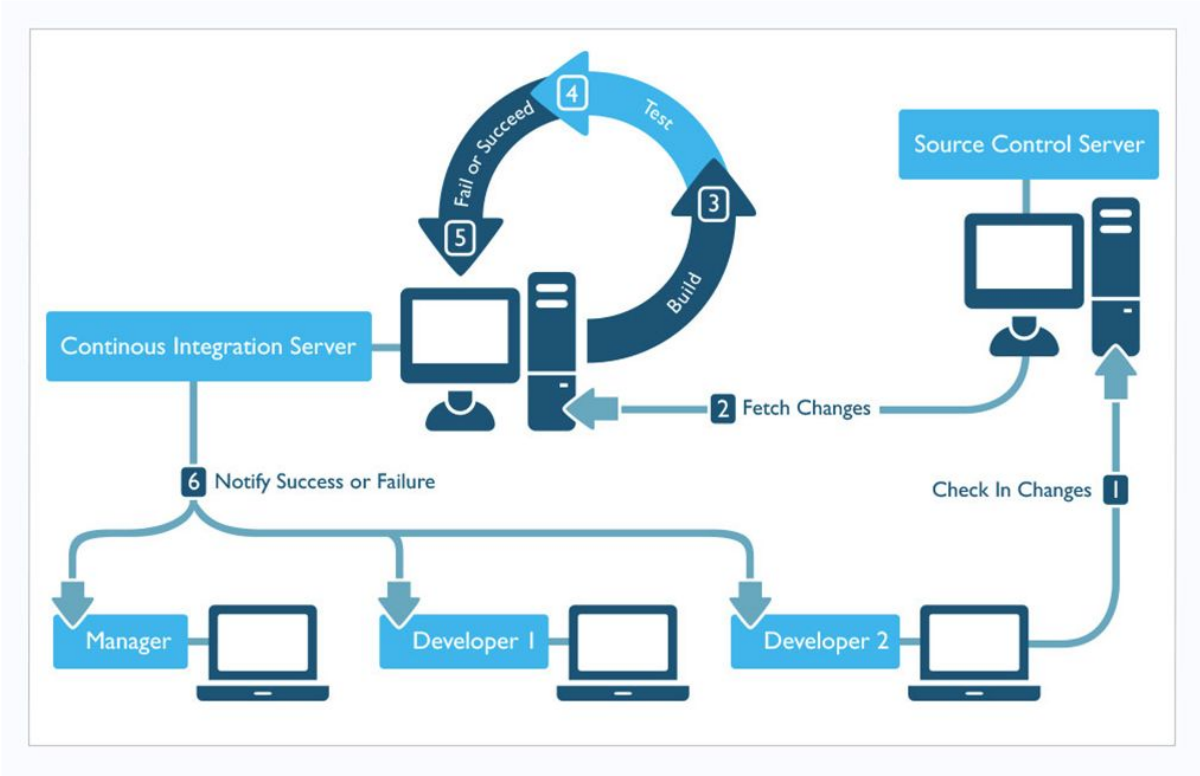
    - A külső rendszerekkel való együttműködést ellenőrzi
- Regressziós teszt (white/gray)
    - Ellenőrzi, hogy a módosítások után (frissítés, más hibák javítása) a szoftver megtartja eredeti hibamentes viselkedést
- Elogadási teszt (black)
    - “User Acceptance Testing”
    - A végfelhasználó végzi saját környezetében
- Alfa teszt (black)
    - Nem végleges verzió ellenőrzése belső vagy meghívott külső felhasználók által
- Béta teszt (black)
    - Széleskörű teszt az alfa teszt után
    - Jellemzően külső felhasználók végzik

31. Mit jelent a continuous delivery? Mutasd be!

- Continuous Delivery is the **ability to get changes of all types** — including new features, configuration changes, bug fixes and experiments — **into production**, or into the hands of users, **safely** and **quickly** in a **sustainable** way.
- Continuous delivery rests on three foundations:
    - Configuration management
    - Continuous integration
    - Continuous testing

32. Mit jelent a continous integration? Mutasd be!

- Continuous Integration is a software development practice where **members of a team integrate their work frequently, usually each person integrates at least daily** - leading to multiple integrations per day.
- **Each integration is verified by an automated build** (including test) to detect integration errors as quickly as possible.
- Many teams find that this approach **leads to significantly reduced integration problems** and allows a team to **develop cohesive software** more rapidly.



33. Foglald össze a continuos integration előnyeit!

CD előnyei:

- Accelerated time to market
- Building the right product
- Improved productivity and efficiency (automation)
- Reliable releases
- Improved product quality
- Improved customer satisfaction

CI előnyei:

- Reduced risk (no integration phase)
- Bugs – CI doesn't get rid of bugs, but it does make them dramatically easier to find and remove
    - the degree of this benefit is directly tied to how good the test suite is
- CI removes one of the biggest barriers to frequent deployment (rapid feedback, collaborative development cycle)

34. Hogyan javasolt bevezetni a continuos integration-t?

- Get the build automated
    - Get everything we need into source control
    - Goal: build the whole system with a single command
- Introduce some automated testing into the build
- Speed up the commit build

- New project? begin with CI from the beginning
- Find someone who has done CI before to help

Agilis fejlesztési módszerek

35. Foglald össze az agilis módszerek értékeit és elveit!

Értékek:

- Kommunikáció
    - „What matters most in team software development is communication”
    - Minden probléma visszavezethető a kommunikációra
        - Programozó – Programozó
        - Programozó – Ügyfél
        - Manager – Programozó
    - Egy cél: minden fejlesztő ugyanúgy lássa a rendszert, ahogy a majdani felhasználók is látni fogják.
- Egyszerűség ○ „What is simplest thing that could work?”
    - Az agilis módszer arra fogad, hogy…
    - YAGNI
    - Az egyszerűség segíti a kommunikációt
- Visszacsatolás
    - „Az optimizmus szakmai ártalom a programozóknál, és a visszajelzés rá a gyógyír.”
    - „Elsőre jól?”
        - Nem tudjuk hogyan
        - Ha tudjuk, holnap már lehet a jó rossz
        - Sok és gyors visszacsatolással tudunk közel kerülni a jó megoldáshoz.
- Bátorság
    - Tenni valamit a félelem ellenére
        - Betartani a YAGNI-t
        - Refaktorálni
        - Revertálni
    - Vagy nem tenni
    - Főleg a többi értéket támogatja
        - Kimondani jót és rosszat
        - Eldobni a rossz megoldásokat
        - Igazi konkrét válaszokat keresni
- Respect
    - Ha nem törődünk egymással, akkor…
    - Ha nem törődünk a projekttel, akkor…
    - „I am important and so are you”

Elvek/Értékek:

- Egyének és interakció inkább, mint folyamatok és eszközök

- Működő szoftver inkább, mint jól dokumentált szoftver
- Felhasználó bevonása inkább, mint szerződések tárgyalása
- Reagálás a változásokra inkább, mint egy terv követése

- **Az agilis fejlesztés 12 pontja**

36. Foglald össze az agilis tervezés jellemzőit és szintjeit!

- Az agilis tervezés: Csapatmunka, Rövid iterációk, Prioritás, Adaptáció
- Célja: Kockázatok csökkentése, Bizonytalanság csökkentése, Döntéstámogatás, Bizalomszerzés, Információátadás

Szintjei:

- Vízió
    - Mit szeretnénk elérni? De a Hogyan? nem része a víziónak
    - Vízió dokumentum: Mi a projekt célja? Specifikus, de nem előíró. Miért értékes? És sikerkritériumok.
- Kiadás tervezés
    - Gyakori kiadások, nem feltétlenül a végfelhasználónak (~3 hónap)
    - A felhasználó jobban átlátja, könnyebben veszi át
    - Felhasználói sztorik: Olyan dolgot ír le, amit a csapatnak meg kell valósítania, mini víziók
        - [VALAKI]ként azt szeretném elérni, hogy [CÉL] azért, hogy [MIÉRT].
- Iteráció tervezés: Az iteráció az agilis projekt szívverése, hossza: 1 vagy 2 hét
    - A legértékesebb sztorik a kiadás tervezési tábláról
    - Becslés és vállalás
    - A felhasználó és a termékmenedzser már az előző iteráció közben kifejti a sztorikat
    - Iteráció közben scope módosítható, határidő nem. Félkész sztorikat rollbackelni és átvinni a következő iterációba.
- Stand-up
    - Naponta 1x (2x), Max. 10-15 perc
    - Felállva
    - Ha másképp nem megy: Mit csináltam tegnap? Mit fogok csinálni ma? Mi akadályoz meg abban, hogy haladjak?

37. Mutasd be az agilis módszerek során alkalmazott kiadás tervezést és annak jellemzőit! Térj ki a felhasználói sztorik szerpére!

- Gyakori kiadások, nem feltétlenül a végfelhasználónak (~3 hónap)
- A felhasználó jobban átlátja, könnyebben veszi át
- Korábban problémás volt
- Egyszerre egy projekt
- Felhasználói sztorik: Olyan dolgot ír le, amit a csapatnak meg kell valósítania, mini víziók

    - Céljuk, hogy tervezni tudjunk
    - Megrendelő számára értékkel rendelkezik: teljesítési kritérium
    - [VALAKI]ként azt szeretném elérni, hogy [CÉL] azért, hogy [MIÉRT].

38. Foglald össze az agilis iterációk jellemzőit és az iteráció tervezés lépéseit!

- Iteráció:
    - Az iteráció az agilis projekt szívverése
    - Hossza: 1 vagy 2 hét -> Időkorlát
    - Az iteráció nem előzi meg a problémát, csak felszínre hozza őket
- Az iteráció lépései:
    - 1 hét
    - Demo
    - Visszatekintés
    - Iteráció-tervezés
    - Vállalás
    - A sztorik megvalósítása
    - A kiadás elkészítése
- Iterációtervezés:
    - A legértékesebb sztorik a kiadás tervezési tábláról
    - Mérnöki feladatokra bontás
    - Becslés és vállalás
    - A felhasználó és a termékmenedzser már az előző iteráció közben kifejti a sztorikat
    - Néhány óra
- Iteráció közben scope módosítható, határidő nem. Félkész sztorikat rollbackelni és átvinni a következő iterációba.

39. Mutasd be a napi stand-up esemény és a “Kész, kész” (done, done) jellemzőit!

- Naponta 1x (2x), Max. 10-15 perc
- Felállva
- Ha másképp nem megy: Mit csináltam tegnap? Mit fogok csinálni ma? Mi akadályoz meg abban, hogy haladjak?
- “Kész, kész” (done, done)
    - Megtervezett
    - Lekódolt
    - Integrált
    - Tesztelt
    - A build lefut
    - A termék installálódik
    - Migrálódik
    - A végfelhasználó ellenőrizte és elfogadta
    - Megfixált

40. Foglald össze az eXtreme Programming (XP) jellemzőit és eszközeit!

- A programozók párokban programoznak.
- A fejlesztést a tesztesetek irányítják
    - Először tesztelünk, utána kódolunk. Amíg nem fut az összes teszteset, nem vagyunk készen. Amikor az összes teszt fut, és nem jut eszünkbe olyan teszt, ami esetleg hibát jelezne, kész vagyunk az új funkciók hozzáadásával.
- A párok nem csak a teszteseteket teszik futtathatóvá, hanem a rendszer tervezését is előremozdítják.
    - A változtatások nem csak egy adott területet érintenek.
    - A párok szervesen részt vesznek a rendszer analízisének, tervezésének, implementációjának és tesztelésének értékesebbé tételében.
- Az integráció közvetlenül követi a fejlesztést, beleértve az integrációs teszteket.
- Szélsőséges mértékben
    - Kód ellenőrzés - párprogramozás
    - Tesztelés – mindenki mindig tesztel
    - Tervezés – mindenki mindig tervez (refactoring)
    - Egyszerűség
    - Integráció – folyamatos integrálás
    - Architektúra – mindenki finomítja
    - Rövid iterációk

41. Foglald össze az eXtreme Programming (XP) értékeit és elveit!

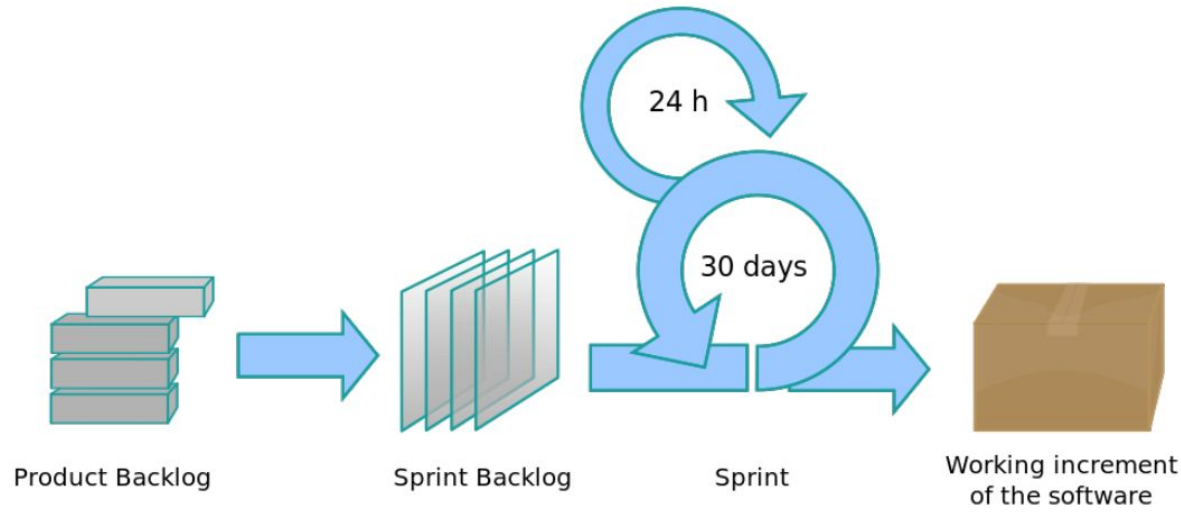
- Szélsőséges mértékben
    - Kód ellenőrzés - párprogramozás
    - Tesztelés – mindenki mindig tesztel
    - Tervezés – mindenki mindig tervez (refactoring)
    - Egyszerűség
    - Integráció – folyamatos integrálás
    - Architektúra – mindenki finomítja
    - Rövid iterációk
- Tervkészítési játék:
    - Üzleti szakemberek eldöntik a(z)
        - …hatókört
        - …prioritásokat
        - …a kibocsátott verzió összetételét
        - …a kibocsátott verziók dátumát
    - A műszaki szakemberek eldöntik a(z)
        - …becsléseket
        - …üzleti döntések következményeit
        - …folyamatokat
        - …részletes időbeosztást
- Kis méretű kibocsátott verziók
- Az egyszerű terv …

    - > minden tesztet futtat
    - > nincs duplikált logika
    - > kifejezi a programozó szándékát
    - > a legkevesebb osztályt és metódust tartalmazza
- Tesztelés
- Kódátírás (refactoring)
- Programozás párban
- Közös kód
- Folyamatos integráció
- 40 órás munkahét
- A megrendelő rendelkezésre áll helyben
- „Az egyszer és csakis egyszer” szabály

42. Mutasd be a Scrum fejlesztési folyamatot és jellemzőit!

Folyamat:

- Minden sprint (2-4 hét) során a csapat egy működő szoftveregységet hoz létre.
- A sprint során megvalósítandó funkciók a Product Backlog-ból (termék teendőlistája) kerülnek ki, ami az elvégzendő munka magas szintű követelményeiből álló, fontossági sorrendbe állított lista.
- A Product Owner közli a csapattal, hogy a teendők listájából melyek azok, amiket leghamarabb szeretné, hogy elkészüljenek.
- Ezután a csapat eldönti, hogy ezek közül melyek azok, amelyeket a következő sprint során meg tud valósítani, és ezek megvalósítására ígéretet tesz.
- A sprint folyamán a Spring Backlog-ot nem lehet megváltoztatni, a sprint során elvégzett tevékenységek rögzítettek.
- Amint a sprint a végéhez ért, a csapat bemutatja az elkészült funkciókat (demo).



Jellemzők:

- A Scrum egyik legfontosabb alapelve az, hogy felismeri és elfogadja, hogy a megrendelő a fejlesztés során meggondolhatja magát a követelményekkel kapcsolatban, és a váratlan változások nem kezelhetők könnyen a hagyományos, előzetes tervezési fázison alapuló módszerekkel.

- Ezért a Scrum gyakorlati megközelítést választ, és elfogadja hogy nincs lehetőség a probléma teljes megértésére és definiálására. Inkább azt próbálja maximálisan elősegíteni, hogy a csapat gyorsan meg tudja valósítani a funkciókat és gyorsan tudjon reagálni a változó követelményekre.

43. Mutasd be a Scrum ceremóniák jellemzőit!

- Napi megbeszélés
    - A megbeszélés ideje 15 perc
    - Állva: ez elősegíti, hogy a megbeszélés ne húzódjon el
    - Minden nap ugyanazon a helyen és ugyanabban az időpontban tartják
    - A megbeszélés során minden résztvevő ugyanazokat a kérdéseket válaszolja meg:
        - Mi az, amit a tegnapi megbeszélés óta csináltam?
        - Mi az, amit a mai nap tervezek csinálni?
        - Vannak-e akadályok, amik gátolnak a cél elérésében?
- Sprint planning
    - Elvégzendő feladatok kijelölése a termék teendőlistájáról (product backlog) a
    - terméktulajdonos közreműködésével.
    - A sprint teendőlistájának előkészítése, amely a teljes csapat figyelembevételével részletezi az egyes részfeladatok időszükségleteit.
    - Annak meghatározása és kommunikálása, hogy mennyi feladat elvégzése várható el az aktuális sprint során.
- Demo
    - Annak áttekintése, hogy mely munkák készültek el és melyek nem.
    - Az elkészült munka bemutatása a terméktulajdonos és a fejlesztésben érdekeltek részére
- Retrospective
    - Csapattagok véleményt alkotnak az elmúlt sprintről. A vélemény lehet egy puszta benyomás is, nem kell kidolgozott, szilárd álláspontnak lennie.
    - Javaslatokat tesznek a folyamatok továbbfejlesztésére. A javaslatoknak nem kell kiérleltnek lenniük, a kidolgozás nem a visszatekintés része.
    - Két kérdés merül fel a megbeszélésen:
        - Mi az, ami jól ment a sprint alatt?
        - Mi az, amit a következő sprint során jobban lehetne csinálni?

44. Mutasd be a Scrum szerepköröket és a sprint tervezés jellemzőit!

Szerepek

- Product Owner
    - A megrendelőt képviseli
    - Biztosítja hogy a csapat az üzleti szempontból fontos dolgokkal foglalkozzon
- Scrum Master
    - Feladata hogy elhárítsa az akadályokat amelyek gátolják a csapatot abban, hogy a sprint célját megvalósítsa

    - Nem a csapat vezetője (a csapat önszerveződő)
- Team
    - A csapat azért felelős hogy a termék elkészüljön
    - 5-9 főből áll

Sprint planning

- Elvégzendő feladatok kijelölése a termék teendőlistájáról (product backlog) a
- terméktulajdonos közreműködésével.
- A sprint teendőlistájának előkészítése, amely a teljes csapat figyelembevételével részletezi az egyes részfeladatok időszükségleteit.
- Annak meghatározása és kommunikálása, hogy mennyi feladat elvégzése várható el az aktuális sprint során.

45. Mik a SOLID elvek? Röviden ismertesd mind az ötöt! Válassz ki egyet és jellemezd részletesen!

- **Single Responsibility Principle**: Egy osztály csak egyetlen dologért legyen felelős!
    - Ha ezt megsértjük, **törékeny lesz** az architektúra: egy mósodítás más, látszólag független helyen hibát eredményez.
    - Betartásához **ne engedjük meg a felesleges komplexitást** és az osztály **kohéziója legyen maximális**: minden metódusa használja az összes tagváltozót.
- **Open Closed Principle:** Egy modul nyitott kell legyen a továbbfejlesztésre (könnyű a viselkedést kiterjeszteni), de zárt a módosításra (új dolog fejlesztéséhez nem kell a régi kódhoz nyúlni).
    - Új funkció hozzáadásakor új kódot kelljen készíteni (pl. értelmes leszármaztatás) és nem a meglévő módosítani.
- **Liskov Substitution Principle:** A leszármazott osztály helyettese lehet az ős osztálynak.
    - A leszármazott mindig álljon az ős helyett: ennek megszegése pl. A **visszautasított öröklés**, amikor az altípus egy függvénye mást csinál/mellékhatással csinálja azt, amit a felhasználó elvár (az ős alapján).
    - Veszélyes lehet a valós élet beli viszonyokat egy az egyben átvenni (**reprezentánsok szabálya**: téglalap a négyszer leszármazottja?)
- **Interface Segregation Principle:** Ne függjön az osztály olyan dolgoktól, amire nincs szüksége.
    - Legjobb példája: **kövér interfészek**, felesleges metódusok implementálása “csak mert az interfészben szerepel”.
- **Dependency Inversion Principle**: A magas szintű logika nem szabad függjön az alacsony szintű részletekről, hanem fordítva.
    - Futás idejű függőségek
        - Amikor a vezérlés futás alatt egyik modult elhagyva a másikba lép be
        - Akkor beszélünk futás idejű függőségről, ha két modul kapcsolatba kerül egymással futás közben
    - Fordítás idejű függőségek (forrás kód függőség)

        - Egy név (pl. osztálynév) definiálva van egyik modulban és megjelenik egy másik modulban is
    - Dependency Inversion: ha a kód függőség iránya ellentétes a vezérlés irányával
    - Egy interfészt készítünk ami tartalmazza a függvényt.

Projektmenedzsment

46. Jellemezd a szervezeti stratégiát és viszonyát a projekthez!

- Szervezeti stratégia: Lényegében a környezet kihívásaira adott válasz
    - A fejlődés, változás tudatos alkalmazása
    - Kívánatos jövőbeni állapot, pálya
    - Döntések kiválasztása
    - Eszközök, erőforrások előteremtése

Stratégiától a projektekig

- Hierarchikus rendszer
    - Jövőkép
        - A mi cégünk a legnagyobb online cipőbolt!
    - Célkitűzések (kvalitatív)
        - Terjeszkedjünk külföldön!
    - Konkrét célok (vegyes)
        - Készítsünk testreszabott weboldalt a legnagyobb 10 országnak!
    - Stratégiai programok és akciók (kvantitatív)
        - Ki a legnagyobb 10 ország?
        - Mik a jellegzetességei?
        - Tervezzünk, implementáljunk!
- Feltételrendszer kialakítása
    - Realizálás: projektek
    - Projekt segítségével kerül a szervezet az egyik állapotból a másikba
    - Projekt menedzsment: a stratégia egy-egy jól körülhatárolható, komplex, egyszeri feladata

47. Jellemezd a lineáris-funkcionális szervezeti formát!

- A tevékenységeket a funkcionális szervezeti egységek végzik
    - Például: oktatás, kutatás, bérszámfejtés stb.
- A projektvezető a felsőbb vezetésnek felel
- Nincs projektmenedzseri hatáskör
    - Nem allokálhat erőforrást, nem utasít, nem ellenőriz (ezeket a felsővezető végezi)
    - Koordinátori szerep, információs központ
- A felelősség és hatáskör nincs összhangban
- Kicsi koordinatív kapacitás
    - Standard tevékenységfolyamatok

- Előnyök:
    - Azonos szakmai kompetenciával rendelkezők együtt dolgoznak, munkaidejük hatékonyan kihasználható
    - Egymás segítségét közvetlenül igénybe vehetik
    - Tapasztalatok megőrzése, átadása, újrahasznosítása
- Hátrányok:
    - A projektvezető hatásköre kisebb, mint a felelőssége
    - A funkcionális szakmai szempontok dominálnak a projektszempontok előtt
    - A projektfeladatokkal lassabban haladhatnak az egyes funkcionális területek, mert közben folyamatosan operatív feladatokat látnak el
    - A több területet érintő döntések nehézkesek
    - Az információ áramlás nehézkes, mert különálló funkcionális területek között kell kommunikálni

48. Jellemezd a projektorientált szervezeti formát!

- Elkülönült szervezeti egység végzi a projekt teljesítését
- A szervezeti egység összegyűjti a projekt teljesítéséhez szükséges embereket a különböző funkcionális területekről
    - a projekt időtartamára
    - a projekt futása alatt ez dinamikusan változhat
- A projektvezető a felsőbb vezetésnek felel
- Döntési, utasítási, ellenőrzési joga van
- Felelősség és hatáskör azonos szinten
- Jó koordináció, rossz skálázhatóság
- Előnyök:
    - Nincsenek prioritási problémák
    - Az szükséges erőforrásokat a projektre lehet koncentrálni
    - Hatékony információ áramlás
- Hátrányok:
    - A projekt érdek háttérbe szoríthat más, a funkcionális területek által képviselt, az egész vállalatot érintő érdekeket
    - Ha a tagok nem teljes munkaidejüket fordítják a projektre, akkor a hatékonyságuk csökken
    - A projekt szervezet ideiglenes, így a vele való azonosulás nehézkes
    - A felhalmozott tapasztalatokat nehéz megőrizni és újra felhasználni egy másik projektben

49. Jellemezd a mátrix szervezeti formát!

- Megosztott hatáskörök
- Funkcionális szervezeti egységek teljesítenek
    - hogyan és ki?
- A projektmenedzsmentnek is van hatásköre
    - mit és mikor/ra?
    - Kompromisszumok a funkcionális terület vezetői és a projekt vezetők között: kiegyensúlyozott mátrix

    - gyenge/funkcionális mátrix
    - erős/projekt mátrix
- Sérülékeny de van kohézió
- Jó koordinációs kapacitás
- Előnyök:
    - Nincsenek prioritásbeli konfliktusok, azt a vezetők tisztázzák
    - Azonos szakmai kompetenciák koncentrációja nagyobb hatékonyságot eredményez
    - Stabil struktúra biztosítja a tapasztalatok újrafelhasználását
    - Hatékonyabb információáramlás mint a funkcionális megközelítésnél
- Hátrányok:
    - A kompromisszumok törékenységet visznek be a megoldásba, ha gyakori felsőbb szintű döntésre van szükség az lassítja a projekt folyamatot
    - Ha egy kolléga sok projektben vesz részt akkor a gyakori fókusz váltás rontja a hatékonyságot
    - Ilyen esetben a elkötelezettség (projekt), csapatépítés (funkcionális terület) is nehézségekbe ütközik

50. Írj a projekt tipológiáról!

- Beruházási projektek: létesítmény
    - Új létesítmény jön létre vagy kerül módosításra
    - Műszaki paraméterekkel egyértelműen jellemezhető
    - Fizikai teljesítés, materiális jellegű
    - Kevéssé prototipizálható
- Kutatási és fejlesztési projektek: termék, technológia
    - Új termék, technológia vagy javítás
        - technológia: minden olyan eljárásmód, amely meghatároz egy tevékenységfolyamatot
    - Többnyire rögzíthető kvantitatív módon
    - Szellemi erőforrások a meghatározóak
    - Prototípusok készülnek a termékhez
- Szellemi szolgáltatási projektek
    - A szervezet működési körülményeinek, keretfeltételeinek új minősége jön létre
        - pl: privatizáció, ISO minősítés megszerzése, …
    - Nem jól kvantifikálható de modellezhető
- Külső, belső projektek: ki a megvalósító / közreműködő

51. Mi a projekt karakterisztika, kik egy projekt résztvevői?

Karakterisztika:

- cél (eredmény, követelmények, funkciók) - MIT
- időtartam (kezdet és vég határidők) - MIKORRA
- költségkeret (erőforrások) - MENNYIÉRT
- Megfelel a minőségi követelményeknek
- Elégedett a projekt szponzor is

Résztvevők (stakeholders):

- Akik a projektbe bevonásra kerültek vagy érintettek a projekt aktivitásai által
    - Projekt szponzor
    - Projekt menedzser
    - Projekt csapat
    - Támogató személyzet
    - Ügyfelek
    - Felhasználók
    - Beszállítók
    - Ellenfelek

52. Sorold fel a PMBOK folyamatcsoportjait és tudásterületeit!

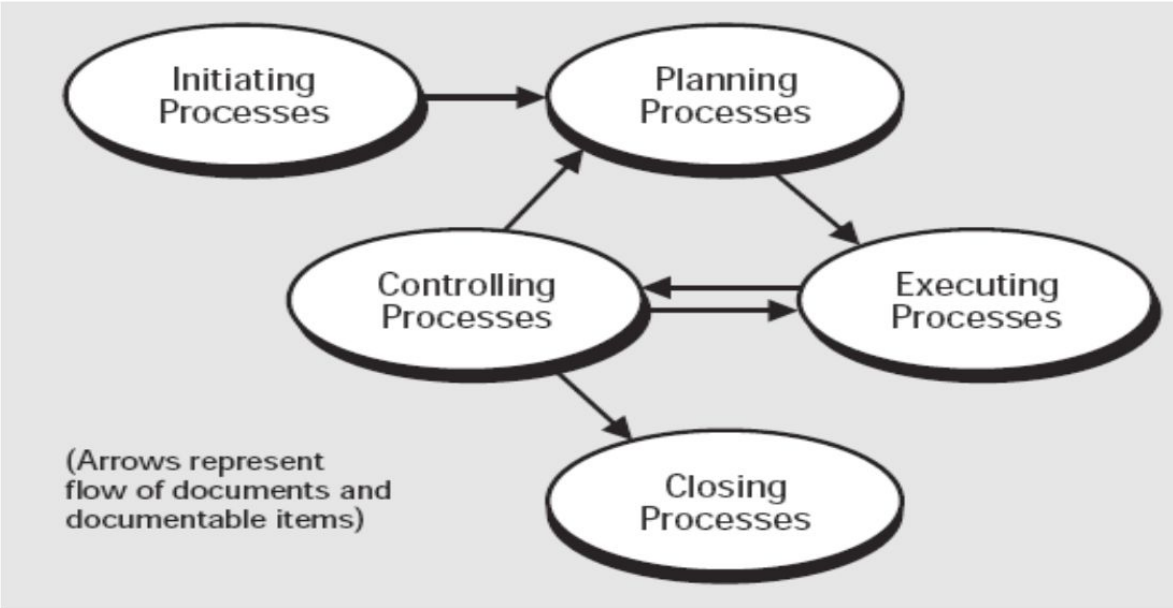
- PMBOK: The Project Management Body of Knowledge
- Széles körben elfogadott projekt menedzsment tudás, best practices gyűjtemény folyamatok leírásával, részletes ajánlásokkal, sok projekttípussal (általános, nem konkrét, testre kell szabni)
- Nemzetközi standard, 4 évente gyakorló szakemberek frissítik

Folyamatcsoportok:

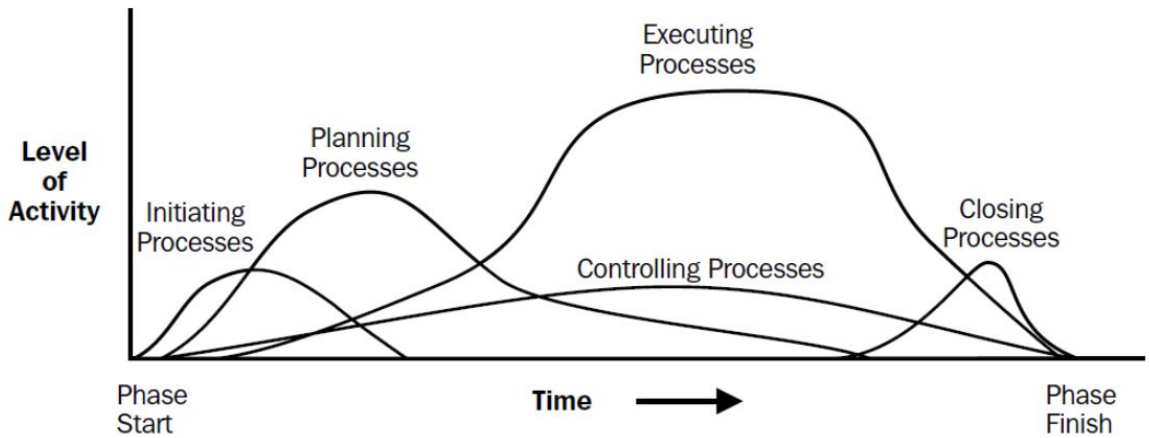
- 1. Kezdeményezés: a következő fázis elkezdése
    - Stratégiai üzleti igények felmérése
    - Variációk kialakítása
        - Megvalósíthatósági tanulmányok bekérése
        - Költség becslések
    - Variáció kiválasztása
- 2. Tervezés
    - **Szkóp tervezés:** írott szkóp dokumentum ami a többi döntést megalapozza
    - **Szkóp definíció:** a nagyobb elemek szétbontása több kisebb darabra
    - **Aktivitások definiálása:** milyen tevékenységek szükségesek az egyes projektelemek létrehozásához
    - **Aktivitások sorrendezése:** a tevékenységek függőségeinek feltérképezése
    - Aktivitások hosszának becslése: milyen hosszúak az egyes tevékenységek
    - **Ütemterv készítése:** a fenti tervek és az erőforrásigények elemzésével a projekt ütemterv elkészítése
    - **Erőforrás tervezés:** el kell dönteni, hogy az egyes projektlépések milyen és mennyi erőforrást használnak
    - **Költség becslés:** az erőforrás terv alapján a költségek megbecslése
    - **Költségvetés készítése:** az egyes munkalépések várható költségeinek meghatározása
    - **Projekt terv készítése:** konzisztens dokumentum készítése a fenti tervek alapján
    - **Minőség tervek készítése:** meg kell határozni, hogy mely minőségi mutatók érdekesek a projekt szempontjából és hogyan lehet azokat kielégíteni
    - **Szervezeti tervezés:** projekt szerepkörök, felelősségi körök és jelentési irányok meghatározása, dokumentálása, hozzárendelése

    - **Projekt személyzet kiválasztása:** az emberi erőforrások lefoglalása és hozzárendelése a projekthez
    - **Kommunikációs terv:** a projekttagok információ és kommunikáció igényeinek meghatározása: kinek milyen információra van szüksége, mikor és hogyan kapja meg
    - **Kockázatok azonosítása:** a projektet várhatóan érintő kockázati elemek meghatározása, azok jellemzőinek dokumentálása
    - **Kockázatok minősítése:** kockázatok és összefüggéseik értékelése a projekt lehetséges kimenetelei szempontjából
    - **Kockázat kezelés kialakítása:** a kockázatok csökkentésére tett lépések definiálása
    - **Beszerzési terv:** mire van szükség és mikor
- 3. Végrehajtás
    - **Projekt terv végrehajtása:** a benne foglalt lépések végrehajtása
    - **Szkóp ellenőrzés:** a projekt terjedelmének formalizált elfogadása
    - **Minőségbiztosítás:** rendszeres, átfogó projektteljesítmény értékelés hogy biztosak lehessünk abban, hogy kielégítjük a meghatározott minőségi követelményeket
    - **Csapat fejlesztés:** egyéni és csoport képességek fejlesztése a projekt teljesítmény növelése érdekében
    - **Információ megosztás:** a szükséges információ eljuttatása a projekt tagokhoz a megfelelő időben és módon
    - **Ajánlatok bekérése:** szükség szerint ajánlatok bekérése
    - **Forrás kiválasztás:** választás a potenciális szállítók közül
    - **Szerződés adminisztráció:** a szállítói kapcsolat menedzselése
- 4. Követés/felügyelet
    - **Általános változáskövetés:** változások kezelése a teljes projektben
    - Szkóp változások felügyelete: terjedelmi változások kezelése
    - **Ütemterv felügyelet:** ütemterv változásainak kezelése
    - **Költség felügyelete:** a költségek változásainak kezelése
    - **Minőség felügyelet:** a projekt bizonyos eredményeinek követése azért, hogy a projekt a minőségi követelményeknek megfeleljen és a nem kielégítő teljesítmény okait megszüntessük
    - **Teljesítmény jelentés:** teljesítmény információ (státusz jelentés, progress report, előrejelzés) összeállítás és megosztása.
    - **Kockázat kezelés:** reagálás az idő során változó kockázatokra
- 5. Zárás
    - **Adminisztrációs lezárás:** információ előállítása, összegyűjtése és megosztása egy fázis vagy a projekt formális lezárásához
    - **Szerződés lezárása:** a szerződés formális lezárása, minden nyitott kérdés tisztázása

A folyamatcsoportok összefüggése



Egy fázis során a különböző folyamatcsportok átfedései



Tudásterületek:

- **1. Integráció menedzsment:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy a többi tudásterület folyamatait azonosítsuk, definiáljuk, kombináljuk, egyesítsük és koordináljuk

- **2. Terjedelem kezelés:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy sikeres projekthez azt, és csak azt szállítsuk, amit a megrendelő kért
- **3. Ütemezés:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy a projekt befejezéséhez az ütemezés elkészüljön és megvalósuljon
- **4. Költség kezelés:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy kezeljük a projekt költségvetését
- **5. Minőség menedzsment:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy a projekt kielégítse azokat a megállapított és beleértett igényeket, amelyek miatt el lett indítva
- **6. Emberi erőforrások kezelése:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy hatékonyan használjuk a projekthez rendelt embereket. Az emberek a projekt legfontosabb eszközei.
- **7. Kommunikáció:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy időben és megfelelő módon létrehozzuk, összegyűjtsük, elterjesszük, tároljuk és megsemmisítsük a projekttel kapcsolatos információkat. A projekt alapvető fontosságú része, nélküle a projekt nehézkes.
- **8. Kockázat kezelés:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy a projekttel kapcsolatos kockázatokat azonosítsuk, minősítsük és kezeljük
- **9. Beszerzés menedzsment:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy beszerezzük és kezeljük azokat az erőforrásokat, eszközöket és javakat, amelyeket szükségesek a projekt munkához
- **10. Projekt érintettek kezelése:** Azok a folyamatok, módszerek, amelyek szükségesek ahhoz, hogy minden projekt tag bevonódjon, hozzáadjon a projekthez, megértse a döntéseket, így a projekt támogatója lesz és nem az ellensége