Szoftwertechnológiák

Egy modell a valóság egy egyszerű sített változata.
Egy jó modell kiemelkedően a lényeges részleteket és elhanyagolja az irrelevánsokat.
↳ Ezek függnek: - mire használjuk a modellt
- ki fogja olvasni a modellt

UML

Szabvány szoftverrendszerek modellezésére
↳ az Object Management Group (OMG) által kiadott szabvány
Az UML szabvány többféle diagramtípust definiál
↳ mindegyik a rendszer egy részletét adja

Két dolgot ír le: - szintaxis: hogyan kell írni, modellt
-szemantika: mit jelent a modell (mivel modellezik át a szintaktikus szabályok a valóságot)

Szerintem UML modelltípus van: • Strukturális szemantika (statikus szemantika): a rendszer egyes elemeinek jelentését definiálja egy adott időpillanatban
• Viselkedési szemantika (dinamikus szemantika): a rendszer elemei jelentésének időbeli változását definiálja

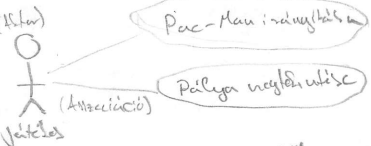
Strukturális UML diagramok: - Component diagram
- Deployment diagram
- Class diagram
- Package diagram
- Object diagram
- Composite Structure diagram
- Profile diagram

Viselkedési UML diagramok: - Use case diagram
- Activity diagram
- Sequence diagram
- Communication diagram
- State diagram
- Timing diagram
- Interaction overview diagram

Use Case Diagram

Követelmények → Tervezés → Implementáció → Tesztelés → Átadás → Karbantartás

A rendszer funkcionális követelményeit reprezentálja
- mit tesz ellene a rendszernek csinálnia, hogyan fogják a rendszert használni
- modellezi a rendszer felhasználóit és interakcióit



Aktor (actor): - Egy felhasználói szerepkör ami a rendszerrel interakcióba lép
- lehet ember, vagy külső rendszer is

Használati eset (use case): - A felhasználó és a rendszer közötti interakciót reprezentálja a felhasználói szemszögből
- Egy cél s interakciók sorozata az aktor és rendszer között

Asszociáció (association): - összekapcsolja az aktort azzal a use caseel amelyikben részt vesz

Specializáció: • Aktorok közötti: öröklődés (generalization): - a leszármazott az ős egy speciális változata
- a leszármazott mindenben use caseket tudja mint az ős is

• Use Caseek közötti: öröklődés (generalization): - a leszármazott az ős egy speciális változata
- a leszármazott use case pontosítja az ősét öröklődést
-> «extend»: - a kiterjesztő use case néhány ponton növeli a kiterjesztett use case lépéseit meghatározott kiterjesztési pontokon
-> «include»: - a tartalmazó use case egy ponton meghívja a tartalmazott use caset

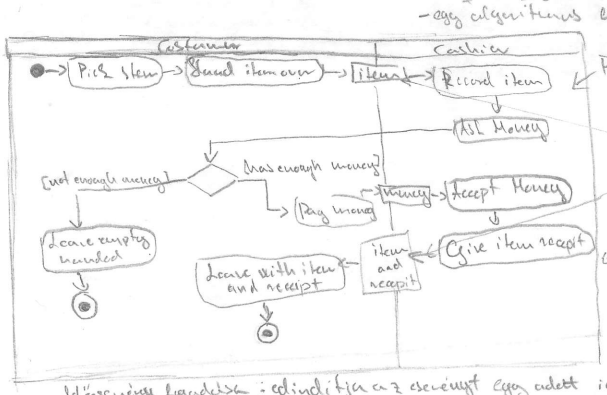
Rendszer határai (System Boundary Box): - a use case pedig a rendszer határait jelzi
- a rendszer belső részei use caseek nem részei a rendszernek

Csomag (package): célja hierarchia elemek csoportosítása

Aktivitásdiagram (Activity Diagram)

Követelmények → Tervezés → Implementáció → Tesztelés → Átadás → Karbantartás

Az aktivitásdiagramok munkafolyamatok lépéseit ábrázolják grafikusan
A fejlesztők ezt modellezésre használják: - adott use case leírása
- a felhasználó s rendszer közti üzleti vagy munkafolyamat
- egy algoritmus lépései



Partíció (partition): - egy felhasználó vagy rendszer egy része egy adott dimenzió mentén
- a sávban lévő aktivitásokat az adott sáv hajtja végre
- a partíciók lehetnek hierarchikusak is

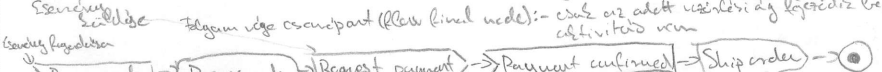
Objektum (object): - az aktivitások által használt adatot reprezentálja

Objektum él (object flow): - adatfolyamot visz át aktivitások között

Hívás aktivitás

Hívás aktivitás (call activity): - meghív egy másik aktivitást
- a meghívott aktivitást egy másik diagram is leírhatja, lehet az aktivitásdiagramnak egyszerűbb ábrázolásmódja

Folyam vége csomópont (flow final node): - csak az adott vezérlési ág fejeződik be, az egész aktivitás nem



Szerepkör kijelölése

Aktivitásokat jelzések is elindíthatnak
Esemény fogadása (accept event): - megvárja míg az esemény bekövetkezik, majd elindítja a vezérlést
Esemény küldése (send signal): - jelzést küld meghatározott eseményt fogadására alkalmas

Időeseményt fogadása: elindítja az eseményt egy adott időpillanatban

Komponens diagram

Követelmények → Tervezés → Implementáció → Tesztelés → Átadás → Karbantartás

Egy szoftverkomponens a szerepeztési alapegysége, szerződésszerűen specifikált interfészekkel rendelkezik s csak explicit módon függ a környezetétől. Egy szoftverkomponens önállóan telepíthető és használható fel más szerepeztési egységekben történő felhasználásra alkalmas.
A komponensdiagram egy rendszer komponenseit és a közöttük lévő kapcsolatot mutatja.

A összetett komponenst más komponensek implementálják

Telepítési diagram (Deployment Diagram)

Követelmények → Tervezés → Implementáció → Tesztelés → Átadás → Karbantartás

A telepítési diagram a rendszer fizikai architektúráját ábrázolja
A telepítési diagram segítségével a rendszer hardware és szoftware topológiáját is ábrázolhatjuk
A rendszer architektúrájának csomópontjaival (nodes) s a közöttük lévő kommunikációs kapcsolatokkal írjuk le (communication path)
Egy csomópont lehet egy hőgépet reprezentáló erőforrás vagy a szoftverfejlesztési folyamat, vagy pedig a rendszer működése állít elő vagy használ fel, ez egy komponens
megjelenítését is jelentheti, lehet ezek csomópontjaira telepítendő fizikai egységeket is definiálni.
A telepítendő artifactok deploy kapcsolattal is lehet a node-okhoz kapcsolni.

Osztálydiagram (Class Diagram)

Az OO modellek kialakításának szabványos módja, a leggyakrabban használt UML diagram. Az osztálydiagramok szerkezetileg leírják rendszerünk

Az osztálydiagram elemei:
- Osztály (class): • Objektumok leírása viselkedését írja le
  • Viselkedés: Operációk
  • Állapot: Tulajdonságok, értékek
- Interfész (Interface): • publikus operációk halmaza melyek egy adott viselkedést írnak le } lazán mennek összefüggés
  • nem definiál implementációt, egy osztály implementálhatja

Lapcsolatok az elemek közt: • Megvalósítás: Osztály megvalósít interfészt ---▷
• Öröklődés: Osztályok v. interfészek közt ——▷
• Asszociáció: Egy osztályon változást egy másik észlelhet ——→
• Függőség: Egy osztály függhet egy másik osztálytól (pl. ——→ / ——)

Az operációnak s tulajdonságoknak láthatósága: • private (-)
• protected (#)
• public (+)
• package (~)

Operáció példa: + add (left: double, right: double): double
Tulajdonság példa: - name: String [0..1] = null ∈ kezdőértékadás

A statikus tagokat aláhúzással jelöljük.

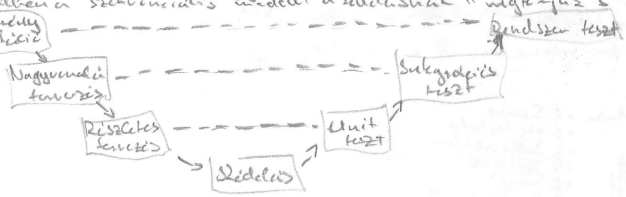
Az elemek nevükön, jelölés után attributumaikon, operációikon keresztül érhetők el.

A függőség azon a tényen alapul, hogy egy osztályunk használ egy másik osztályt, ekkor az osztály változása befolyásolhatja a függő osztályunk működését. Ilyen pl. egy operáció paramétere, lokális változója, példányosítás.

Az asszociáció nem tűnik el függőségnek akkor, ha egy osztály típusú változót is tartalmaz. Ekkor is asszociáció áll fenn, de ez is függőséget jelent

...ció s üzemeltetés s időnként egy beállítási fázis.
...szeftverfejlesztési módszertanok a szeftverfejlesztés tevékenységeit modellezik.
...szeftver életciklus modellek típusai: • szekvenciális modellek
• Sukcesszív modellek
• Iterativ modellek

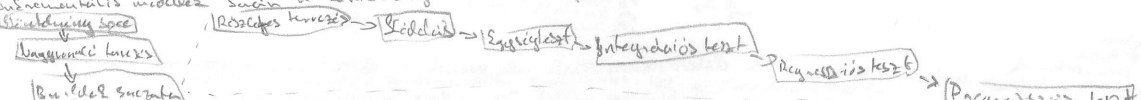
...szekvenciális modellek lineárisan szekvenciális folyamatként ábrázolták, fázisokra különül el nelyek nem lapolódnak át. Sőt ismert példa: vízesés, s V modell
...vízesés modellben a fázisok egyenerőt követik s az előző fázis elmenete a következő fázis bemenete.
...V modellben a szekvenciális modellt a kódolásnál "megtörjük s visszahajtjuk".



vízesés modell

V modell

...nkrementális modellek során a követelményeket kisebb csoportokra bontják s erre értelmezik a tervezés, kódolás, tesztelés lépéseket.



Definiálás	Értékelés
A következő fázis tervezés	Fejlesztés s unifikáció

Spirális modell

Inkrementális modell

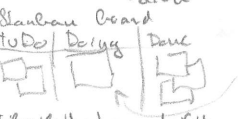
...iterativ modellek ciklikusak, egy kisebb csoportját implementálják a követelményeknek egyszerre.
...MP: egy iterativ, inkrementális ... csak vezérelt életciklus modell/módszertan.
...tném programozás (XP), ggors változtatás, rugalmas projektek módszertana. Tanú elkészítésekor ugorsan buildeket készítenek.
...egy másik életciklus modell a TDD (Test Driven Development). A programozó először a teszteket írja meg.
...Scram egy management eszköz nagy hosszú felmentek részletre bontását csinálja. A beatrecket egy sorrendben részletesebben egyesével implementálják.
...Általános szabvány: ISO/IEC 12207:2017 kompatibilis a V modellel. Nem ír le konkrét modellt de minden modell felépíthető a szabvány folyamatokat.
...Nagyonalú szeftverfejlesztés: szekvenciális, több dokumentációt igényel. Agilis szeftverfejlesztés: iterativan történik, kevésbé formális, elfogadva a követelmények
...szeleges változásait. A folyamatokat is részt vesz a fejlesztésben.
...Agilis alapelv: 1) A vevő elégedett legyen
2) A követelmények változhatnak
3) Gyakran szállítsunk
4) A fejlesztők dolgozzanak együtt
5) Építsük motivált egyénekre a projektet
6) A szemleges beszélgetés a leghatékonyabb inté átadására
7) A haladás mércéje a működő szeftver
8) Az agilis módszertanok elősegítik a fenntartható fejlesztést
9) Folyamatos figyelem a technikai kiválóságokra s jó tervezésre
10) Egyszerűség
11) Összeszerűdő csoportmunkák
12) Változtatható, viselkedés.

...Az agilis az időt helyezi előtérbe. A szeftverfejlesztésen belül mindent lehet agilisan csinálni. Az agilis csapat kevesz funkcionális s önszerűdő, egyéltt
...tezik.
...Egy másik filozófia a lean. Lényege azon folyamatok identifikálása s eltávolítása melyek nem teremtenek értéket.
...7 pazarlás: • Felig kész munka
• Felesleges funkciók
• Feladat átadása másnak
• Feladat változtatása
• Újratanulás
• Várakozás, csúszás
• Hiba

} Ezeket szeretné eliminálni a lean

Szervezet -> érettség
szeftver -> képesség

Kanban board



max 1 cetli!

...Filozófiák: Lean, Agilis
...Módszertanok: Scram, Kanban, XP
...Szerepek: sprint, napi Scram ...
...Szeftverfejlesztés: folyamatok célja a szeftverrel kapcsolatos folyamatok hatékonyságának növelése. A szeftverfolyamat érettsége: egy folyamat mennyire definiált, tervezett,
...követett, mért, ellenőrzött s hat ékony. Szerszer érettség helyett képességről beszélünk. Egy szervezet annál érettebb minél több érett folyamatot működtet.
...Ha egy szervezetnek érettsége egy szeftvernek pedig képessége van. CMM = Capability Maturity Model. CMMI = Capability Maturity Model
...Integration. A képesség érettségi modellek jól meghatározottak.
...Fele képességi érettségi modell van: • Lépesés modell: a teljes szervezetet vizsgálják, SW-CMM
• Folyamatos modell: egyes folyamatokra koncentrálnak, SPICE, Automotive SPICE
• Kombinált modellek: a fenti kettőt kombinálják, CMMI

...CMM egy képesség-érettségi modell. A egyéknek 1-5 szintet érnek el. 1: A projektek tipikusan átlépik az idő s költség keretet. 2: Az egyes emberek munkájukat
...sémelhetnek tudják, megjelennek bizonyos irányítási folyamatok. 3: Az egyes emberek jó gyakorlatait dokumentálják s ezt az egész szervezetben követik
...A megjelenik a peer review. 4: A folyamatok egyre hatékonyabban s ezt mérve hajtják végre. 5: A szervezet képes optimalizálni a saját folyamatait.
...-> kaotikus
...-> ismételhető
...-> meghatározott
...-> mért, statisztikailag menedzselt
...-> optimalizálható.
...SPICE (Software Process Improvement and Capability dEtermination) egy szervezet egyéni folyamatainak vonatkozik. Létrejött az ISO/IEC TR 15504 szabvány
...SPICE képesség szintjei: • 0. Nem teljes
• 1. Végrehajtott
• 2. Menedzselt
• 3. Meghatározott/bevezetett (dokumentált)
• 4. Jósolható (mért)
• 5. Optimalizálé (előző tapasztalatok alapján fejlesztett)

A SPICE folyamatok: • Vevői/beszállítói
• Mérnöki/fejlesztési
• Projekt menedzsment
• Támogatói
• Szervezeti
• stb. amiben a ágnál van

...SPICE folyamat dimenziói a folyamatokat írják le, a képesség dimenzió pedig a folyamattal kapcsolatos gyakorlatokat mérik.
...CMMI egy integrált modell összekapcsolja a CMM s a SPICE folyamatmodelleket, vonatkozhat az egész cégre, vagy csak egy folyamatra. Minden folyamatunk kell
...írás, ennek vannak kötelező s nem kötelező elemei, a kötelező elemek a célok.

Folyamatok ← Érettségi szintek

Folyamatok → Sajátos célok → Sajátos gyakorlatok
Folyamatok → Általános célok → Általános gyakorlatok

Az érettségi szintek a szervezet egészére vonatkoznak, azonosak a CMM modellel.
Az érettségi szintek a folyamatokra vonatkoznak: • 1 Nem végrehajtott
• 2 Végrehajtott
• 3 Menedzselt
• 4 Definiált

...folyamatokat típusok szerint is csoportosíthatjuk CMMI-ben (folyamat menedzsment, projekt menedzsment, támogató, mérnöki (fejlesztési))
...CMMI képességi szintek: 1. a folyamat elkészült (CL1)
2. egy menedzselt folyamat intézményesítése + CL1 (CL2)
3. egy folyamat meghatározott s a hozzá tartozó infókat gyűjtik + CL2 (CL3)
...CMMI-nek való megfelelő auditálása SCAMPI-vel történik (Standard CMM, Appraisal Method for Process Improvement). 3 osztálya van csökkenő
...zigorral: A, B, C. Az audit eredménye 3 évig érvényes.
...TMMI modell a CMMI-n alapuló tesztelési folyamatok érettségét vizsgálé modell.
...CMMI s a TMMI is alkalmazható agilis környezetben.
...Követelmény: olyan feltétel vagy képesség amely a felhasználó szemszögére azért szükséges, hogy megoldjon egy problémát vagy elérjen egy cél. Követelmény fejlesztés
...requirements engineering (RE), ennek 4 alapterekonyiége: • Meghatározás, egyeztetés, felmérés
• Dokumentálás
• Validáció
• Követelmények menedzsmentje

...CMMI is tartalmazza a követelmény fejlesztést 2-es s 3-as érettségi szinten. 2-es szinten követelmény menedzsment. Követelmény megértése, követelmények kezelése
...Követelmény változások menedzselése kötirányú követhetőség fenntartása, követelmények s projekt közti ellentmondások azonosítása. 3-as érettségi szinten követelmény
...fejlesztése célja a vevői, tanúsz s tanúsz-szempontok követelmények felmérése, elemzése s dokumentálása. A követelmények szerkezhatnak felhasználóktól, más tanúsz
...rendszerek, általános tapasztalat. Fontosak a vevő által explicitan nem mondott követelmények. Hasznos a követelmények forrását folyamatok. Fontos a követelmények
...egyeztetés s elemzés/priorizálni. Követelmények két típusa van: funkcionális (mit csinál), nem funkcionális (hogy csinálja) (minőségi követelmények)
...Functional Requirements Specification dokumentum tartalmazza a funkcionális követelmények. Véhány nem funkcionális követelmény fiszámíthatóság használhatóság
...tekenysége. A konkrétek is követelmények s egy külön követelmény típust alkotnak szerűjük a tervezési s implementációs megoldásokat követelményeken
...modell. A követelmények 3 szempontból modellezhetjük: adatok, funkcionalitás, viselkedés. A követelmények modellezése közben történik a követelmények priorizálása.
...Tradicionális priorizálás ben. kötelező, kívánatos, opcionális. A modellek leírását a követelmények dokumentálását szolgálják. Követelmények hasznos mérőszámai
...A követelmények menedzsmentje a követelmények gondis kezelése. A kötirányú követhetőség azt jelenti, hogy tudjuk a kapcsolatot a követelmények-kód-tesztesetek
...Sőt.
...Tervezés az a folyamat melyben a követelmények alapján megértjük a rendszert szet a megértést formalizáljuk. Tervezés során általában 2 nézetet
...Tanúsz le: statikus s dinamikus. Szeftver architektúra terv, szeftver részletes terve. Tervezés során modelleket készítünk a követelmények alapján. Különösen
...Ha kell figyelni a szeftver felbontására/szerkezetére tervezés során. Egy architektúra stílus a szeftver szerkezetének magas szintű leírását nyújtja.
...Jó tervezés néhány alapelve: • Tanulhatóság
• Szerős a felhasználónak
• Kiszámítás
• minimális megepetés
• visszaállíthatóság
• felhasználó támogatása
• felhasználói sokféleségre figyelés

...CMM-ben a Design kapcsolódik a követelmény fejlesztéshez, műszaki megoldás. Fontos a Design dokumentációja is. Egy terv jó ha szabványos, kiszámítás s
...nem redundáns s design után jön az implementáció. A belső fontosak a tervezés során is. A funkcionálisok számolás a rendszer mérőtérnek egy számolása
...Az egy jó belső mértékegység. A funkcionálisok számolási módszer. Albrecht módszer. Cosmic módszer. Az utóbbi a modernebb s jobb rendszer. Agilis design
...MX = Tech + Feel + Usability. Az agilis designnál lebeszik.

Hiba: különbség, ami nem felel meg a követelményeknek vagy specifikációnak.
Szindróma: probléma, programhiba.
Verifikáció: az adott követelmény teljesülésének vizsgálata
Validáció: a rögzített követelményeknek megfelelően működik a szoftver
A szoftvertesztelés 7 alapelve:
1. Hibák kétségtelen hiánya → nem jelenti, hogy hibátlan a rendszer
2. Nem lehetséges kimerítő teszt
3. Korai tesztelés
4. Hibafürtök megjelenése
5. Féregirtó paradoxon → ugyanazok a tesztek kiirtják ezeket a hibákat, érdemes újraírni a teszteseteket
6. A tesztelés függ a körülményektől
7. A hibamentes rendszer tévhite → hiába hibátlan, ha nem éri el a követelményt

A tesztelés tevékenységei:
- Teszttervezés
- Teszt felügyelet és irányítás
- Tesztelemzés
- Műszaki tesztelemzés
- Teszt megvalósítása
- Teszt végrehajtás
- Teszt lezárás

Tesztelési jegyzőkönyv elemei:
- Tesztelési irányelvek
- Teszterv
- Tesztelési/hibajegyzőkönyv
- Teszteset
- Tesztszkript
- Tesztadat
- Tesztelési jegyzőkönyv
- Hibalista, hibák leírása
- Átadási/átvételi dokumentumok

Hibák rangsorolása:
- 1. Enyhe
- 2. Mérsékelt
- 3. Kellemetlen
- 4. Zavaró
- 5. Súlyos
- 6. Nagyon súlyos
- 7. Extrém
- 8. Elfogadhatatlan
- 9. Katasztrofális
- 10. Fertőző

Jelentéktelen / jelentős teszt

Tesztek sokféleképp csoportosíthatók, pl.: teszt alanya szerint, alkalmazott technikák szerint, életciklusmodellbeli hely szerint. Az ISTQB szerint 4 teszt szint van: Komponens, Integrációs, Rendszer, Átvételi teszt. Ezen kívül beszélhetünk manuális és automatizált tesztelésről is. Statikus tesztelés: teszt program futtatása nélkül. Dinamikus tesztelés: teszt program futtatása közben. Statikus tesztelés: fekete doboz, fehér doboz is; statikus elemzés, átvizsgálás
A tesztelési technikák segítenek a teszt eseteket kiválasztani. Funkcionális teszt, fehér doboz tesztelés: strukturális tesztelés
A dinamikus tesztelés két típusa: fekete doboz: funkcionális teszt, fehér doboz: strukturális tesztelés
Határérték analízis alapelvei: 1. Ha a változók értéktartományt alkotnak, akkor a határérték értékekkel teszteljünk: min, min+, nominális, max−, max
2. Ha a hibák pontosan egy változó generálja, legyen minden min+, max− és egy min/max értékkel teszteljünk
n változós függvény esetén határérték analízisben 4n+1 esetet kell. Közben figyeljük a min− és a max+ esetek is, ez a robusztussági tesztelés. A legrosszabb eset tesztelés
[illegible] feltételezzük, hogy csak egy dolog okozza a hibát ⇒ 5^n tesztesetet
Ekvivalencia partícionálás során a bemeneteket ekvivalens partíciókra bontjuk, s a tesztesetekhez mindegyik partícióból egy elemet választunk, ezzel garantáljuk a teljes lefedettséget
A fekete doboz tesztelésben van döntési tábla, ezt is ide lehet tartozni
A fekete doboz tesztelésben állapotátmenet teszt is lehetséges, ugyanis helyes és helytelen átmeneteket tesztelünk
A fekete doboz tesztelésben use case teszt is létezik, mely az adott use caseket követi végig, ehhez feltesszük a jó use case-t.
Egy másik fekete doboz tesztelési módszer a tapasztalat alapú tesztelés
Strukturális tesztelés során a lefedettség számítását fontos. A tesztesetekhez a programmal gráfot építünk. A gráf csomópontjai az utasítások
A döntés-döntés gráfban (DD Path) a döntések közt szereplő utasítások kerülnek végig, s csak döntések lehetnek csomópontok
Az adatfolyam tesztelés azokat a pontokat keresi, ahol a változók értéket kapnak, s ahol ezeket az értékeket felhasználják. Ugyanis a kódot változóként szeletelik s tesztelik
Változtatások után regressziós s progressziós tesztelést végezni.
A projekt tevékenységek egy egységes cél érdekében. Az alapvető projektvezetői feladatok: Plan, Do, Check, Act. A projektmenedzsment alapvetően kilenc összetevő területtel foglalkozik: Integrációs menedzsment, Terjedelem menedzsment, Ütemezés menedzsment, Költség menedzsment, Minőség menedzsment, Emberi erőforrás menedzsment, Kommunikációs menedzsment, Kockázat menedzsment, Beszerzés menedzsment. Projektmenedzsment módszertanok: PRINCE, PRINCE2, Rational Unified Process, Scrum
CMMI-ben a projektvezetés 2. szintű követelmény.

A Gantt diagram egy projekt vezetői diagram, mely a tevékenységek időbeliségét szimbolizálja.
Az elsőbbségi háló egy precedencia gráf a projekt folyamatairól. Azon tevékenységek, melyek kellenek a kritikus tevékenységek, kritikus út. Ez azt jelöli a projekt minimális átfutási idejét.
Tevékenységeket ábrázolunk ve dobozzal

Legkorábbi kezdés	Időtartam	Legkorábbi befejezés
Tevékenység száma Tevékenység leírás		
Legkésőbbi kezdés	Teljes "időjáték"	Legkésőbbi befejezés

Work Breakdown Structure
Szervezeti lebontási struktúra

Adott időszakra ütemezett munka tervezett költsége (BCWS Budget Cost of Work Scheduled)
Felhasznált költségek (ACWP Actual Cost of Work Performed)
Megvalósult érték (BCWP Budget Cost of Work Performed)
Cost variance = BCWP − ACWP
Schedule variance = BCWP − BCWS
Cost Performance Index = BCWP / ACWP
Schedule Performance Index = BCWP / BCWS

} Ezek alapján becsléseket adhatunk részleges állapotból

Az agilis projektmenedzsmentet olyan emberek végzik, akik értenek az adott projekthez is

Agilis projektekben a projektvezetési, fejlesztési tevékenységeket gyakran a műszaki tevékenységekkel együtt kezeljük.
A Burndown chart a sprintből hátralévő munkát mutatja.
Agilis projekt indítás: Inception deck:
1) Miért vagyunk itt?
2) Elevator Pitch
3) Termék doboz
4) "Nem" lista
5) Ismerd a szomszédokat
6) Mutasd be a megoldást
7) Mi nem hagy aludni?
8) Becsüld fel
9) Mit adunk?
10) Mibe fog kerülni?

A konfigurációmenedzsment a fejlesztői munkák integrálásának menedzselését jelenti. Megjelenik a CMMI 2. szinten. Ma a verziókezelés a konfigurációmenedzsment (CM) része. Kétféle verziókezelő rendszer van: központosított s elosztott
A kockázatmenedzsment lépései:
- Kockázatok azonosítása
- Kockázat elemzése
- A kockázati tényezők fontossági sorrendje
- A kockázat elhárítását célzó intézkedések kidolgozása → elkerülés, csökkentés, kompenzálás, megegyezés
- Az intézkedés bevezetése
- Az intézkedés követése

A kockázatkezelés a CMMI-ben 3. szintű követelmény. Kockázat értékelése agilisan = probability · impact

- függőség
- asszociáció
- aggregáció
- kompozíció